Laboratorinis darbas Nr. 4.

Slėgio jutiklio LPS25HB nuskaitymas per I2C sąsają.

**Turinys**

[1 Darbo tikslai 1](#_Toc116312121)

[2 LPS25HB temperatūros nuskaitymas 1](#_Toc116312122)

[3 Užduotys 14](#_Toc116312123)

# Darbo tikslai

1. Išmokti nuskaityti bei apdoroti skaitmeninių sensorių duomenis per I2C sąsają.

# LPS25HB temperatūros nuskaitymas

Kamami (KA-Nucleo-Multisensor - shield for STM32 Nucleo with display and MEMS sensors) makete per I2C sąsaja yra prijungtas skaitmeninis slėgio sensorius LPS25HB (angl. MEMS pressure sensor), kuris matuoja slėgį bei temperatūra.

Diagram, schematic

Description automatically generated A close-up of a computer chip

Description automatically generated with medium confidence

A picture containing graphical user interface

Description automatically generated

Panagrinėsime pavyzdinį projektą kuriame 1-os sekundės periodiškumu per I2C sąsaja nuskaitomi skaitmeninio sensoriaus (LPS25HB) temperatūros duomenys kurie sveikąją forma (be kablelio) atvaizduojami Kamami makete sumontuotame 4 skaitmenų 7 segmentų ekranėlyje. Atsisiųskite pavyzdinį projektą iš svetainės Github.com:

<https://github.com/euronymous1991/LPS25HB_MCU_LAB>

Graphical user interface, text, application, email

Description automatically generated

**1 lentelė.** Projekte naudojamų papildomų failų funkcijos

|  |  |
| --- | --- |
| Failo pavadinimas | Atliekama funkcija |
| Core/Src/4DIG\_7SEG\_DISPLAY.c  Core/Inc/4DIG\_7SEG\_DISPLAY.h | 4 skaitmenų 7 segmentų ekrano aptarnavimo C kodas |
| Core/Src/LPS25HB.c  Core/Inc/LPS25HB.h | Skaitmeninio LPS25HB sensoriaus aptarnavimo C kodas |

Toliau detaliau aptarsime LPS25HB.c ir LPS25HB.h failus, bei kaip jų pagalba nuskaitomi LPS25HB temperatūros duomenys, žemiau pateikti šių failų kodai:

*LPS25HB.c*

|  |
| --- |
| #include "main.h"  #include "LPS25HB.h"  uint8\_t chipID**;**  uint8\_t datatowrite **=** 0**;**  uint8\_t TEMP\_OUT\_H**,** TEMP\_OUT\_L**;** // RAW DATA  int16\_t val**;**  uint8\_t LPS25HB\_Initialise**(**I2C\_HandleTypeDef **\***i2cHandle**){**    // Pradzioje patikriname ID  HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_WHO\_AM\_I**,** 1**,** **&** chipID**,** 1**,** 1000**);**  if **(** chipID **!=** LPS25HB\_DEVICE\_ID**)** **{**  return 1**;}** //jei ID nesutampa inicializacija nutraukiama    datatowrite **=** 0x01**<<**7**;** //power on PD  if **(**HAL\_I2C\_Mem\_Write**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_CTRL\_REG1**,** 1**,** **&**datatowrite**,** 1**,** 100**)** **!=** HAL\_OK**)** **{**  return 1**;**  **}**  return 0**;**  **}**  float LPS25HB\_Measure\_Temperature**(**I2C\_HandleTypeDef **\***i2cHandle**)** **{**    datatowrite **=** 0x01**<<**0**;** // One shot mode  if **(**HAL\_I2C\_Mem\_Write**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_CTRL\_REG2**,** 1**,** **&**datatowrite**,** 1**,** 100**)** **!=** HAL\_OK**)** **{**  return 1**;**  **}**    HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_TEMP\_OUT\_L**,** 1**,** **&**TEMP\_OUT\_L**,** 1**,** 100**);**  HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_TEMP\_OUT\_H**,** 1**,** **&**TEMP\_OUT\_H**,** 1**,** 100**);**  val **=** **(**TEMP\_OUT\_H **<<** 8**)** **|** **(**TEMP\_OUT\_L**);**  return 42.5f **+** val **/** 480.0f**;**  **}** |

*LPS25HB.h*

|  |
| --- |
| /\*---------------------Adresai------------------------------------------\*/  #define LPS25HB\_ADDRESS 0x5C << 1 //Note that SA0 = 0 so address is 1011101 and not 1011100  #define LPS25HB\_DEVICE\_ID 0xBD //Device ID, the value in the WHO\_AM\_I Register  /\*---------------------Registrai-------------------------------\*/  #define LPS25HB\_WHO\_AM\_I 0x0F //Who am I register location  #define LPS25HB\_STATUS\_REG 0x27  //Tells whether the Pressure Data is ready or is being overrun  #define LPS25HB\_TEMP\_OUT\_L 0x2B // Temperature output value (LSB)  #define LPS25HB\_TEMP\_OUT\_H 0x2C // Temperature output value (MSB)  #define LPS25HB\_CTRL\_REG1 0x20 //Contains PD, BDU and more  #define LPS25HB\_CTRL\_REG2 0x21 //Contains one-shot mode and FIFO settings  #define LPS25HB\_RES\_CONF 0x10 //Pressure and temperature Resolution  /\*---------------------Konfiguraciniai bitai--------------------------------\*/  #define LPS25HB\_CTRL\_REG1\_PD 0x80 //Power Down when 0, active mode when 1 (Default 0)  #define LPS25HB\_CTRL\_REG2\_ONE\_SHOT 0x1 //One shot mode enabled, obtains a new dataset  #define LPS25HB\_RES\_CONF\_AVGP0 0x1 //Pressure resolution Configuration  #define LPS25HB\_RES\_CONF\_AVGP1 0x2 //Pressure resolution Configuration  #define LPS25HB\_STATUS\_REG\_PDA 0x2 //Pressure data available  /\* Private function prototypes -----------------------------------------------\*/  /\* USER CODE BEGIN PFP \*/  uint8\_t LPS25HB\_Initialise**(**I2C\_HandleTypeDef **\***i2cHandle**);**  float LPS25HB\_Measure\_Temperature**(**I2C\_HandleTypeDef **\***i2cHandle**);**  /\* USER CODE END PFP \*/ |

LPS25HB.h preambulės faile surašyti reikalingi adresai bei registrai reikalingi LPS25HB konfigūracijai norint nuskaityti temperatūrą, ši informacija yra paimta iš LPS25HB datasheet (LPS25HB\_dokumentacija/LPS25HB\_datasheet.pdf) registrų žemėlapio (Register mapping).

Table

Description automatically generated

Bei dviejų vienintelių funkcijų prototipai:

|  |
| --- |
| uint8\_t LPS25HB\_Initialise**(**I2C\_HandleTypeDef **\***i2cHandle**);** |

**ir**

|  |
| --- |
| float LPS25HB\_Measure\_Temperature**(**I2C\_HandleTypeDef **\***i2cHandle**);** |

LPS25HB.c faile yra pateikiamas prieš tai minėtų dviejų funkcijų kodai. LPS25HB\_Initialise**(**I2C\_HandleTypeDef **\***i2cHandle**);**

Ši funkcija skirta sensoriaus inicializavimui panagrinėkime šią funkcija atidžiau.

|  |
| --- |
| uint8\_t LPS25HB\_Initialise**(**I2C\_HandleTypeDef **\***i2cHandle**){**    // Pradzioje patikriname ID  HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_WHO\_AM\_I**,** 1**,** **&** chipID**,** 1**,** 1000**);**  if **(** chipID **!=** LPS25HB\_DEVICE\_ID**)** **{**  return 1**;}** //jei ID nesutampa inicializacija nutraukiama    datatowrite **=** 0x01**<<**7**;**  if **(**HAL\_I2C\_Mem\_Write**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_CTRL\_REG1**,** 1**,** **&**datatowrite**,** 1**,** 100**)** **!=** HAL\_OK**)** **{**  return 1**;**  **}**  return 0**;**  **}** |

LPS25HB turi naudinga registrą pavadinimu WHO\_AM\_I (0Fh), kurį nuskaičius gaunamas sensoriaus ID, kuris LPS25HB atveju turėtų būti 1011 1101 (0xBD).

Chart

Description automatically generated Tai galima atlikti pasinaudojant žemiau pavaizduota funkcija:

|  |
| --- |
| HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_WHO\_AM\_I**,** 1**,** **&**chipID**,** 1**,** 1000**);** |

Naudojantis šia funkcija nuskaitomas LPS25HB\_WHO\_AM\_I registre esanti reikšmė kuri išsaugoma kintamajame chipID, jeigu registre esanti reikšmė nesutampa su datasheet pažymėta reikšme tai yra 0xBD, tolimesnė LPS25HB inicializacija yra beprasmė, dėl to išeinama iš funkcijos ir pranešamas klaidos pranešimas šiuo atveju return 1.

|  |
| --- |
| if **(** chipID **!=** LPS25HB\_DEVICE\_ID**)** **{**  return 1**;}** //jei ID nesutampa inicializacija nutraukiama |

Atkreipkime dėmesį į registrą CTRL\_REG1 (20h)

Table

Description automatically generated

CTRL\_REG1 (20h) registre vyriausio bito (PD) pagalba galima aktyvuoti arba deaktyvuoti sensorių, kaip matome iš datasheet, pradinė vertė yra nulis, kas reiškia, kad sensorius yra neaktyvus. Norint atlikti sensoriaus aktyvacijai į vyriausiąjį bitą įrašomas loginis vienetas:

Ir įrašomas į CTRL\_REG1 registrą naudojantis žemiau pateikta funkcija:

|  |
| --- |
| datatowrite **=** 0x01**<<**7**;**  if **(**HAL\_I2C\_Mem\_Write**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_CTRL\_REG1**,** 1**,** **&**datatowrite**,** 1**,** 100**)** **!=** HAL\_OK**)** **{**  return 1**;**  **}** |

Jei komunikacija tarp sensoriaus ir mikrovaldiklio nepavyksta ir HAL\_I2C\_Mem\_Write funkcija gražina ne HAL\_OK statusą, išmetamas klaidos pranešimas (return 1), jei viskas įvyksta be klaidų baigdami inicializacijos procedūrą gražiname 0 (return 0).

Taip pat galima atkreipti dėmesį į kitus konfigūracinius bitus kaip ODR [2:0], kurių pagalba galima konfigūruoti duomenų atnaujinimo dažnį tačiau šiam pavyzdžiui LPS25HB bus naudojamas „ONESHOT“ rėžimas, dėl to jų keisti šiai akimirkai nereikia ir galima palikti pradines vertes tai yra „000“.

Toliau projekte belieka vienintelė funkcija tai yra:

|  |
| --- |
| float LPS25HB\_Measure\_Temperature**(**I2C\_HandleTypeDef **\***i2cHandle**);** |

Kuri atlieka temperatūros nuskaitymą, bei neapdorotų duomenų konvertavimą į Celsijus, panagrinėkime šią funkcija atidžiau.

|  |
| --- |
| float LPS25HB\_Measure\_Temperature**(**I2C\_HandleTypeDef **\***i2cHandle**)** **{**    datatowrite **=** 0x01**<<**0**;** // One shot mode  if **(**HAL\_I2C\_Mem\_Write**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_CTRL\_REG2**,** 1**,** **&**datatowrite**,** 1**,** 100**)** **!=** HAL\_OK**)** **{**  return 1**;**  **}**    HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_TEMP\_OUT\_L**,** 1**,** **&**TEMP\_OUT\_L**,** 1**,** 100**);**  HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_TEMP\_OUT\_H**,** 1**,** **&**TEMP\_OUT\_H**,** 1**,** 100**);**  val **=** **(**TEMP\_OUT\_H **<<** 8**)** **|** **(**TEMP\_OUT\_L**);**  return 42.5f **+** val **/** 480.0f**;**  **}** |

Kaip buvo pastebėta iš registro CTRL\_REG1 jei parenkiamas „ONESHOT“ veikimo rėžimas reikia pasinaudoti CTRL\_REG2 registru norint gauti kiekvieną kartą naujus duomenis iš LPS25HB sensoriaus. CTRL\_REG2 (21h) registras:

Table

Description automatically generated

Norint kad LPS25HB išduotų naujus duomenis ONE\_SHOT rėžimu, CTRL\_REG2 registro jauniausiame bite „ONE\_SHOT“ reikia įrašyti loginį vienetą:

|  |
| --- |
| datatowrite **=** 0x01**<<**0**;** // One shot mode  if **(**HAL\_I2C\_Mem\_Write**(** i2cHandle**,** LPS25HB\_ADDRESS**,** LPS25HB\_CTRL\_REG2**,** 1**,** **&**datatowrite**,** 1**,** 100**)** **!=** HAL\_OK**)** **{**  return 1**;**  **}** |

Tai padarius naujausi LPS25HB temperatūros registruose yra atnaujinami temperatūros duomenys kuriuos galima tada nuskaityti, temperatūros sensorius yra 16 bitų, dėl to šie duomenys yra saugomi dvejuose registruose TEMP\_OUT\_L(2Bh) LSB dalis ir TEMP\_OUT\_H(2Ch) MSB dalis.

Table

Description automatically generated

Table

Description automatically generated

Šie abu registrai nuskaitomi dvejomis funkcijomis ir išsaugomi kintamuosiuose TEMP\_OUT\_L ir TEMP\_OUT\_H.

|  |
| --- |
| HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** 0x2B**,** 1**,** **&**TEMP\_OUT\_L**,** 1**,** 100**);**  HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** 0x2C**,** 1**,** **&**TEMP\_OUT\_H**,** 1**,** 100**);** |

Kurių duomenys po turi būti apjungiami viename short 16 bitų tipo kintamajame:

|  |
| --- |
| val **=** **(**TEMP\_OUT\_H **<<** 8**)** **|** **(**TEMP\_OUT\_L**);** |

Pabandykime nuskaityti šias reikšmes:

|  |  |
| --- | --- |
| Šešioliktainis formatas | Dešimtainis formatas |
|  |  |

Matome kad nei šešioliktainiame nei dešimtainiame formate šie duomenys neneša mums suprantamos informacijos, tai yra dėl to nes šie duomenys neapdoroti (angl. RAW DATA), jų apdorojimas pavaizduotas žemiau, (iškarpa iš failo LPS25HB\_dokumentacija/ LPSHB\_duomenu\_interpretacija.pdf )

Graphical user interface

Description automatically generated with medium confidence

Naudojantis čia informacija gaunama funkcijos (LPS25HB\_Measure\_Temperature) gražinama temperatūros vertė Celsijais.

|  |
| --- |
| return 42.5f **+** val **/** 480.0f**;** |

Derinimo režime kintamųjų peržiūros lange patikrinkite, kaip kinta neapdoroti temperatūros duomenys (TEMP\_OUT\_L, TEMP\_OUT\_H, val) bei konvertuoti temperatūra Celsijais (temperatura). Temperatūra galima keisti priliečiant temperatūros jutiklį pirštu, taip jį šiek tiek sušildant.

Graphical user interface, text, application

Description automatically generated

O ekranėlyje turėtų būti matoma LPS25HB sensoriaus išmatuota temperatūra vertė be kablelio.

A picture containing qr code

Description automatically generated

Naudodamiesi praeituose dvejuose laboratoriniuose įgautomis žiniomis savarankiškai išsiaiškinkite kaip šiame pavyzdiniame projekte veikia duomenų atvaizdavimas ekranėlyje, bei kaip yra sukonfigūruotas Timer2 laikmatis.

Taip pat programų pavyzdžių šablonų galima rasti kataloge C:\Users\XXX\STM32Cube\Repository\STM32Cube\_FW\_L0\_V1.11.3\Projects\NUCLEO-L073RZ\Examples\I2C\. \* XXX yra katalogas, priskirtas Windows sistemos vartotojui „User“ darbiniame kompiuteryje.

Graphical user interface, text, application

Description automatically generated

# Užduotys

1. Pakeiskite programa taip jog 4 skaitmenų 7 segmentų ekranėlyje būtų atvaizduojama temperatūra su dviem skaičiais po kablelio (kablelis turi būti atvaizduojamas ekranėlyje).
2. Sukurkite naują funkciją kurios pagalba būtų nuskaitomi slėgio (mBar) duomenys, bei jie atvaizduojami 4 skaitmenų 7 segmentų ekranėlyje.
3. Pakeiskite duomenų nuskaitymo periodą, tokių principu: studento AIS kodo galutiniai du skaičiai \* 100 ms. Pvz.: jei galiniai skaičiai yra 15 tai duomenys turi būti nuskaitomi kas 1.5s. Jei abu galiniai skaičiai yra 00 tai nuskaitymo periodas turi būti 0.1 sek.
4. Pakeiskite programą taip kad kiekvieną kartą paspaudus mygtuką pasikeistų ekrane atvaizduojami duomenys iš slėgio į temperatūra ir atgal.