

Systemy mikroprocesorowe

Instrukcja laboratoryjna

Spotkanie 3

Interfejsy komunikacyjne

Autorzy: dr inż. Paweł Dąbał,
dr inż. Krzysztof Sieczkowski
mgr inż. Sylwia Zawadzka

Ostatnia aktualizacja: 10.12.2021 r.

Wersja: 1.0.3

Spis treści

1.	Wprowadzenie.....	3
1.1.	Cel ćwiczenia	3
1.2.	Wymagania wstępne.....	3
1.3.	Opis stanowiska laboratoryjnego.....	3
1.4.	Wprowadzenie teoretyczne	3
1.4.1.	Interfejs szeregowy UART	4
1.4.2.	Interfejs I2C	5
1.5.	Sposób realizacji ćwiczenia laboratoryjnego.....	7
2.	Zadania podstawowe do realizacji (12 pkt.)	9
2.1.	Zagadnienia wstępne	9
2.1.1.	Dołączenie do wirtualnej klasy i utworzenie repozytorium bazowego dla zadania	9
2.1.2.	Pobranie repozytorium i konfiguracja lokalna	9
2.1.3.	Konfiguracja środowiska STM32CubeIDE	10
2.1.4.	Opis projektu bazowego.....	11
2.2.	Obsługa interfejsu szeregowego (6 pkt.).....	11
2.2.1.	Konfiguracja modułu LPUART oraz standardowego wejścia/wyjścia (3 pkt.)	11
2.2.2.	Transmisja danych w trybie blokowania i przerwania (3 pkt.)	13
2.3.	Obsługa interfejsu I2C (6 pkt.).....	14
2.3.1.	Konfiguracja modułu I2C1 i odczyt temperatury (3 pkt.)	14
2.3.2.	Konfiguracja modułu I2C3 i odczyt z układu akcelerometru (3 pkt.).....	16
3.	Zadania rozszerzające do realizacji (14 pkt.).....	19
3.1.	Rozszerzona obsługa interfejsu szeregowego (7 pkt.)	19
3.1.1.	Konsola wiersza poleceń (3 pkt.).....	19
3.1.2.	Generowanie próbek funkcji sinus (4 pkt.).....	19
3.2.	Rozszerzona obsługa interfejsu I2C (7 pkt.)	19
3.2.1.	Zaawansowane funkcje czujnika temperatury STLM75F (3 pkt.)	19
3.2.2.	Zaawansowane funkcje czujnika przyspieszenia LSM303C (4 pkt.).....	20

1. Wprowadzenie

Instrukcja ta zawiera zadania związane z tworzeniem programów obsługujących interfejsy komunikacyjne dostępne w mikrokontrolerze. W trakcie zajęć student będzie korzystał z środowiska programistycznego [STM32CubeIDE](#), rozbuduje dostarczony projekt aplikacji dla mikrokontrolera STM32L496ZGT6 umieszczonego na płycie uruchomieniowej *KAmLeon* o funkcjonalność umożliwiającą obsłużenie interfejsów komunikacyjnych. Ponadto będzie potrafił obsługiwać narzędzia wspomagające kontrolę wersji oprogramowania w celu dokumentowania własnych postępów.

1.1. Cel ćwiczenia

Ćwiczenie laboratoryjne ma na celu:

- nabycie umiejętności konfiguracji stanowiska pracy w oparciu o oprogramowanie [STM32CubeIDE 1.7.0](#);
- poznanie płyty uruchomieniowej [KAmLeon](#) z mikrokontrolerem [STM32L496ZGT6](#);
- przypomnienie i usystematyzowanie wiedzy i umiejętności z zakresu posługiwania się językiem C;
- przypomnienie wiedzy z zakresu budowy wyświetlaczy siedmiosegmentowych oraz tekstowych LCD;
- praktyczne korzystanie z systemu kontroli wersji [Git](#) i serwisu [GitHub](#).

1.2. Wymagania wstępne

Przed przystąpieniem do wykonywania ćwiczenia laboratoryjnego należy :

- zapoznać się z schematem płyty uruchomieniowej [KAmLeon](#) oraz dokumentacją mikrokontrolera STM32L496ZGT6 ([Product Specifications](#), [Reference Manuals](#));
- zapoznać się z składnią języka C – pojęcie zmiennej, stałej, funkcji, prototypu funkcji, parametru funkcji, wyrażenia warunkowego, pętli, wskaźnik, struktury i tablicy;
- zapoznanie się z sposobem działania interfejsu UART oraz I2C;
- utworzyć konto na platformie [GitHub](#);
- zapoznać się z dokumentacją środowiska [STM32CubeIDE](#) i biblioteki [STM32CubeL4](#).

1.3. Opis stanowiska laboratoryjnego

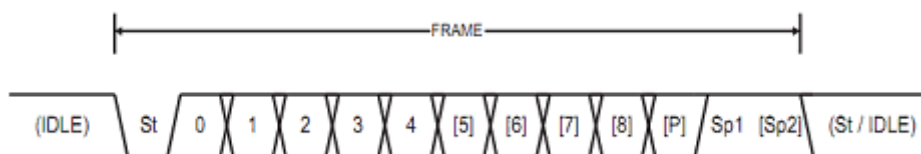
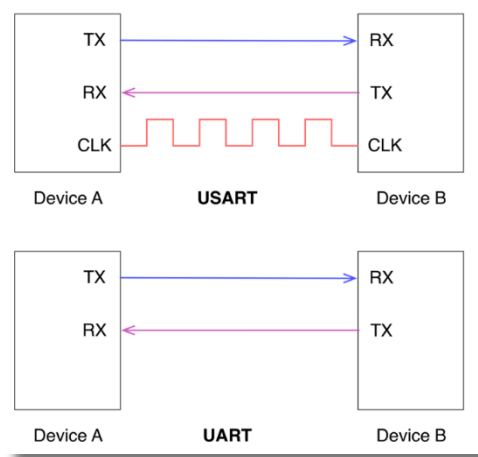
Stanowisko do przeprowadzenia zajęć składa się z komputera PC z zainstalowanym oprogramowaniem koniecznym do realizacji zajęć: *STM32CubeIDE*, dedykowaną do układu biblioteką *STM32CubeL4* i systemem *Git* oraz płyty *KAmLeon* podłączonej do komputera za pomocą przewodu USB do złącza programatora *SWD PRG/DBG/VCOM* płyty uruchomieniowej. Połączenie to również odpowiada za zasilanie płyty. Ponadto w trakcie zajęć użyty może być oscyloskop cyfrowy.

1.4. Wprowadzenie teoretyczne

Współczesne mikrokontrolery wyposażone są w szereg różnych interfejsów. Podzielić możemy je na dwie zasadnicze grupy: lokalne łączące układy scalone w obrębie urządzenia (UART, SPI, I2C, SAI) oraz rozległe łączące różne systemy cyfrowe (UART, CAN, USB).

1.4.1. Interfejs szeregowy UART

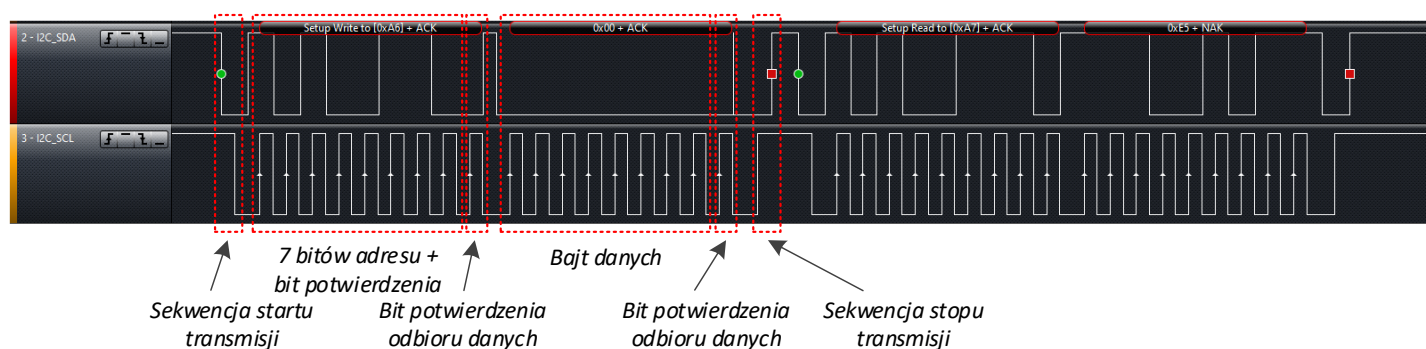
Interfejs szeregowy UART stanowi podstawowy interfejs komunikacyjny występujący we współczesnych mikrokontrolerach. Standard ten wywodzi się ze standardu RS232, który został opracowany w 1969 r. przez *Electronic Industries Association*. UART pozwala na dwukierunkową, asynchroniczną transmisję danych pomiędzy dwoma urządzeniami pracującymi w tym standardzie. Asynchroniczna transmisja w odróżnieniu od transmisji synchronicznej nie posiada dodatkowego sygnału synchronizującego (**CLK**). Zatem do dwukierunkowej transmisji danych w standardzie UART wymagane są tylko dwie linie transmisyjne, tj. **RX** oraz **TX**. Linia **RX** stanowi wejście danych napływających z zewnętrznego urządzenia, natomiast linia **TX** stanowi wyjście wyprowadzanych danych wyjściowych. Urządzenia komunikujące się w standardzie UART często określane są mianem *Host* oraz *Device*, gdzie *Host* w przypadku transmisji synchronicznej traktowany jest jako urządzenie nadrzędne. W transmisji synchronicznej *Host* zapewnia sygnał zegarowy. Ponieważ w transmisji UART nie występuje sygnał zegarowy to urządzenia te nie mają określonych priorytetów działania i mogą niezależnie od siebie, w dowolnym czasie transmitować dane. Podstawowa ramka transmisyjna, stosowana w komunikacji UART przedstawiona została na poniższym rysunku.



Brak transmisji UART na linii określane jest przez występowanie na niej stanu wysokiego. Urządzenie chcące wysłać ramkę pierwotnie wysyła bit startu (ustawia linię w stan niski), następnie transmitowane są dane (od 5 do 9 bitów), w zależności od konfiguracji interfejsu, wysyłany jest bit parzystości. Zakończenie transmisji realizowane jest poprzez wysłanie bitu stopu (ustawienie linii w stan wysoki). Wszystkie bity danych przesyłane są od najmniej znaczącego (*LSB*) do najbardziej znaczącego (*MSB*) bitu danych. Format ramki, gdzie występuje bit startu, 8 bitów danych, brak bitu parzystości oraz bit stopu o rozmiarze 1 bita danych, określany jest w skrócie: *8N1* (8 bitów danych, *N* – brak bitu parzystości, bit stopu o rozmiarze 1 bitu). Format ramki *8N1* jest najczęściej stosowanym formatem danych, który występuje w transmisji UART. Rozmiar każdego bitu danych jest ściśle związany z szybkością transmisji (ang. *baud rate*). Szybkość transmisji pozwala na określenie czasu trwania poszczególnego bitu jest to tzw. *Bod*, które mogą być przetransmitowane w czasie 1 sekundy. Powszechnie stosowanymi szybkościami transmisji UART są: 9600, 19200, 38400, 57600, 115200. W wielu systemach cyfrowych możliwa jest niestandardowa konfiguracja prędkości do np. 1 Mb, 2 Mb. Ważne, aby szybkość transmisji wyrażaną w *Bod*-ach, nie mylić z szybkością transmisji wyrażaną w bajtach, gdyż np. w konfiguracji *8N1* do przesłania 1 B danych wymagane jest przesłanie 10 *Bod*. Obecnie transmisja danych przy pomocy standardu UART najczęściej stosowana jest do komunikacji systemu z użytkownikiem. Zatem, transmitowanymi danymi są najczęściej litery zakodowane w kodzie ASCII. Nie ma natomiast przeciwwskazań do stosowania interfejsu UART do transmisji danych binarnych. Na poniższym rysunku przedstawiono zrzut

zegarowego) na poziomie 100kHz. Występują również kolejne wersje standardu I²C, które m.in. oferują zwiększoną szybkość transmisji sięgającą nawet pojedynczych MHz. W niniejszym laboratorium zastosowana zostanie podstawowa szybkość transmisji (100kHz). Istotnym wymogiem transmisji I²C jest stałość poziomu logicznego na linii *SDA* podczas trwania stanu wysokiego na linii *SCL*. Zatem aktualny stan występujący na linii *SDA* musi być stabilny najlepiej już nieco przed zboczem narastającym sygnału *SCL* i utrzymywać się nieco po wystąpieniu zbocza opadającego sygnału zegarowego.

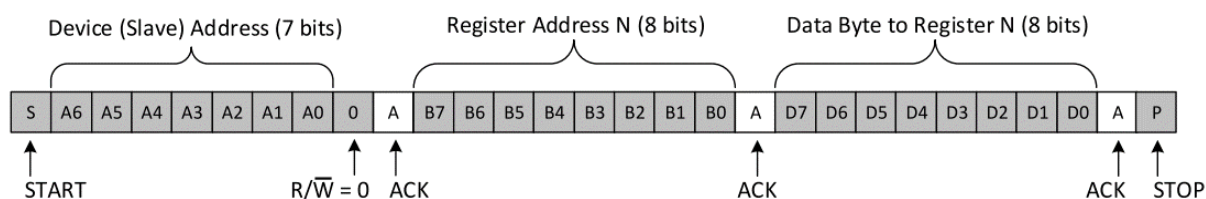
Rozróżnianie wielu urządzeń *slave* występujących na jednej magistrali możliwe jest przy pomocy indywidualnych adresów urządzeń. Każde urządzenie *slave* posiada własny 7 lub 10-bitowy adres. Stan niezajętej magistrali widoczny jest poprzez dłuższe utrzymywanie się stanu wysokiego na liniach *SCL* i *SDA*. Rozpoczęcie transmisji realizowane jest poprzez zmianę stanu linii *SDA* z wysokiego na niski, podczas trwania wysokiego stanu na linii *SCL*. Zatrzymanie transmisji odbywa się poprzez zmianę stanu na linii *SDA* ze stanu niskiego na wysoki podczas występowania wysokiego stanu na linii *SCL*. Prawidłowa transmisja danych występuje po wysłaniu znaku startu. Stan magistrali *SDA* nie zmienia się w momencie występowania stanu wysokiego na *SCL*. Zmiana stanu na magistrali *SDA* może nastąpić podczas stanu niskiego na linii *SCL*. Na poniższym rysunku przedstawiono przykładowy przebieg transmisji I²C uzyskany z analizatora stanów logicznych.



Każdą transmisję rozpoczyna znak startu i kończy znak stopu. Informacje wysyłane linią *SDA* mają rozmiar 8 bitów (1 bajt). Dane wysyłane są od najbardziej znaczącego (*MSB*) do najmniej znaczącego (*LSB*) bitu. Każdy wysyłany bajt musi być potwierdzony przez urządzenie odbiorcze przy pomocy bitu potwierdzenia. Bit potwierdzenia występuje zawsze po 8 bitach danych. Jeżeli bit potwierdzenia jest w stanie wysokim to urządzenie odbiorcze nie potwierdziło faktu odebrania danych lub urządzenie odbiorcze nie będzie dalej odbierało danych. Jeżeli natomiast bit ten jest w stanie niskim to urządzenie odbiorcze poprawnie odebrało dane i domyślnie jest gotowe na odbiór kolejnych ramek I²C.

W kompletnej transmisji I²C wyróżniamy dwa typy ramek: ramkę zawierającą adres urządzenia wraz ze wskazaniem zamiaru transmisji oraz ramkę danych która zawiera 1 bajt informacji. Każdy typ ramki zawiera bit potwierdzenia transmisji. Transmisja danych za pośrednictwem interfejsu I²C ukierunkowana jest albo na odczyt danych z urządzenia *slave* przez *master* albo zapis danych w urządzeniu *slave* przez *master*. W zależności od zamiaru transmisji całkowity przebieg transmisji zawierający ramki danych nieznacznie różni się pomiędzy sobą. Odczyt danych przez *master* realizowany jest najczęściej w poniższy sposób. Po sekwencji startu wysyłany jest 7-bitowy adres urządzenia *slave*. Za tym adresem w polu R/W ustawiany jest bit informujący o bieżącym zamiarze. Ponieważ bit ten jest 0 to urządzenie *slave* „wie” że będzie u niego występował zapis danych. Zapis danych w tym przypadku polega na ustawieniu aktywnego rejestru poprzez wskazanie go przy pomocy adresu. Ustawienie aktywnego rejestru powoduje, że najbliższy odczyt lub zapis będzie dotyczył właśnie tego rejestru.

Po prawidłowym odebraniu adresu urządzenia i bitu R/W urządzenie *slave* zeruje bit A informując tym samym, że dane zostały odebrane. Następnie urządzenie Master wysyła drugi bajt danych, który teraz zawiera adres rejestru, do którego będzie chciał mieć dostęp. Po poprawnym odbiorze danych przez *slave* bit A jest również zerowany. W tym momencie (połowa transmisji) następuje ponowne wysłanie sekwencji startu. Trzeci bajt danych również zawiera ten sam 7-bitowy adres urządzenia *slave*, ale bit R/W jest ustawiony. Ustawiony bit R/W informuje, że urządzenie Master spodziewa się odbioru danych od *slave*. Po trzeciej poprawnie odebranej ramce danych *slave* potwierdza (bit A), że odebrał dane. W czwartej ramce urządzenie *slave* wysyła zawartość rejestru o adresie wskazanym w ramce drugiej. Jeżeli *master*, po odebraniu danych (1 bajtu, który stanowi zawartość rejestru) zamierza zakończyć transmisję to przed sekwencją stopu ustawia bit NA w stan wysoki co świadczy, że nie jest gotowy na odbiór kolejnych danych. W przypadku zapisu danych do rejestru, sytuacja jest nieco prostsza.



Tak samo jak poprzednio, po sekwencji startu, wysyłany jest 7-bitowy adres urządzenia *slave* oraz bit sygnalizujący zamiar transmisji. Bit R/W również jest zerowany co świadczy, że *slave* otrzyma dane, które będzie musiał zapisać. Po odebraniu pierwszej ramki *slave* potwierdza poprawność odczytu poprzez wyzerowanie bitu A. W drugiej ramce wysyłany jest 8-bitowy adres rejestru wewnętrznego *slave*. Po odebraniu *slave* również potwierdza poprawność (poprzez wyzerowanie bitu A). W trzeciej ramce wysyłana jest wartość, którą należy zapisać rejestr o poprzednio wskazanym adresie. Po odebraniu trzeciej ramki *slave* potwierdza odbiór danych. W większości urządzeń obsługujących transmisję I2C po wysłaniu kolejnych ramek, nastąpi zapisanie kolejnych wewnętrznych rejestrów urządzenia *slave*. Jednak dokładne zachowanie urządzenia peryferyjnego jest sprecyzowane w jego nocie katalogowej.

I2C jest dostępny w mikrokontrolerze STM32L496ZGT6 w czterech modułach. Wspiera szereg trybów pracy w tym *SMBus* i *PMBus*. Przed pierwszym użyciem interfejsu należy go skonfigurować tzn. włączyć zegar dla modułu, wyprowadzenia mikrokontrolera. Obsługa I2C z użyciem biblioteki HAL odbywa się z użyciem funkcji grupy *HAL_I2C*. Można wyróżnić kilka trybów pracy: odpytywania z blokowaniem (ang. *polling in blocking*), odpytywania w trybie pamięci (ang. *polling in blocking memory*), przerwania (ang. *interrupt*), transmisji z obsługą bezpośredniego dostępu do pamięci.

1.5. Sposób realizacji ćwiczenia laboratoryjnego

Każde zajęcie składać będą się z następujących elementów:

- zadań obowiązkowych - do wykonania krok po kroku na podstawie instrukcji – do uzyskania od 0 do 12 pkt.;
- zadań uzupełniających – do samodzielnego zbudowania i napisania programu – do uzyskania od 0 do 14 pkt.;
- pytań sprawdzających rozumienie działania programu – zadawane przez prowadzącego przyjmującego wykonanie zadania – odpowiedź wpływa na punktację z zadań obowiązkowych i uzupełniających, tzn. czy przyznać maksymalną możliwą liczbę punktów za zadanie czy tylko część przy ewidentnym braku zrozumienia problemu.

Po zrealizowaniu każdego z zadań należy poprosić prowadzącego o sprawdzenie i przyznanie punktów. Na koniec zajęć wystawiana jest ocena na podstawie sumy uzyskanych punktów: **2,0** <0; 10), **3,0** <10; 13), **3,5** <13; 16), **4,0** <16; 19), **4,5** <19; 23), **5,0** <23; 26>. W każdych zajęciach należy uczestniczyć. Przy każdym zadaniu określona jest liczba punktów jakie można uzyskać. W przypadku zadań uzupełniających premiowana jest **jakość** i **czas realizacji** zaprezentowanego rozwiązania. Ocena końcowa z laboratorium to średnia arytmetyczna ocen z każdego spotkania.

2. Zadania podstawowe do realizacji (12 pkt.)

W tej części instrukcji zamieszczone są treści, z którymi obowiązkowo należy się zapoznać i praktycznie przećwiczyć. Ważne jest, aby zapamiętać wykonywane przedstawione czynności, aby móc na kolejnych zajęciach wykonywać je na kolejnych zajęciach bez potrzeby sięgania do niniejszej instrukcji.

Uwaga: Załączone wycinki z ekranu są poglądowe i pomagają jedynie w wskazaniu lokalizacji elementów interfejsu. Należy używać wartości podanych w tekście.

2.1. Zagadnienia wstępne

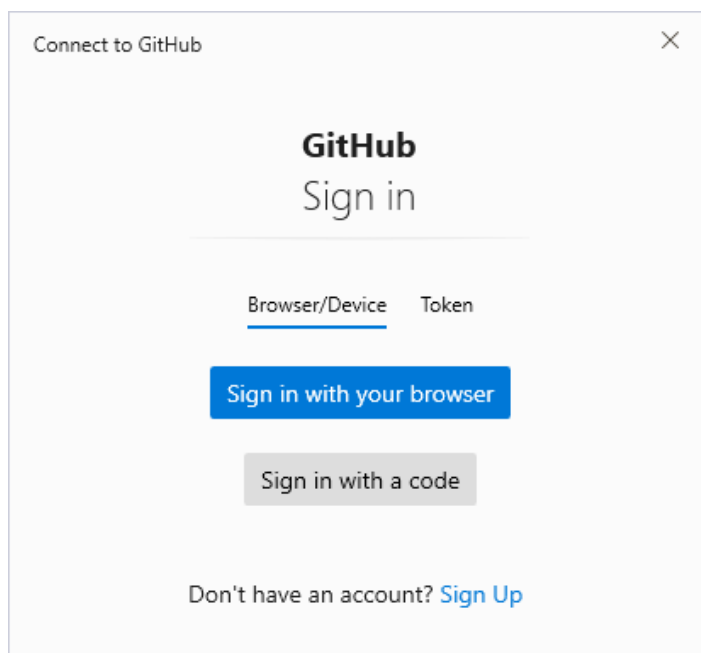
Przed przystąpieniem do pracy należy skonfigurować zainstalowane oprogramowanie i uzyskać dostęp do repozytorium, tzn. system kontroli wersji *Git*, instalacji biblioteki do obsługi mikrokontrolerów rodziny STM32L4 w środowisku *STM32CubeIDE* oraz sklonować repozytorium dla zajęć.

2.1.1. Dołączenie do wirtualnej klasy i utworzenie repozytorium bazowego dla zadania

Na zajęciach należy dołączyć do wirtualnej grupy w ramach *Classrom GitHub* za pomocą udostępnionego odnośnika prowadzącego do zadania, po otwarciu którego tworzone jest zadanie – indywidualne repozytorium. Z listy należy wybrać swój adres e-mail i potwierdzić przyjęcie zadania (ang. *assignment*). Automatycznie zostanie utworzone prywatne repozytorium indywidualnie dla każdego studenta na podstawie przygotowanego repozytorium-szablonu. Na pierwszych zajęciach jest to wersja minimalna, a na kolejnych będzie zawierała już wstępnie skonfigurowane projekty. W sytuacji, jeżeli student nie może odszukać się na liście (np. ktoś inny podłączył się pod daną osobę) proszę zgłosić to prowadzącemu zajęcia. W celu skorygowania nieprawidłowości. Zaakceptowanie zadania wiąże się również z dołączeniem do organizacji – wirtualnego konta organizacji w ramach którego tworzone są indywidualne repozytoria do zadań, z którego prowadzący zajęcia mają dostęp do wszystkich repozytoriów tworzonych w ramach zajęć.

2.1.2. Pobranie repozytorium i konfiguracja lokalna

W celu pobrania repozytorium należy odszukać na pulpicie skrót o nazwie *LabGitConfig*, który uruchomi skrypt wiersza poleceń, w którym należy podać: 1) swoje imię i nazwisko, 2) adres e-mail, 3) symbol grupy, 4) numer ćwiczenia, 5) adres indywidualnego repozytorium uzyskany w wcześniejszym punkcie. Po wykonaniu punktu 5) pojawi się okno logowania do serwisu GitHub podobne do zamieszczonego obok. W celu połączenia należy wybrać przycisk *Sign in with your browser* co spowoduje otworzenie nowej karty przeglądarki w którym należy udzielić dostępu do konta. Po uzyskaniu zgody nastąpi sklonowanie projektu z serwera do lokalizacji wynikającej z symbolu grupy. W kolejnym pytaniu można uruchomić środowisko programistyczne *STM32CubeIDE*. Po jego uruchomieniu nie należy zamykać okna wiersza poleceń. Przykładowy przebieg wykonania skryptu zamieszczony został poniżej.



```

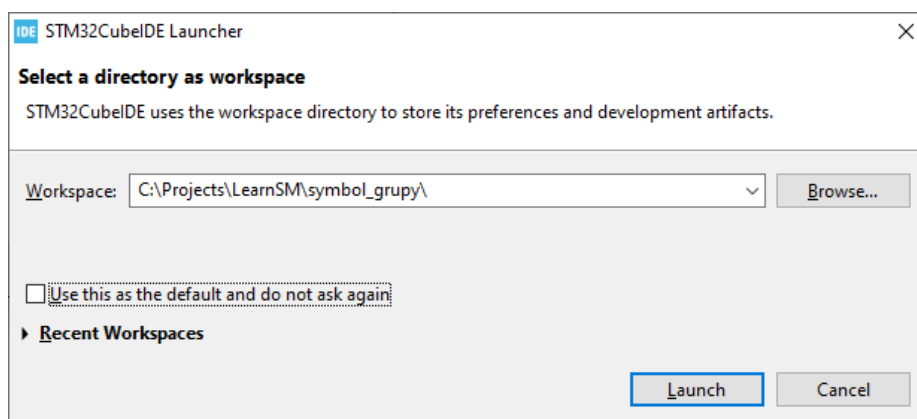
Windows PowerShell
Konfiguracja systemu Git dla zajęć laboratoryjnych Systemy Mikroprocesorowe
Proszę wprowadzić imię i nazwisko: Paweł Dabał
Proszę wprowadzić adres e-mail (@student.wat.edu.pl): pawel.dabal@student.wat.edu.pl
Proszę podać pełny symbol grupy (np. WEL19EL1S0): WEL19ExySz
Podaj numer ćwiczenia: 1
Podaj adres indywidualnego repozytorium: https://github.com/ztc-wel-wat/sm-lab-1-gpio-exti-pdabal

CMDKEY: Nie można odnaleźć elementu.
Cloning into 'C:\Projects\LabSM\WEL19EXYSZ\Lab-1'...
info: please complete authentication in your browser...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 32 (delta 4), reused 28 (delta 1), pack-reused 0
Receiving objects: 100% (32/32), 215.44 KiB | 1.89 MiB/s, done.
Resolving deltas: 100% (4/4), done.
Ustawiona nazwa użytkownika:
Paweł Dabał
Ustawiony adres e-mail:
pawel.dabal@student.wat.edu.pl
Aby uruchomić STM32CubeIDE naciśnij Enter:

```

2.1.3. Konfiguracja środowiska STM32CubeIDE

Po uruchomieniu środowiska programistycznego *STM32CubeIDE* pojawi się pytanie o wskazanie katalogu, który będzie pełnił rolę przestrzeni roboczej. Należy wskazać na katalog skopiowanego w poprzednim kroku repozytorium (w przykładzie: *C:\Projects\LabSM\symbol_grupy*). Wybór katalogu zatwierdzamy przyciskiem *Launch*. Można używać wielu różnych przestrzeni roboczych w przypadku pracy z różnymi projektami. Po załadowaniu środowiska należy zamknąć zakładkę *Information Center*.



W kolejnym kroku należy zaimportować projekt do przestrzeni roboczej poprzez wybranie z menu *File* → *Import*, a następnie w oknie, które się pojawi wybrać *General* → *Existing Projects into Workspace*. Zatwierdzić przyciskiem *Next*. Za pomocą przycisku *Browse...* wybrać katalog repozytorium (np. *C:\Projects\LabSM\symbol_grupy*). Zaktualizuje się lista dostępnych projektów i na niej należy wybrać *BasicInterfacesHAL* i zatwierdzić przyciskiem *Finish*.

W celu zarządzania repozytorium dla zadania należy w środowisku *STM32CubeIDE* należy otworzyć widok (ang. *perspective*) zarządzania repozytorium. W tym celu z menu wybieramy: *Window* → *Perspective* → *Open Perspective* → *Other...*, gdzie z listy należy wybrać *Git*. Nastąpi przełączenie widoku i po lewej stronie pojawi się zakładka *Git Repositories* z listą dostępnych repozytoriów. Proszę zweryfikować obecność pobranego repozytorium, a w przypadku braku zgłosić prowadzącemu zajęcia.

W trakcie zajęć ograniczymy się do prostej liniowej struktury migawek wykonywanych po zakończeniu każdego z zadań instrukcji opatrzonej stosownym komentarzem. W dalszej części instrukcji będą podane treści komentarzy jakimi należy opatrzyć realizowane migawki. Powstanie zatem coś w rodzaju sprawozdania z zajęć, które będzie *przechowywane* w serwisie *GitHub*.

2.1.4. Opis projektu bazowego

W sklonowanym repozytorium znajduje się projekt bazowy korzystający z biblioteki HAL o nazwie *BasicInterfacesHAL*. Posiada on skonfigurowane wyprowadzenia mikrokontrolera do których dołączone są diody LED (*LED0 ... LED7*) oraz pięć styków joysticka joystick (*SW_RIGHT, SW_LEFT, SW_DOWN, SW_UP, SW_OK*). W pliku *gpio.c* zdefiniowane zostały podstawowe funkcje do obsługi diod LED oraz odczytu stanu joysticka. Ponadto skonfigurowana i dodana została obsługa wyświetlacza 7-segmentowego oraz tekstowego LCD (pliki *display.c, lcd.c*).

2.2. Obsługa interfejsu szeregowego (6 pkt.)

Interfejs szeregowy (USART – ang. *Universal Synchronous/Asynchronous Receiver Transmitter*) jest dostępny w mikrokontrolerze STM32L496ZGT6 w sześciu modułach o zróżnicowanych funkcjonalnościach. Najważniejsze z nich to wsparcie dla sprzętowego wsparcia kontroli przepływu danych, obsługa transmisji z bezpośrednim dostępem do pamięci (DMA), komunikacją między procesorową. Jeden z modułów LPUART (ang. *Low Power UART*) dodatkowo może pracować w niższych poziomach zasilania. Na płycie *KAmLeon* komunikacja między mikrokontrolerem a komputerem z użyciem interfejsu szeregowego LPUART podłączonego do układu programatora pełniącego rolę konwertera UART – USB. W tym celu należy skonfigurować wyprowadzenia PC0 i PC1.

2.2.1. Konfiguracja modułu LPUART oraz standardowego wejścia/wyjścia (3 pkt.)

W celu skonfigurowania modułu LPUART należy w projekcie otworzyć plik *BasicInterfacesHAL.ioc* w celu otworzenie okna konfiguratora. Należy w widoku *Pinout & Configuration* rozwinąć *Connectivity* i wybrać *LPUART1*. Następnie po prawej stronie w oknie *LPUART1 Mode and Configuration* w sekcji *Mode* w polu wyboru *Mode* należy wybrać *Asynchronous* (standardowy tryb pracy). Poniżej w sekcji *Configuration* zakładce *Parameter Settings* należy skonfigurować w grupie *Basic Parameters*: prędkość transmisji (*Baud Rate*) *115200*, długość słowa (*Word Length*) *8 Bits (including Parity)*, parzystość (*Parity*) *None*, liczba bitów stop (*Stop Bits*) *1*; w grupie *Advanced Features* zmienić *Overrun* na *Disable*. Następnie należy otworzyć zakładkę *NVIC Settings* i zaznaczyć

The screenshot displays the STM32CubeMX configuration interface for the LPUART1 module. On the left, the 'Connectivity' tree shows various peripherals, with LPUART1 highlighted. The main configuration window is titled 'LPUART1 Mode and Configuration'. It is divided into two main sections: 'Mode' and 'Configuration'. In the 'Mode' section, 'Asynchronous' is selected. In the 'Configuration' section, the 'Parameter Settings' tab is active, showing 'Basic Parameters' (Baud Rate: 115200, Word Length: 8 Bits (including Parity), Parity: None, Stop Bits: 1) and 'Advanced Features' (Overrun: Disable). The 'NVIC Settings' tab is also shown, with 'LPUART1 global interrupt' enabled. The 'DMA Settings' tab shows 'LPUART_RX' and 'LPUART_TX' configured with DMA2 Channel 7 and 6 respectively.

pole *Enable* przy *LPUART1 global interrupt*. W zakładce *DMA Settings* należy korzystając z przycisku *Add* dodać dwa kanały do transmisji w trybie DMA wybierając odpowiednie pozycje *DMA Request*, pozostałe parametry pozostawić bez zmian. Zapisać konfigurację i zezwolić na wygenerowanie kodu startowego aplikacji.

Dodanie obsługi interfejsu szeregowego w funkcjonalności biblioteki standardowej *STDIO* języka C wymaga dodania implementacji metod `__io_putchar` (wejście) oraz `__io_getchar` (wyjście). W tym celu należy otworzyć plik *Core->Src->syscalls.c*, dodać plik nagłówkowy *usart.h* za pomocą dyrektywy *#include*, dodać dyrektywę *#define STDIO_ECHO_ENABLED 1* oraz dodać poniższy kod:

```

45 /* Functions */
46 int __io_putchar(int ch){
47     HAL_UART_Transmit(&hlpuart1, (uint8_t*) &ch, 1, HAL_MAX_DELAY);
48     return ch;
49 }
50 int __io_getchar(void) {
51     uint8_t ch =0;
52
53     __HAL_UART_CLEAR_OREFLAG(&hlpuart1);
54     HAL_UART_Receive(&hlpuart1, &ch, 1, HAL_MAX_DELAY);
55     #if STDIO_ECHO_ENABLED
56     HAL_UART_Transmit(&hlpuart1, &ch, 1, HAL_MAX_DELAY);
57     #endif
58     return ch;
59 }

```

Funkcja `__io_putchar` odpowiedzialna jest za wysyłanie znaków. W linii 47 wywoływana jest funkcja *HAL_UART_Transmit*, która rozpoczyna wysyłanie 1 znaku *ch*. Z racji podania parametru *Timeout* wartości *HAL_MAX_DELAY* funkcja ta będzie blokować do momentu skutecznego wysłania danych. W funkcji `__io_getchar` w pierwszej kolejności kasujemy przepełnienie bufora odbiorczego (linia 54), a następnie za pomocą funkcji *HAL_UART_Receive* podjęta jest próba odebrania 1 znaku *ch*. Z racji podania parametru *Timeout* wartości *HAL_MAX_DELAY* funkcja ta będzie blokować do momentu skutecznego odebrania danych. Jeżeli stała *STDIO_ECHO_ENABLED* ma wartość różną od 0 wtedy zostanie odesłany zwrótnie odebrany znak. Jest to tak zwane *echo*.

W celu przetestowania funkcjonalności należy uzupełnić plik *main.c* przez dodanie odwołania do pliku nagłówkowego *stdio.h* oraz modyfikację kodu funkcji *main* znajdującej się w pliku *Core->Src->main.c* po niższą zawartością. Kluczowym dla prawidłowej pracy jest wywołanie funkcji *setvbuf* (linia 100) odpowiedzialnej za ustawienie wielkości bufora odbieranych znaków na 0. Domyślnie ustawiony jest on na 1024 znaki. Za pomocą funkcji *scanf* wczytywana jest wpisana liczba całkowita do zmiennej *value*. Następnie za pomocą funkcji *SEG_DisplayDec* jest prezentowana na wyświetlaczu 7-segmentowym.

```

97 /* Infinite loop */
98 /* USER CODE BEGIN WHILE */
99 uint16_t value;
100 setvbuf(stdin, NULL, _IONBF, 0);
101 while (1) {
102     printf("\nPodaj liczbę do wyświetlenia: ");
103     scanf("%d", (int*) (&value));
104     SEG_DisplayDec(value);
105     /* USER CODE END WHILE */
106
107     /* USER CODE BEGIN 3 */
108 }
109 /* USER CODE END 3 */

```

Przygotowany program należy skompilować, uruchomić i przetestować we współpracy z programem obsługującym interfejs szeregowy np. *Tera Term*. Po uruchomieniu aplikacji należy z menu wybrać *File->New connection*, zaznaczyć *Serial* i wybrać właściwy port szeregowy. Następnie wybrać w menu *Setup->Terminal* i ustawić *Receive: Auto* oraz *Transmit: CR*. Na zakończenie konfiguracji wybrać w menu *Setup->Serial Port* i ustawić *Speed: 115200*. Działanie programu przedstawić prowadzącemu do oceny. Po zatwierdzeniu wykonać migawkę z komentarzem „**Zadanie 2.2.1 – standardowe wejście/wyjście**”.

2.2.2. Transmisja danych w trybie blokowania i przerwania (3 pkt.)

Funkcje biblioteki *HAL_UART_Transmit* i *HAL_UART_Receive* umożliwiają odpowiednio wysyłanie i odbieranie wskazanej ilości słów w określonym czasie. Jeżeli nie będzie to możliwe funkcja zwróci stosowny kod błędu wykonania. Taka konstrukcja pozwala na pisanie kodu, który będzie blokował możliwość wykonania innych instrukcji przez mikrokontroler. W związku z czym ich użycie preferowane dla transmisji małych porcji danych. Spowodowane jest to przez wartość ostatniego parametru tych funkcji o nazwie *Timeout*. Kiedy chcemy wysłać lub odebrać większą ilość danych warto użyć odpowiednio funkcji *HAL_UART_Transmit_IT* i *HAL_UART_Receive_IT*. Ich wywołanie inicjuje transmisję danych, która jest później kontynuowana w odpowiedniej bibliotecznej funkcji obsługi przerwań dla danego modułu UART. W ten sposób nie ma blokowania i oczekiwania na transmisję przez stosunkowo wolny interfejs. Dla typowych nastaw interfejsu szeregowego (115200, 8, N, 1) transmisja jednego 8-bitów informacji wymaga 8680,5 ns co przy zegarze taktującym mikrokontroler (4 MHz) o okresie 250 ns oznacza możliwość wykonania maksymalnie 34 instrukcji. Przy maksymalnej możliwej częstotliwości taktowania wynoszącej 80 MHz można w tym czasie wykonać do 680 instrukcji. Programista może dodać funkcję *HAL_UART_TxCpltCallback* i *HAL_UART_RxCpltCallback*, które zostaną wywołane na zakończenie transmisji danych. Ponadto można zaimplementować funkcje *HAL_UART_TxHalfCpltCallback* oraz *HAL_UART_RxHalfCpltCallback*, które zostaną wywołane po wysłaniu połowy danych.

```
39  /* Private define ----- */
40  /* USER CODE BEGIN PD */
41  #define TEST_UART_IT 0
42  /* USER CODE END PD */
```

Poniższy kod pozwoli zaobserwować wpływ blokowania na wykonywanie programu w porównaniu do pracy z użyciem przerwania. W pierwszej kolejności należy dodać definicję symbolu *TEST_UART_IT*:

Następnie w pliku *mian* należy dodać poniższy kod:

```
189 void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
190     if (huart->Instance == LPUART1) {
191         LED_Toggle(7);
192     }
193 }
```

W przypadku, kiedy *TEST_UART_IT* ma wartość 0 dioda *LED0* będzie zmieniać swój stan po każdorazowym odebraniu 8 znaków. Dioda *LED6* będzie się świecić przez czas potrzebny na odebranie 8 znaków. Zmiana wartości symbolu *TEST_UART_IT* na 1 spowoduje, że zamiast diody *LED6* zapali się dioda *LED7*, która zgaśnie po odebraniu 8 znaków. Różnica w działaniu obu konfiguracji polega na tym, że w drugim przypadku dioda *LED0* cały czas zmienia swój stan. Spowodowane jest to przez fakt, że po wywołaniu funkcji w linii 106 odbieranie jest rozpoczęte. Sam odczyt z rejestru odbiorczego wykonywany jest w funkcji obsługi przerwania modułu LPUART1.


```

99      /* Infinite loop */
100     /* USER CODE BEGIN WHILE */
101     uint8_t rxBuff[16];
102     while (1) {
103         LED_Toggle(0);
104     #if TEST_UART_IT
105         LED_On(7);
106         HAL_UART_Receive_IT(&hlpuart1, rxBuff, 8);
107     #else
108         LED_On(6);
109         HAL_UART_Receive(&hlpuart1, rxBuff, 8, HAL_MAX_DELAY);
110         LED_Off(6);
111     #endif
112         HAL_Delay(1000);
113         /* USER CODE END WHILE */
114
115         /* USER CODE BEGIN 3 */
116     }
117     /* USER CODE END 3 */

```

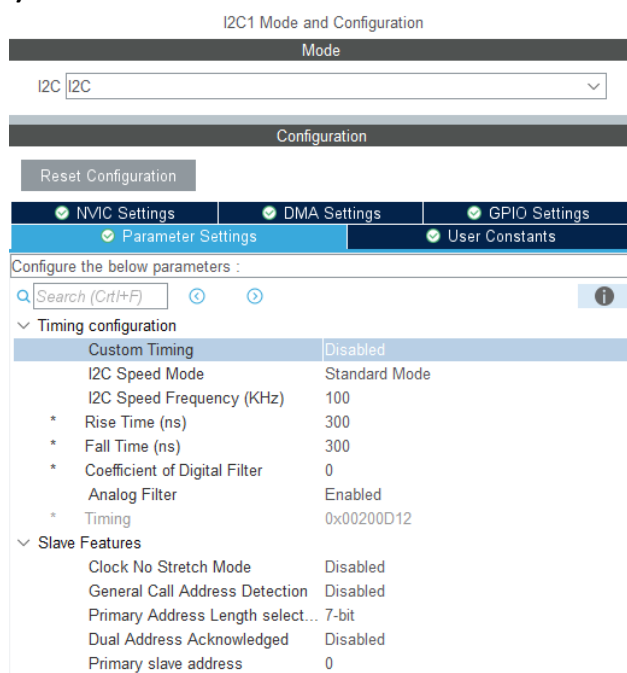
Przygotowany program należy skompilować, uruchomić i przetestować we współpracy z programem obsługującym interfejs szeregowy np. *Tera Term* dla dwóch różnych trybów pracy przez zmianę wartości symbolu *TEST_UART_IT*. Działanie programu przedstawić prowadzącemu do oceny. Po zatwierdzeniu wykonać migawkę z komentarzem „**Zadanie 2.2.2 – wysyłanie w trybie blokowania i przerwania**”.

2.3. Obsługa interfejsu I2C (6 pkt.)

Na płycie *KAmLeon* dostępne są 2 układy cyfrowe komunikujące się z mikrokontrolerem z użyciem interfejsu I2C. Pierwszy z nich to cyfrowy czujnik temperatury [STLM75M2F](#) dostępny pod adresem 0x48 lub 0x49 w zależności od konfiguracji zwory JP17, podłączony do interfejsu I2C1. Drugi układ to [LSM303C](#) moduł e-kompasu wyposażony w 3-osiowy akcelerometr (adres 0x1D) i 3-osiowy magnetometr (adres 0x1E) podłączony do interfejsu I2C3. Interfejs I2C jest dostępny w mikrokontrolerze STM32L496ZGT6 w czterech modułach. Najważniejsze z nich to wsparcie dla adresowania o długości 7 i 10 bitów, prędkości transmisji do 1 Mb/s, obsługa protokołu SMBus/PMBus. Na płycie *KAmLeon* interfejs I2C1 jest użyty na wyprowadzeniach PG13 (SDA), PG14 (SCL), natomiast interfejs I2C3 jest dostępny na wyprowadzeniach PG7 (SCL) i PG8 (SDA).

2.3.1. Konfiguracja modułu I2C1 i odczyt temperatury (3 pkt.)

W celu skonfigurowania modułu I2C1 należy w projekcie otworzyć plik *BasicInterfacesHAL.ioc* w celu otworzenia okna konfiguratora. Należy w widoku *Pinout & Configuration* rozwinąć *Connectivity* i wybrać *I2C1*. Następnie po prawej stronie w oknie *I2C1 Mode and Configuration* w sekcji *Mode* w polu wyboru *I2C* należy wybrać *I2C*. Poniżej w sekcji *Configuration* zakładce *Parameter Settings* należy skonfigurować w grupie *Timing Configuration: Rise Time* oraz *Fall Time* na 300, a pozostałe opcje pozostawić bez zmian. Wyprowadzeniom PG13 i PG14 należy nadać odpowiednio etykiety *TEMP_LM75_I2C1_SDA* i *TEMP_LM75_I2C1_SCL*. Ponadto układ ten podłączony jest do wejścia PG15 (etykieta *LM75_INT*), które służy do poinformowania mikrokontrolera o przekroczeniu ustawionej temperatury.



Do obsłużenia czujnika temperatury należy przepisać poniższe definicje symboli za pomocą dyrektywy `#define`. Pierwsza z nich przechowuje adres na magistrali I2C, cztery kolejne to adresy rejestrów wewnętrznych układu, a ostatnia przedstawia rozdzielczość przetwornika. Do odczytu temperatury wystarczy odczyt 2 bajtów z rejestru `TEMP_LM75_TEMP_REG` oraz przeliczenie na rzeczywistą wartość odebranych danych. Odczyt nie powinien odbywać się jednak częściej niż co 150 ms. Więcej informacji można znaleźć w dokumentacji do układu.

```
40 /* Private define ----- */
41 /* USER CODE BEGIN PD */
42 #define TEMP_LM75_I2C_ADDR      0x48
43
44 #define TEMP_LM75_TEMP_REG      0
45 #define TEMP_LM75_CONF_REG      1
46 #define TEMP_LM75_THYS_REG      2
47 #define TEMP_LM75_TOS_REG      3
48
49 #define TEMP_LM75_RESOLUTION    0.5f
50 /* USER CODE END PD */
```

Powyższe definicje zostaną użyte w funkcjach, które zapewniają odczyt ze wskazanego rejestru. Pierwsze dwie umożliwiają odczyt odpowiednio 8 i 16 bitów ze wskazanego w parametrze funkcji rejestru układu czujnika. W celu komunikacji z układem użyta została funkcja `HAL_I2C_Mem_Read`, która najpierw wpisuje numer rejestru, a później odczytuje dane. Funkcje `TEMP_GetTempI` oraz `TEMP_GetTempF` pozwalają na odczytanie rzeczywistej temperatury wyrażonej w stopniach Celsjusza, przy czym druga funkcja zwraca wartość z dokładnością $\pm 0,5^{\circ}\text{C}$.

```
69 /* Private user code ----- */
70 /* USER CODE BEGIN 0 */
71 uint8_t LM75_ReadReg8(uint8_t regAddr) {
72     uint8_t regValue;
73     if (HAL_I2C_Mem_Read(&hi2c1, TEMP_LM75_I2C_ADDR << 1, regAddr, 1,
74         (uint8_t*) &regValue, 1, 1000) != HAL_OK)
75         Error_Handler();
76     return regValue;
77 }
78
79 uint16_t LM75_ReadReg16(uint8_t regAddr) {
80     uint8_t regValue[2];
81     if (HAL_I2C_Mem_Read(&hi2c1, TEMP_LM75_I2C_ADDR << 1, regAddr, 1,
82         (uint8_t*) regValue, 2, 1000) != HAL_OK)
83         Error_Handler();
84     return (((uint16_t) regValue[0]) << 8) | regValue[1];
85 }
86
87 int16_t TEMP_GetTempI(void) {
88     return (LM75_ReadReg16(TEMP_LM75_TEMP_REG) >> 8);
89 }
90
91 float TEMP_GetTempF(void) {
92     return ((LM75_ReadReg16(TEMP_LM75_TEMP_REG) >> 7)
93         * TEMP_LM75_RESOLUTION);
94 }
95 /* USER CODE END 0 */
```

W celu sprawdzenia poprawności odczytu temperatury należy zmodyfikować zawartość pętli `while` w funkcji `main` na poniższą. Wartość rzeczywista temperatury jest odczytywana z użyciem funkcji `TEMP_GetTempF` i zapisywana do zmiennej `temp`. Następnie w celu wyświetlenia jej na wyświetlaczu 7-segmentowym pomnożona jest przez 10 i przekazana jako pierwszy parametr funkcji `BSP_SEG_DisplayFixedPoint` po

wcześniej zmianie na liczbę całkowitą. Z racji, że drugi parametr ustawiony został na 1 to kropka dziesiętna zapali się na module 1.

```

131  /* Infinite loop */
132  /* USER CODE BEGIN WHILE */
133  while (1) {
134      LED_Toggle(0);
135      float temp = TEMP_GetTempF();
136      BSP_SEG_DisplayFixedPoint((int16_t) (temp * 10), 1);
137      HAL_Delay(1000);
138      /* USER CODE END WHILE */
139
140      /* USER CODE BEGIN 3 */
141  }
142  /* USER CODE END 3 */

```

Przygotowany program należy skompilować, uruchomić i przetestować. Zmianę temperatury można wywołać przez przyłożenie palca do czujnika temperatury. Działanie programu przedstawić prowadzącemu do oceny. Po zatwierdzeniu wykonać migawkę z komentarzem „**Zadanie 2.3.1 – konfiguracja modułu I2C1 i odczyt temperatury**”.

2.3.2. Konfiguracja modułu I2C3 i odczyt z układu akcelerometru (3 pkt.)

W celu skonfigurowania modułu I2C3 należy w projekcie otworzyć plik *BasicInterfacesHAL.ioc* w celu otworzenie okna konfiguratora. Należy w widoku *Pinout & Configuration* rozwinąć *Connectivity* i wybrać *I2C3*. Następnie po prawej stronie w oknie *I2C3 Mode and Configuration* w sekcji *Mode* w polu wyboru *I2C* należy wybrać *I2C*. Poniżej w sekcji *Configuration* zakładce *Parameter Settings* należy skonfigurować w grupie *Timing Configuration*: *I2C Speed Mode* ustawić na *Fast Mode*, *Rise Time* oraz *Fall Time* ustawić na 300, a pozostałe opcje pozostawić bez zmian. Następnie należy odszukać wyprowadzenie PG7, wybrać opcje *I2C3_SCL* i nadać etykietę *LSM303C_I2C3_SCL*. Natomiast wyprowadzeniu PG8 nadać etykietę *LSM303C_I2C3_SDA*. Wprowadzone zmiany należy zapisać w celu wygenerowania niezbędnych zmian w konfiguracji projektu.

W pierwszej kolejności należy dodać plik nagłówkowy *lsm303c.h* dyrektywą *#include*. Zawiera on zdefiniowane symbole reprezentujące adresy rejestrów oraz większość możliwych wartości jakie mogą być w tych rejestrach ustawione.

```
32 #include "lsm303c.h"
```

Następnie z racji, że można pozyskać dane o przyspieszeniu w trzech osiach i są one dostępne w postaci liczby typu *int16_t* zdefiniować należy strukturę *ACC_XYZ_TypeDef*. Pozwoli ona na opakowanie odczytanych wartości w jedną zmienną.

```

36 /* Private typedef -----*/
37 /* USER CODE BEGIN PTD */
38 typedef struct {
39     int16_t x;
40     int16_t y;
41     int16_t z;
42 } ACC_XYZ_TypeDef;
43 /* USER CODE END PTD */

```

Kolejnym elementem do dodania jest definicja symbolu *ACC_LSM303C_I2C_ADDR*, który określa adres na magistrali I2C.

```
56 #define ACC_LSM303C_I2C_ADDR    0x1D
```

Następnie należy przepisać kod poniższych dwóch funkcji. Pierwsza z nich *ACC_LSM303C_Init* sprawdza w pierwszej kolejności czy wykryto układ przez odczyt jego identyfikatora *ID*, a jeżeli identyfikacja wypadła pozytywnie wykonywana jest konfiguracja rejestru *CTRL_REG1_A* (częstotliwość próbkowania, wybór aktywnych osi) oraz rejestru *CTRL_REG4_A* (zakres pracy, automatyczna inkrementacja adresu rejestru podczas odczytu). Funkcja *ACC_LSM303C_ReadXYZ* dokonuje odczytu 6 kolejnych bajtów począwszy od adresu rejestru *OUT_X_L_A*, które zapisywane są do struktury *acc*. W każdej funkcji w przypadku wystąpienia błędu komunikacji za pośrednictwem magistrali I2C program zostanie przekierowany do funkcji *Error_Handler*.

```

103 /* USER CODE BEGIN 0 */
104 HAL_StatusTypeDef ACC_LSM303C_Init(uint8_t acc_datarate, uint8_t acc_fullscale) {
105     uint8_t reg[1];
106     HAL_StatusTypeDef status;
107     // Odczyt ID układu
108     if (HAL_I2C_Mem_Read(&hi2c3, ACC_LSM303C_I2C_ADDR << 1, LSM303C_WHO_AM_I_ADDR, 1, reg, 1, 1000) != HAL_OK) {
109         Error_Handler();
110     } else {
111         if (reg[0] != LSM303C_ACC_ID)
112             return HAL_ERROR;
113     }
114     // Włączenie każdej z osi oraz ustawienie częstotliwości próbkowania
115     reg[0] = (acc_datarate | LSM303C_ACC_AXES_ENABLE);
116     status = HAL_I2C_Mem_Write(&hi2c3, ACC_LSM303C_I2C_ADDR << 1, LSM303C_CTRL_REG1_A, 1, reg, 1, 1000);
117     // Ustawienie zakresu pomiarowego
118     reg[0] = (acc_fullscale | 4);
119     status = HAL_I2C_Mem_Write(&hi2c3, ACC_LSM303C_I2C_ADDR << 1, LSM303C_CTRL_REG4_A, 1, reg, 1, 1000);
120
121     return status;
122 }
123
124 void ACC_LSM303C_ReadXYZ(ACC_XYZ_TypeDef *res) {
125     if (HAL_I2C_Mem_Read(&hi2c3, ACC_LSM303C_I2C_ADDR << 1, LSM303C_OUT_X_L_A | 0x80, 1, (uint8_t*) res, 6, 1000)
126         != HAL_OK)
127         Error_Handler();
128 }
129 /* USER CODE END 0 */

```

W celu sprawdzenia poprawności odczytu przyspieszenia należy zmodyfikować zawartość pętli *while* w funkcji *main* na poniższą. W pierwszej kolejności deklarujemy zmienną *acc* do której będziemy wczytywać wartości przyspieszenia. Następnie jednorazowo wywołana jest funkcja *ACC_LSM303C_Init*, która przyjmuje dwa parametry. Pierwszy z nich określa częstotliwość próbkowania (ang. *Output Data Rate* - ODR), która może przyjąć wartość 10, 50, 100, 200, 400 lub 800 Hz. Drugi parametr określa zakres pomiarowy wyrażony w jednostkach przyspieszenia ziemskiego *g* ($1g = 9,81 \text{ m/s}^2$) i może wynieść ± 2 , ± 4 lub $\pm 8 g$. W każdej iteracji pętli *while* następuje zmiana stanu diody *LED0*, odczytanie wartości przyspieszenia z użyciem funkcji *ACC_LSM303C_ReadXYZ* wysłanie za pośrednictwem interfejsu szeregowego pracującego w trybie bezpośredniego dostępu do pamięci DMA. Tryb ten różni się od trybu IT tym, że przesłanie danych z pamięci do rejestru wysyłania nie wymaga wykonywania instrukcji przez rdzeń procesora, lecz dzieje się sprzętowo.

```

165 /* Infinite loop */
166 /* USER CODE BEGIN WHILE */
167 ACC_XYZ_TypeDef acc;
168 ACC_LSM303C_Init(LSM303C_ACC_ODR_10_HZ, LSM303C_ACC_FULLSCALE_2G);
169 while (1) {
170     LED_Toggle(0);
171     ACC_LSM303C_ReadXYZ(&acc);
172     HAL_UART_Transmit_DMA(&h1puart1, (uint8_t *)&acc, 6);
173     HAL_Delay(100);
174     /* USER CODE END WHILE */
175     /* USER CODE BEGIN 3 */
176 }
177 /* USER CODE END 3 */

```

Przygotowany program należy skompilować, uruchomić i przetestować we współpracy z programem obsługującym interfejs szeregowy np. *Tera Term* oraz programem *Serial Plot* umożliwiającym wizualizację odebranych danych. W katalogu *SerialPlotCfg* umieszczona został plik konfigurujący program, który można wczytać za pomocą menu *File->Load File* i wskazanie pliku *Zad-2.3.2.ini*. Działanie programu przedstawić prowadzącemu do oceny. Po zatwierdzeniu wykonać migawkę z komentarzem „**Zadanie 2.3.2 – konfiguracja modułu I2C3 i odczyt z układu akcelerometru**”.

3. Zadania rozszerzające do realizacji (14 pkt.)

W rozdziale tym przedstawione zostały zadania dodatkowe, za realizację których można podnieść ocenę końcową za wykonane ćwiczenie. Ich realizację należy wykonać w dotychczasowym projekcie.

3.1. Rozszerzona obsługa interfejsu szeregowego (7 pkt.)

3.1.1. Konsola wiersza poleceń (3 pkt.)

W zadaniu tym należy zaimplementować możliwość wykonywania komend przesłanych z komputera, które spowodują wykonanie odpowiedniej akcji przez mikrokontroler. Komendy można podzielić na dwa rodzaje ustawiające ('=') oraz pytające ('?'). Należy zaimplementować obsługę komendy ustawiającej w postaci „<komenda>=<wartość>\r\n”. Gdzie komenda to „LED”, a wartość może być liczbą od 0 do 7 i spowoduje zmianę stanu odpowiedniej diody. Ponadto należy zaimplementować obsługę komendy pytającej w postaci „<komenda>?\r\n”. Gdzie komenda to „TEM” i w odpowiedzi otrzymujemy wartość temperatury z czujnika STLM75M2F. Wykonanie komendy potwierdzone jest odesłaniem do komputera „OK\r\n”. Znak końca linii jest informacją o długości komendy.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „**Zadanie 3.1.1 – konsola wiersza poleceń**”.

3.1.2. Generowanie próbek funkcji sinus (4 pkt.)

W zadaniu tym należy przygotować program, który będzie wysyłał do komputera kolejne próbki funkcji sinus wyznaczone dla kąta, który będzie zwiększany o stałą wartość koku fazy (*phaseStep*) 100 razy na sekundę. Krok fazy będzie domyślnie ustawiony tak aby 100 próbek reprezentowało 1 okres funkcji sinus ($phaseStep = 2\pi f / 100$, dla jednego okresu $f = 1$). Za pomocą joysticka należy umożliwić zmianę wartości zmiennej f w zakresie od 0 do 100, którą później należy uwzględnić w wyznaczeniu nowej wartości zmiennej *phaseStep*. Wartość zmiennej f należy prezentować na wyświetlaczu 7-segmentowym. Wysyłana wartość powinna być reprezentowana w formacie tekstowym z dokładnością trzech miejsc o przecinku.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „**Zadanie 3.1.2 – generowanie próbek funkcji sinus**”.

3.2. Rozszerzona obsługa interfejsu I2C (7 pkt.)

3.2.1. Zaawansowane funkcje czujnika temperatury STLM75F (3 pkt.)

Na bazie zadania 2.3.1 oraz w oparciu o dokumentację układu [STLM75M2F](#) należy skonfigurować wartości rejestrów $T_{HYS} = 26,0\text{ }^{\circ}\text{C}$ oraz $T_{HYS} = 27,0\text{ }^{\circ}\text{C}$. Linii *LM75_INT* podłączona do portu PG15 mikrokontrolera będzie ustawiana w stan niski za każdym razem po przekroczeniu temperatury $27,0\text{ }^{\circ}\text{C}$ i ustawiana w stan wysoki, kiedy temperatura spadnie poniżej $26,0\text{ }^{\circ}\text{C}$. Z racji, że pomiar odbywa się automatycznie i trwa 150 ms można przyjąć, że należy sprawdzać stan portu PB15 co 200 ms. Do tego celu można rozbudować funkcję *HAL_SYSTICK_Callback* tak aby sprawdzać stan PG15 i kiedy jest niski zapalić diodę LED7 i zgasić ją, kiedy na PG15 będzie stan wysoki. Cały czas na wyświetlaczu 7-segmentowym odświeżana jest wartość odczytywanej temperatury co 1000 ms

oraz będzie ona wysyłana za pośrednictwem interfejsu szeregowego w postaci tekstowej z dokładnością jednego miejsca po przecinku.

Przygotowany program należy skompilować, uruchomić i przetestować przez przyłożenie palca do czujnika temperatury. Spowodować powinno to wzrost temperatury. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „**Zadanie 3.2.1 – alarm temperaturowy**”.

3.2.2. Zaawansowane funkcje czujnika przyspieszenia LSM303C (4 pkt.)

Na bazie zadania 2.3.1 oraz w oparciu o dokumentację układu [LSM303C](#) należy skonfigurować układ do pracy z kolejką *FIFO* w trybie *Stream*, która umożliwia zbieranie do 32 próbek i jednocześnie umożliwia rzadsze odpytywanie mikrokontrolera o przesłanie danych. W tym celu należy skonfigurować rejestr *CTRL_REG3_A* = 0x80, *FIFO_CTRL* = 0x40. Przy częstotliwości próbkowania 10 Hz można ograniczyć częstotliwość odczytu danych o przyspieszeniu do pojedynczego wywołania funkcji raz na sekundę. Liczbę próbek jakie znajdują się w kolejce można odczytać z rejestru *FIFO_SRC* na podstawie pięciu najmniej znaczących bitów. Odebrane dane należy zapisywać do tablicy typu *ACC_XYZ_TypeDef*, która może pomieścić co najmniej 10 próbek. Następnie próbki należy przesłać z użyciem interfejsu szeregowego do komputera.

Przygotowany program należy skompilować, uruchomić i przetestować. Działanie programu przedstawić prowadzącemu zajęcia. Po zatwierdzeniu i ocenieniu wykonać migawkę z komentarzem „**Zadanie 3.2.2 – praca z buforowaniem próbek**”.