**Dispozitiv pentru Măsurarea Temperaturii**

*Proiect pentru Disciplina “Structura Sistemelor de Calcul”*

*2024*

Universitatea Tehnică din Cluj-Napoca

Facultatea de Automatică și Calculatoare

Catedra: Calculatoare și Tehnologia Informației



Student: Pop Tudor-Ștefan

Îndrumător: Radu Darius Răzvan

Grupa: 30236

Cuprins

[I. Introducere 3](#_Toc187892950)

[II. Fundamentare teoretică 3](#_Toc187892951)

[1. Familia de Microcontrollere STM32 & Placa de Dezvoltare STM32L496G-DISCOVERY 3](#_Toc187892952)

[2. Senzorul de Temperatură LM35 5](#_Toc187892953)

[3. Placa de Testare (Breadboard) 6](#_Toc187892954)

[4. Convertorul PMod USB-UART 7](#_Toc187892955)

[III. Proiectare și Implementare 9](#_Toc187892956)

[Configurarea Componentelor Hardware (Proiectare) 9](#_Toc187892957)

[Partea de Software (Implementare) 9](#_Toc187892958)

[1. Mediul de dezvoltare STM32CubeIDE 9](#_Toc187892959)

[a. STM32CubeMX & Configurarea Perifericelor 10](#_Toc187892960)

[b. Codul C pentru Logica Proiectului 12](#_Toc187892961)

[c. Funcția main() & Familia HAL 14](#_Toc187892962)

[2. Ecranul Terminal HTerm 18](#_Toc187892963)

[IV. Mod de Utilizare (Sumar al Părții III.) 19](#_Toc187892964)

[V. Rezultate Experimentale 19](#_Toc187892965)

[VI. Concluzii 19](#_Toc187892966)

[VII. Bibliografie 20](#_Toc187892967)

# Introducere

Acest proiect presupune proiectarea și implementarea unui dispozitiv care măsoară temperatura camerei prin intermediul unui modul **STM32L** pe post de microcontroller, a unui senzor de temperatură (ex. **LM35**) pentru măsurarea efectivă a temperaturii, și a unui convertor **PMod USB-UART** pentru a face legătura dintre microcontroller și un ecran terminal (în acest caz, se va utiliza programul “**HTerm**” pe post de ecran terminal).

Programul va afișa în fiecare secundă pe ecranul terminal temperatura curentă măsurată de către senzorul de temperatură, în urma citirii datelor dintr-un convertor analog-numeric (CAN) (eng. Analog-Digital Converter = ADC) la care va fi legat senzorul LM35.

# Fundamentare teoretică

În cadrul acestui proiect au fost alese următoarele componente hardware:

* O placă de dezvoltare **STM32L496G-DISCOVERY**
* Un senzor de temperatură **LM35**
* O **placă de testare (breadboard) + 9 fire de curent (3 LM35 + 6 PMod)**
* Un convertor **PMod USB-UART**
* **Un cablu USB** pentru conectarea plăcii de dezvoltare la ecranul terminal

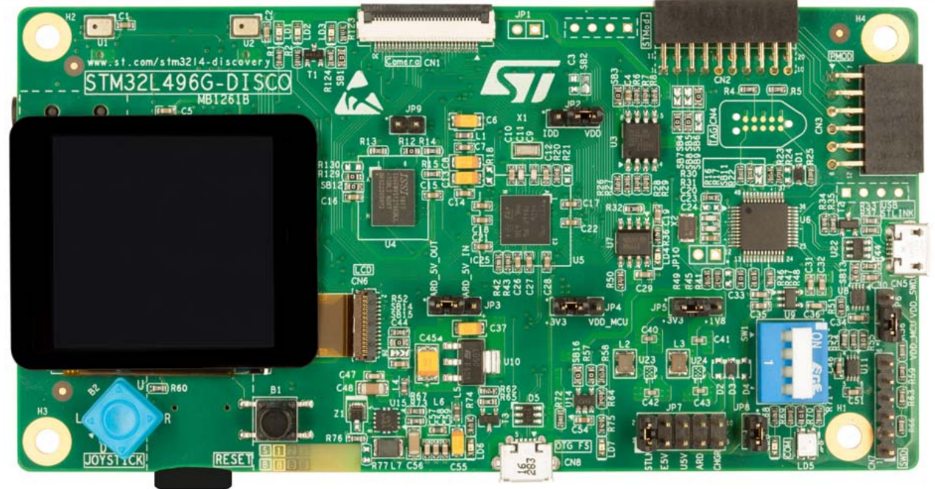
## Familia de Microcontrollere STM32 & Placa de Dezvoltare STM32L496G-DISCOVERY

**STM32** este o familie de microcontrollere pe 32 de biți creată de către compania **STMicroelectronics**. Acestea folosesc procesoare **ARM Cortex-M pe 32 de biți**. Chip-urile STM32 sunt grupate în diferite serii care sunt înrudite în raport cu procesorul ARM pe care îl folosesc, după cum urmează:

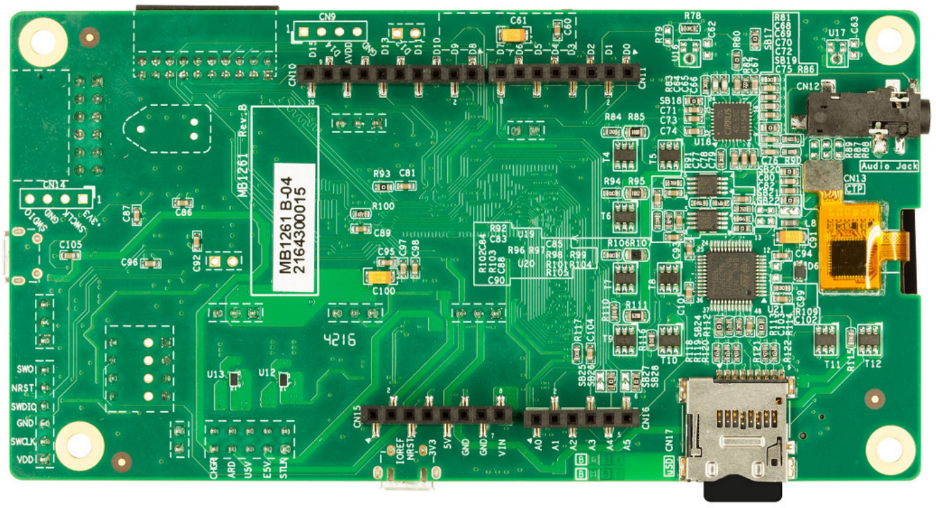
* Seria **F0** utilizează **Cortex-M0**
* Seriile **C0**, **G0**, **L0** și **U0** utilizează **Cortex-M0+**
* Seriile **F1**, **F2** și **L1** utilizează **Cortex-M3**
* Seriile **F3**, **F4**, **G4**, **L4**, și **L4+** utilizează **Cortex-M4**
* Seriile **WB** și **WL** utilizează un **Cortex-M4** și un **Cortex-M0+** (dual core)
* Seria **F7** utilizează **Cortex-M7**
* Seria **H7** utilizează **Cortex-M7** (single core) sau **Cortex-M7 & M4** (dual core)
* Seriile **H5**, **L5**, **U5** și **WBA** utilizează **Cortex-M33**

Microcontroller-ul utilizat în cadrul acestui proiect este **STM32L496AG** din seria **L4**, iar placa de dezvoltare aferentă este **STM32L496G-DISCOVERY** din gama plăcilor de dezvoltare **ST Discovery**, care sunt plăci programabile dedicate utilizării microcontrollerelor corespondente.

Avantajul utilizării acestui microcontroller din seria L4 este că acesta utilizează caracteristici orientate pe **consumul extrem de redus de energie**, **memorie RAM extinsă**, precum și **performanță grafică ridicată**. Prin urmare, acest kit Discovery este proiectat să faciliteze prototipizarea a numeroase aplicații, inclusiv aplicații orientate pe sunet sau grafică, cu o eficiență energetică ultra-modernă. Pentru mai multă ușurință de utilizare, placa conține un program de depanare numit **ST-LINK/V2-1**.



**Figura 1. Placa de dezvoltare STM32L496G-DISCOVERY (Partea Ventrală).**

****

**Figura 2. Placa de dezvoltare STM32L496G-DISCOVERY (Partea Dorsală).**

Pentru mai multe detalii se poate consulta pagina oficială a companiei STMicroelectronics, unde placa de dezvoltare utilizată este descrisă în detaliu:

<https://www.st.com/resource/en/user_manual/um2160-discovery-kit-with-stm32l496ag-mcu-stmicroelectronics.pdf>

## Senzorul de Temperatură LM35

**LM35** este un **senzor analogic de temperatură** **de tip circuit integrat**, creat de către compania **National Semiconductor Corporation** care este compus din dispozitivul de măsurare în sine precum și **3 pini**, dintre care **primul pin (stânga)** este pentru **sursa de alimentare** (senzorul va fi alimentat cu o valoare constantă de **3.3V** de tensiune), **al doilea pin (mijloc)** servește ca **pin de ieșire (Vout)** pentru a fi **conectat la CAN (ADC),** pentru a converti datele analogice citite prin senzor și traductor în date numerice/digitale/discrete, și **al treilea pin (dreapta)** servește ca și **pinul de împământare (GND)**.

Traductorul intern din circuitul integrat LM35 generează exact **10mV/°C**. Așadar, pentru a determina numărul de °C, tensiunea Vout a senzorului **va trebui înmulțită cu 100** pentru a compensa acest factor de scalare.

Însă, chiar și în urma înmulțirii cu 100, tensiunea citită de către CAN din senzorul LM35 **tot nu va fi egală cu temperatura adevărată**, însă, **va fi direct proporțională cu aceasta (în °C)**, spre deosebire de comportamentul altor senzori de temperatură. Spre exemplu, senzorul de temperatură **TMP36** NU crează tensiuni direct proporționale cu temperatura, deoarece la valoarea de **0°C**, CAN-ul va citi o tensiune de **0.5V (500 mV),** care este, prin urmare, un mic defazaj care distruge relația de proporționalitate directă dintre valoarea temperaturii mediului și cea a tensiunii transmise către CAN, defazaj de care programatorul va trebui să țină cont în scrierea codului aferent acestui proiect.

Astfel, valoarea de tensiune care se va afla în CAN va trebui înmulțită cu o constantă pentru determinarea temperaturii adevărate (**cu o marjă de eroare de ±0.25°C**), constantă care va fi discutată în amănunt în secțiunea de descriere a codului.



**Figura 3. Senzorul de temperatură LM35.**

Pentru mai multe considerente teoretice, precum și detalii de funcționalitate ale senzorului, se poate consulta următoarea pagină: <https://www.ti.com/lit/ds/symlink/lm35.pdf>

## Placa de Testare (Breadboard)

O **placă de testare (eng. Breadboard)** este o bază de construcție utilizată pentru construirea unor prototipuri semi-permanente de circuite electronice.

După cum a fost menționat anterior, aceasta va fi utilizată ca și componentă auxiliară pentru conectarea senzorului LM35 la placa de dezvoltare folosită. Pinii senzorului nu pot fi conectați direct la placă din pricina topologiei defavorabile a pinilor plăcii, deoarece aceștia sunt prea îndepărtați unul de altul pentru a putea fi conectați direct la pinii senzorului, după cum se poate observa în cadrul figurilor 2 și 3. (În partea inferioară a figurii 2 se pot observa dezavantajele topologice. În conectorii **CN15** și **CN16** (rândul de pini negri de jos), există pinul **3V3 (alimentarea cu 3.3V)**, unul din cei doi pini **GND (împământarea)** (irelevant care dintre ei), și pinul **A0 (datele de intrare analogice de la senzor + CAN/ADC)**.

Așadar, pe lângă breadboard, vom necesita încă **trei fire de curent** pentru a conecta pinii corespondenți între ei. Prin urmare, datorită numărului redus de fire de curent care ni se impune, este suficient un breadboard de dimensiuni mici, pentru a reduce costul final al proiectului. Placa de testare utilizată în acest proiect va fi, astfel, un **breadboard mini**, care este unul dintre cele mai mici breadboard-uri standard găsite pe piață.



**Figura 4. Un breadboard mini.**

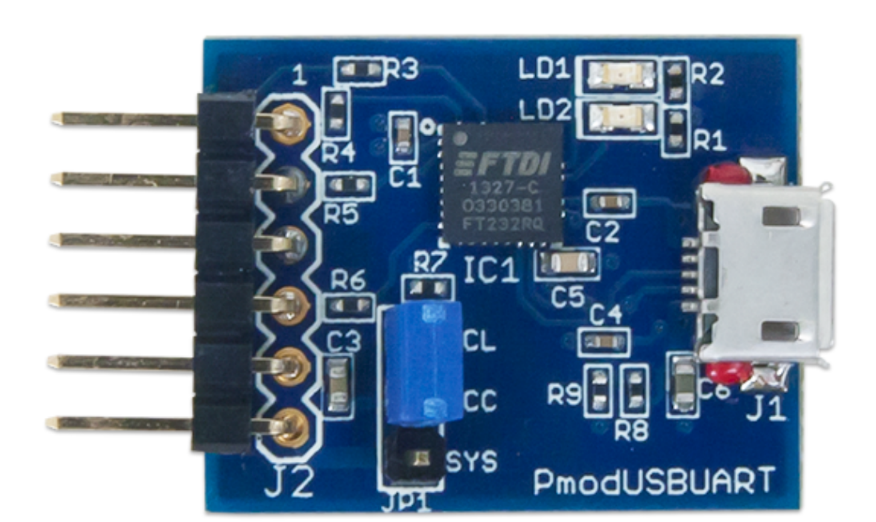
Acum, având figura 4 la îndemână, putem discuta în amănunt funcționalitatea unei plăci de testare și modul în care aceasta ne va facilita munca în cadrul proiectului. Întâi, se pot observa două porțiuni pline de pini în breadboard, care sunt separate de către un șanț central longitudinal. În ciuda faptului că există foarte mulți pini pe placă, **numărul de pini efectivi dintr-o jumătate este egal cu numărul de coloane (i.e. câți pini sunt pe un rând) în acea jumătate.** **Pinii sunt conectați între ei pe verticală, iar șanțul central servește la a separa jumătatea de sus de cea de jos. Spre exemplu, toți pinii care au fost încercuiți cu roșu vor fi legați între ei, însă nu vor fi legați de pinii încercuiți cu verde, și nici de cei marcați cu albastru.**

Ca și consecință, dacă legăm, spre exemplu, pinul de alimentare a senzorului pe rândul marcat cu roșu, putem să legăm un fir între un pin de pe același rând și sursa de alimentare 3V3. Analog, dacă legăm pinul GND pe rândul albastru, putem conecta un fir între un alt pin de pe rândul albastru și pinul GND de pe placa Discovery. Procesul se aplică în mod analog pe orice rând ales de pe breadboard.

## Convertorul PMod USB-UART

Interfața **PMod (Peripheral Module)** este un **standard deschis** definit de către compania **Digilent** pentru conectarea modulelor periferice la plăci de dezvoltare FPGA sau pe bază de microcontrollere, utilizând **6 pini**. Placa STM32L496G-DISCOVERY dispune de conectori PMod în cadrul conectorului **CN3** de pe partea ventrală a plăcii (figura 1, partea din dreapta sus).

**PMod USB-UART** este un modul periferic de conversie serială din USB în UART care este capabil de a transmite date la o rată care poate atinge 3 Mbaud. Modulul utilizează un chip **FTDI FT232RQ** creat de **FTDI (Future Technology Devices International Limited)** al cărui scop este să convertească semnalele UART în semnale USB și viceversa. Conectorul UART (**J2**) permite comunicațiilor UART să pătrundă de la placa conectată (în acest caz, placa Discovery) la chip-ul FTDI, iar conectorul USB (**J1**) permite comunicațiilor să pătrundă de la chip la portul USB.



**Figura 5. Modulul PMod USB-UART**

După cum a fost menționat anterior, și după cum se poate observa în figura 5, acest modul dispune de 6 pini. Pinul 1 se consideră ca fiind pinul din partea de sus, pinul 6 e cel din partea de jos, iar funcționalitățile pinilor vor fi listate acum:

1. *“****RTS****”* ***(Ready to Send)*** -> Un semnal **output** din modul către placa conectată (Discovery) care dictează faptul că modulul PMod este pregătit să primească date de la placă. Acest semnal este fundamental în buna comunicare dintre modul și placă, deoarece previne modulul din a fi supraîncărcat cu date.
2. *“****RXD****”* ***(Receive)*** -> Pinul **input** prin care intră datele din placă în modul. Odată intrate în modul, datele sunt imediat convertite și înaintate către portul USB. Datele pot intra din placă în acest pin numai dacă semnalul RTS (pinul 1) este activ. Evident, acest pin este fundamental comunicării.
3. *“****TXD****”* ***(Transmit)*** -> Pinul **output** din care ies datele din modul înspre placă. Datele ce vor intra în TXD sunt datele provenite din portul USB, și ulterior convertite în semnale UART. Datele vor putea fi înaintate către placă numai dacă pinul CTS (pinul 4) este activ. Acesta este un alt pin, evident, fundamental pentru comunicare.
4. *“****CTS****”* ***(Clear to Send)*** -> Un semnal **input** care provine de la placă spre modulul PMod, care servește ca rol de “undă verde” pentru modul să înainteze datele primite de la portul USB la placa UART. Pentru intuiție, acest semnal se comportă exact ca și semnalul RTS (pin 1), doar că sensul este inversat. În RTS, modulul îi spune plăcii că este pregătit pentru a primi date, iar în CTS, placa îi spune modulului că este pregătită pentru a primi date. Acest pin este fundamental comunicării deoarece asigură flux de date neted și evită transmiterea eronată de date și riscul pierderii datelor în timpul comunicării.
5. *“****GND****”* ***(Ground)*** -> Un pin **bidirectional** care furnizează o referință electrică comună de împământare între cele două dispozitive conectate (modul și placă). Acest pin este necesar pentru comunicarea adecvată între dispozitive.
6. *“****VCC****”* ***(Power Supply)*** -> Sursa de alimentare a modulului. Aceasta poate fi fie de 5V, fie de 3.3V. În contextul nostru, dorim ca aceasta să fie de 3.3V, pentru a fi în concordanță cu sursa de alimentare a senzorului de temperatură. Însă, întrebarea este, **cum reglăm tensiunea de alimentare a modulului?**

Pentru a răspunde la această întrebare, să observăm că în figura 5, se poate observa header-ul **JP1** (format din pinii **LCL, VCC** și **SYS**, precum și un **jumper** de plastic albastru cu lățime de 2 pini). Dacă setăm jumper-ul pe ultimii doi pini (VCC și SYS), modulul va fi alimentat cu 5V de către chip-ul FTDI, iar dacă setăm jumper-ul pe primii doi pini (LCL și VCC), modulul va fi alimentat de către aceeași sursă de alimentare a plăcii conectate prin UART (în acest caz, placa noastră Discovery). Astfel, pentru a alimenta modulul cu 3.3V, va trebui să setăm jumper-ul pe pinul LCL.

O ultimă problemă pe care o prezintă modulul PMod USB-UART este că pinii săi nu se află în aceeași ordine cu pinii conectorului CN3 al plăcii Discovery (pinii modulului se află în ordinea **RTS->RXD->TXD->CTS->GND->VCC**, iar pinii plăcii sunt **CTS->TXD->RXD->RTS->GND->VCC**), motiv pentru care mai avem nevoie de încă 6 fire pentru breadboard.

# Proiectare și Implementare

## Configurarea Componentelor Hardware (Proiectare)

Pentru scopurile acestui proiect, placa de dezvoltare STM32L496G-DISCOVERY va fi ținută cu partea dorsală în sus, deoarece acolo se află pinii necesari proiectului, așadar s-ar facilita conectarea firelor din breadboard în placă, și s-ar evita riscul deteriorării acestora. Cum partea ventrală este cu fața în jos, și modulul PMod USB-UART va trebui ținut cu fața în jos pentru ca pinii acestuia să fie în concordanță cu pinii plăcii Discovery. Acum, tot ce rămâne de făcut este să conectăm componentele:

* Placa se va ține cu partea dorsală în sus și se va lega un cablu Micro USB de la sursa de alimentare (laptop) la placă
* Se va lega senzorul de temperatură LM35 la breadboard
* Se vor trage din breadboard 3 fire din coloanele corespondente fiecărui pin al senzorului în pinii aferenți de pe placa de dezvoltare (Vs+ la 3V3, Vout la A0, și GND la GND)
* Se va lega modulul PMod USB-UART la breadboard
* Precum în cazul senzorului de temperatură, se vor trage încă 6 fire din breadboard de la modul la pinii corespondenți din placă
* Jumperul JP1 al modulului PMod USB-UART se va seta pe pinii LCL și VCC
* Se va lega un alt cablu Micro USB de la modulul USB-UART înapoi la laptop pentru a face conexiunea cu ecranul terminal HTerm, program pe care îl vom descrie ulterior.

## Partea de Software (Implementare)

În cadrul acestui proiect au fost folosite următoarele programe software pentru logica aplicației:

* Mediul de dezvoltare **STM32CubeIDE** pentru a programa logica plăcii de dezvoltare
* Programul terminal **HTerm** pentru comunicarea serială dintre placă și laptop

## Mediul de dezvoltare STM32CubeIDE

**STM32CubeIDE** este un mediu de dezvoltare avansat C/C++ creat de către compania STMicroelectronics, care poate fi utilizat în conjuncție cu plăcile de dezvoltare ST Discovery sau doar cu microcontrollerele STM32 (în funcție de caz) pentru a dicta comportamentul microcontrollerului intern în scopul implementării logicii dorite a unei aplicații. STM32CubeIDE se bazează pe **framework-ul Eclipse®/CDT™** și **toolchain-ul GCC** pentru dezvoltare, respectiv **GDB** pentru depanare.

Acest mediu de dezvoltare dispune de numeroase caracteristici, multe dintre care au fost utilizate în cadrul acestui proiect:

* Configurarea manuală interactivă a perifericelor (o vom descrie ulterior)
* Generarea automată de cod C/C++ care să implementeze configurația hardware specificată anterior
* Redactarea de cod adițional C/C++
* Depanarea prin ST-LINK (program de depanare discutat anterior)

### STM32CubeMX & Configurarea Perifericelor

După crearea unui proiect nou în STM32CubeIDE și alegerea unității de dezvoltare dorite (placă/MCU/MPU), programatorului îi va fi prezentat un fișier care poartă numele proiectului și extensia **.ioc** . Acesta este un fișier bazat pe **XML** dedicat **configurației pinilor, ceasului intern, perifericelor și componentelor middleware.** Fișierul *ioc* furnizează o interfață grafică utilizator pentru configurarea plăcii prin mediul de dezvoltare **STM32CubeMX**, reprezintă o modalitate care facilitează inițializarea componentelor hardware necesare, și este crucial pentru generarea codului C/C++ aferent configurației.



**Figura 6. Fișierul *ioc* al Proiectului & Interfața Grafică STM32CubeMX pentru placa STM32L496G-DISCOVERY**

În partea centrală a figurii 6 se poate observa interfața grafică furnizată de către STM32CubeMX, unde bulinele verzi/galbene/albe/gri reprezintă diferiții pini și surse de alimentare ale plăcii noastre de dezvoltare. Bulinele verzi reprezintă configurații complete și funcționale, al căror cod va fi generat automat de către componenta MX. Bulinele galbene reprezintă configurații *parțial complete:* placa Discovery dispune de diferite protocoale de comunicație precum **3 SPI-uri, 4 I2C-uri, un SDIO, FMC, Quad-SPI,** **3 USART-uri, un ULP UART**, și multe altele, care necesită configurarea a mai multor pini pentru o legătură completă, iar, în cazul în care nu toți pinii sunt mapați, legătura este incompletă, iar pinii vor fi reprezentați cu galben. Bulinele albe reprezintă surse de alimentare **VDD, VSS, VSSA**, iar bulinele gri reprezintă pini nemapați.

Mulți dintre acești pini sunt deja mapați implicit în momentul inițializării proiectului cu această placă, prin urmare, la partea de generare a codului se vor crea funcții de inițializare pentru două protocoale I2C, un protocol SPI, **trei** **convertoare ADC pe 12 biți**, **două USART-uri**, FMC-ul, ULP UART-ul, un SAI, și **pinii GPIO**, dintre care marea majoritate sunt funcții redundante în contextul acestui proiect. Singura dintre aceste inițializări pe care o vom utiliza este unul dintre ADC-uri (**ADC1**) pentru conversia tensiunii generate de senzor în temperatura dorită, și unul dintre cele două USART-uri (**USART1**) pentru comunicarea serială USART (**U**niversal **S**ynchronous **A**synchronous **R**eceiver **T**ransmitter), care este un caz particular al protocolului UART, cu utilizarea adițională a unui semnal de ceas pentru sincronizarea comunicării. IDE-ul mapează automat protocolul USART1 la pinii conectorului CN3 (PMod) în ordinea discutată anterior. Din nefericire, funcționalitatea (ordinea) acestor pini NU poate fi modificată, în sensul în care pinul RTS NU poate fi transformat în pin CTS, și așa mai departe, motiv pentru care încă 6 fire sunt necesare pentru breadboard. Însă, pinul analogic A0 (reprezentat ca **ARD-A0** sau **PC4** în IDE) trebuie setat pe opțiunea **ADC1\_IN13**, cu opțiunea **IN13 Single Ended**.

Printre aceste mapări implicite, IDE-ul va seta următorii parametri impliciți pentru comunicarea serială:

* Rata de transmisie (**Baud rate**) va fi setată la **9600 de biți pe secundă**
* **Lungimea unui cuvânt** va fi de **8 biți** (inclusiv biții de paritate)
* **Nu** vom avea **biți de paritate**
* **Un bit de stop**

După toate aceste mapări, se va da click dreapta pe fișierul *ioc* și se va selecta opțiunea “**Generate Code**” pentru generarea codului C/C++ aferent configurației setate în pasul anterior (în acest caz a fost ales limbajul de programare **C**).

### Codul C pentru Logica Proiectului

După generarea codului C, se va deschide automat fișierul **main.c**,unde vor putea fi observate următoarele linii de cod remarcabile implicite:

1. Liniile 20 și 21:

**#include** "main.h"

**#include** "usb\_device.h"

Întâi, se include biblioteca ”**main.h**”, bibliotecă ce servește pe post de abstracție modulară generată automat de către IDE pentru a seta denumirile pinilor în concordanță cu maparea din partea de configurare cu STM32CubeMX și pentru a implementa componentele necesare realizării proiectului. Spre exemplu, să considerăm următoarele linii de cod din main.h, respectiv, linia 30, linia 196 și linia 254:

**#include** "stm32l4xx\_hal.h"

**#define** ARD\_A0\_Pin GPIO\_PIN\_4

**#define** STMOD\_ADC\_Pin GPIO\_PIN\_4

Linia 30 include biblioteca care conține toate funcțiile și caracteristicile necesare lucrului cu microcontrollere STM32 din seria L4. Denumirea de “**HAL**” din numele bibliotecii este o abreviere pentru “**Hardware Abstraction Layer**”, o familie de funcții și constante care servesc pe post de strat de abstractizare a interacțiunilor codului cu componenta hardware STM32 conectată, pentru ușurința în înțelegere și utilizare.Mai multe detalii despre familia HAL vor fi discutate ulterior. Relația de incluziune este una tranzitivă, așadar, cum biblioteca “stm32l4xx\_hal.h” este inclusă în ”main.h”, și “main.h” este inclusă în “main.c”, știm automat că biblioteca HAL va fi inclusă în fișierul main.c.

Linia 196 (primul *define*) este generată implicit mereu de către IDE, și reprezintă cele două denumiri ale pinului A0 discutate anterior: *ARD-A0* (denumirea de pe placă), și *PC4* (denumirea internă), iar linia 254 (al doilea *define*) este partea pe care am configurat-o noi manual, respectiv, setarea ADC-ului pe același pin (PC4/GPIO\_PIN\_4).

Acum, cea de-a doua incluziune din main.c, respectiv, **#include** "usb\_device.h", are scopul de a include biblioteci relevante pentru munca cu dispositive conectate prin USB. În interiorul acestei biblioteci există trei incluziuni, respectiv: **stm32l4xx.h, stm32l4xx\_hal.h,** și **usbd\_def.h**, precum și o funcție **void MX\_USB\_DEVICE\_Init(void);** care servește ca și inițializare a conexiunii prin USB, funcție a cărei implementare se găsește în **usbd\_def.c**.

1. Liniile 44-68:

ADC\_HandleTypeDef hadc1;

ADC\_HandleTypeDef hadc2;

ADC\_HandleTypeDef hadc3;

DCMI\_HandleTypeDef hdcmi;

DFSDM\_Channel\_HandleTypeDef hdfsdm1\_channel1;

DFSDM\_Channel\_HandleTypeDef hdfsdm1\_channel2;

I2C\_HandleTypeDef hi2c1;

I2C\_HandleTypeDef hi2c2;

UART\_HandleTypeDef hlpuart1;

UART\_HandleTypeDef huart1;

UART\_HandleTypeDef huart2;

SAI\_HandleTypeDef hsai\_BlockA1;

SAI\_HandleTypeDef hsai\_BlockB1;

SD\_HandleTypeDef hsd1;

SPI\_HandleTypeDef hspi2;

SRAM\_HandleTypeDef hsram1;

SRAM\_HandleTypeDef hsram2;

Acestea sunt variabile private generate automat de către IDE pentru a corespunde cu fiecare protocol de comunicare/componentă periferică utilizată în cadrul generării automate de cod. Se pot observa cele 3 ADC-uri, SPI-ul, I2C-urile, USART-urile, ULP UART-ul, și altele. Singurele variabile utilizate în cadrul acestui proiect vor fi **hadc1** (corespondent convertorului ADC1), și **huart1** (corespondent protocolului USART1).

Tipul de date **ADC\_HandleTypeDef** se află în biblioteca **stm32l4xx\_hal\_adc.h** (inclusă în biblioteca ”mare” HAL), și tipul de date **UART\_HandleTypeDef** se află în biblioteca cu denumire analoagă **stm32l4xx\_hal\_uart.h** (de asemenea inclusă în stm32l4xx\_hal.h).

1. Liniile 75-91:

**void** **SystemClock\_Config**(**void**);

**void** **PeriphCommonClock\_Config**(**void**);

**static** **void** **MX\_GPIO\_Init**(**void**);

**static** **void** **MX\_ADC1\_Init**(**void**);

**static** **void** **MX\_ADC2\_Init**(**void**);

**static** **void** **MX\_ADC3\_Init**(**void**);

**static** **void** **MX\_DCMI\_Init**(**void**);

**static** **void** **MX\_DFSDM1\_Init**(**void**);

**static** **void** **MX\_FMC\_Init**(**void**);

**static** **void** **MX\_I2C1\_Init**(**void**);

**static** **void** **MX\_I2C2\_Init**(**void**);

**static** **void** **MX\_LPUART1\_UART\_Init**(**void**);

**static** **void** **MX\_USART1\_UART\_Init**(**void**);

**static** **void** **MX\_USART2\_UART\_Init**(**void**);

**static** **void** **MX\_SAI1\_Init**(**void**);

**static** **void** **MX\_SDMMC1\_SD\_Init**(**void**);

**static** **void** **MX\_SPI2\_Init**(**void**);

Acestea sunt funcțiile de inițializare discutate anterior. Majoritatea dintre ele sunt, după cum a fost menționat, redundante, însă se generează automat datorită configurării implicite a plăcii de dezvoltare în partea de configurare STM32CubeMX. Funcțiile care sunt în interesul nostru sunt **SystemClock\_Config()**, **PeriphCommonClock\_Config()**, **MX\_ADC1\_Init()**, respectiv, **MX\_USART1\_UART\_Init()**. Implementarea acestor funcții este, de asemenea, generată automat de către IDE, motiv pentru care nu se va insista asupra codului intern ale acestora.

### Funcția main() & Familia HAL

Acum vom prezenta atracția principală a acestui proiect, respectiv, implementarea propriu-zisă a logicii aplicației în cadrul funcției **int** **main**(**void**). (Liniile 108-177)

Întâi, se poate face o paralelă dintre comportamentul funcției **main()** din STM32CubeIDE și comportamentul funcțiilor ***void setup()*** și ***void loop()*** din cadrul mediului de dezvoltare **Arduino IDE** dedicat plăcilor de dezvoltare **Arduino**. Pentru redactarea unui cod Arduino, funcțiile *setup()* și *loop()* sunt obligatoriu mereu prezente, și funcționalitatea lor este următoarea:

* Funcția *setup()*: Se rulează o singură dată în momentul lansării programului, scopul acesteia fiind să dicteze starea inițială a plăcii de dezvoltare Arduino.
* Funcția *loop()*: Este funcția care dictează logica efectivă a programului în timpul rulării. Este o funcție care este programată în așa fel încât să ruleze în buclă infinită, până la momentul deconectării plăcii Arduino.

STM32CubeIDE nu dispune de funcții setup() și loop() predefinite, însă, comportamentul acestora poate fi recreat cu ușurință în cadrul funcției main() astfel:

* Partea de *setup*: Primele linii de cod din main() pot fi folosite pe post de setup. Implicit, IDE-ul va apela toate inițializările predefinite descrise anterior. (În acest caz, setup-ul are loc la liniile 108-155)

**int** **main**(**void**)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* Configure the peripherals common clocks \*/

PeriphCommonClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_ADC1\_Init();

MX\_ADC2\_Init();

MX\_ADC3\_Init();

MX\_DCMI\_Init();

MX\_DFSDM1\_Init();

MX\_FMC\_Init();

MX\_I2C1\_Init();

MX\_I2C2\_Init();

MX\_LPUART1\_UART\_Init();

MX\_USART1\_UART\_Init();

MX\_USART2\_UART\_Init();

MX\_SAI1\_Init();

MX\_SDMMC1\_SD\_Init();

MX\_SPI2\_Init();

MX\_USB\_DEVICE\_Init();

/\* USER CODE BEGIN 2 \*/

**printf**("Initializing the thermometer.\n");

uint32\_t adcValue;

**float** temperature;

/\* USER CODE END 2 \*/

Întâi, se pot observa o mână de perechi de comentarii de forma **/\* USER CODE BEGIN *n* \*/**, respectiv, **/\* USER CODE END *n* \*/**. Acestea sunt comentarii predefinite ale IDE-ului care dictează locațiile unde poate fi scris cod customizat de către utilizator. Orice cod scris în afara acestor spații delimitate va fi ignorat în timpul compilării, și va fi șters în momentul reîncărcării proiectului. Acesta este un mecanism de delimitare a apelurilor implicite generate de către IDE de codul redactat de către utilizator, motiv pentru care **nu se recomandă ștergerea comentariilor automate ale IDE-ului în momentul scrierii de cod**.

După ce au fost făcute inițializările automate ale IDE-ului, există o porțiune unde se poate scrie cod personalizat pentru setup, respectiv, porțiunea delimitată de către marcajele comentate **/\* USER CODE BEGIN 2 \*/** și **/\* USER CODE END 2 \*/**. În această secțiune (liniile de cod 151-155) au fost implementate următoarele:

1. O linie de cod care afișează faptul că termometrul nostru a fost inițializat cu succes prin apelarea funcției **printf()**.
2. Declararea unei variabile de tip **unsigned int (uint32\_t)** în care vor păstra valorile de tensiune din ADC, și temperatura adevărată (de tip **float**).

* Partea de *loop*: În urma secțiunii precedente de cod se află o buclă infinită **while (1)** care recrează comportamentul funcției loop() din Arduino IDE. Aici are loc întreaga logică a aplicației noastre.

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

**while** (1)

{

HAL\_ADC\_Start(&hadc1);

**if** (HAL\_ADC\_PollForConversion(&hadc1, HAL\_MAX\_DELAY) == *HAL\_OK*) {

adcValue = HAL\_ADC\_GetValue(&hadc1);

temperature = (adcValue \* 3.3 / 4095) \* 100;

**printf**("Temperature: %.2f °C\r\n", temperature);

}

HAL\_ADC\_Stop(&hadc1);

HAL\_Delay(1000);

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

Întâi, observăm aceeași delimitare de cod discutată anterior, doar că, de data aceasta, codul se află între delimitatoarele **/\* USER CODE BEGIN WHILE \*/** și **/\* USER CODE END WHILE \*/**. Acum, să explicăm codul:

* Întâi, apelul “HAL\_ADC\_Start(&hadc1);” pornește convertorul ADC1, fapt indicat de către referința trimisă către **hadc1**.
* Apoi, se apelează “HAL\_ADC\_PollForConversion(&hadc1, HAL\_MAX\_DELAY)”, funcție al cărei scop este să aștepte ca valoarea din ADC1 să fie convertită, și îi este acordată limita maximă posibilă de timp HAL\_MAX\_DELAY. Dacă această conversie s-a făcut în timpul acordat, funcția va returna valoarea “*HAL\_OK*”, și codul poate rula în continuare.
* Se preia valoarea citită în ADC prin apelul “HAL\_ADC\_GetValue(&hadc1)”, și această valoare îi este asignată valorii adcValue definită anterior.
* Acum vine momentul mult așteptat al calculului temperaturii efective. Formula este următoarea: temperature = (adcValue \* 3.3 / 4095) \* 100 (Linia 166). Înmulțirea finală cu 100 a fost explicată anterior în secțiunea de descriere a senzorului de temperature LM35, respectiv, aceasta este necesară deoarece senzorul LM35 generează exact 10 mV/°C măsurat. Însă, înmulțirea cu **3.3 / 4095** pare puțin abstractă. Această formulă derivă din faptul că ADC-ul returnează o valoare rațiometrică, în sensul în care valoarea maximă a ADC-ului (în acest caz, este pe 12 biți, prin urmare, valoarea maximă este 212 – 1 = 4095) este asociată valorii maxime de tensiune (în acest caz, sursa de alimentare, adică 3.3V). Astfel, cum 0V corespunde valorii 0 pe ADC, și 3.3V corespunde valorii 4095 pe ADC, orice altă valoare intermediară se calculează cu regula de trei simplă. Astfel se deduce formula finală de calcul a temperaturii.
* Acum, valoarea temperaturii va fi afișată pe ecran la următoarea linie de cod, cu o precizie de două cifre după virgula zecimală, urmată de un caracter *newline* pentru a lăsa loc următoarei afișări de temperatură.
* Acum, se apelează funcția HAL\_Delay(1000), care întrerupe execuția firului current timp de o secundă, iar, în acest timp, pentru a minimiza consumul de energie, se oprește ADC-ul prin funcția HAL\_ADC\_Stop(&hadc1).
* După ce trece acea secundă, codul începe să se repete până în momentul deconectării plăcii. Astfel, a fost realizat un termometru de tip circuit integrat care afișează temperatura reală a camerei (cu o marjă de eroare de ±0.25°C) în fiecare secundă.

O ultimă mențiune în cadrul codului C aferent logicii proiectului trebuie să fie făcută pentru buna utilizare a funcției **printf()**. În limbajul C, funcția printf() servește pe post de funcție de abstracție pentru afișarea în terminal a informațiilor dorite, pentru ușurința în utilizare. Una dintre funcțiile apelate de către printf() este funcția **int** **\_write**(), care este bine implementată în limbajul canonic C, însă, în STM32CubeIDE, aceasta este implementată ca și funcție **weak**, ceea ce permite suprascrierea ei în orice mod dorește utilizatorul. În cazul nostru, dorim ca printf() să afișeze datele pe ecranul terminal, așadar, trebuie să calibrăm **\_write()** în așa fel încât să comunice cu terminalul prin UART. Implementarea noastră este următoarea (Liniile 98-101):

**int** **\_write**(**int** file, **char** \*data, **int** len) {

HAL\_UART\_Transmit(&huart1, (uint8\_t\*)data, len, HAL\_MAX\_DELAY);

**return** len;

}

Această implementare a funcției **\_write()** apelează funcția **HAL\_UART\_Transmit(),** funcție care este responsabilă de trimiterea datelor prin protocolul USART1 (huart1) la ecranul terminal. Astfel, funcția **printf()** va apela în spate funcția **HAL\_UART\_Transmit()**, și datele vor fi transmise corespunzător către HTerm, program care urmează a fi descris acum.

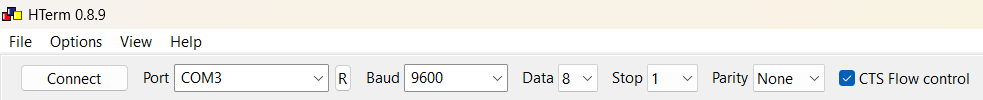
## Ecranul Terminal HTerm

**HTerm** este un program de ecran terminal gratuit utilizat pentru comunicarea serială dintre unitatea centrală (în acest caz, laptop-ul), și un alt modul de comunicare serială precum un dispozitiv UART (în acest caz, modulul PMod USB-UART).

După cum a fost specificat în partea de configurare hardware, se va lega un cablu USB pentru transferul de date dintre modulul PMod și laptop, iar acum, cu codul C scris și cu componentele hardware conectate corespunzător, se va deschide programul HTerm. Acesta trebuie configurat în așa fel încât parametri de comunicare serială să fie în concordanță cu cei din STM32CubeMX, respectiv, în acest caz, aceștia sunt următorii:

* **Baud Rate** de **9600 de biți pe secundă**
* **Lungimea unui cuvânt** de **8 biți** (inclusiv biții de paritate)
* **Niciun bit de paritate**
* **1 bit de stop**

Într-un final, se va alege portul de comunicare serială correspondent mufei în care este conectat cablul USB (spre exemplu, **COM3**). Mai mult, se va bifa opțiunea “**CTS Flow Control**” pentru a dicta terminalului faptul că fluxul de date va fi controlat de către semnalul ***Clear to Send (CTS)***. Astfel, totul este pregătit, și se poate apăsa pe butonul “**Connect**”, după care se va rula codul din STM32CubeIDE. Dacă fiecare pas a fost executat corect, ar trebui să se afișeze textul “Initializing the thermometer.” pe terminal, urmat de un newline, după care, din secundă în secundă ar trebui să se afișeze temperatura curentă a mediului.



**Figura 7. Configurarea Terminalului HTerm pentru Comunicarea Corespondentă cu Placa Discovery**

# Mod de Utilizare (Sumar al Părții III.)

Pentru partea de hardware, se vor configura conform instrucțiunilor prezentate în partea III. Pentru partea de software, se va deschide proiectul din STM32CubeIDE aferent acestei aplicații, precum și terminalul HTerm, care va fi configurat conform figurii 7 de pe pagina anterioară. Acum, tot ce rămâne de făcut este să selectăm opțiunea “**Connect**” din HTerm, și opțiunea “**Run <nume proiect>**” din IDE, iar proiectul ar trebui să funcționeze corespunzător, i.e., să se afișeze mesajul “**Initializing the thermometer.**” în HTerm, urmat de caracterul *newline*, și, apoi, din secundă în secundă, ar trebui să se afișeze mesajul “**Temperature: <temperatura curentă>**” urmat de câte un caracter *newline* până la momentul deconectării a uneia dintre componentele hardware, moment în care comunicarea serială va eșua, și rularea programului se va încheia.

# Rezultate Experimentale

În urma conectării componentelor hardware și a rulării programului din STM32CubeIDE, se pot observa rezultate experimentale prin manipularea datelor citite de la senzorul LM35 prin modificarea manuală a temperaturii în momentul în care ținem în mână senzorul și îl încălzim, sau, dacă îl scoatem pe pervaz și îl răcim. Spre exemplu, la rulare, terminalul poate arăta astfel:

Initializing the thermometer.

Temperature: 25.4 °C

Temperature: 25.8 °C

Temperature: 27.1 °C

Temperature: 28.0 °C

# Concluzii

Acest proiect a fost o experiență deosebit de instructivă de dezvoltare a unei aplicații practice bazate pe diferitele noțiuni învățate în cadrul disciplinei *Structura Sistemelor de Calcul*. Au fost învățate, aprofundate, și aplicate, noțiuni precum:

* Arhitectura ST & Familia de Microcontrollere STM32
* Mediul de dezvoltare STM32CubeIDE și redactarea de cod C pentru plăci STM32
* Protocolul de comunicare UART (mai concret, USART)
* Comunicarea serială între mai multe module hardware & Utilizarea terminalului HTerm
* Rata de transmisie a datelor (Baud Rate)
* Convertorul analog-numeric (ADC) și conversia din tensiune în temperatură
* Interconectarea corespunzătoare a componentelor hardware utilizând un breadboard

# Bibliografie

* <https://www.st.com/resource/en/user_manual/um2160-discovery-kit-with-stm32l496ag-mcu-stmicroelectronics.pdf> — Manual de Referință pentru placa de dezvoltare STM32L496G-DISCOVERY
* <https://www.ti.com/lit/ds/symlink/lm35.pdf> — Manual de Referință pentru senzorul de temperatură LM35
* <https://en.wikipedia.org/wiki/STM32> — Pagina Wikipedia dedicată familiei de microcontrollere STM32
* <https://digilent.com/reference/_media/pmod:pmod:pmodusbuart_rm.pdf> — Manual de Referință pentru modulul PMod USB-UART
* <https://www.st.com/en/evaluation-tools/32l496gdiscovery.html> — Pagina de prezentare a plăcii de dezvoltare STM32L496G-DISCOVERY
* <https://mihai.utcluj.ro/wp-content/uploads/dmp/labs/pmp-lab07.pdf> — Prezentare succintă a intrărilor analogice & convertorului ADC