

UNIVERSITATEA TEHNICĂ „Gheorghe Asachi” din IAȘI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DOMENIUL: Calculatoare și Tehnologia Informației
SPECIALIZAREA: Calculatoare Încorporate

INPUT 1: DHT22

Student:

Grosu Gabriela
Andrei Chirica
Blănaru Radu

Iași, 2026

Cuprins

1	Introducere	2
1.1	Descriere	2
1.1.1	Arhitectura Sistemului și Distribuția Echipelor	2
1.1.2	Schema Logică a Transmisiei	2
1.2	Placa de dezvoltare	2
1.3	Arhitectură Microcontroler (MCU)	3
1.4	Caracteristici Hardware și Periferice	3
1.5	Ecosistem de Dezvoltare	3
1.6	Implementare Tehnică	3
2	Implementarea Controlului pentru DHT22 și FlexCAN	5
2.1	Prezentarea generală	5
2.2	Arhitectura hardware	5
2.3	Fluxul de date DHT22	5
2.3.1	Nivel fizic: Protocolul DHT22	5
2.3.2	Algoritmul de citire (Implementare Asincronă)	6
2.3.3	Interpretarea datelor	6
2.3.3.0.1	Procesare:	7
2.4	Transmisia Datelor pe CAN Bus	7
2.4.1	Arhitectura Software CAN	7
2.4.2	Mașina de Stări a Aplicației	7
2.4.3	Formatul Pachetelor CAN	7
2.4.3.0.1	Detalii despre codificare:	8

Capitolul 1. Introducere

1.1. Descriere

Acest proiect implementează un sistem de control distribuit utilizând protocolul de comunicație industrială **CAN (Controller Area Network)**. Obiectivul principal este transferul de date de la nodurile de intrare (senzori) către nodurile de execuție (actuatori), sub supravegherea unui nod dedicat de monitorizare. Întregul sistem este construit pe platforma **NXP FRDM-MCXN947**.

1.1.1. Arhitectura Sistemului și Distribuția Echipelor

Sistemul este compus din 5 noduri, fiecare fiind gestionat de o echipă specifică:

1. **Potențiometru (Input):** achiziția analogică (ADC) și transmiterea poziției către rețea pentru controlul unghiular sau de intensitate.
2. **Senzor de Temperatura/Umiditate (Input):** monitorizarea condițiilor de mediu și partajarea acestor parametri pe magistrală.
3. **Monitor:** interceptarea pachetelor. Acesta nu modifică datele, ci analizează traficul pentru validarea comunicării între celelalte noduri.
4. **Servomotor (Output):** Interpretează datele primite (de la potențiometru și senzor) și generează semnalul PWM necesar poziționării mecanice.
5. **LED RGB (Output):** Oferă feedback vizual pe baza datelor de la senzori.

1.1.2. Schema Logică a Transmisiei

Figura 1.1 ilustrează modul în care datele circulă de la cele două surse de intrare către restul componentelor sistemului.

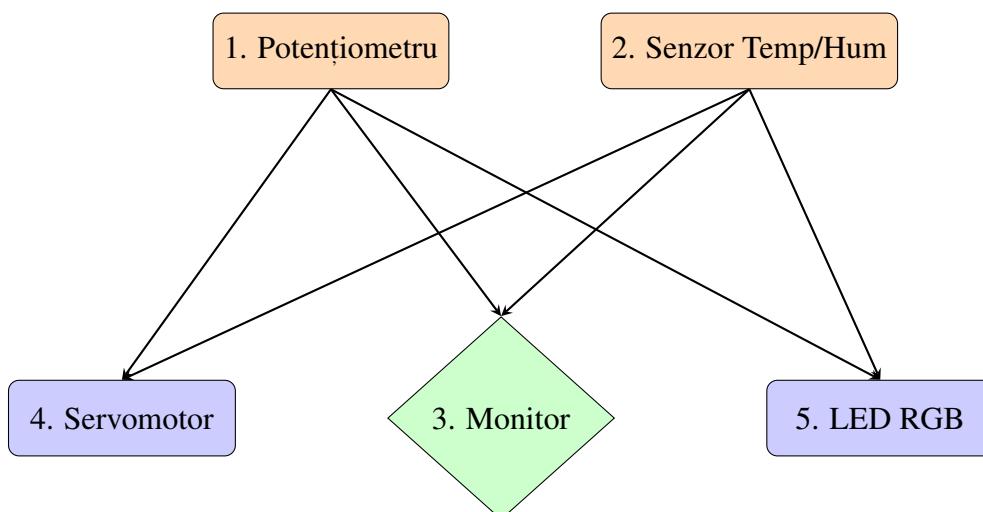


Figura 1.1. Schema de distribuție

1.2. Placa de dezvoltare

Placa de dezvoltare **FRDM-MCXN947** este o platformă compactă și scalabilă, concepută pentru evaluarea microcontrolerelor din seria **MCX N94x**. Aceasta este ideală pentru aplicații de tip Edge Computing, automatizări industriale și Internet of Things (IoT).

1.3. Arhitectură Microcontroler (MCU)

Nucleul plăcii este reprezentat de MCU-ul **MCX N947**, care dispune de următoarele caracteristici:

- **Dual-Core:** 2x Arm® Cortex®-M33 rulând la frecvențe de până la 150 MHz.
- **Accelerator AI/ML:** Include unitatea de procesare neurală **eIQ® Neutron NPU** pentru sarcini de inteligență artificială la nivel hardware.
- **Securitate:** Subsistem de securitate **EdgeLock® Secure Enclave** (Core Profile).
- **Memorie:** Până la 2 MB de memorie Flash și 512 KB de memorie SRAM cu paritate/ECC.

1.4. Caracteristici Hardware și Periferice

Placa FRDM-MCXN947 integrează o gamă largă de componente pentru prototipare rapidă:

- **Conecțivitate Ethernet:** Port RJ45 integrat pentru aplicații de rețea.
- **Interfețe USB:** Port USB Type-C High-Speed pentru date și alimentare.
- **Expansiune:** Compatibilitate cu shield-uri **Arduino Uno R3** și conectori **MikroBus™** pentru senzori și actuatori.
- **Debug Integrat:** On-board **MCU-Link** debugger bazat pe CMSIS-DAP, eliminând necesitatea unui programator extern.
- **Interfață Audio:** Suport pentru ieșiri audio de tip MQS (Medium Quality Sound).

1.5. Ecosistem de Dezvoltare

Suportul software este asigurat prin **MCUXpresso Developer Experience**, oferind:

1. **IDE:** MCUXpresso IDE, VS Code (cu extensii NXP) sau IAR/Keil.
2. **SDK:** Drivere periferice optimizate, stive de protocoale și exemple de cod (RTC, PWM, Ethernet).
3. **Configurare:** Instrumente grafice pentru rutarea pinilor și configurarea ceasurilor (Pins, Clocks, Peripherals Tool).

1.6. Implementare Tehnică

Pentru realizarea comunicării, s-au utilizat următoarele resurse hardware ale microcontrolerului MCX N947:

- **FlexCAN:** Configurarea baud-rate-ului și a filtrelor de acceptare (Hardware Acceptance Filters).
- **GPIO & PWM:** Pentru controlul direct al servomotorului și al LED-ului.
- **ADC:** Pentru conversia semnalului analogic de la potențiometru.

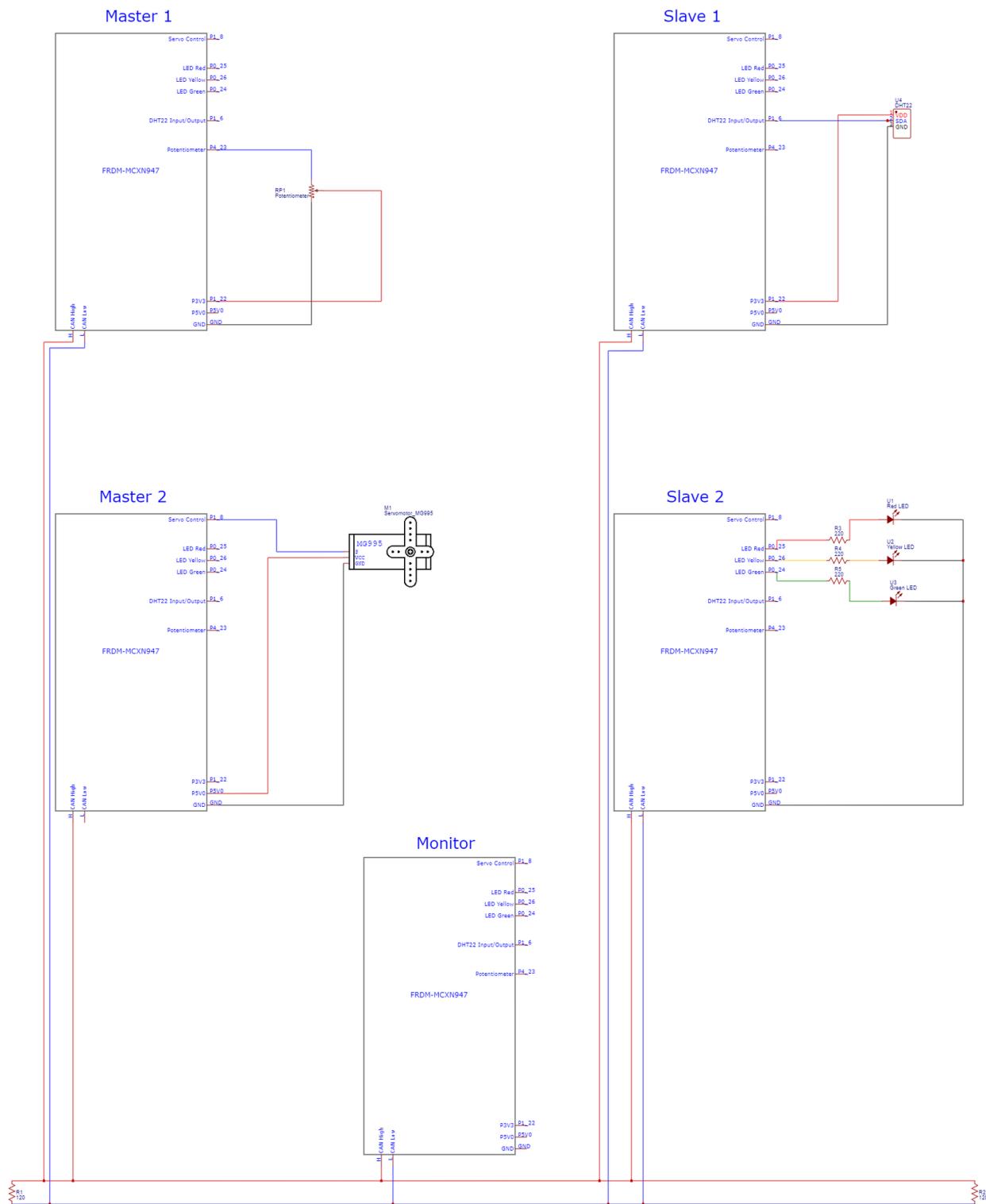


Figura 1.2. Schema de conectare generală

Capitolul 2. Implementarea Controlului pentru DHT22 și FlexCAN

2.1. Prezentarea generală

Acest document descrie implementarea software pentru achiziția de date de la un senzor de umiditate și temperatură **DHT22** și integrarea acestuia într-un sistem bazat pe placa de dezvoltare **NXP FRDM-MCXN947**. Soluția utilizează un mecanism asincron bazat pe întreruperi pentru citirea senzorului și o mașină de stări pentru transmisia datelor via **CAN-FD**.

2.2. Arhitectura hardware

Sistemul de input este format din placă FRDM-MCXN947 (master) și senzorul DHT22. Comunicația CAN se realizează prin transceiver-ul integrat, conectat la controller-ul FlexCAN al microcontroller-ului.

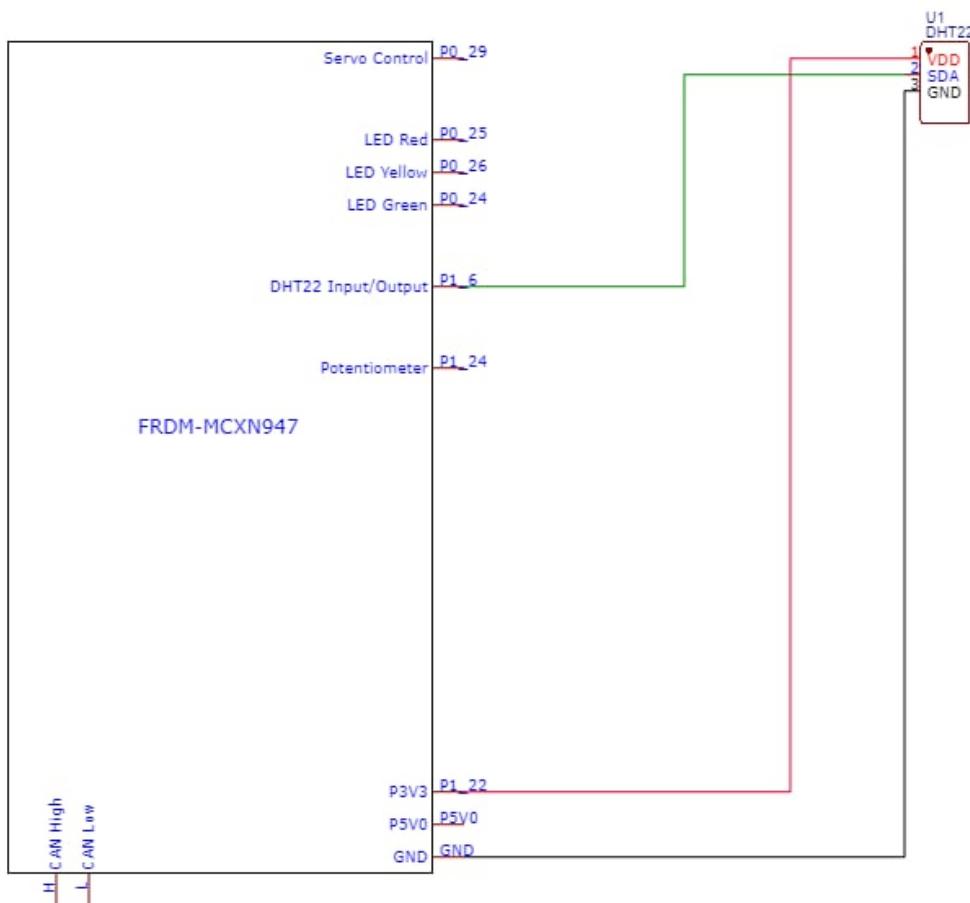


Figura 2.1. Schema de conectare FRDM-MCXN947 și DHT22

2.3. Fluxul de date DHT22

2.3.1. Nivel fizic: Protocolul DHT22

Senzorul comunică printr-un singur fir (One-Wire), folosind lungimea impulsurilor pentru a codifica biți. Protocolul este sensibil la timp și necesită o rezoluție de microsecunde.

Secvența de citire:

- **Start Signal:** Microcontroller-ul configurează pinul ca OUTPUT și îl menține în starea LOW timp de 18ms pentru a "trezi" senzorul.
- **Handshake:** Pinul trece în modul INPUT. Senzorul răspunde cu un semnal LOW (80μs) urmat de HIGH (80μs).
- **Transmisia Bițiilor (40 biți):** Fiecare bit începe cu un nivel LOW de 50μs. Diferența dintre '0' și '1' este dată de durata nivelului HIGH:
 - **Bit '0':** Puls HIGH de ~26-28 μs.
 - **Bit '1':** Puls HIGH de ~70 μs.

2.3.2. Algoritmul de citire (Implementare Asincronă)

Spre deosebire de metodele clasice de tip "bit-banging" blocant, implementarea actuală utilizează perifericele hardware (**SCTimer** și **GPIO Interrupts**) pentru a nu bloca procesorul în timpul achiziției.

Procesul este gestionat de o mașină de stări internă senzorului (g_dhtState):

1. Inițiere (SCTimer): Funcția Start_DHT_Read configurează **SCTimer (SCT0)** pentru a genera pulsul de start de 18ms. La expirarea timpului, întreruperea SCTimer comută pinul pe INPUT și activează întreruperile GPIO pe front descrescător (Falling Edge).

2. Captura Datelor (GPIO ISR): Rutina de tratare a întreruperii (DHT_IRQ_HANDLER) este apelată la fiecare front descrescător.

- Se citește valoarea curentă a contorului SCTimer (SCT0->COUNT).
- Se calculează lățimea pulsului: width = current_time – last_edge_time.
- Pe baza lățimii pulsului se decide valoarea bitului:

$$\text{width} > 100\mu\text{s} \implies \text{Bit '1'}$$

$$\text{width} \leq 100\mu\text{s} \implies \text{Bit '0'}$$

(Pragul de 100μs este ales pentru a diferenția suma timpilor Low+High pentru bitul 1 vs bitul 0).

3. Timeout Software: Funcția DHT_HandleTimeout este apelată periodic în bucla principală pentru a detecta blocajele (de exemplu, senzor deconectat) și a reseta mașina de stări dacă citirea durează mai mult de 100ms.

2.3.3. Interpretarea datelor

După capturarea celor 40 de biți, datele sunt stocate în vectorul g_dhtBits [5] și procesate în bucla principală (main).

Tabelul 2.1. Structura datelor DHT22

Octet	Conținut	Descriere
Byte 0	RH_Integral	Partea întreagă a umidității.
Byte 1	RH_Decimal	Partea zecimală a umidității.
Byte 2	T_Integral	Partea întreagă a temperaturii (Bitul 7 = semn).
Byte 3	T_Decimal	Partea zecimală a temperaturii.
Byte 4	Checksum	Suma de control.

2.3.3.0.1 Procesare:

- **Validare CRC:** Checksum == (Byte0 + Byte1 + Byte2 + Byte3)&0xFF.
- **Temperaturi Negative:** Se verifică bitul cel mai semnificativ al Byte 2 (0x80). Dacă este setat, temperatura se consideră negativă.
- **Conversie:** Valorile sunt convertite în format float pentru afişare şi în format întreg/byte pentru transmisia CAN.

2.4. Transmisia Datelor pe CAN Bus

Sistemul implementează o mașină de stări pentru gestionarea comunicării pe magistrala CAN, utilizând standardul **CAN-FD** (Flexible Data-rate).

2.4.1. Arhitectura Software CAN

- **FIFO Buffer:** Deoarece citirea senzorului și transmisia CAN sunt proceze asincrone, se utilizează un buffer circular (FIFO_Push / FIFO_Pop) pentru a stoca mesajele ce urmează să fie trimise.
- **Transmisie Non-Blocking:** Se folosește funcția FLEXCAN_TransferFDSendNonBlocking pentru a nu bloca execuția CPU în timpul arbitrajului pe magistrală.

2.4.2. Mașina de Stări a Aplicației

Logica principală este divizată în trei stări, controlate de timere software (CTIMER):

1. **STATE_LISTEN (1000ms):** Dispozitivul ascultă magistrala pentru mesaje de intrare sau așteaptă expirarea timer-ului pentru a iniția transmisia propriilor date.

2. STATE_SEND:

- Verifică dacă există date în coada FIFO (Umiditate sau Temperatură).
 - Extragă datele și le trimit folosind ID-urile configurate (TX_MSG_ID_RECEIVER1 pentru umiditate, TX_MSG_ID_RECEIVER2 pentru temperatură).
 - Dacă transmisia eșuează sau buffer-ul este ocupat, mesajul este reîncercat sau aruncat.
3. **STATE_WAIT (500ms):** O stare de așteptare introdusă după receptia sau transmisia unui mesaj pentru a evita congestia magistralei (rate limiting).

2.4.3. Formatul Pachetelor CAN

Conform funcției SendCanMessageNonBlocking, datele sunt împachetate într-un frame **CAN-FD Standard** cu o lungime a datelor (DLC) de **2 octeți**.

Structura payload-ului este organizată astfel:

Tabelul 2.2. Structura Payload-ului CAN (2 Bytes)

Byte Index	Conținut	Descriere
0	Date Senzor	Valoarea întreagă a umidății sau temperaturii (uint8_t).
1	ID Sursă	Identifierul nodului emițător (definit ca SENDER1).

2.4.3.0.1 Detalii despre codificare:

- **Tipul Datelor:** Valorile float citite de la senzor sunt convertite prin trunchiere la uint8_t înainte de transmisie (ex: 24.5°C devine 24).
- **Identificare Tip Date:** Distincția dintre umiditate și temperatură se face pe baza ID-ului mesajului CAN (Arbitration ID), nu a payload-ului:
 - TX_MSG_ID_RECEIVER1: Conține date despre **Umiditate**.
 - TX_MSG_ID_RECEIVER2: Conține date despre **Temperatură**.
- **Endianness:** Microcontroller-ul plasează octetul de date pe poziția cea mai puțin semnificativă (LSB - Byte 0), iar ID-ul sursei pe următorul octet (Byte 1), conform operației: dataByte | (SENDER1 « 8).