Laboratorinis darbas Nr. 4.

Slėgio jutiklio LPS25HB nuskaitymas per I2C sąsają.

**Turinys**

[1 Darbo tikslai 2](#_Toc115109348)

[2 Analoginio signalo įvestis 2](#_Toc115109349)

[3 Signalo generavimas panaudojant Skaitmeninį-Analoginį Keitiklį (SAK, angl. DAC) 5](#_Toc115109350)

[3.1 Prievadų funkcijų konfigūravimas 5](#_Toc115109351)

[3.2 Taktavimo sistemos konfigūravimas (Clock system configuration) 7](#_Toc115109352)

[3.3 Signalo generavimo laikmačio konfigūravimas 7](#_Toc115109353)

[3.4 Keil projekto šablono generavimas 9](#_Toc115109354)

[3.5 Projekto šablono papildymas C programos tekstu 9](#_Toc115109355)

[3.5.1 Kuriamo programos kodo paskirtis 9](#_Toc115109356)

[3.5.2 Programinė įranga 10](#_Toc115109357)

[3.6 Programos veikimo tikrinimas 11](#_Toc115109358)

[4 Užduotys 11](#_Toc115109359)

[5 Priedas 1. Skaitmeninio kodo konvertavimas į įtampos vienetus 12](#_Toc115109360)

# Darbo tikslai

1. Išmokti realizuoti analoginio signalo kvantavimą ir diskretizavimą užsiduota sparta (diskretizavimo dažniu), panaudojant ASK (Analoginį-Skaitmeninį Keitiklį, angl. ADC);
2. Išmokti realizuoti analoginio bet kokios formos signalo generavimą su užduotais parametrais (dažniu ir amplitude), panaudojant SAK (Skaitmeninį-Analoginį- Keitiklį, angl. DAC)

# Analoginio signalo įvestis

Kamami makete yra prijungtas skaitmeninis slėgio sensorius LPS25HB (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

Parodysime, kaip išmatuoti LDR1 varžą. Tam pradžioje išmatuosime įtampą prievade PA1, o paskui pasinaudodami VDD ir R8 vertėmis apskaičiuosime varžą naudodami išraišką (išvedama iš varžinio daliklio įtampos perdavimo lygties)

*,* (1)

čia maketo maitinimo įtampa VDD=3,3 V, atraminė varža .

ASK gali būti nuskaitomas nuolatinės apklausos būdu (polling), aptarnaujant pertrauktį iš keitiklio arba panaudojant tiesioginių mainų kanalą. Kiekvieno iš būdų realizavimui galima naudoti atitinkamas HAL bibliotekos funkcijas ir pavyzdinį šabloną. 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

Panagrinėsime pavyzdį, kai ASK nuskaitoma panaudojant nuolatinės apklausos būdą ir jį realizuojančias HAL bibliotekos funkcijas. CubeMx projekto sukūrimui parsisiųskite iš Moodle projekto šabloną *\*.ioc* iš laboratorinio darbo katalogo. Papildykite main.c ir stm32l0xx\_it.c failą žemiau parodytais fragmentais.

*Main.c*

|  |
| --- |
| /\* USER CODE BEGIN Includes \*/  #include "LPS25HB.h"  /\* USER CODE END Includes \*/ |
| /\* USER CODE BEGIN PV \*/  float temperatura**;** // Temperatura  uint32\_t laikas**=**0**;**  /\* USER CODE END PV \*/ |
| /\* USER CODE BEGIN 2 \*/  LPS25HB\_Initialise**(&**hi2c2**);**  /\* USER CODE END 2 \*/  /\* Infinite loop \*/  /\* USER CODE BEGIN WHILE \*/  while **(**1**)**  **{**  if**(**laikas**>=**500**){**  temperatura **=** LPS25HB\_Measure\_Temperature**(&**hi2c2**);**  laikas**=**0**;**  **}**  /\* USER CODE END WHILE \*/ |

Iš Moodle puslapio atsisiunčiame Pagalbinius failus lab. darbui Nr. 4, LPS25HB.c ir LPS25HB.h šių failų turiniai pateikti žemiau.

*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.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 **=** LPS25HB\_CTRL\_REG1\_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 **=** LPS25HB\_CTRL\_REG2\_ONE\_SHOT**;** // 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**,** 0x2B**,** 1**,** **&**TEMP\_OUT\_L**,** 1**,** 100**);**  HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** 0x2C**,** 1**,** **&**TEMP\_OUT\_H**,** 1**,** 100**);**  val **=** **(**TEMP\_OUT\_H **<<** 8**)** **|** **(**TEMP\_OUT\_L**);**  return 42.5f **+** val **/** 480.0f**;**  **}** |

LPS25HB.h preambulės faile surašyti reikalingi adresai, registrai bei konfiguraciniai bitai reikalingi temperatūros nuskaitymui, ši informacija yra paimta iš LPS25HB datasheet registro žemėlapio (Register mapping).

Table

Description automatically generated

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 **=** LPS25HB\_CTRL\_REG1\_PD**;**  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 iš LPS25HB.h paimamas konfiguracinis bitas:

|  |
| --- |
| #define LPS25HB\_CTRL\_REG1\_PD 0x80 //Power Down when 0, active mode when 1 (Default 0) |

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

|  |
| --- |
| datatowrite **=** LPS25HB\_CTRL\_REG1\_PD**;**  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 nuskaitymo 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 **=** LPS25HB\_CTRL\_REG2\_ONE\_SHOT**;** // 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**,** 0x2B**,** 1**,** **&**TEMP\_OUT\_L**,** 1**,** 100**);**  HAL\_I2C\_Mem\_Read**(** i2cHandle**,** LPS25HB\_ADDRESS**,** 0x2C**,** 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 vienetą, kurį galima pasiimti iš LPS25HB.h failo.

|  |
| --- |
| #define LPS25HB\_CTRL\_REG2\_ONE\_SHOT 0x1 //One shot mode enabled, obtains a new dataset |

Bei jį įrašyti panašiu principu kaip inicializacijoje:

|  |
| --- |
| datatowrite **=** LPS25HB\_CTRL\_REG2\_ONE\_SHOT**;** // 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 temperatūros duomenys atsiranda temperatūros registruose kuriuos galima 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

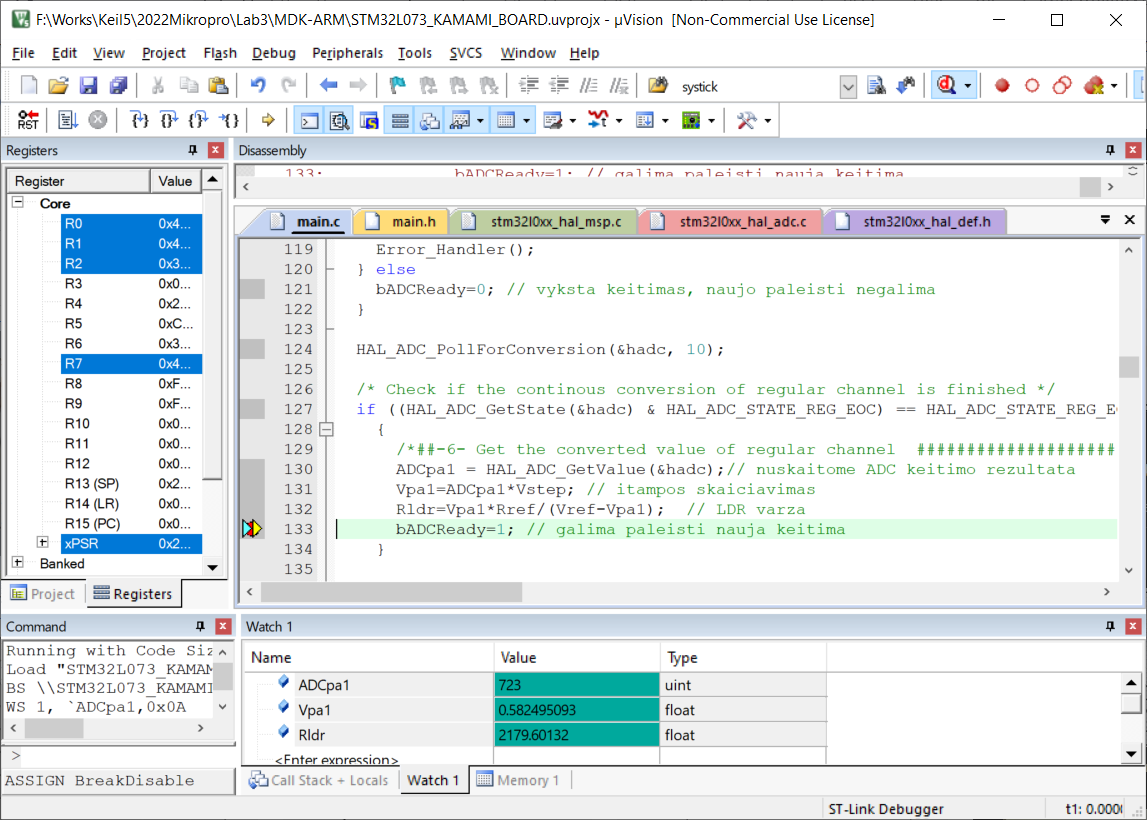
Table

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**);** |

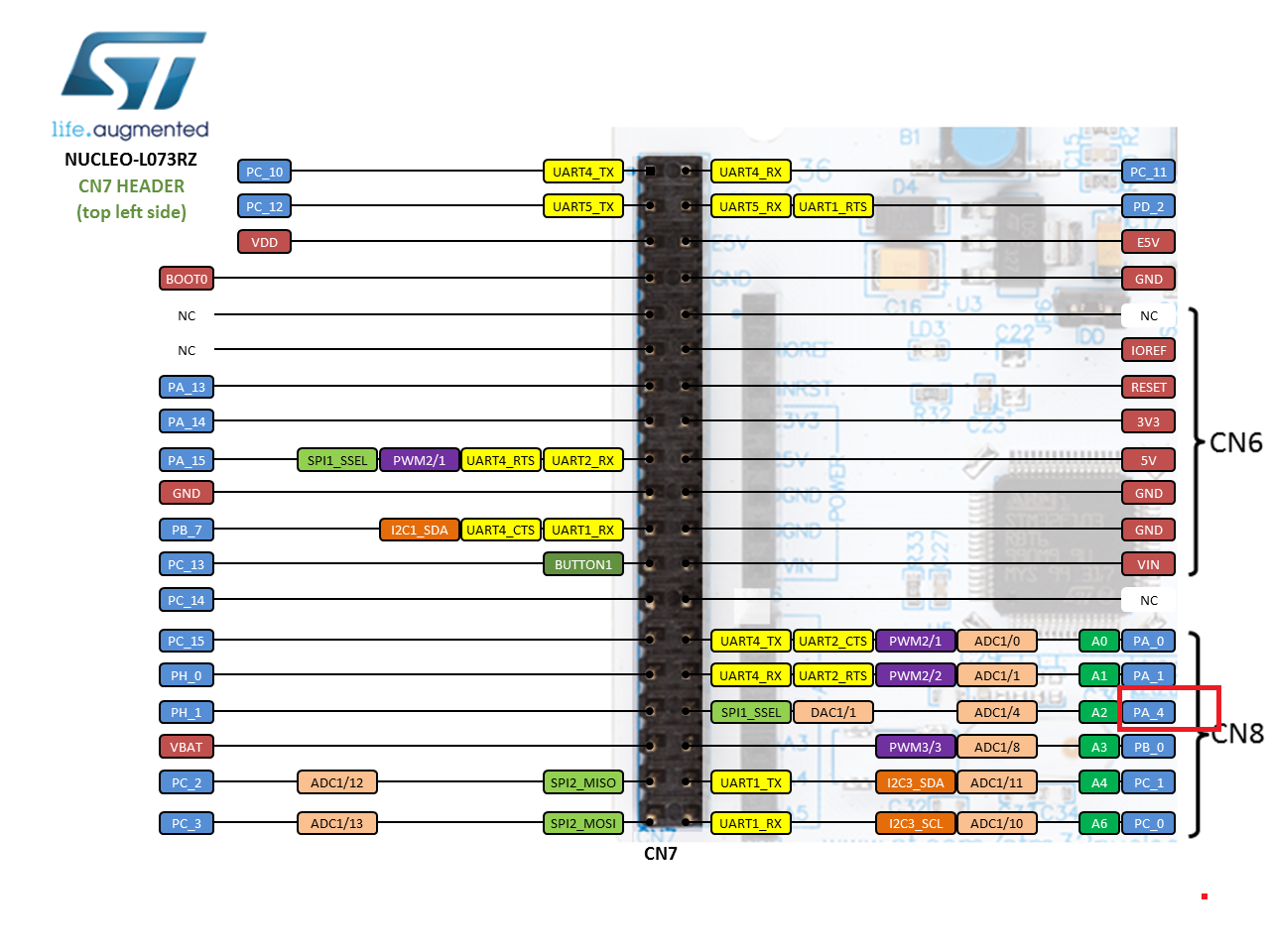
stabdymo tašką (breakpoint) kaip parodyta žemiau ir derinimo režime kintamųjų peržiūros lange patikrinkite, kaip priklauso ASK išmatuojamas kodas (ADCpa1), suskaičiuota įtampa (Vpa1) ir suskaičiuota fotorezistoriaus varža (Rldr), priklausomai nuo LDR1 apšvietimo. Apšvietimą galima kaitalioti pridengiant lapeliu LDR1 korpusą.

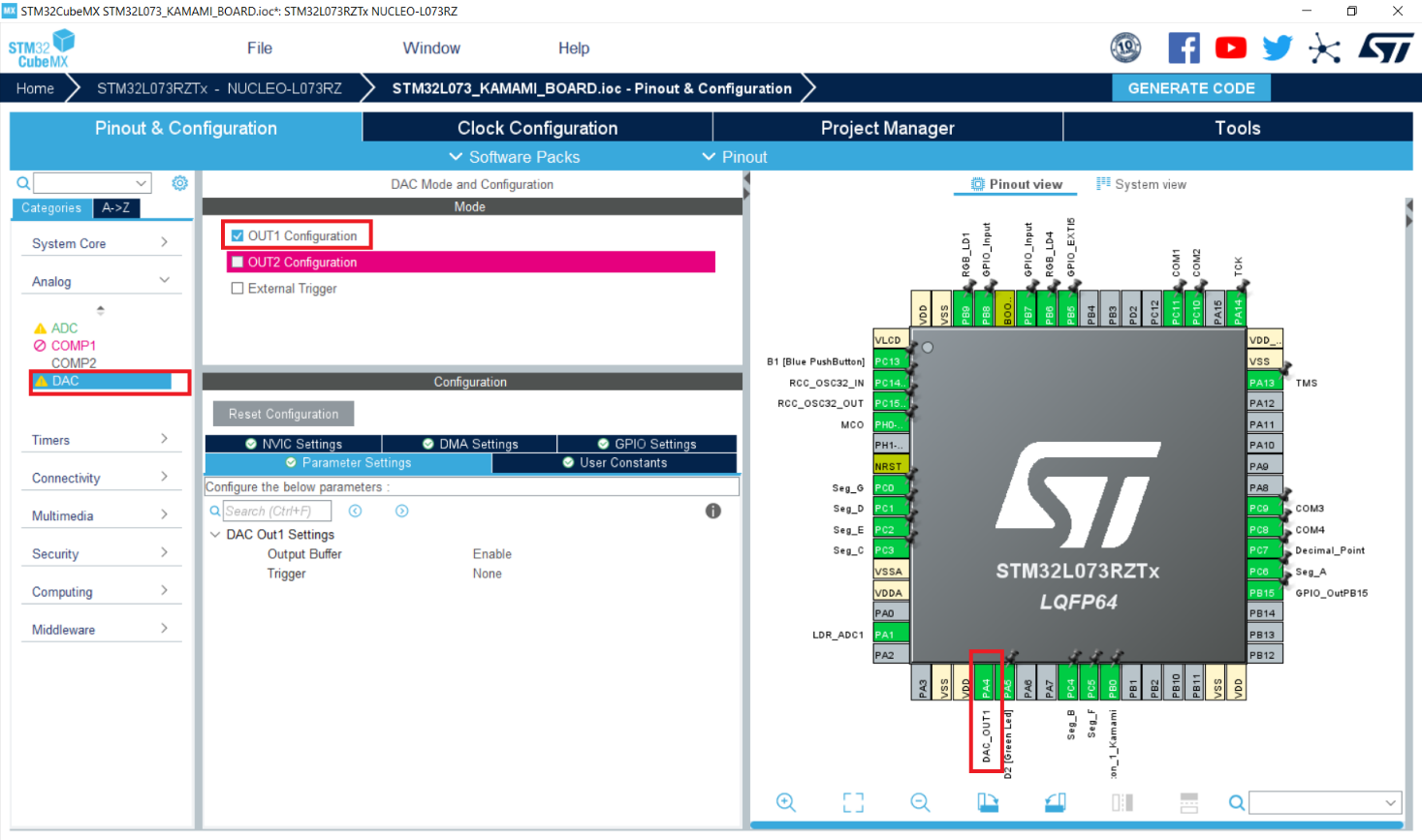


# Signalo generavimas panaudojant Skaitmeninį-Analoginį Keitiklį (SAK, angl. DAC)

## Prievadų funkcijų konfigūravimas

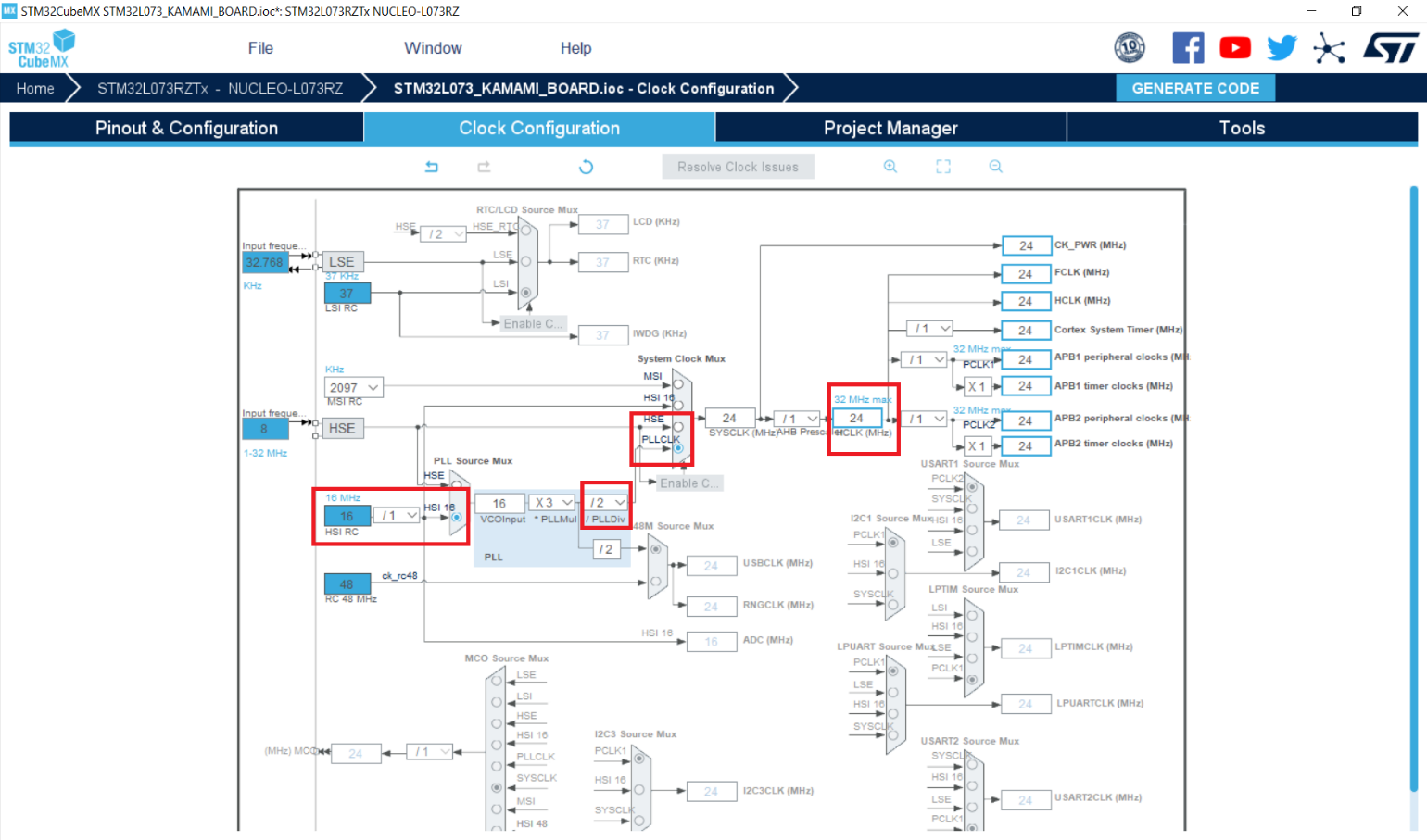
SAK išėjimas DAC\_OUT1 yra prijungtas prie prievado PA4 (jungtis CN7) makete Nucleo-L073RZ. CubeMx projekte atitinkamai parinkime prievado PA4 darbo režimą DAC\_OUT1 (žr. žemiau).





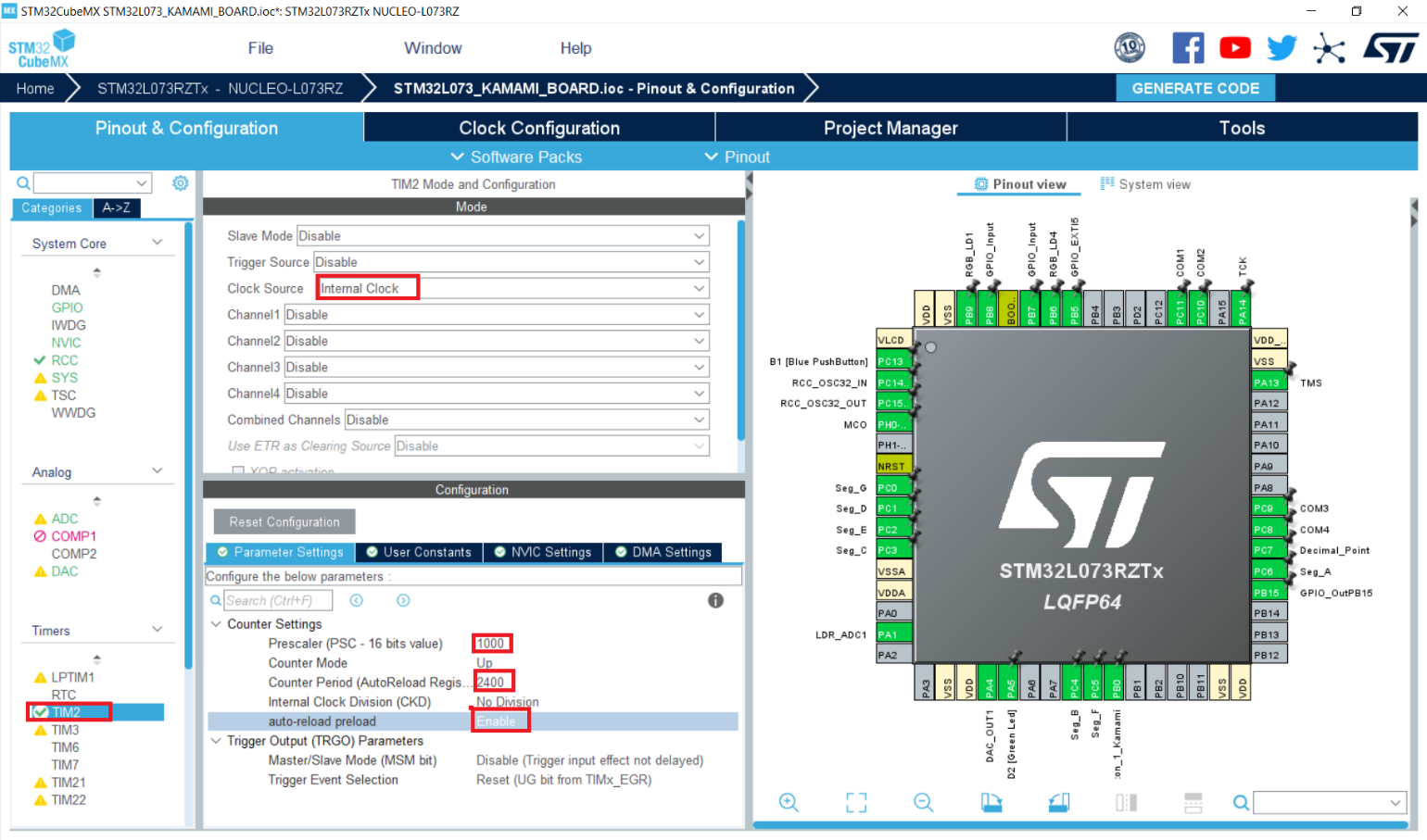
## Taktavimo sistemos konfigūravimas (Clock system configuration)

Panelėje **Clock Configuration** pasirinkite žemiau parodytus nustatymus.



## Signalo generavimo laikmačio konfigūravimas

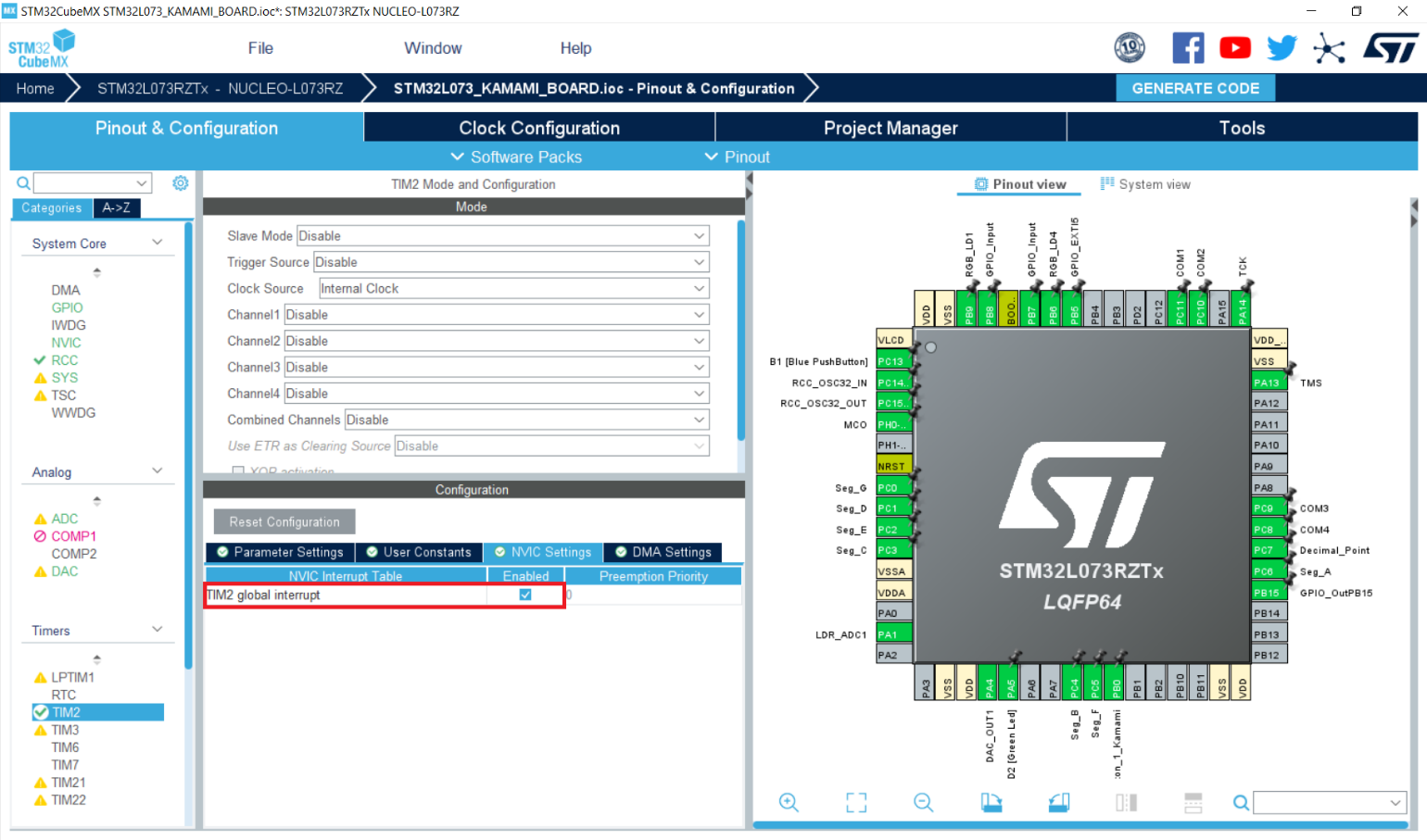
Laikmatį TIM2 panaudosime signalo generavimui (diskretizavimo periodo nustatymui). Sukonfigūruokite laikmatį kaip parodyta žemiau.



Pasirinkę šiuos nustatymus, sukonfigūruojame laikmatį TIM2 generuoti pertrauktis tokiu dažniu

**FINT=FSOURCE/ Prescaler / Counter Period= 24 MHz/1000/2400=10 Hz.**

Panelėje **NVIC Settings** aktyvuokite TIM2 globalų pertraukties aptarnavimo pasirinkimą (bitą, vėliavėlę).



## Keil projekto šablono generavimas

Nepamirškite reikiamų nustatymų panelėje **Project Manager**, su kuriais susipažinote praeituose laboratoriniuose darbuose. Sugeneruokite projekto šabloną aplinkai Keil MDK-ARM, paspausdami **GENERATE CODE** mygtuką.

## Projekto šablono papildymas C programos tekstu

### Kuriamo programos kodo paskirtis

Projekto šabloną papildykime programos kodu, kuris realizuoja pjūklo (angl. *sawtooth*) formos signalo generavimą, kaip parodyta 1 pav.



1 pav. Pjūklo formos signalas

### Programinė įranga

Projekto modulyje *main.c* aktyvuokite laikmatį TIM2.

/\* USER CODE BEGIN 2 \*/

if**(**HAL\_TIM\_Base\_Start\_IT**(&**htim2**)** **!=** HAL\_OK**)**

**{**

//ErrorHandler();

**}**

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

while **(**1**)**

**{**

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

**}**

Projekto modulyje *main.c* realizuokite laikmačio TIM2 aptarnavimo (Callback) funkciją HAL\_TIM\_PeriodElapsedCallback.

*main.c*

#define SIGNAL\_STEPS\_IN\_PERIOD 20 // in samples

#define SIGNAL\_MAGNITUDE 4095 // in samples

static volatile int DACValue**;**

int sample\_no**=**0**;**

/\* TIM2 and TIM6 elapse callback function. Invoked from TIM2\_IRQHandler \*/

void HAL\_TIM\_PeriodElapsedCallback**(**TIM\_HandleTypeDef **\***htim**)**

**{**

if **(**htim**==&**htim2**)**

**{**

DACValue**=**SIGNAL\_MAGNITUDE**\***sample\_no**/**SIGNAL\_STEPS\_IN\_PERIOD**;** /\* calculate new value \*/

HAL\_DAC\_SetValue**(&**hdac**,** DAC\_CHANNEL\_1**,** DAC\_ALIGN\_12B\_R**,** DACValue**);** /\* output DAC \*/

HAL\_DAC\_Start**(&**hdac**,** DAC\_CHANNEL\_1**);** /\* start conversion \*/

if **(**sample\_no**<**SIGNAL\_STEPS\_IN\_PERIOD**)** /\* update DAC sample number \*/

sample\_no**++;**

else

sample\_no**=**0**;**

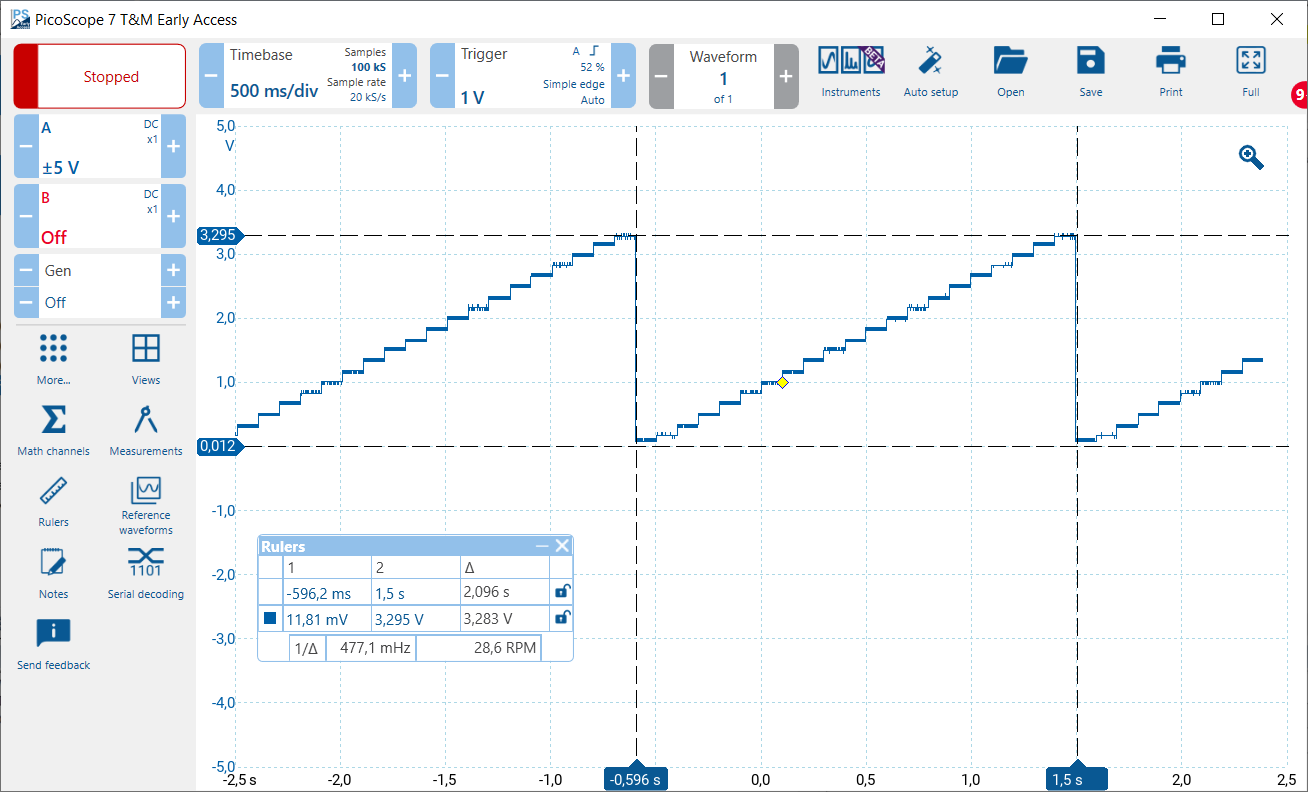
**}**

**}**

/\* USER CODE END 0 \*/

## Programos veikimo tikrinimas

Įkelkite programą, ją paleiskite derinimo režime ir prijungę osciloskopą prie Nucleo-L073 maketo prievado PA4 įsitikinkite, kad yra generuojamas užduotų parametrų pjūklo formos signalas. Kadangi laikmačio persipildymo dažnis buvo nustatytas 10 Hz, o vieną pjūklo formos signalo periodą formuojame iš SIGNAL\_STEPS\_IN\_PERIOD=20 žingsnelių, tai signalo periodas turi būti lygus 2 sek.



# Užduotys

1. Sukurkite programą, kuri nuskaito LDR (Light Dependent Resistor) kas 0,5 sek. ir išsaugo 10 paskutinių verčių masyve. Programa turi uždegti RGB\_LED1 raudoną diodą, jeigu LDR varža viršija 10 kΩ.
2. Susipažinkite su pjūklo formos generavimo ir diskretizavimo pavyzdžiu. Pakeiskite SAK signalo formą taip, kad pjūklo signalo parametrai būtų tokie:
   1. Dažnis F=2 Hz, o laiptelių skaičius per periodą 30.
   2. Įtampų diapazonas nuo Vmin=0,5 V iki Vmax=1 V. Įsitikinkite, kad būtent toks yra sugeneruoto signalo įtampų diapazonas, panaudojant osciloskopą.
3. Pakeiskite SAK projektą taip, kad generuojamo signalo dažnis būtų F=0,5 Hz , o jo forma - sinusas bei diskretizavimo periodas Fs=5 Hz. Signalo forma turi būti saugoma lentelės pavidalu (masyvas) atmintyje (angl. Look-up table).

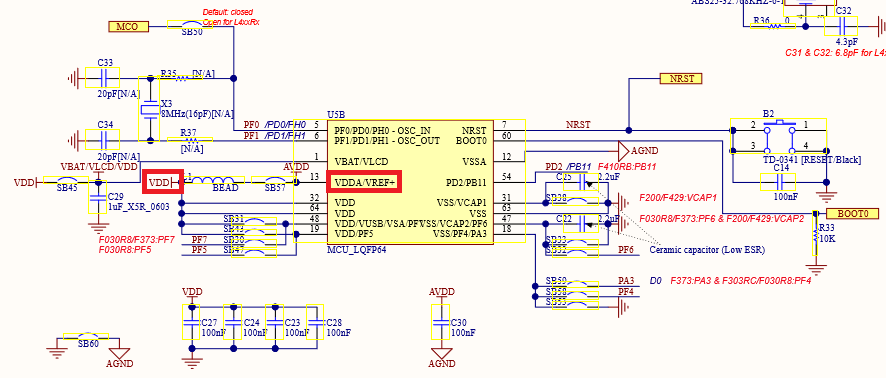
# Priedas 1. Skaitmeninio kodo konvertavimas į įtampos vienetus

Išmatuota įtampa gali būti apskaičiuojama panaudojant išraišką

, (2)

kur yra ASK sugeneruotas imties kodas (galima matyti ir naudojant derinimo langą Watch Keil uVision aplinkoje) ir *n* yra ASK bitų.

Makete Nucleo-L073RZ atraminė įtampa ASK ir SAK moduliams yra VDDA = 3,3 V yra prijungta prie maitinimo linijos VDD, kaip parodyta schemoje žemiau.



Tikslesnę atraminės įtampos VDDA vertę galima nustatyti panaudojant vidinį atraminį šaltinį ir kalibravimo koeficientus saugomus mikrovaldiklyje, kaip aprašyta dokumentacijoje ***Reference Manual. Ultra-low-power STM32L0x3 advanced ARM®-based 32-bit MCUs***:

Calculating the actual VDDA voltage using the internal reference voltage

The VDDA power supply voltage applied to the microcontroller may be subject to variation or not precisely known. The embedded internal voltage reference (VREFINT) and its calibration data acquired by the ADC during the manufacturing process at VDDA = 3 V can be used to evaluate the actual VDDA voltage level. The following formula gives the actual VDDA voltage supplying the device:

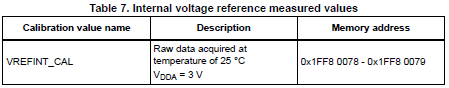
VDDA = 3 V x VREFINT\_CAL / VREFINT\_DATA

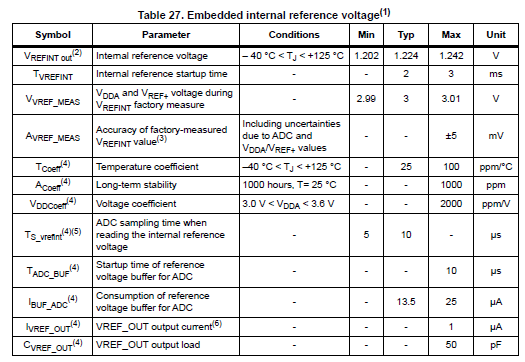
Where:

• VREFINT\_CAL is the VREFINT calibration value

• VREFINT\_DATA is the actual VREFINT output value converted by ADC

**Internal voltage reference (VREFINT)** The internal voltage reference (VREFINT) provides a stable (bandgap) voltage output for the ADC and Comparators. VREFINT is internally connected to the ADC\_IN17 input channel. It enables accurate monitoring of the VDD value (when no external voltage, VREF+, is available for ADC). The precise voltage of VREFINT is individually measured for each part by ST during production test and stored in the system memory area. It is accessible in read-only mode.





if **(**HAL\_ADCEx\_EnableVREFINT**()** **!=** HAL\_OK **)**

**{**

Error\_Handling**();**

**}**

HAL\_SYSCFG\_VREFINT\_OutputSelect**(**SYSCFG\_VREFINT\_OUT\_PB1**);**

The TSEN bit must be set to enable the conversion of ADC\_IN18 (temperature sensor) and the VREFEN bit must be set to enable the conversion of ADC\_IN17 (VREFINT).

#define VREFINT\_CAL\_ADDR 0x1FF80078 (????)

uint16\_t vrefint\_cal**;** // VREFINT calibration value

vrefint\_cal**=** **\*((**uint16\_t**\*)**VREFINT\_CAL\_ADDR**);** // read VREFINT\_CAL\_ADDR memory location

Vdd **=** 3300**\*(\***VREFINT\_CAL\_ADDR**)/**ADC\_raw**;**