FACULTATEA DE INGINERIE ELECTRICĂ ŞI
ŞTIINŢA CALCULATOARELOR
Calculatoare și tehnologia informației

ing. **Cornel VENTUNEAC**

TEZA de DOCTORAT

--- rezumat ---

Contribuții privind implementarea unui gateway IoT utilizând extensia ModbusE a protocolului Modbus

---

Conducător științific:

Prof. univ. dr. ing. **Vasile-Gheorghiță GĂITAN**

Suceava, 2024

Această lucrare a beneficiat de suport financiar prin proiectul *“ Excelența academică și valori antreprenoriale - sistem de burse pentru asigurarea oportunităților de formare și dezvoltare a competențelor antreprenoriale ale doctoranzilor și postdoctoranzilor (ANTREPRENORDOC)”,* cofinanţat din Fondul Social European prin Programul Operaţional Capital Uman, 2014-2020, Contract nr. 36355/23.05.2019 POCU/380/6/13 - Cod SMIS: 123847.”

Doresc să adresez mulțumirile mele domnului Vasile Gheorghiţă GĂITAN pentru îndrumarea și susținerea oferită de-a lungul întregii perioade de formare prin studii doctorale. Pentru observațiile făcute și sfaturile date asupra tezei de doctorat, mai doresc să mulțumesc domnilor: Prof. univ. dr. ing Cornel TURCU, Conf. univ. dr. ing. Ioan UNGUREAN și Ș. l. dr. ing. Ionel ZAGAN.

Cuprins

1 Obiectivele și structura lucrării

În ziua de azi în care tehnologia este în continuă evoluție, inovația este motorul progresului industrial. Transformările digitale, de-a lungul timpului, au remodelat modul în care interacționăm cu mediul înconjurător, prin redefinirea standardelor și modificarea paradigmelor. Industria 4.0 și Internet of Things (**IoT**) sunt două concepte interconectate, dar distincte, care aduc contribuții semnificative la transformarea digitală a societății și a economiei. Industria 4.0 reprezintă o ramură specifică a transformării digitale care se concentrează pe industrie și producție. Internetul Lucrurilor (IoT) este o tehnologie care se referă la interconectarea dispozitivelor și obiectelor la rețeaua de internet, asigurându-le în acest fel să comunice atât între ele și cât și cu alte sisteme și servicii. Aceste dispozitive pot fi diverse, de la electrocasnice inteligente și senzori la autovehicule conectate și echipamente industriale. De fapt Internetul lucrurilor [1] este baza industriei 4.0 [2],[3]. Conceptul de Industrial Internet of Things (**IIoT**) a captat atenția lumii industriale și a marcat o nouă era a automatizării și conectivității în producție. IIoT nu este doar o simplă extensie a Internet of Things în mediul industrial ci reprezintă un catalizator pentru transformări în mai multe domenii (energetic, agricol, manufacturii, etc). Prin conectarea senzorilor, dispozitivelor, echipamentelor și proceselor la internet și la sistemele de comunicații, IIoT deschide drumul către o eficiență sporită și o vizibilitate îmbunătățită în operațiunile industriale. IIoT reprezintă o paradigmă tehnologică care utilizează conectivitatea și capacitatea de calcul a dispozitivelor IoT pentru a transforma industria prin optimizarea proceselor, îmbunătățirea calității produselor și serviciilor, și creșterea productivității. Acesta permite colectarea și analiza datelor în timp real, luând decizii în baza acestor informații pentru a maximiza performanța și rentabilitatea operațiunilor industriale. O altă definiție ar fi că IIoT prin integrarea tehnologiilor IoT în procesele de fabricație și producție facilitează automatizarea și optimizarea operațiunilor industriale. Acest lucru presupune folosirea tuturor dispozitivelor care sunt conectate în scopul de a monitoriza și controla echipamentele și procesele în timp real, în vederea îmbunătățirii performanței și a eficienței. Gateway-urile IIoT (Industrial Internet of Things) sunt esențiale în cadrul proceselor industriale moderne, ele asigurând legătura diverselor dispozitive (ce folosesc protocoale de comunicație Modbus) , senzori și elemente de execuție cu internet-ul pentru a facilita monitorizarea și controlul proceselor industriale.

1.1 Obiective

Obiectivele propuse pentru teză sunt :

- Definirea arhitecturii IIoT și implementarea acesteia pe o platformă cu mai multe nuclee de prelucrare care să permită dezvoltarea de aplicații specifice protocolului Modbus cu extensia ModbusE și server/client OPC UA.
- Îmbunătățirea fluxului de date la nivelul slot-urilor și a ciclului de achiziție, utilizând arhitecturi cu mai multe nuclee, unul care se va ocupa de ModbusE și unul de putere de calcul mare care se va ocupa de implementarea server-ului OPC UA.
- Portarea în condiții de performanță a serverului OPC UA (serverul OPC UA pe procesorul Arm Cortex A8 ce rulează pe sistemul de operare Debian).
- Integrarea modulelor prezentate în obiectivelor anterioare (ciclu de achiziție (CA), server OPC UA) într-un gateway IIoT utilizând extensia ModbusE a protocolului Modbus.

- Atașarea de contoare software pentru sloturi. Astfel este necesar un contor pentru acele mesaje care au fost recepționate corect, un contor pentru erorile depășirii duratei de 3,5 caractere specific Modbus, un contor necesar erorilor CRC și un contor pentru durata slotului (time-out). Astfel: dacă au apărut erori de depășire a timpului de 3,5 caractere între două mesaje consecutive Modbus, este de preferat înlocuirea stației fiindcă nu este conformă specificației Modbus; dacă există erori semnificative care se referă la durata slotului (time-out), atunci durata estimată pentru execuția comenzilor este greșită și este necesară mărirea duratei slot-ului. Această mărire a duratei slot-ului se face până când erorile pentru durata slotului sunt acceptabile; dacă erorile CRC sunt multe și nu sunt însoțite de alte erori, precum cele de 3,5 caractere sau cele de durata slotului, atunci ori rețeaua este afectată de zgomote ori rețeaua are defecte fizice sau există defecte fizice la nivelul dispozitivului server (slave).

1.2 Structura lucrării

Teza de doctorat este structurată pe 8 capitole. Capitolul 2 realizează o abordare de ansamblu a paradigmei industriei 4.0 (Industry 4.0), a arhitecturilor IoT, a protocoalelor de comunicație folosite de IoT-uri și a arhitecturilor cloud, edge și fog. De asemenea se face referire la rețelele industriale locale (RIL) și la rolul lor în comunicația dintre dispozitivele industriale precum și la protocoalele de comunicație folosite de aceste dispozitivele industriale pentru a putea comunica. Tot în capitolul 2 se face referire și la modelele OSI și TCP/IP precum și la diferențele dintre ele.

Capitolul 3 prezintă protocolul de comunicație Modbus, acest protocol fiind unul dintre cele mai simple și folosite protocoale de comunicație din industrie. Se prezintă doar varianta pe linia serială. De asemenea acest capitol prezintă și extensia ModbusE a protocolului Modbus. Implementarea gateway-ului IIoT din teza de doctorat urmărește prin folosirea unei arhitecturi multiprocesor să îmbunătățească fluxul de date atât la nivelul sloturilor cât și la nivelul ciclului de achiziție (CA) pentru extensia ModBusE.

Capitolul 4 prezintă arhitectura hardware a gateway-ului IIoT. Se prezintă o descriere a platformei de dezvoltare BeagleBone Black, care folosește procesorul Sitara AM335x de la Texas Instruments. Procesorul Sitara AM335x este constituit dintr-un procesor ARM Cortex A8 RISC pe 32 de biți care operează la o frecvență de până la 1 GHz și două procesoare PRU pe 32 de biți pentru operații în timp real care funcționează la o frecvență de 200 MHz.

Capitolul 5 prezintă arhitectura software a gateway-ului IIoT. Se prezintă aplicațiile cu firele de execuție care alcătuiesc partea software a gateway-ului IIoT. Astfel este prezentată aplicația PRU_RPMsg_ModBusE_Master ce reprezintă firul de execuție pentru ciclul de achiziție (CA); aplicația dispatcher BBB_ARM_A8_MBE_IOT_GATEWAY care este alcătuită din firele de execuție ModBusE_Listening_Thread, ModBusE_Eth_Servers_Thread, ModBusEThreadDispatcherForServer. Aceste fire de execuție au rolul de a stabili conexiuni TCP/IP prin intermediul socket-urilor cu clienți ModbusE (clienți OPC UA sau alți clienți ce folosesc ModbusE), preiau comenzile venite după care le trimit mai departe către ciclul de achiziție, preiau răspunsul de la ciclul de achiție și trimit mai departe răspunsul către clienții care au solicitat răspunsul; aplicațiile PRU1_RPMsg_ModBusE_Master și ListenServer folosite pentru a putea verifica anumite variabile din ciclul de achiziție; aplicația

ServerOPCUAModBusE care reprezintă serverul OPC UA (server ce preia comenzi ModbusE de la un client OPC UA ce rulează pe un PC); aplicația MBE_IOT_GATEWAY_BBB_CLIENT și clientul UaExpert OPC UA ce reprezintă clienții pentru dispatcher-ul BBB_ARM_A8_MBE_IOT_GATEWAY.

Capitolul 6 prezintă modelarea canalului de comunicație a ciclului de achiziție, fluxul de mesaje de comunicație între ciclul de achiziție (pe PRU0), dispathcer (pe ARM Cortex A8 cu Linux), server OPC UA (pe ARM Cortex A8 cu Linux), client OPC UA (pe Windows), și de asemenea analiza performanțelor comunicațiilor de date gateway IIoT, client OPC UA. Se prezintă timpii măsurați cu osciloscopul între anumite fire de execuție de când serverul OPC UA care rulează pe procesorul Cortex A8 primește comanda de la clientul OPC UA care rulează pe un calculator și până când serverul OPC UA primește răspuns la comandă.

Capitolul 7 prezintă contribuțiile aduse pentru această lucrare, propunerile pentru cercetările viitoare precum și diseminarea rezultatelor în reviste de specialitate

Capitolul 8 reprezintă bibliografia.

2 Introducere.

Automatizările din domeniul industriei reprezintă un domeniu care este în plină dezvoltare și se bazează pe progresele tehnologice. Aceste progrese făcute în domeniul tehnologiei de producție mai poartă și denumirea de revoluție industrială. Companiile care vor să fie relevante în domeniul economic actual trebuie să adapteze întreg ciclu de producție la era digitală a tehnologiei industriale.

2.1 Industria 4.0

Revoluția industrială sau mai bine zis industria 4.0 (Industry 4.0) are un impact puternic asupra omenirii zilelor noastre. Astfel această revoluție industrială se referă la dispozitivele conectate la internet, la prezența tot mai mare a senzorilor, la răspândirea tot mai largă a comunicațiilor fără conexiune prin fir, la dezvoltarea și utilizarea pe scară largă a dispozitivelor și a echipamentelor inteligente, de asemenea și la controlul și la analiza în timp real a dispozitivelor și sistemelor automatizate..

O clasificare a revoluțiilor industriale ar putea fi :

- Prima revoluție industrială a fost reprezentată de trecerea de la producția manuală la mașinile care funcționau cu aburi.
- A doua revoluție industrială sau revoluția tehnologică caracterizată de instalațiile telegrafice, rețelele feroviare și liniile de producție moderne prin utlizarea electrificării.
- A treia revoluție industrială sau revoluția digitală caracterizată de apariția calculatoarelor
- A patra revoluție industrială sau strategia germană „Industrie 4.0”. „Industrie 4.0” a apărut ca urmare a unui proiect cerut de guvernul german pentru computerizarea producțiilor.

Industria 4.0 a fost menționată pentru prima data de o echipă de oameni de știință care dezvoltau strategii în tehnologie pentru guvernul german și a evoluat de la o idee de a face ca industria germană să fie mai competitivă la un termen ce a devenit foarte răspândit în toată lumea. Guvernul german a introdus termenul industrie 4.0 pentru a defini a patra revoluție industrială [1],[4]. Se poate susține că industria 4.0 a reprezentat o nevoie pentru modernizarea și îmbunătățirea proceselor de producție și reprezintă automatizarea continuă a metodelor tradiționale industriale de fabricație folosind noile tehnologii. Industria 4.0 realizează trecerea de la un sistem de control industrial central la unul în care produsele inteligente definesc pașii de producție și reprezintă o eră nouă a producției industriale [5]. Industria 4.0 accepta OPC UA ca standard înglobat. Industria 4.0 constă din patru principii organizatorice de bază:

- Interconectare
- Transparența informațiilor
- Asistență tehnică
- Decizii descentralizate

Industria 4.0 se referă la nevoia tot mai mare de înglobare a tehnologiilor digitale și a comunicației prin intermediul internetului în industrie. Automatizarea și schimbul de date include: internetul lucrurilor (IoT), internetul industrial al lucrurilor (IIoT), cloud computing, calculul cognitiv, inteligența artificială și crearea de fabrici smart. Avantajele industriei 4.0 pentru companiile din producție:

- **timp**: lucrând în procese optimizate angajații devin mai eficienți deci se reduce timpul de livrare a comenzilor.
- **cost**: prezentarea precisă a datelor duce la o planificare mult mai bună a costurilor și veniturilor.
- **flexibilitate**: datorită sistemelor flexibile schimbările se pot face cu ușurință.
- **integrare**: timpul datorat întreruperilor producției scade cu aproximativ 80% ca urmare a integrării sistemelor cu resursele din fabrică.

Tranziția de la fabrica tradițională la fabrica viitorului de tip smart presupune:

- **digitalizare** - accesul la date în timp real prin solulția software potrivită.
- **mobilitate** – se caracterizează prin conectarea operatorilor cu sistemele informatice din fabrică prin utilizarea calculatoarelor, tabletelor și a telefoanelor inteligente.
- **automatizare** - presupune automatizarea proceselor de producție.
- **informatizarea fabricii** – presupune achiziții de sisteme informatice cu ajutorul cărora să se integreze resursele din fabrică și procesele de execuție.

Fabrica smart digitală optimizează toate fazele unui ciclul de viață al produselor. Reprezintă un spațiu cu condiții de lucru sigure în care oamenii utilizează roboții nu doar pentru sarcini simple ci și pentru etape mai complexe de producție, în care se folosește o abordare predictivă bazată pe inteligența artificială. În fabrica inteligentă, ființele umane, mașinile și resursele comunică între ele la fel de natural ca într-o rețea socială [6]. Figura 2-1 prezintă caracteristicile industriei 4.0.

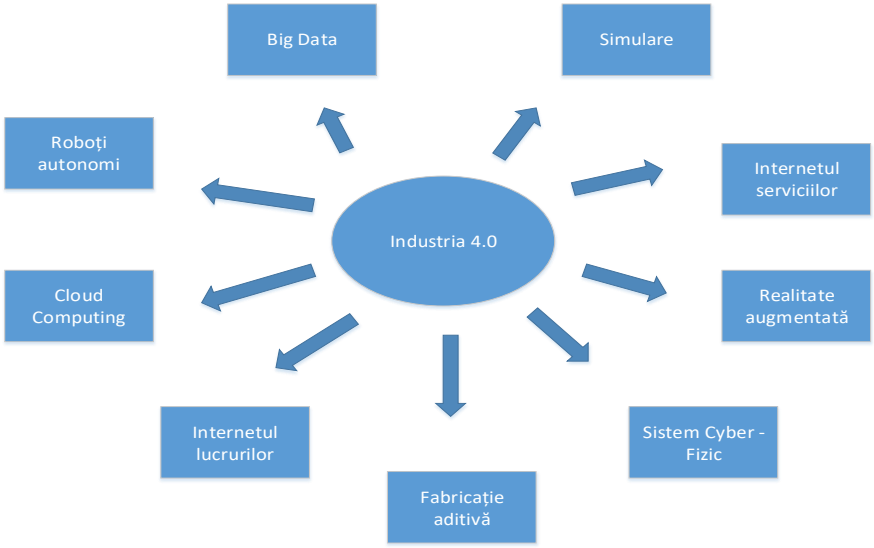


Figura 2-1 Caracteristici ale industriei 4.0 [7]

2.1.1 Internetul lucrurilor (IoT – Internet of Things)

Internetul lucrurilor (sau obiectelor) reprezintă conexiunile dintre obiectele (lucrurile) fizice cum ar fi senzorii, echipamentele industriale și internetul. O definiție a IoT este dată de [8] și anume: *„Internetul lucrurilor (IoT) a fost definit ca o infrastructură globală pentru societatea informațională, care permite servicii avansate prin interconectarea lucrurilor (fizice și virtuale) pe baza informațiilor interoperabile existente și în evoluție și tehnologii de comunicație"*. Internetul industrial al lucrurilor IIoT este similar cu IoT, numai că IIoT

reprezintă internetul lucrurilor în mediul industrial și se referă la conexiunile dintre oameni, date și mașini, în cadrul proceselor industriale. Fabricile smart valorifică puterea combinată dintre tehnologia informației, comunicarea și procesele de producție [9]. A fost lansată versiunea 1.9 a arhitecturii de referință pentru IIoT de către cei de la Industrial Internet Consortium (IIC). Această versiune 1.9 oferă definiții standard de vocabular pentru punctele de vedere și domenii funcționale și descrie rolurile pentru utilizatorii umani. De asemenea include informații despre comunicațiile fără fir pentru sistemele de automatizare industrială [10]. Digitalizarea tot mai accentuată combinată cu IIoT care este următorul pas în automatizarea industrială. Această automatizare industrială necesită combinarea tehnologiei de automatizare tradiționale (OT) cu tehnologia informației (IT). Internetul industrial al lucrurilor (IIoT) este exact ceea ce face posibil acest lucru. Rețelele în care sunt conectate IoT – urile se caracterizează prin acoperiri pe distanțe mari (15-20 de kilometri), costuri reduse de conectare, bateriile ce alimentează senzorii trebuie să țină foarte mult și capacitatea de a conecta simultan un număr mare de dispozitive.

2.2 Protocoale de comunicație IoT

Conceptul de internet of things (Internetul lucrurilor sau obiectelor) a dus la îmbunătățirea modului în care trăim, muncim și ne distrăm și a dus și la dezvoltarea unor noi afaceri și la automatizarea întreprinderilor. Comunicația între dispozitivele unei rețele se face utilizând protocoale de comunicație. Astfel IoT – urile necesită protocoale de comunicație specifice . Aceste protocoale reprezintă reguli care fac ca două sau mai multe dispozitive inteligente să comunice între ele. La ora actuală există mai multe protocoale de comunicație: MQTT(Message Queuing Telemetry Transport), serviciul de distribuție a datelor (DDS), AMQP (Protocol avansat pentru ordonarea mesajelor), protocolul CoAP (Constrained Application Protocol), REST (REpresentational State Transfer), XMPP (Extensible Messaging and Presence Protocol), WiFi, etc.

2.3 Gateway-urile IIoT

Gateway-urile IIoT (Industrial Internet of Things) sunt elemente esențiale în infrastructura proceselor industriale moderne, conectând diverse dispozitive, senzori și elemente de execuție la Internet pentru a facilita monitorizarea și controlul proceselor industriale. Aceste gateway-uri permit colectarea, procesarea și transmiterea datelor într-un mod sigur și eficient.

Gateway-urile IIoT sunt esențiale pentru digitalizarea și modernizarea proceselor industriale. Ele asigură o conectivitate robustă și securizată între dispozitivele industriale și rețelele IT, facilitând colectarea și analiza datelor, ceea ce duce la o mai bună eficiență operațională și reducerea costurilor. Cu progresele continue în tehnologie, aceste gateway-uri vor juca un rol și mai important în transformarea digitală a industriilor la nivel global.

De asemenea gateway-urile IIoT joacă un rol crucial în implementarea și succesul Industriei 4.0 (paradigmă), asigurând o conectivitate robustă și securizată între dispozitivele industriale și internet. Prin colectarea și analiza datelor în timp real, aceste gateway-uri permit automatizarea, optimizarea și îmbunătățirea continuă a proceselor industriale, conducând la o eficiență operațională sporită, reducerea costurilor și îmbunătățirea calității produselor și serviciilor.

Putem menționa câteva caracteristici și funcții ale gateway-urilor:

- Se pot utiliza protocoale de comunicații industriale (Modbus, CAN, etc)
- Se pot utiliza diferite tipuri de protocoale de conectivitate wireless (Wi-Fi, LTE - internet, Zigbee, LoRaWAN – procese industriale)
- Se pot utiliza porturi fizice Ethernet, RS232, RS485, USB.
- Edge computing: oferă capacitatea de a prelucra datele local, reducând latența și volumul de date transmis către cloud.
- Criptare de date: TLS/SSL pentru securitatea comunicației.

Beneficiile gateway-urilor:

- Monitorizarea datelor în timp real pentru a se lua decizii rapide.
- Automatizarea proceselor industriale.
- Întreținere predictivă prin identificarea și remedierea problemelor înainte ca acestea să apară.
- Optimizarea resurselor
- Integrarea cu diverse sisteme și platforme software.

În Figura 2-2 este prezentată arhitectura generală a sistemului client-server ModBusE, unde avem:

- IIoT ModBusE Gateway care implementează ciclul de achiziție și serverul OPC UA.
- Serverul IIoT ModBusE este serverul (slave-ul) care preia date de la senzori sau trimite comenzi către elemente de execuție sau actuatoare.

În continuare se prezintă câteva arhitecturi de procesoare ce se pot folosi pentru a se implementa gateway-ul IIoT ce utilizează extensia ModbusE:

1) STM32MP157C/F – processor dual - ARM Cortex A7 800 MHz + Cortex M4 MPU [11] . Dispozitivele STM32MP157C/F se bazează pe ARM dual-core de înaltă performanță. Procesorul RISC Cortex A7 pe 32 de biți care funcționează până la 800 MHz. Procesorul Cortex A7 include un cache de instrucțiuni L1 de 32 kbyte pentru fiecare CPU și un cache de nivel 2 de 256 Kbyte. Procesorul Cortex A7 este un procesor foarte eficient din punct de vedere energetic proiectat pentru a oferi performanțe bogate în wearable high-end și alte aplicații integrate de consum redus de energie. Oferă cu până la 20% mai multă performanță pe un singur thread decât Cortex A5 și oferă performanțe similare cu Cortex A9. Dispozitivele STM32MP157C/F încorporează, de asemenea, un nucleu RISC Cortex M4 pe magistrală de 32 de biți și care funcționează la frecvența a ceasului de 209 MHz. Procesorul Arm Cortex M4 are o unitate flotantă în virgulă mobilă (FPU) de simplă precizie care acceptă instrucțiuni și tipuri de date Arm cu simplă precizie pentru prelucrarea datelor. Cortex M4 are un set de instrucțiuni DSP complet și o unitate de protecție a memoriei (MPU) care crește securitatea aplicației. Dispozitivele STM32MP157C/F încorporează și o unitate de procesare grafică 3D (Vivante - OpenGL ES 2.0) care rulează la o viteză de până la 533 MHz. Dispozitivele STM32MP157C/F oferă o interfață SDRAM externă care acceptă extern memorii cu densitate de până la 8 Gbiți (1 Gbyte), LPDDR2/LPDDR3 sau DDR3/DDR3L pe 16 sau 32 biți până la 533 MHz. Procesorul Cortex A7 încorporează toate caracteristicile de înaltă performanță Cortex A15 și Cortex A17, inclusiv suport de virtualizare în hardware, NEON™ (Arm Neon este o arhitectură SIMD (SIMPLE INSTRUCTION MULTIPLE DATA) care poate procesa date în

paralel folosind registre pe 64 sau 128 de biți) și AMBA (The Advanced Microcontroller Bus Architecture) pe 128 de biți. De asemenea Cortex A7 beneficiază de un cache L2 integrat conceput pentru consum redus de energie, cu mai puține latențe ale tranzacțiilor și suport îmbunătățit al sistemului de operare pentru întreținerea cache-ului.

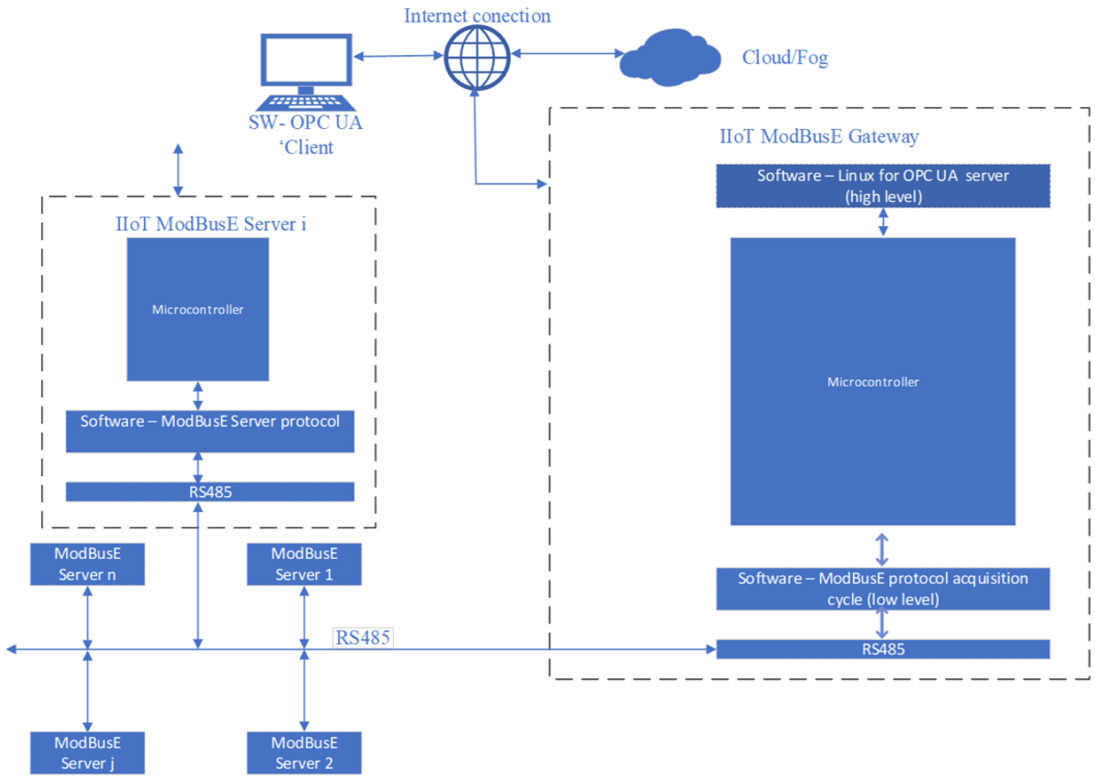


Figura 2-2 Arhitectura generală a sistemului experimental pentru gateway IIoT ce utilizează extensia ModbusE.

2) Seria STM32H747xI/G de la STMicroelectronics [12] oferă performanța nucleelor ce pot rula până la 480 MHz și respectiv 240 MHz, permițând mai multă procesare și partiționare a aplicațiilor. Liniile de produse STM32H7 dual-core sunt disponibile cu un **SMPS step-down converter** încorporat pentru o eficiență dinamică îmbunătățită a consumului de energie. Versiunile cu un singur nucleu, Cortex M7, oferă fie performanță excelentă la 550 MHz, fie o combinație unică de performanță și economie de energie la 280 MHz și 34 µA (tipic) în modul STOP.

3) BeagleBone Black folosește procesorul Sitara AM335x de la Texas Instruments. Procesorul Sitara AM335x este constituit dintr-un procesor ARM Cortex A8 RISC cu o magistrală de 32 de biți care operează la o frecvență a ceasului de până la 1 GHz și două procesoare PRU pe 32 de biți pentru operații în timp real care funcționează la o frecvență de 200 MHz.

În această lucrare, am ales varianta cu BeagleBone Black pentru că nucleul PRU la 200 MHz, permite o comunicație serială de maxim 12 Mb/s (tipică pentru Profibus), poate să ajute la îmbunătățirea utilizării fluxului de date pentru comunicația serială, iar procesorul Cortex A8 la o conexiune IIoT performantă. Astfel pe unul din cele două procesoare PRU va fi implementat ciclul de achiziție (CA) al extensiei ModbusE bazat pe modelul client (master) -

server (slave). Serverul OPC UA poate fi implementat pe procesorul ARM Cortex A8 care este un procesor de mare putere. De asemenea, se va utiliza driver-ul specific Sitara AM335x care să permită comunicația între cele două procesoare (PRU - client/server ModbusE și ARM Cortex A8 RISC - OPC UA server) pentru depanarea aplicației (ciclu de achiziție , modulul dispatcher (intermediar între ciclul de achiziție și serverul OPC UA)).

În arhitectura unui gateway putem avea:

- Modbus RTU – Modbus RTU (de exemplu STM32f429) conexiune in sistem pe Modbus RTU (virtual com).

- Modbus RTU - Modbus TCP/IP(de exemplu STM32f756). Procesor Modbus RTU, PRU (de exemplu Sitara A335x) – procesor cortex A8 cu conexiune de nivel înalt in sistem (OPC UA , MQTT, etc).

Extensia ModbusE implementează nivelul 1 și 2 din stiva de protocoale OSI astfel:

- La nivelul 1 (nivelul fizic) – avem ModbusE punct la punct prin utilizarea standardului de linie RS232 și ModbusE multipunct prin utilizarea standardului de linie RS485. Problema comună este comutarea direcție pentru driverul de rețea multipunct.
- La nivelul 2 (nivelul legăturii de date) - se realizează implementarea ciclului de achiziție (CA). În cadrul ciclului de achiziție sunt transmise și recepționate toate mesajele ModbusE. Structurile de date pentru ModbusE de la nivelul legăturii de date pot fi accesate fie prin comenzi Modbus RTU, sau prin alte metode de citire din structurile Modbus RTU. Aceasta este sarcina clientului (master-ului). De aici, informația poate fi transmisă mai departe fie utilizând un client Modbus RTU, Modbus TCP/IP, sau la un nivel mai înalt folosind mecanismele sistemului de operare din Linux Debian pentru Sitara care permite citirea datelor din structură folosind memoria partajată (shared memory) între nucleele PRU și procesorul ARM Cortex A8 (Sitara AM335x). Datele astfel achiziționate, sub sistemul de operare Linux, ajung la un distribuitor de date în cazul acesta serverul OPC UA. Clienții OPC UA pot distribui informația chiar în cloud. Există soluții mai simple în care informația poate fi transmisă pe Modbus RTU, Modbus TCP/IP, MQTT către nivelurile superioare.

Avantajul soluției cu server OPC UA pe Sitara AM335x este aceea că integrează pe un singur sistem atât facilitățile necesare pentru ModbusE cât și conexiunea în cloud folosind OPC UA. Pentru a îndeplini aceste obiective este foarte clar că avem nevoie de un microcontroller de viteză pentru ModbusE (PRU) cât și de un microcontroller (Cortex A) pentru implementarea server-ului OPC UA. Variantele (care nu au procesorul Cortex A) care sunt mai simple și implementează eventual un driver TCP/IP pe microcontrollere care implementează și ModbusE permit conexiunea către cloud fie prin intermediul unui virtual com fie prin TCP/IP.

3 Protocolul Modbus cu extensia Modbus EXTINS.

3.1 Istoric al rețelelor industriale locale

Acceptarea și utilizarea PLC-urilor (Programmable Logic Controllers) în procesele industriale a avut ca rezultat apariția rețelelor industriale. La început rețelele industriale erau deținute de diferiți producători de PLC-uri. Utilizarea rețelelor locale LAN (Local Area Network) pentru interconectarea PC-urilor și a dispozitivelor în cadrul sistemelor industriale de automatizare a început să prindă contur începând cu anii 80. Câștigul concretizându-se în comunicația de cost scăzut și cu capacitate mare oferit de aceste rețele locale (LAN) a ajutat la dezvoltarea de sisteme distribuite, dar și a serviciilor de automatizare. Aceste sisteme industriale de automatizare folosesc arhitecturi distribuite deschise și comunică date prin intermediul rețelelor digitale. Este de actualitate acum pentru toți utilizatorii interconectați la o rețea locală să comunice cu PC-uri sau diferite dispozitive ce se află în alte rețele locale prin intermediul unor rețele mai mari WAN - Wide Area Network.

Introducerea rețelelor industriale locale (RIL) ca suport de comunicație între dispozitive a fost printre puținele evoluții tehnologice care au schimbat profund automatizările. Rețelele industriale au evoluat pe măsură ce aplicațiile industriale și alte dispozitive industriale au devenit mai sofisticate. În zilele noastre rar se găsesc instalații industriale care să nu folosească sisteme fieldbus sau Ethernet. Internetul industrial este tendința în ziua de azi a sistemelor de comunicație industriale.

Rețele industriale locale sunt utilizate cu precădere în cadrul fabricilor, uzinelor și altor facilități industriale pentru a conecta diverse dispozitive, echipamente și sisteme automatizate. RIL-urile permit schimbul de informații între controlerele logice programabile (PLC-uri), senzori, actuatori, sisteme de monitorizare și alte dispozitive industriale, facilitând controlul și monitorizarea proceselor industriale. De fapt rețelele industriale formează nucleul fabricilor industriale moderne prin facilitarea automatizării și controlul proceselor

Rețelele de tip *fieldbus* au influențat flexibilitatea și performanța sistemelor de automatizare fiind create pentru nivelurile ierarhice de automatizare apropriate de proces. Prin cuvântul field se înțelege pe teren adică aproape de proces și departe de centru iar fieldbus descrie rețelele de cabluri industriale [13]. Fundația (fundation) Fieldbus definește fieldbus ca o comunicație bidirecțională între dispozitive inteligente [14]. Prin urmare fieldbus reprezintă o rețea locală (LAN) pentru controlul proceselor și automatizarea fabricilor. Deci prin RIL vom înțelege atât rețele locale industriale cât și rețele de tip fieldbus. Procesele industriale automatizate implică multe dispozitive, deci prin urmare conectarea tuturor dispozitivelor implică un număr foarte mare de cabluri de legătură. Dacă dispozitivele care preiau datele de la senzori sunt plasate aproape de aceștia se pot utiliza două fire răsucite pentru a conceta între ele aceste dispozitive.

În general dimensiunile instalațiilor de automatizare ale proceselor industriale sunt deosebit de mari, deci utilizarea RIL-urilor în aceste instalații de automatizare ar duce la reducerea creșterii numărului mare de cabluri dar și la ierarhizarea instalației de automatizare a proceselor [13]. În prezent, Ethernetul este tehnologia cea mai des utilizată în rețelele LAN. Deși în trecut Ethernet-ul tradițional prin mecanismul MAC (CSMA/CD) prezenta un obstacol pentru utilizarea sa în mediul industrial, astăzi datorită evoluției Ethernet-ului comutat se poate vorbi de Ethernet-ul industrial. În Ethernet-ul clasic stațiile împart canalul de comunicație în schimb

la Ethernet-ul comutat fiecărei stații îi este dedicată o conexiune în felul acesta coliziunile nu mai apar. Bineînțeles că trebuie să se acorde o mare atenție asupra fiabilității și capacității de comunicație în timp real a Ethernet-ului industrial când se folosește pentru a conecta dispozitivele din teren [15]. Progresul tehnologic al Ethernet-ului industrial va influența dezvoltarea rețelelor industriale locale. Rețelele locale industriale sunt mai optimizate pentru sarcinile specifice de automatizare decât Ethernet-ul. Datorită diversificării echipamentelor și serviciilor furnizate de diverși producători a fost necesară elaborarea de standarde pentru rețele. ISO (International organization of Standardization) a dezvoltat modelul de referință pentru interconectarea calculatoarelor numit modelul OSI (Open Systems Interconnection). Modelul asigură că sistemele dezvoltate de producători diferiți sunt compatibile și pot comunica cu ușurință. Ca și particularitate, rețelele industriale locale folosesc doar 3 niveluri definite în modelul OSI, mai exact, nivelul fizic, nivelul legăturii de date iar cel de al 3 lea este nivelul aplicație. Trebuie de precizat că în legătură cu nivelul fizic și nivelul legătură de date , stațiile finale ar trebui să aibă aceeași configurație a canalului.

3.2 Protocolul ModBus.

Protocolul ModBus a fost dezvoltat de firma Modicon în anul 1979 pentru a interconecta dispozitivele PLC proprii. A devenit și este un standard de comunicație în industrie și reprezintă în prezent unul din cele mai utilizate protocoale de comunicație folosite în procesele industriale. Modbus este un protocol deschis și independent de tipul de rețea. Fiind un protocol simplu și foarte ușor de implementat s-a dezvoltat o multitudine de aplicații pentru controlul și monitorizarea sistemelor industriale.

Protocolul ModBus folosește la nivelul 7 al stivei OSI modelul client / server iar la nivelul 2 al modelului OSI modelul master / slave (care poate fi abordat ca o arhitectura client/server unde rolul de master este preluat de client iar rolul de slave este preluat de server). În Figura 3-1 este prezentat modul de comunicație ModBus în modelul client/srever. Comunicația între clientul Modbus și serverul Modbus începe atunci când stația client Modbus trimite un mesaj către serverul Modbus pentru a schimba informații sau pentru a executa o anumită comandă. Serverul Modbus când primește cererea execută comanda și răspunde clientului și furnizează datele cerute. În cadrul unui sistem care folosește protocolul Modbus timpul de răspuns depinde de timpul necesar clientului pentru a trimite si recepționa răspunsul și de timpul necesar serverului de a răspunde la cererea primită.

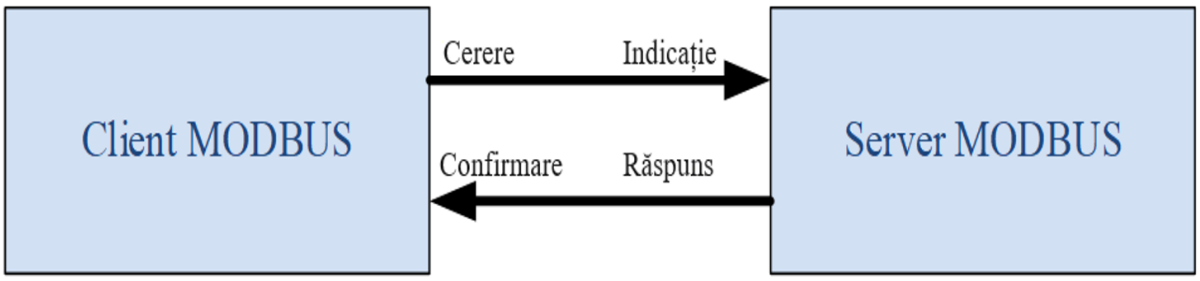


Figura 3-1 Comunicație Modbus în modelul client / server [16]

Aplicații ca HMI - Human Machine Interface sau SCADA - Supervisory Control And Data Acquisition implementează serviciul client în timp ce dispozitive de intrare/ieșire (senzori, actuatori, etc.) implementează serviciul server.

În Figura 3-2 se prezintă un exemplu de arhitectură a unei rețele Modbus în care se pot observa dispozitivele de tip porți de access (gateway) care au rolul de a conecta dispozitivele de pe nivelul inferior, în general dispozitive care nu sunt smart, si nu se pot conecta pe internet.

Prin utilizarea mesajelor Modbus se face schimbul de date pentru protocoalele de aplicație. Modelul client / server acre este utilizat de Modbus folosește diferitele posibilități oferite de rețelele de comunicație de pe nivelul inferior (unu-la-unu, unu-la-mai mulți, verificarea erorilor, pipelining, etc.). Protocolul Modbus poate fi folosit de oricare dintre dispozitivele PLC - Programmable Logic Controller, HMI - Human Machine Interface, dispozitive de intrare/ieșire pentru a iniția o operație în rețea. Aceeași comunicație se poate face atât pe rețele seriale cât și pe rețele TCP/IP , dispozitivele gateway permițând comunicația între diferitele tipuri de magistrale sau rețele folosind protocolul Modbus [17].

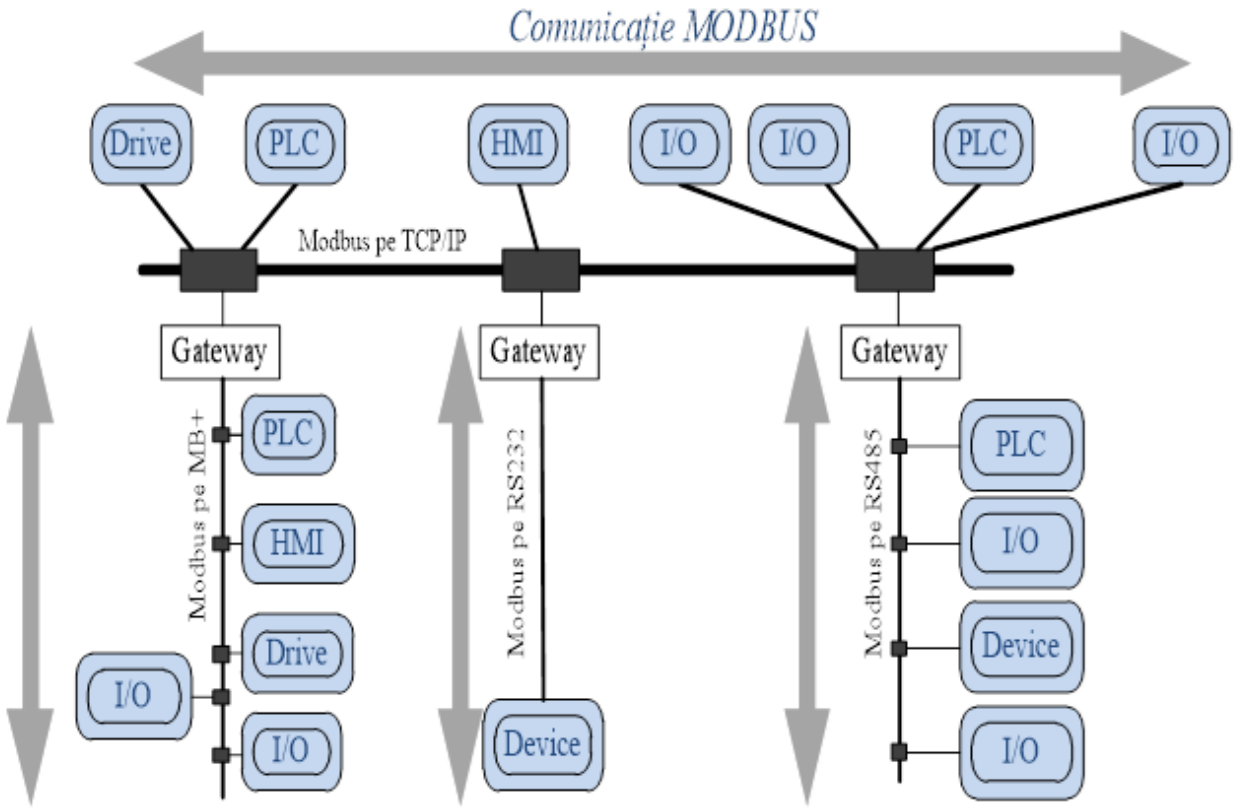


Figura 3-2 Exemplu de arhitectură a unei rețele Modbus [17]

3.2.1 Analiza bibliografică privind evoluția protocolului Modbus

În ziua de azi, numărul de fieldbus-uri existente este foarte mare, o parte sunt complet definite iar altele sunt incomplet definite. Pentru fieldbus-urile incomplet definite, variabila de timp care este cheia principală în achiziția datelor în timp real, nu există în mod explicit. Protocoale

ca M-bus , Modbus RTU, sunt câteva exemple de protocoale incomplet definite. Modbus Extins (ModbusE) care reprezintă o extensie a protocolului Modbus introduce variabila de timp prin implementarea unui model de achiziție cu ciclu flexibil pe o stație client. Acest model de achiziție este bazat pe arhitectura client (master) – sever (slave). Această extensie ModbusE a fost dezvoltată ca fiind o soluție simplă și ieftină și care se bazează pe hardware-ul existent.

În lucrarea [18], autorii descriu ce avantaje are limbajul Protege într-un sistem industrial. Astfel autorii utilizând acest limbaj au realizat implementarea protocolului Modbus atât pe TCP/IP cât și pe RTU (linia serială) și au folosit un gateway industrial pentru a o testa. Autorii au demonstrat că utilizarea limbajului Protege, pentru implementarea protocolului Modbus, facilitează implementarea stivei de protocol și face mai ușoară întreținerea.

În [19], autorii au propus extinderea, într-un mod compatibil înapoi a spațiului de adresare Modbus de la adrese de 8 biți la adrese de 16 biți. În acest fel, doar dispozitivele care implementează această extensie vor accepta acest mod adresare, iar celelalte dispozitive vor ignora aceste mesaje. Tot în această lucrare autorii au propus o arhitectură multi-client (multi-master) în care fiecare dispozitiv poate deveni client (master) la un moment dat printr-un protocol de alegere al clientului (master-ului).

În lucrarea [20], autorii descriu un canal (ascuns) pentru comandă și control pentru Modbus TCP/IP. Astfel canalul folosește biții care sunt mai puțin semnificativi ai regiștrilor pentru a stoca și schimba informații între o stație client și o stație server folosind metode de scriere și citire. Astfel rezultă un compromis între canalul ascuns și lățimea de bandă. Implementarea canalului ascuns a fost realizată în Python prin utilizarea bibliotecii Modbus-tk [21]. În timp ce arhitectura canalului este de tip client server pentru cerere, comunicația de date este bidirecțională. Canalul ascuns descris de autori nu acoperă toate variantele posibile ale canalelor SCADA [22].

În lucrarea [23], jiter-ul comunicației Modbus TCP/IP este evaluat prin date experimentale și analize teoretice. Autorii arată că mecanismul de recunoaștere TCP/IP , parte a protocolului TCP/IP, introduce o cantitate apreciabilă de jitter în comunicația Modbus TCP/IP. S-au efectuat 2 seturi de experimente pentru a se evalua supracontrolul și jiter-ul. Primul set de experimente măsoară timpul RTD (Round Trip Delay), iar cel de al doilea set se referă la configurația TCP/IP ce se folosește ca referință. Experimentele realizate au fost făcute prin transferul conținuturilor ce au fost înregistrate uniform distribuite între sistemele client (master) și server (slave). Sistemele folosesc un microcontroler low cost NXP LPC2468 folosit pentru aplicații industriale [24] bazat pe un procesor ARM7 și pe care rulează un sistem de operare de timp real FreeRTOS. Analiza descrisă de autori arată că jiter-ul de comunicație este afectat în mod semnificativ de mecanismul de recunoaștere a segmentului TCP ceea ce a periclitat folosirea protocolului Modbus TCP în sisteme de timp real.

În lucrarea [25], autorii prezintă modelul de protocol Modbus RS-485 server (slave) și implementarea acestuia. Autorii au propus un hardware ARM pentru a demonstra faptul că modulul server (slave) Modbus poate face parte dintr-un sistem de comunicații stabil și fiabil. Execuția programului software fiind rulată pe un sistem de timp real. Arhitectura software este portabilă putând fi aplicată în orice fel de sistem care suportă SOTR. În procesul de testare

calculatorul este conectat la modulul server (slave) prin intermediul unui circuit de conversie (ce folosește UART – RS-485 transceiver IC). Se trimite o solicitare la fiecare 200 ms la modulul server (slave) pentru a se realiza reîmprospătarea valorilor intrărilor și registrelor discrete. Testarea comunicației cu PLC S7-200 ce folosește Modbus RTU pe linia serială RS232 s-a făcut prin utilizarea unui circuit de conversie. În momentul în care conexiunea este stabilă , PLC S7-200 acționează ca un client (master) și cere serverului (slave-ului) să modifice stările ieșirilor digitale.

În lucrarea [26], autorii au demonstrat că există posibilitatea de a îmbunătăți transmisia în comunicațiile industriale care folosesc protocolul Modbus RTU prin folosirea de dispozitive cu retransmisie care au capacitatea de detecta erori. Autorii descriu implementarea schemei de detecție de erori într-o rețea ce folosește Modbus RTU ce se bazează pe codul REED-SOLOMON. Detectarea și corectarea erorilor se face folosind dispozitive de tip repetor din transmițător și receptor. În urmă măsurătorilor de timp a rezultat că maximul informațiilor redundante care pot fi adăugate la rețea este de 4 caractere de paritate. Codificarea și decodificarea s-a efectuat prin software prin adaptarea algoritmilor de codare SOLOMON.

În lucrarea [27], autorii prezintă un model de adaptare a protocolului Modbus pentru CAN (Controller Area Network), care mai este și denumit MODBUS CAN. Rezultatele experimentale descrise de autori arată că performanța MODBUS CAN într-un sistem care costă puțin (low cost) încorporat se poate compara în mod favorabil cu o implementare a protocolului Modbus TCP/IP, care se bazează pe o rețea Ethernet la o viteză de 100Mbps. În această lucrare atât proiectarea cât și validarea MODBUS CAN are ca obiectiv fragmentarea și reasamblarea Modbus PDU (Modbus Protocol Data Units) pentru a fi încadrate în cadre CAN, cadre ce dețin cel mult 8 octeți de date utile. Spre deosebire de protocolul Modbus RTU, MODBUS CAN prezintă performanțe mai mari, având atât costuri similare cât și cablaje simplificate care se bazează pe tehnologia magistralei de tip rețea. Astfel, pentru un transfer de date într-un proces industrial, se confirmă faptul că MODBUS CAN poate întrece performanțele Modbus TCP. Cu siguranță performanța Modbus TCP/IP se poate îmbunătăți fie prin alegerea unui procesor cu viteză mare, fie prin rescrierea stivei de protocoale TCP/IP. Ambele cazuri conduc la creșterea costurilor de implementare. Fiecare PDU Modbus este codificat de către MODBUS CAN printr-un nivel de fragmentare situat sub entitatea protocolului Modbus. Se pot defini gateway-uri pentru a permite o interconectare transparentă între dispozitivele Modbus existente cu subrețele bazate pe CAN.

În lucrarea [28] este propusă îmbunătățirea arhitecturii protocolului Modbus RTU prin utilizarea comunicației hibride ce folosește Modbus RTU cu fir și IEEE 802.15.4 fără fir. Astfel protocolul hibrid pentru comunicații propus crește controlul și limitele topologice care sunt cerute de Modbus RTU, permițând astfel o topologie cu fir/fără fir de tip tree-bus și multiplexarea client (master). În urma testelor această arhitectură prezintă o rată scăzută de eroare de comunicație, deci poate să satisfacă cerințele rețelelor de comunicație din industrie.

În lucrarea [29], se descrie un instrument care analizează timpul de răspuns pentru rețelele Modbus RTU (RS485). Acești timpi de răspuns pentru mesaje sunt preluați de un dispozitiv specializat după care sunt trimiși la aplicația software care face analiza. Autorii acestei lucrări propun două abordări pentru a se evalua timpii de răspuns. Astfel în cadrul primei abordări,

dispozitivul client (master) este înlocuit cu un dispozitiv specializat ce se ocupă cu evaluarea timpilor de răspuns pentru o colecție de mesaje care sunt cunoscute. În abordarea a doua dispozitivul specializat este utilizat ca un dispozitiv pasiv ce colectează informații cu privire la mesajele ce comunică între dispozitivele rețelei și timpii de răspuns. Prin intermediul acestui instrument s-a putut studia cum variază timpii de răspuns în ceea ce privește caracteristicile dispozitivelor din rețea.

În lucrarea [30] se prezintă un sistem care folosesc protocolul de comunicație Modbus pe procesorul ARM Cortex M0 pe 32 de biți. Astfel sistemul implementează în mod fiabil achiziția și transmiterea de date. Comunicația dintre client (master) și server (slave) ce utilizează protocolul Modbus RTU este implementată pe sistemul de timp real uC/OS-II. Stația server (slave) este implementată pe procesorul ARM Cortex-M0 iar stația client (master) este implementată pe PC. Din teste rezultă că, comunicația este fiabilă, viteza de transmisie este rapidă, deci poate fi folosită în industrie.

În lucrarea [31] se prezintă o metodă prin care se recuperează cadrele corupte în cazul erorilor de emisie în cadrul comunicației seriale Modbus RTU. Această metodă folosește coduri de corectare a codurilor înainte, adăugând-se în acest fel informații de paritate în ferestrele de timp atunci când magistrala este în repaus. Metoda permite extinderea caracteristicilor Modbus RTU, dar menține în același timp și compatibilitatea cu alte echipamente industriale. Atât repetoarele cât și corectorii de eroare sunt proiectați utilizând un codor/decodificator RS (255,223) pe 8 biți.

Diversitatea cooperărilor dintre industrie și mediul academic a avut ca și rezultat inovații spectaculoase în domeniul protocoalelor de comunicație. Rezultatele cercetărilor din mediul academic au contribuit semnificativ în inovațiile din rețelele de automatizare industriale.

3.3 Modbus Extins - ModbusE

Extensia protocolului Modbus, denumită Modbus Extins (Modbus Extension) se adresează nivelului legăturii de date al stivei OSI și este o extensie care îmbunătățește varianta de protocol Modbus RTU. Pentru a se introduce variabila de timp este nevoie de o stație client, denumită Base Station Gateway (BSG), stație care permite accesul la fieldbus, adică rețeaua industrială locală (RIL), folosind internetul [32]. BSG-ul poate adăuga o ștampilă de timp informațiilor. Protocolul ModbusE definește un ciclu de achiziție la nivelul stației BSG corespunzător protocoalelor incomplet definite pentru a se adăuga ștampila de timp. Totodată trebuie să reamintim și faptul că structura ciclului de achiziție (CA) depinde de ce fel de tip de rețea industrială este utilizată [33]. ModbusE definește la nivelul ciclului de achiziție (CA) două tipuri de obiecte [32]:

- PDO (Process Data Object). Acest tip de obiecte se folosește în comunicația de date de la proces la proces.
- SDO (Service Data Object). Acest tip de obiecte se folosește pentru configurarea stațiilor, mentenanță și testare.

Se pot utiliza trei tipuri de mesaje în cadrul comunicației folosind ciclu de achiziție și anume [32]:

- Trimite date cu recunoaștere (Send Data With Acknowledgment - SDA) – adică mesajele tip cerere – răspuns.
- Trimite date fără recunoaștere (Send Without Acknowledgment - SDN) – adică mesaje cerere.
- Trimite și solicită date (Send And Request Data - SDR) – mesaje cu solicitare de răspuns.

Protocolul ModBusE folosește modele de comunicație Client- Server, Producător – Consumator, Master – Slave (care poate fi abordat ca un model client - server unde rolul de master este preluat de client iar rolul de slave este preluat de server). Referitor la ModBusE se pot defini următoarele pentru ciclul de achiziție [32], [34]:

- Tick-ul (θ) – care reprezintă unitatea de bază de timp , această mărime trebuie să fie acceptată de toate stațiile din rețea care folosesc protocolul ModBusE.
- Slot-urile S de lungime l – ciclu de achiziție este format din slot-uri care pot avea lungimi diferite. Lungimea l fiind multiplul unui tick θ. Pentru o funcționare normală slot-urile au lungime fixă.
- Prioritatea PRi – slot-urile pot avea priorități.
- Slot-uri periodice și aperiodice – slot-urile periodice transmit mesaje periodice. Există cel puțin un slot aperiodic pentru comunicația obiectelor SDO.
- Slot-uri nedefinite - este un caz special, în acest caz obiectele de tip PDO și obiectele de tip SDO sunt transmise prin folosirea de cozi separate. Atunci când sunt obiecte de tip SDO în coadă, mediul de comunicație se alocă în mod alternativ între obiectele SDO și PDO.
- Subciclii ASC (Acquisition Sub-Cycle) – ciclul de achiziție poate avea subciclii.
- Mesajele sunt indivizibile – timpul maxim de transfer pentru un mesaj este Mi (tipmesaj, t).
- Tranzacțiile sunt indivizibile – o tranzacție TRi(t) constă din mai multe mesaje.
- Tranzacțiile sau mesajele se efectuează în slot-uri – un mesaj sau o tranzacție este efectuată într-un slot, deci o eroare de comunicație sau un mesaj care este incorect nu mărește lungimea slot-ului din ciclul de achiziție.
- Slot pentru situații de urgență, alarme sau sincronizare.

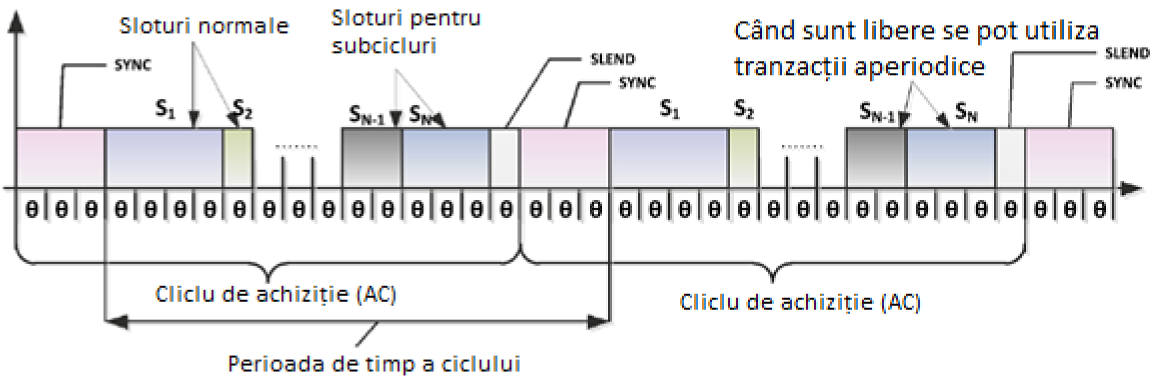


Figura 3-3 Structura generală a unui ciclu de achiziție [32]

După cum se poate vedea din Figura 3-3 structura ciclului de achiziție constă din slot-uri printre care un slot opțional SYNC care este folosit de BSG pentru a efectua o comandă cu difuzare

(broadcast) de exemplu să se înceapă scanarea intrărilor sau să se înceapă un nou ciclu de achiziție. După cum se observă din Figura 3-3 ciclul de achiziție se încheie cu slot-ul opțional SLEND care semnalează sfârșitul ciclului și de asemenea poate să trateze erorile ce au dus la creșterea unor slot-uri. Celelalte slot-uri de la S1 până la Sn au rolul de a permite efectuarea tranzacțiilor, slot-uri ce au lungimi de multipli de tick (θ). După cum s-a menționat mai înainte cel puțin un slot va fi folosit pentru traficul aperiodic, și acest slot trebuie să fie destul de lung pentru a se realiza tranzacția. Se poate alege și varianta ca toate slot-urile să aibă aceeași lungime. Timpul ciclului de achiziție se poate calcula stfel: $t_{AC} = t_{SYNC} + t_{END} + t_{S1} + \ldots + t_{SN}$.

După cum se vede în Figura 3-4 atunci când se utilizează o buclă de control formată din fieldbus-uri, și dacă ciclul unei astfel bucle de control este mai mic ca ciclul de achiziție (CA) există o soluție prin care se face planificarea slot-urilor dacă intervalul sub-cicli (Sc) dintre două slot-uri consecutive este mai mare decât cel mai mare slot. Un exemplu de buclă de control ar putea fi formată dintr-un controler PID și un element de acționare [35] care primește valori de la un senzor [32].

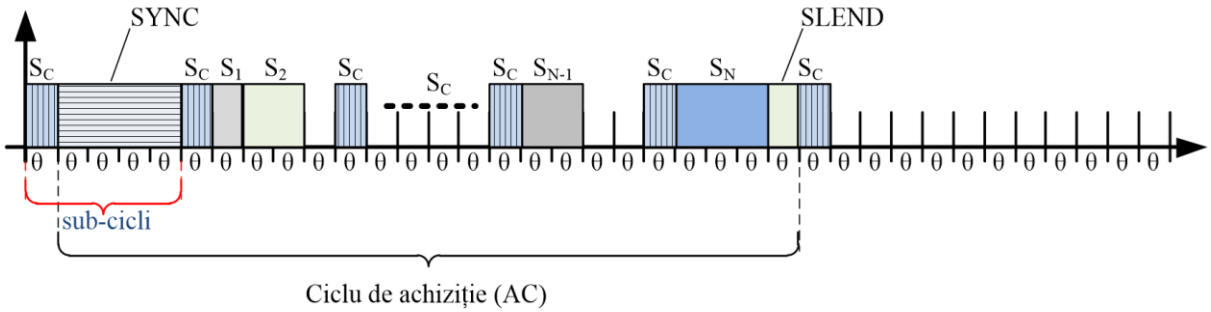


Figura 3-4 Soluție pentru buclele de control cu perioada de achiziție mai mică decât ciclul de achiziție [32]

În cazul în care sunt mai multe bucle în același subciclu planificarea se face la fel ca și în cazul unei singure bucle doar dacă diferența rămasă este mai mare decât cea mai mare durată a spațiului planificat. Se folosesc două variante pentru umplerea golurilor care apar între slot-urile Sc și pentru crearea ciclul de achiziție (CA) și anume fie se folosesc numerele slot-urilor ca priorități fie umplerea optimă a golurilor adică folosirea slot-urilor cât mai eficient astfel încât să nu rămână tick-uri nefolosite. Când diferența dintre perioadele slot-ului este mare se poate utiliza fie un slot sau mai multe slot-uri pentru tranzacții aperiodice caz în care se folosește metoda round – robin de comunicare a mesajelor fie se rezervă un interval în ciclul de achiziție care este folosit atunci când perioada este depășită sau se apropie de momentul de expirare.

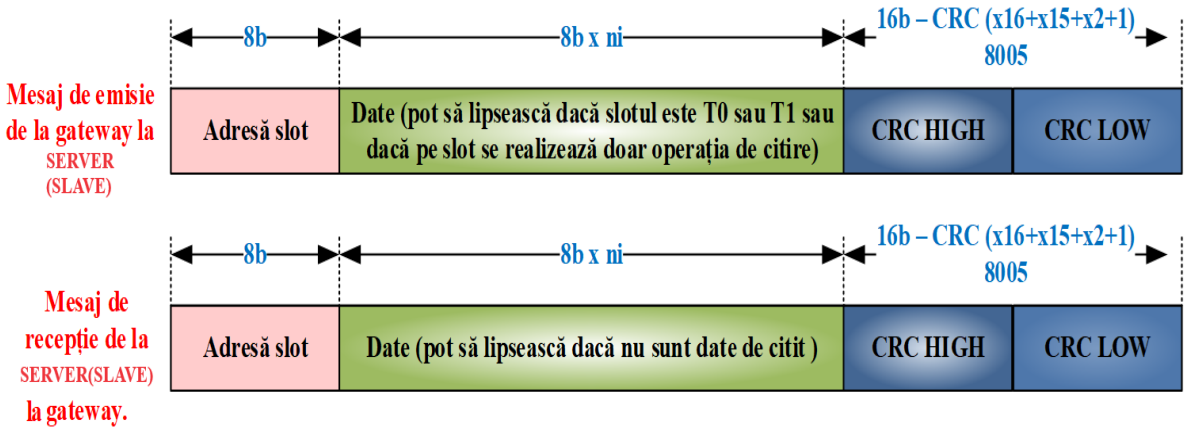
Ciclul de achiziție al protocolului ModbusE este de tip time-triggering și se dorește ca protocolul să devină determinist spre deosebire de protocoalele non-deterministe ca Modbus, M-Bus etc.

Dintre variantele Modbus pe linia serială doar Modbus RTU este potrivit pentru comunicațiile de timp real conform specificațiilor Modbus precum și a altor extensii non-standard [36]. În cazul variantei Modbus ASCII distanța maximă dintre caractere este de 6 secunde deci nu se poate folosi pentru o comunicație de timp real [37]. După cum s-a menționat anterior în

cazul protocolului Modbus RTU mesajele sunt separate prin intermediul unui interval de minim 3,5 caractere de inactivitate, de asemenea dacă există inactivitate de 1,5 caractere între două caractere consecutive înseamnă că mesajul este considerat ca fiind incomplet și este respins. Astfel durata maximă a unui mesaj este de (256 + 3,5 + 255x1,5) x 11 = 7062 biți, unde valoarea 256 reprezintă numărul maxim al caracterelor, iar valoarea 255 reprezintă numărul maxim al spațiilor. Lungimea maximă a unei perechi comandă / răspuns este 2 x (256 + 3,5 + 255 x 1,5) x 11) + t_{procsv} = 14,124 biți + t_{procsv}, unde t_{procsv} este timpul pentru procesarea comenzii de interogare a stației server, timp ce nu este specificat în protocolul Modbus. O soluție ar fi că la nivelul stației BSG să se poată estima un timp pentru slot și se pot utiliza patru contoare pentru fiecare slot. Un contor pentru mesajele care au fost recepționate corect, un contor pentru folosit pentru erorile datorate depășirii intervalului de 1,5 caractere între caracterele consecutive, un contor pentru erori de tip CRC și de asemenea un contor folosit pentru erori de tipul depășirea timpului (time-out). Dacă apar erori generate de timeout înseamnă ca tipul estimat pentru procesarea comenzilor este prea mic, deci durata slot-ului trebuie mărită. Dacă apar erori datorate intervalului de 1,5 caractere înseamnă că specificația de protocol nu este respectată. Dacă apar numai erori de CRC e posibil să fie zgomote în rețea sau stația server sau rețeaua este defectă. Se calculează RTT(Round Trip Time) pentru fiecare mesaj păstrându-se valoarea maximă. Astfel durata slot-ului trebui să fie multiplu de 3 care să fie mai mică decât valoarea RTT maximă plus timpul de procesare la nivelul BSG [32]. Slot-ul pentru obiectele SDO trebuie stabilit în funcție de aplicație, și de viteza de modificare ai unor parametrii ai sistemului. Se poate atașa unui slot una sau mai multe comenzi dar este de preferat doar o singură comandă să se atașeze unui slot. Microcontrolerele din ziua de azi [38] au incorporate circuite de tip UART cu performanțe bune cum ar fi viteze mari (mai mari de 10Mb/s), mod de comunicație multiprocesor (MM) utilizând bitul 9, transfer DMA și control automat a direcției la driver-ele RS485. Pentru comunicație MM, extensia ModbusE păstrează caracteristicile protocolului Modbus RTU, doar structura pe biți a unui caracter fiind diferită în sensul că bitul 9 care era folosit ca bit de stop sau paritate este folosit ca bit multiprocesor (MM). Dezavantajul acestei soluții ar fi că transmisia sau recepția unui caracter ar genera o întrerupere. Dar se poate rezolva dacă fiecare caracter din mesajul Modbus ar fi trimis cu bitul 9 pe valoarea 0 logic, excepție fac caracterele de adresă. Extensia ModbusE folosește 2 tipuri de mesaje: mesaje folosite pentru obiectele PDO și mesaje folosite pentru obiectele SDO. Mesajele pentru obiectele PDO sunt formate din adresa slot-ului, datele mesajului și suma de control. Lungimea mesajelor este fixă la nivelul stațiilor și poate fi maxim 256 de octeți. Obiectele PDO sunt mapate în regiștrii Modbus construindu-se astfel structura unui mesaj, iar o stație folosește un obiect de tip PDO pentru a transmite și unul pentru a recepționa. Doar stațiile server se pot abona la PDO-urile ce se transmit în rețea. În Figura 3-5 este prezentată structura mesajelor din cadrul unui slot din ciclul de achiziție al protocolului ModbusE. După cum se poate observa din Figura 3-5 mesajele de transmisie și recepție sunt identice. Bitul 7 al octetului care reprezintă adresa este setat pe 0 logic. Adresele de la 0x70 până la 0x7F sunt adrese rezervate, de exemplu adresa 0x70 va indica adresa unui slot de obiecte SDO. Dacă se dorește adresa slot-ului poate să fie extinsă cu încă un octet și poate să fie însoțită de suma de control, doar primul octet din adresa de slot-ului este MM. În cazul în care adresa are bitul 8 setat pe 1 este folosită pentru obiecte SDO. Valorile cuprinse între 0x81 și 0xEF sunt folosite pentru adresele stațiilor iar valorile 0x80 și 0xF0 până la 0xFF sunt rezervate. Se pot utiliza opțional 4 octeți ce reprezintă ștampila

de timp care precede un mesaj PDO. Folosirea obiectelor PDO și a caracterelor multiprocesor duce la îmbunătățirea utilizării canalului de comunicație deoarece tranzacțiile sunt mai scurte. Implementarea protocolului ModbusE necesită trei noi comenzi și anume [32] :

- comanda de mapare pentru emisia/recepția unui obiect PDO
- comanda de citire a unei emisii/recepții pentru obiectele PDO
- comanda pentru maparea adreselor slot-urilor, a adreselor fizice și a comenzilor Mosbus clasice.



Notă: sloturile T0 și T1 nu emit mesaje de citire pentru gateway și nici slot-urile goale

Figura 3-5 Structura mesajelor din cadrul unui slot (Modbus Extension - ModbusE) [32]

4 Arhitectura IIoT

În lucrarea [39] se prezintă că performanța obținută cu arhitecturile Cortex Mx este de aproximativ 58% date utile din ciclul de achiziție cu STM32F407 la 10,5 Mb / s, 51% date utile din ciclul de achiziție cu STM32F746 la 27Mb / s, și 70% date utile din ciclul de achiziție cu LPC4300 care are 2 Cortex M0 și M4. M0 se ocupă doar de protocol. Procesoarele Cortex M4 și M7 nu au resurse suficiente pentru a putea fi implementat serverul și clientul OPC UA.

Pentru îmbunătățirea utilizării canalului de comunicație propun o arhitectură de gateway IIoT care implementează protocolul de comunicație ModbusE folosind arhitecturi multiprocesor. Pentru implementare gateway-ului IoT s-a folosit platforma de dezvoltare BeagleBone Black, care folosește procesorul Sitara AM335x de la Texas Instruments. Procesorul Sitara AM335x este constituit dintr-un procesor ARM Cortex A8 RISC pe 32 de biți care operează la o frecvență a ceasului de până la 1 GHz și două procesoare PRU pe 32 de biți pentru operații în timp real care funcționează la o frecvență de 200 MHz. Acesta este și motivul alegerii procesorului Sitara AM335x sperând că procesorul PRU ajută la îmbunătățirea utilizării canalului de date iar procesorul Cortex A8 la o conexiune performantă la IoT. Astfel pe unul din cele două procesoare PRU va fi implementat ciclul de achiziție al protocolului Modbus Extension bazat pe modelul client (master) - server (slave). Serverul OPC UA poate fi implementat pe procesorul ARM Cortex A8 care este un procesor de mare putere. De asemenea va fi utilizat driver-ul specific AM335x Sitara care să permită comunicația între cele două procesoare (PRU - client (master) / server (slave) ModbusE și ARM Cortex A8 RISC - server OPC UA). Referitor la implementare a protocolului client / server Modbus Extins se propune îmbunătățirea performanțelor de utilizare a canalelor de comunicație mai bune decât unele din cele prezentate în lucrarea [39] (mai mare de 50%), utilizând procesoarele de timp real PRU. De asemenea procesorul ARM Cortex A8 de 1 MHz din punct de vedere al performanței este superior procesoarelor Cortex M4, M7 pentru implementarea atât a serverului OPC UA și a specificațiilor clientului (specificații acceptate de Industry 4.0) cât și a altor middleware IoT precum MQTT, AMQP, REST, DDS, CoAP (6LowPAN) etc.

În Figura 4-1 este prezentată arhitectura generală a sistemului client-server ModbusE, unde avem:

- IIoT ModbusE Gateway implementează ciclul de achiziție la nivelul PRU și serverul OPC UA la nivelul ARM Cortex A8, precum și utilizarea driver-ului de comunicație între procesoarele ARM și PRU.
- Serverul IIoT ModbusE este serverul (slave-ul) care preia date de la senzori sau trimite comenzi către elemente de execuție sau actuatoare.
- Celelalte servere ModbusE sunt alte stații de lucru realizate cu alte tehnologii.

Modulul IIoT ModbusE Gateway din Figura 4-1 reprezintă stația client (master) care implementează ciclul de achiziție (CA) la nucleul PRU0, serverul OPC UA și aplicația client, precum și aplicația dispecer la procesorul ARM Cortex A8. Serverul OPC UA și aplicația client preia comenzi de la un client OPC UA (care rulează pe un computer cu sistem de operare Windows) prin TCP/IP și le transmite către aplicația de dispatcher.

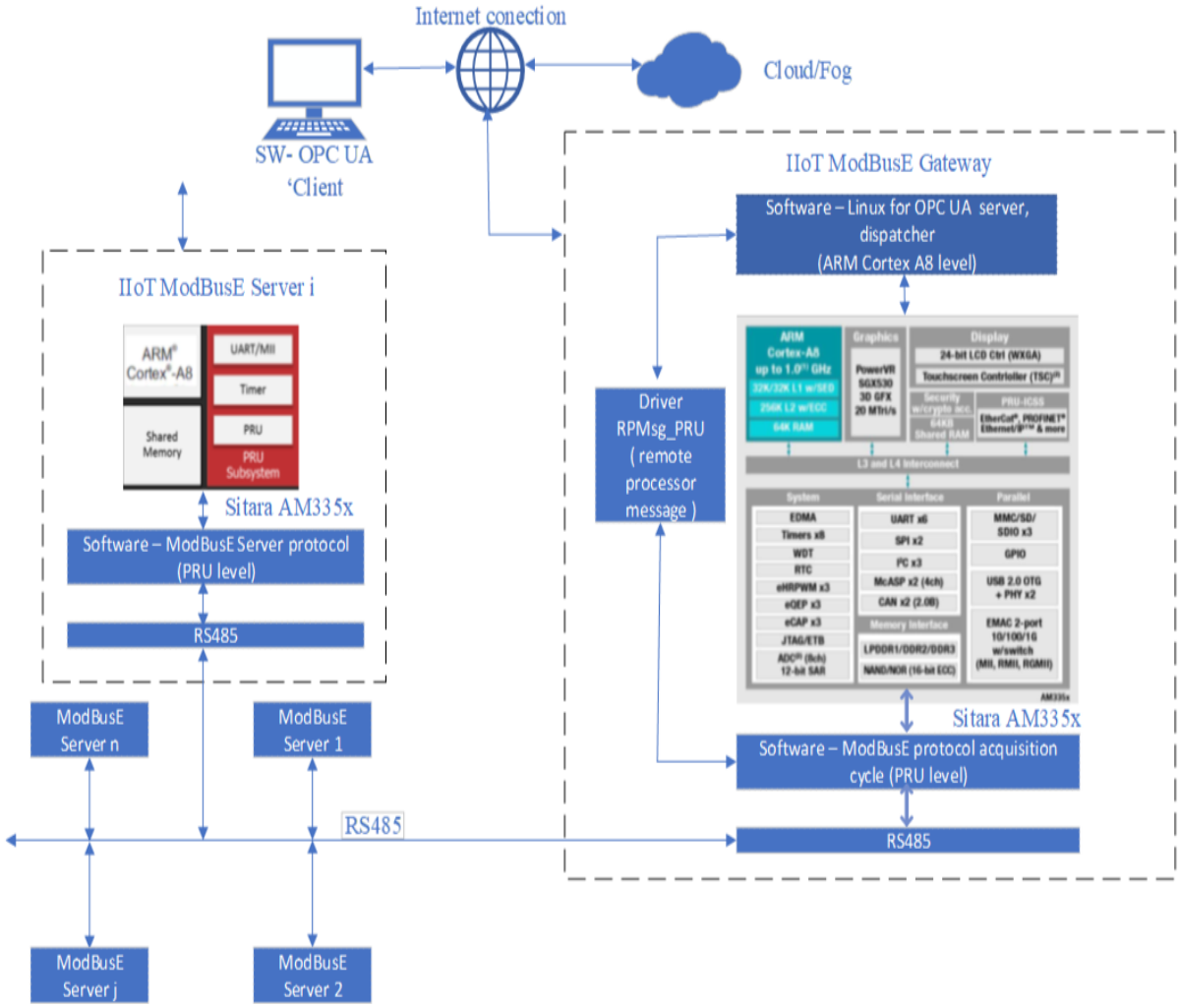


Figura 4-1 Arhitectura principală a sistemului experimental pentru gateway IIoT — ModbusE.

Aplicația dispatcher preia comenzi de la serverul OPC UA și de la aplicația client (aplicație realizată pentru test) și le trimite la ciclul de achiziție (CA). Dispatcher-ul preia răspunsul la comenzile din ciclul de achiziție și îl trimite către serverul OPC UA și aplicația client. Serverul OPC UA la rândul său, înaintează răspunsul către clientul OPC UA prin TCP/IP. Modulul IIoT ModbusE Server i din Figura 4-1 reprezintă stația server (slave) care este implementată și pe procesorul Sitara AM335x. Această stație preia date de la senzori sau trimite comenzi către elementele de execuție sau actuatoare în urma comenzilor de la stația client (master). Celelalte stații (ModbusE Server1, ..., ModbusE Server n) reprezintă alte stații de lucru care folosesc extensia ModbusE dar implementate cu alte tehnologii. Stațiile server (slave) comunică cu modulul ModbusE Gateway IIoT prin porturi seriale utilizând standardul de linie serială RS485. Analiza performanței sistemului experimental IIoT - ModbusE Gateway a fost realizată pentru un ciclu de achiziție format din 10 slot-uri.

4.1 Arhitectura BeagleBone Black.

Placa BeagleBone Black a fost dezvoltată de Texas Instruments în colaborare cu Digi-Key și face parte din familia beagleboard [40].

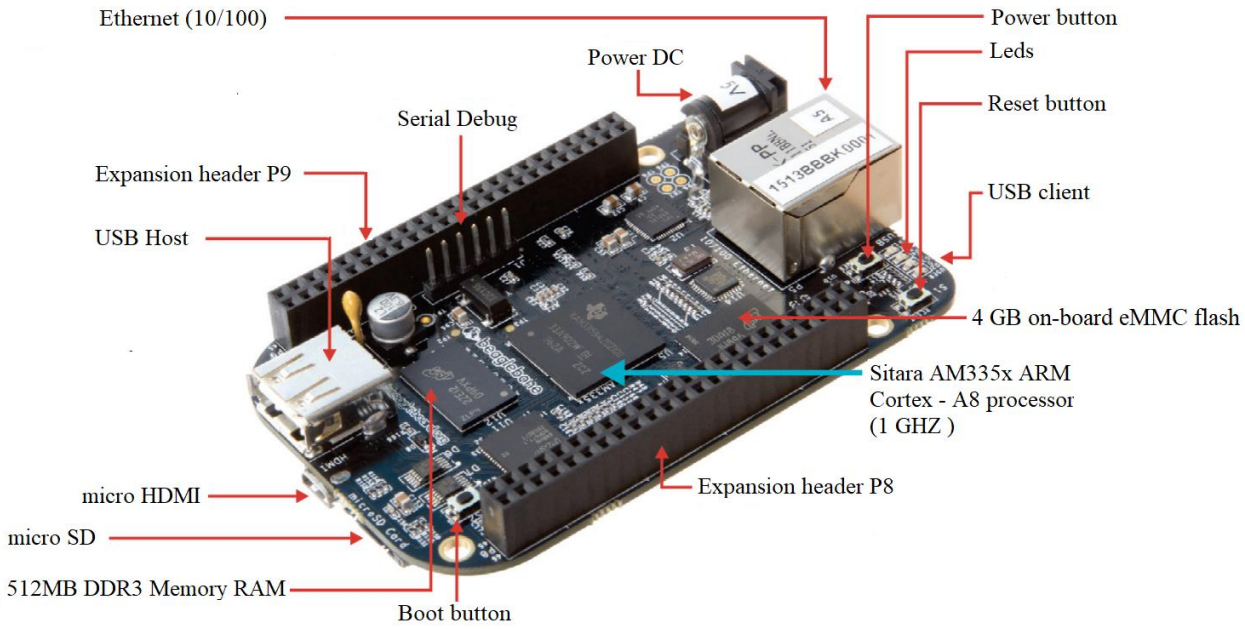


Figura 4-2 Placa BeagleBone Black [41].

După cum se poate observa din Figura 4-2 placa BeagleBone Black constă din [40]:

- Procesorul Sitara AM335x de la Texas Instruments
- 4 GB on board de memorie eMMC flash
- Un slot pentru card de memorie microSD pentru sistem de operare sau date adiționale
- Un port USB 2.0 host (Type A)
- Un port mini USB 2.0 pentru comunicație și alimentare
- 512 MB memorie RAM DDR3 care operează la frecvența de 800 MHZ
- Conectorii (header-ele) de expansiune P8 și P9 (Expansion Headers)
- MicroHDMI audio/video
- Conector cu 6 pini pentru interfața USB la serial și USB la JTAG (serial debug)
- Conector Ethernet 10/100 Mbps.
- Conector pentru alimentare (Consum redus de energie de 210 - 460 mA la 5V)

Placa BeagleBone Black mai este prevăzută și cu un accelerator grafic 3D (Processor Graphics Engine) și poate funcționa cu următoarele sisteme de operare [40]:

- Debian
- Android
- Ubuntu

5 Gateway IIoT — ModbusE

În Figura 4-1 este prezentată arhitectura principală a sistemului experimental client – server ModbusE , unde avem:

- IIoT ModbusE Gateway care implementează ciclul de achiziție (CA) la nivelul **PRU0**, serverul **OPC UA** la nivelul **ARM Cortex A8**, dispatcher-ul pentru comunicația cu serverul OPC UA/clienții **MODBUS TCP/IP** și ciclul de achiziție, precum și driverul de comunicație între procesoarele **ARM** și **PRU**.
- Serverul **IIoT** ModBusE este serverul (**slave-ul**) care preia date de la senzori sau trimite comenzi către elemente de execuție (actuatoare).
- Celelalte servere ModBusE sunt alte stații de lucru realizate cu alte tehnologii.

În Figura 4-1 **IIoT ModBusE Gateway** este client pentru **IIoT ModBusE Server i,** dar în același timp este server pentru **SW-OPC UA client**.

5.1 Modulul IoT ModbusE Gateway client (Master – în terminologia Modbus clasică)

Modulul IIoT Modbus gateway client folosește din punct de vedere hardware:

- procesorul **ARM Cortex A8** pentru implementarea serverului de comunicație cu clienții **Modbus TCP/IP (BBB_ARM_A8_MBE_IOT_GATEWAY)**, acest server preia comenzile de la clienți TCP/IP (de exemplu o aplicație care se execută pe PC) sau de la server-ul OPC UA le interpretează, comunică cu ciclul de achiziție , precum și pentru implementarea server-ului OPC UA.

- nucleul **PRU 0** pentru implementarea ciclului de achiziție (CA).

- nucleul **PRU1** pentru implementarea unei aplicații pentru depanare care face comunicația cu procesorul **ARM Cortex A8** prin intermediul driver-ului **RPMsg_PRU** (remote procesor message).

Comunicația dintre procesorul **ARM Cortex A8** și nucleele **PRU0** și **PRU1** se poate face și prin intermediul memoriei de date partajate de 12K. De fapt comunicația dintre dispatcher (serverul de comunicație cu clienții Modbus TCP/IP) și ciclul de achiziție (CA) se face prin intermediul memoriei partajate de 12K. Deoarece nucleele **PRU0** și **PRU1** dispun fiecare de câte 8K de memorie de instrucțiuni nu s-a putut implementa serverul de comunicație cu clienții TCP/IP pe nucleele **PRU**, memoria de 8K fiind prea mică.

5.1.1 Implementarea Ciclului de achiziție(PRU_RPMsg_ModBusE_Master).

Pentru implementarea ciclului de achiziție se folosește nucleul **PRU0** din subsistemul **PRU-ICSS**. Ciclul de achiziție va fi implementat pe stația client (master) care va trimite mesaje către stația server (slave). Stația server preia date de la senzori sau trimite date către elementele de execuție. Recepția și transmisia mesajelor din sloturi se realizează cu viteze de până la 12 Mb/s, viteza de 12Mb/s necesită un răspuns foarte bun pentru **PRU** (durata unui caracter de 10 biți care trebuie procesat este 0.83 microsecunde (1/12 x 10 (us))).

Software-ul pentru stația client (master) se bazează pe biții de control și stare ai perifericelor utilizate, UART și timer-ul **IEP** (cu două comparatoare utilizate pentru a determina sfârșitul

unui mesaj la recepție și pentru semnalizarea sfârșitului de slot). Când se semnalizează sfârșitul de slot se încarcă registrul de comparare cu durata slot-ului următor și se comută driver-ul RS485 pe emisie, după care se începe emisia mesajului utilizând indicatorul pentru registrul de emisie gol. În tot acest timp se testează și indicatorul de depășire pentru durata slot-ului. Dacă a intervenit o depășire se semnalizează eroare de emisie mesaj slot, se comută driver-ul pe recepție și se trece la slot-ul următor. În cazul în care s-a emis tot mesajul se trece pe recepție și se așteaptă recepția primului caracter sau depășire pentru durata slot-ului, caz în care se semnalizează eroare de emisie se comută driver-ul pe recepție și se trece la slot-ul următor. Dacă s-a primit un caracter se stă pe recepție până când se indică depășire pentru sfârșit de mesaj. Și în această buclă se testează indicatorul pentru depășire slot și se iau aceleași acțiuni amintite anterior. În continuare se comută pe emisie și se așteaptă depășirea indicatorului de terminare slot, după care se reia bucla principală cu următorul slot (**sloturm = slot % NrMaxSlot**).

Software-ul pentru stația server (slave) este realizat într-o manieră asemănătoare cu diferența că la nivelul server-ului apare operația de mutare a mesajului din buffer-ul de recepție în buffer-ul aplicație.

5.1.2 Aplicația PRU1_RPMsg_ModBusE_Master.

Aplicația PRU1_RPMsg_ModBusE_Master folosește nucleul **PRU1**. Se folosește nucleul **PRU1** deoarece nucleul **PRU0** are doar 8K de memorie pentru cod și nu a fost posibil de a se îngloba și codul pentru această aplicație în nucleul **PRU0**. Această aplicație comunică cu procesorul **ARM Cortex A8** prin intermediul driver-ului RPMsg_PRU (remote procesor message). Deci un client (**ListenServer** în cazul nostru) care rulează pe procesorul **ARM Cortex A8** poate cere acestei aplicații PRU1_RPMsg_ModBusE_Master să întoarcă anumite date din memoriile de date a nucleelor **PRU**. Deoarece nu se poate folosi debuger-ul atunci când serverul de comunicație cu clienții **Modbus TCP/IP (BBB_ARM_A8_MBE_IOT_GATEWAY)** pe procesorul **ARM Cortex A8** și ciclul de achiziție pe nucleul **PRU0** rulează și comunică între ele se folosește această aplicație pentru a putea vizualiza mesaje, variabile de tip contor, variabile de tip spioni, etc. Deci această aplicație server și clientul cu care se conectează și comunică sunt unelte pe care le-am creat pentru a putea vizualiza daca datele care comunică între stația client și stația server sunt corecte și că totul funcționează așa cum trebuie.

5.1.3 Aplicația ListenServer.

Pentru a putea vizualiza (spiona) datele și mesajele din ciclul de achiziție s-a realizat o aplicație numită **ListenServer**. Această aplicație rulează pe procesorul **ARM Cortex A8** sub sistemul de operare Debian. **ListenServer** reprezintă un client pentru aplicația PRU1_RPMsg_ModBusE_Master (PRU1_RPMsg_ModBusE_Master rulează pe nucleul de timp real PRU1).

5.1.4 Serverul de comunicație (dispatcher-ul) cu clienții ModbusE TCP/IP sau server-ul OPC UA(BBB_ARM_A8_MBE_IOT_GATEWAY).

Această aplicație rulează pe procesorul ARM Cortex A8. Rolul acestui server este de a prelua comenzile ModbusE de la clienții TCP/IP sau server-ul OPC UA, de a le interpreta, de a le transmite ciclului de achiziție și de a transmite înapoi răspunsul clienților TCP/IP. La fel ca și

ciclul de achiziție sau serverul **PRU1_RPMsg_ModBusE_Master** și serverul **BBB_ARM_A8_MBE_IOT_GATEWAY** folosește memoria de date partajată de 12KDisptacher-ul BBB_ARM_A8_MBE_IOT_GATEWAY este alcătuit din următoarele fire de execuție: *ModBusE_Listening_Thread*, *ModBusE_Eth_Servers_Thread*, *ModBusEThreadDispatcherForServer*.

5.1.4.1 Thread-ul (firul de execuție) ModBusE_Listening_Thread.

Acest thread stă în buclă unde creează socket, leagă socket-ul creat de port (în cazul ModbusE este vorba de portul 502) și așteaptă o cerere de la un client ModbusE , acceptă cererea după care salvează socket-ul într-un buffer de socket-uri. Pentru socket-uri este definit atât buffer-ul de socket-uri ***sckMbeTcpIp[i]***, cu ***i < BBB_MBE_MAX_SKT – 1,*** unde ***BBB_MBE_MAX_SKT* = 5** cât și și structura socket ***BBB_MBE_TCP_SEND_BUFFER*** (se instanțiază: *sb[BBB_MBE_MAX_SKT]*) ce va conține starea (de lucru) a socket-ului (*sb[i].stateSocket = -1*, tip dată S32) care în momentul este creat valoarea este -1 adică starea IDLE (0xFFFFFFFF).

Buffer de socket-uri va conține rezultatul funcției accept. Rezultatul funcției accept este salvată în buffer-ul de socket-uri în ordine crescătoare, adică dacă ***sckMbeTcpIp[0]*** are socket activ și ***sckMbeTcpIp[1]*** nu are socket activ, atunci rezultatul va fi salvat în ***sckMbeTcpIp[1].*** Mai departe, noul socket activ va fi preluat preluat, în ordinea importanței, de firele de execuție *ModBusE_Eth_Servers_Thread și și ModBusEThreadDispatcherForServer*.

5.1.4.2 Thread-ul (firul de execuție) ModBusE_Eth_Servers_Thread

Acest thread execută într-o buclă infinită două bucle ciclice for după ce se scurge un interval de timp. Prima buclă **for** verifică dacă s-au recepționat mesaje pentru socket-urile active (***sckMbTcpIp[i] != -1***) și care nu sunt prinse în alte tranzacții (***sb[i].stateSocket* = -1**). Dacă s-a recepționat mesaj se trece la analiza mesajului. Dacă mesajul recepționat este un mesaj Modbus TCP/IP atunci starea de lucru a socket-ului trece în starea ***sb[i].stateSocket = i+IN_EXECUTION***, socket-ul cu această stare va fi preluat de firul de execuție *ModBusEThreadDispatcherForServer*. A doua bucla **for** verifică dacă sunt socket-uri active (***sckMbTcpIp[i] != -1***) care au de transmis mesaje de tip răspuns (***sb[i].stateSocket = i + EXECUTED***). Dacă există se trece la emiterea acestor mesaje de tip răspuns.

5.1.4.3 Thread-ul (firul de execuție) ModBusEThreadDispatcherForServer

Acest fir de execuție (thread-ul dispatcher) analizează în mod periodic socket-urile care au starea (status-ul) **IN_EXECUTION** și **IN_DISPATCHER**. Atunci când socket-ul este în starea **IN_EXECUTION** firul de execuție schimbă starea socket-ului în **IN_DISPATCHER** atunci când se primește semnal de la ciclul de achiziție sau când expiră time-out-ul. Apoi din starea **IN_DISPATCHER** socket-ul este trecut în starea **EXECUTED** , mai departe socket-ul cu starea **EXECUTED** este preluat de task-ul **ModBusE_Eth_Servers_Thread.** Task-ul **ModBusEThreadDispatcherForServer** are rolul de a analiza cererile lansate de către firul de execuție **ModBusE_Eth_Servers_Thread** pentru slot-urile care sunt în ciclu - ***IN_CYCLE***, și de asemenea pentru slot-urile care sunt în afara ciclului - ***OUT_OF_CYCLE*** și de a trimite informațiile necesare pentru execuție, de a analiza răspunsurile venite de la ciclul de achiziție (CA) și de a le trimite mai departe către task-ul server **ModBusE_Eth_Servers_Thread**, și de a trata diferențiat comenzile pentru slot-urile ***IN_CYCLE*** față de slot-urile ***OUT_OF_CYCLE***

asincrone, de a pregăti cererile pentru execuție (mutarea datelor, calculul CRC, lungimile de emisie și recepție, etc.), de a realiza excluderea mutuală pentru accesul datelor la buffer-ele sloturilor și a socket-urilor fără a folosi susținerea SOTR, de a pregăti răspunsurile (mutarea datelor, calculul CRC, lungimile de emisie și recepție, etc.), de a gestiona stările celor două tipuri de slot-uri pe care le poate trata la un moment dat (***a*** pentru ***SLOT-URI ÎN_CYCLE***, și ***b*** pentru slot-uri ***OUT_OF_CYCLE***) și transferul datelor între buffer-ele asociate slot-urilor și cele asociate socket-urilor.

Slot-urile ***OUT_OF_CYCLE*** cu indirectare < *0x80* suportă ca și slot-urile ***IN_CYCLE*** doar funcțiile Modbus (3, 16 și 23). Sloturile ***OUT_OF_CYCLE >= 0x80*** sunt considerate comenzi Modbus clasice care implementează majoritatea funcțiilor Modbus.

5.2 Aplicație client ModbusE TCP/IP(MBE_IOT_GATEWAY_BBB_CLIENT)

După cum se poate observa și din Figura 4-1 la modulul IIoT ModbusE Gateway se pot conecta clienți ModbusE TCP/IP. Acești clienți trebuie să implementeze protocolul Modbus. Un exemplu de client Modbus TCP/IP este clientul OPC UA. Pentru acest referat pentru a putea testa gateway-ul IoT ModbusE am creat un client ModbusE TCP/IP utilizând limbajul de programare C# și mediul de dezvoltare integrat (IDE) Microsoft Visual Studio.

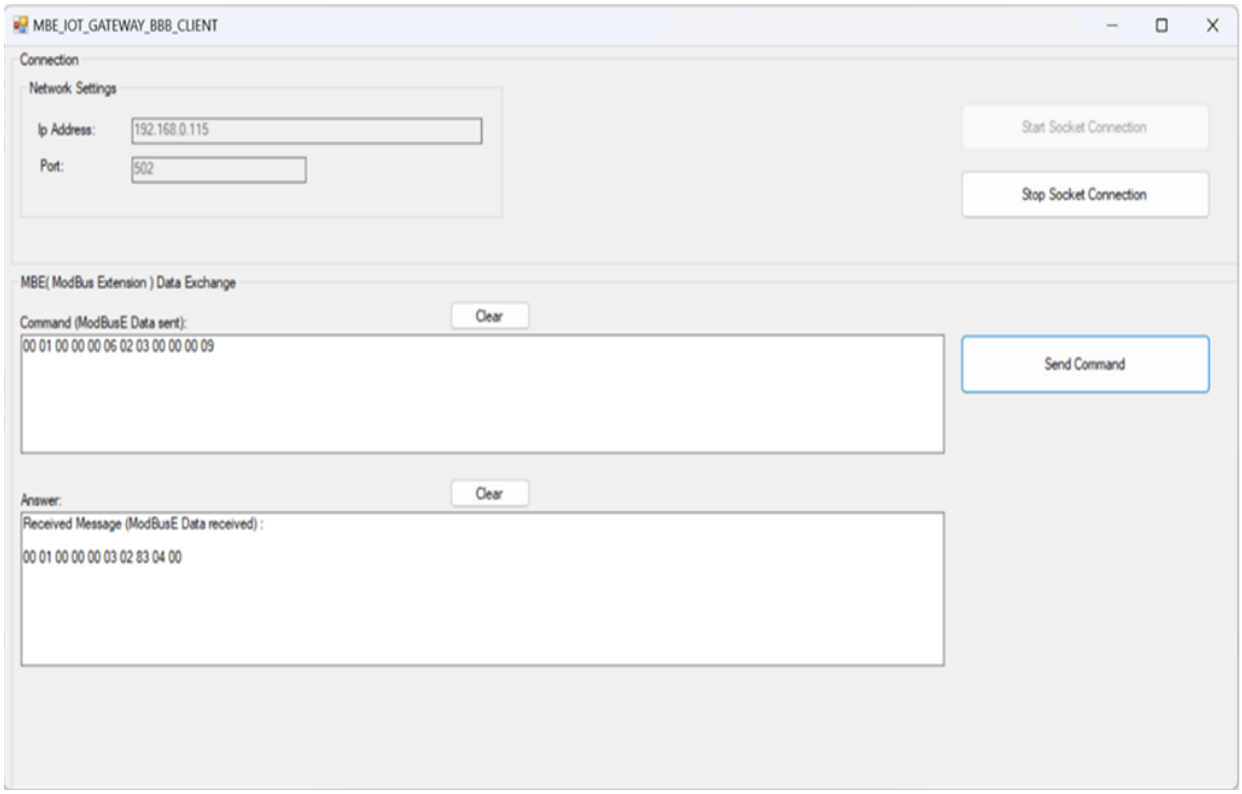


Figura 5-1 Client-ul ModbusE TCP/IP MBE_IOT_GATEWAY_BBB_CLIENT.

În Figura 5-1 este prezentat clientul Modbus TCP/IP ***MBE_IOT_GATEWAY_BBB_CLIENT***. Pentru ca acest client să se poată conecta la modulul **IoT ModbusE** Gateway trebuie să se încarce și să se lanseze în execuție serverul de comunicație pe procesorul **ARM Cortex A8** și

ciclul de achiziție pe procesorul **PRU0**. Lansarea în execuție se realizează prin linia de comandă.

5.3 Server și client OPC UA.

După cum se poate observa și din Figura 4-1 pe procesorul ARM Cortex A8 poate rula serverul OPC UA. Pentru aceasta am portat **open62541** pe procesorul ARM Cortex A8. **open62541** este open source și reprezintă o implementare free a protocolului OPC UA scris în limbajul de programare C++. Serverul OPC UA care rulează pe procesorul ARM Cortex A8 se numește ***ServerOPCUAModBusE.*** Aplicația ***ServerOPCUAModBusE*** este folosită ca aplicație client pentru comunicația cu serverul de comunicație ***BBB_ARM_A8_MBE_IOT_GATEWAY***. Comunicația se realizează prin intermediul socket-urilor. Aplicația ***ServerOPCUAModBusE*** este folosită ca aplicație server pentru comunicația cu clientul **OPC UA**. Deci aplicația ***ServerOPCUAModBusE*** preia comenzi de la clientul OPC UA și le trimite mai departe prin intermediul socket-urilor server-ului (dispatcher-ului) ***BBB_ARM_A8_MBE_IOT_GATEWAY***. Aplicația ***BBB_ARM_A8_MBE_IOT_GATEWAY*** trimite mai departe comenzile ciclului de achiziție preia răspunsul de la ciclul de achiziție și îl trimite clientului OPC UA prin intermediul ***ServerOPCUAModBusE.*** Clientul **OPC UA** folosit este **UaExpert** de la Unified Automation și rulează pe un PC care folosește sistemul de operare Windows. În spațiul de adrese ale serverului ***ServerOPCUAModBusE*** am introdus două variabile de tip String pentru comanda ModbusE și pentru răspunsul ModbusE.

În Figura 5-2 este prezentat clientul UaExpert OPC UA cu cele două variabile de tip string pentru comandă și răspuns și anume “Command ModBusE” și respective “Response ModBusE”.

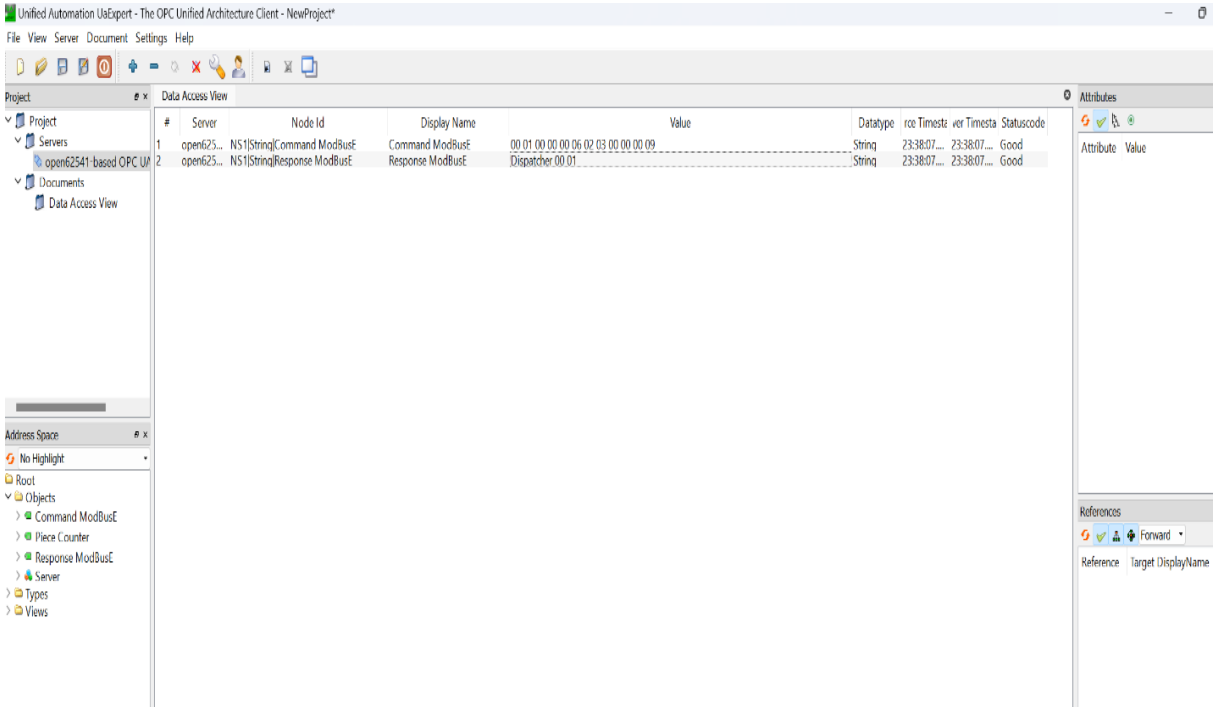


Figura 5-2 Clientul UaExpert OPC UA. [42]

6 Analiza performanțelor ModbusE

6.1 Performanța PRU cu ciclu de achiziție

După cum este prezentat și în Capitolul II al acestei lucrări, nucleul PRU0 de pe procesorul Sitara AM335x a fost ales pentru a implementa ciclul de achiziție (CA) [43], [44]. Acest nucleu este o unitate programabilă în timp real (PRU) și funcționează la o frecvență de până la 200 MHz. Comunicația dintre stația client (master) și stațiile slave (server) se face prin portul serial PRU UART. Portul PRU UART funcționează la viteze de până la 12 Mbps, astfel încât durata unui caracter de 10 biți de procesat este de minim 0,83 microsecunde. Măsurătorile arată că durata unui bit la 12 Mbps este de aproximativ 83,19 ns, iar perioada unui ciclu de achiziție cu 10 slot-uri este de 1,349 ms (Figura 6-1). Deci, rezultă că numărul maxim posibil de biți pe ciclu de achiziție este 16210 (1349µs / 0,08319µs). Un număr de 1621 de cadre pe 10 biți pot fi transmise în condiții de transmisie continuă pe ciclu de achiziție cu 10 slot-uri. Dar, pentru exemplul ales, au fost transmise 1137 de cadre (3 + 4 + 20 + 40 + 80 + 96 + 510 + 128 + 128 +128 – vezi secțiunea 6.2 pentru mai multe detalii) pe ciclu de achiziție cu 10 slot-uri în sistemul experimental. Astfel pentru exemplul prezentat (adresă de start, biți de start și stop, sumă de control CRC) se obține un grad de utilizare pentru CA de 70,1% (11370/16210 = 0,701). Scăzând caracterele adresei de start și caracterele sumei de control CRC din numărul de cadre efectiv trimise pe ciclu de achiziție (cu 10 slot-uri transmise și 8 slot-uri recepționate, slot-ul 0 și slot-ul 1 nu recepționează răspunsuri) obținem un total de 1082 cadre pe ciclu de achiziție, și astfel un mesaj util de 66,7 % (1082/1621 = 0,667). Scăzând biții de start și stop se obține o sarcină utilă de 53,3% ((1082*8) / (1621*10) = 0,533).

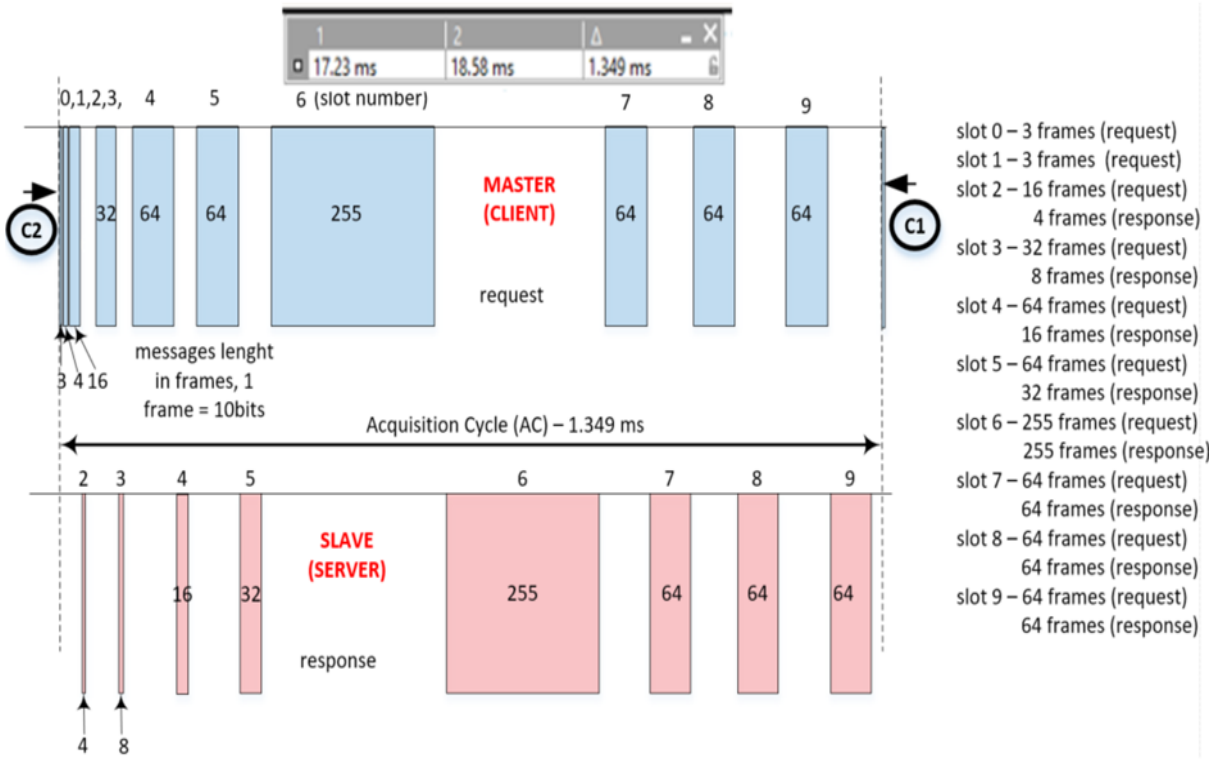


Figura 6-1 Perioada unui ciclu de achiziție (CA) constând din 10 slot-uri [45]

Aplicația software care implementează ciclul de achiziție pe nucleul PRU0 (al subsistemului PRU-ICSS) pentru stația master (client) utilizează biții de stare și control ai porturilor UART (Universal Asynchronous Receiver/Transmitter) și IEP (Industrial Ethernet Peripheral). Portul IEP de tip timer este utilizat cu două comparatoare: un comparator este utilizat pentru a determina sfârșitul unui mesaj la recepție, al doilea comparator este utilizat pentru a semnala sfârșitul duratei unui slot. Când se semnalează sfârșitul duratei slot-ului, registrul comparator este încărcat cu durata următorului slot și driverul RS485 este comutat pe transmisie, după care transmisia mesajului este începută utilizând flag-ul că registrul de transmisie poate transmite următorul caracter. Pe durata noului slot este testat și indicatorul de depășire pentru slot-ul curent. Dacă a avut loc o depășire, este semnalată o eroare de transmitere a mesajului pe respectivul slot, și driverul este comutat pe recepție și se trece la următorul slot. Dacă întregul mesaj a fost trimis, se comută pe recepție și se așteaptă primul caracter recepționat sau poate să se genereze o eroare de depășire a duratei intervalului slot-ului, caz în care este semnalată o eroare de transmisie, și driver-ul este comutat pe recepție și se trece la următorul slot. Dacă a fost primit un caracter, se rămâne pe recepție până când se indică depășirea pentru sfârșitul mesajului. De asemenea, în această buclă, indicatorul de depășire a duratei slot-ului este testat și sunt luate aceleași acțiuni ca mai sus. Apoi se comută driverul pe emisie și se așteaptă setarea indicatorului de sfârșit al duratei slot-ului, după care se reia bucla principală cu următorul slot (slot_next = slot % NrMaxSlot). În Figura 6-2 se arată bucla software folosind porturile UART și IEP pentru a transmite un mesaj către stația server în codul aplicației de pe stația client. Astfel, pe linia de cod 399, caracterul de transmis este depus în registrul THR al portului UART, iar pe linia de cod numărul 400, așteaptă până când caracterul este transmis. Linia de cod 403 verifică existența unei întreruperi de la portul IEP, iar linia de cod 406 verifică timeout-ul slot-ului (adică, dacă timpul alocat slot-ului curent este depășit) [45].

```
393       int m;
394       m = 0;
395       //lMess - the lenght of message
396       while( m < lMess )
397       {
398           // Transmit message section
399           CT_UART.THR = *pmSlot++;
400           while (!CT_UART.LSR_bit.TEMT);
401
402           // Verify that the IEP is the source of the interrupt
403           if( CT_INTC.SECR0 & 0x80)
404           {
405               CT_INTC.SECR0  = 0x80;
406               if( CT_IEP.TMR_CMP_STS_bit.CMP_HIT & 0x01)   //If it is Timeout slot
407               {
408
409                   CT_IEP.TMR_CMP_STS_bit.CMP_HIT  = 0x01;
410                   statusSlot = STATUS_SLOT_TIMER_OVERFLOW;
411                   break;
412               }
413
414
415
416           }
417
418           m++;
419       }
420
421
```

Figura 6-2 Utilizarea portului UART și IEP la transmiterea caracterelor pe stația client [45]

În Figura 6-3 este prezentată utilizarea porturilor UART și IEP pentru primirea unui mesaj de la stația server în codul de pe stația client. Astfel, pe linia de cod 466, se verifică în buclă dacă a fost primit primul caracter al mesajului, iar pe linia de cod 536, se verifică în buclă dacă au fost primite următoarele caractere. La linia de cod 570, se verifică dacă a apărut intreruperea pentru intervalul de 3,5 caractere, iar la linia de cod 574, se verifică dacă întregul mesaj a fost primit, adică dacă CRC partea low și partea high sunt pe 0x00. La linia de cod 591, se verifică dacă a apărut intreruperea de timeout pentru slot, ceea ce înseamnă că timpul alocat slotului curent a fost depășit [45].

```
462   while(1)
463   {
464        /* Wait for first character */
465        //UART RX
466       if( CT_UART.IIR & 0x4 )
467       {
468           recvChar = ReadMessageIn();
469           statusSlot = STATUS_SLOT_RX;
470           break;
471       }
472
473
474
475       // Verify that the IEP is the
476       //source of the interrupt
477       if( CT_INTC.SECR0 & 0x80)
478       {
479            CT_INTC.SECR0  = 0x80;
480            //Timeout slot
481            if( CT_IEP.TMR_CMP_STS_bit.CMP_HIT & 0x01)
482            {
483                CT_IEP.TMR_CMP_STS_bit.CMP_HIT  = 0x01;
484                statusSlot = STATUS_SLOT_TIMER_OVERFLOW;
485                cntT1++;
486                break;
487            }
488       }
532   while(1)
533   {
534       /* take the rest of message */
535       //UART RX
536      if( CT_UART.IIR & 0x4 )
537      {
538          spycnt = 4;
539          recvChar = ReadMessageIn();
540          /* compute CRC */
541          crcIndex = crcLo ^ recvChar;
542          crcLo = ( U8 )( crcHi ^ aucCRCHi[crcIndex] );
543          crcHi = aucCRCLo[crcIndex];
544          /* put the char into receive buffer */
545          *pmSlot++ = recvChar;
546          n++;
547          if( n > lMess)
548          {
549              if( ( workSlot != 0 ) && ( workSlot != 1) )
550              {
551                  cfsp1[workSlot].ErrorFieldbusRecv++;
552                  statusSlot = STATUS_SLOT_TIMER_OVERFLOW;
553                  break;
554              }
555          }
556          //refresh 3.5 characer timer
557          CT_IEP.TMR_CMP1 =  (CT_IEP.TMR_CNT + 600);
558          CT_IEP.TMR_CMP_STS_bit.CMP_HIT  = 0x02;
559          continue;
560       }
```

```
562       //UART timeout 4 characters
563       /* Verify that the IEP is the source of the interrupt I.E Timeout slot*/
564       if( CT_INTC.SECR0 & 0x80)
565       {
566
567            CT_INTC.SECR0 = 0x80;
568            spycnt = 5;
569            //3,5 character timer
570            if( CT_IEP.TMR_CMP_STS_bit.CMP_HIT & 0x02)
571            {
572               CT_IEP.TMR_CMP_STS_bit.CMP_HIT  = 0x02;
573               statusSlot = STATUS_SLOT_IDLE;
574               if( ( crcHi == 0 ) && ( crcLo == 0 ) )
575               {
576                   cfsp1[workSlot].RecvMessages++;
577                   statusSlot = STATUS_SLOT_IDLE;
578                   spycnt = 6;
579                   break;
580               }
581               else
582               {
583                   if( ( workSlot != 0 ) && ( workSlot != 1 ) )
584                       cfsp1[workSlot].ErrorFieldbusRecv++;
585                   statusSlot = STATUS_SLOT_TIMER_OVERFLOW;
586                   spycnt = 7;
587                   break;
588               }
589            }
590            //Timeout slot
591            if( CT_IEP.TMR_CMP_STS_bit.CMP_HIT & 0x01)
592            {
593                spycnt = 8;
594                CT_IEP.TMR_CMP_STS_bit.CMP_HIT  = 0x01;
595               statusSlot = STATUS_SLOT_TIMER_OVERFLOW;
596               cntT1++;
597               if ((workSlot!=0)&&(workSlot!=1))
598                   cfsp1[workSlot].ErrorFieldbusRecv++;
599               break;
600            }
601    }
602 }
```

Figura 6-3 Utilizarea portului UART și IEP la recepționarea caracterelor pe stația client [45]

Software-ul pentru stația server (slave) este implementat într-un mod similar cu diferența că la nivelul serverului (slave-ului) are loc operația de mutare a mesajului din buffer-ul de recepție în buffer-ul aplicației. În Figura 6-4 sunt prezentați timpii obținuți prin măsurarea cu osciloscopul pentru transmisia slot-urilor de la staţia client la staţia server cu 3, 4 şi 16 caractere. Astfel, pentru transmiterea slot-ului de 3 caractere s-a obţinut (măsurat) un timp de

3,027 µs; pentru transmiterea slot-ului de 4 caractere s-a obţinut un timp de 3,784 µs; pentru transmiterea slot-urilor de 16 caractere s-a obţinut un timp de 16,65 µs.

De asemenea, pentru transmiterea slot-urilor de 32 de caractere s-a obţinut un timp de 33,62 µs, pentru transmiterea slot-urilor de 64 de caractere s-a obţinut un timp de 67,18 µs iar pentru transmiterea slot-urilor de 255 de caractere a fost obținut un timp de 268,5 µs.

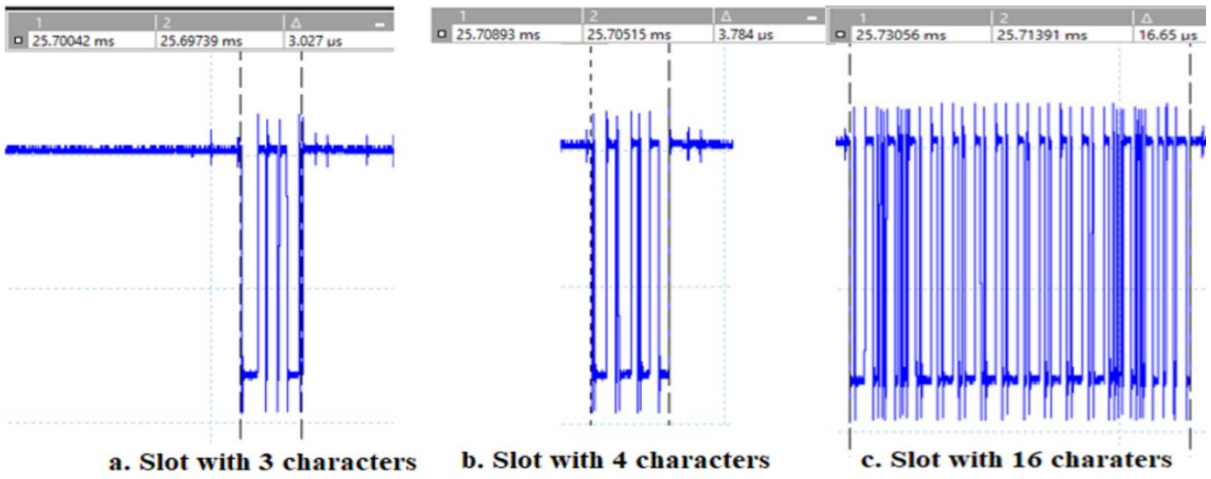


Figura 6-4 Durata transmisiei pentru slot-uri cu 3, 4, 16 de caractere pentru transmisie [45]

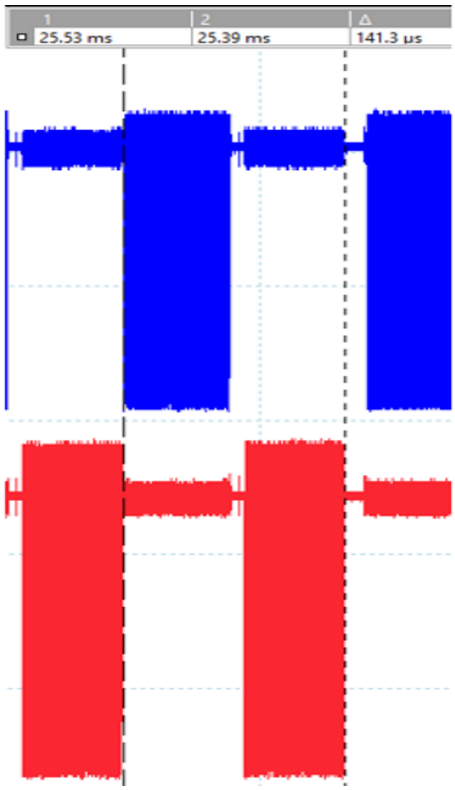


Figura 6-5 Exemplu de măsurare a duratei unui slot care conține transmiterea mesajului de către client ca cerere, transmiterea mesajului de către server ca răspuns, analiza timpului și pregătirea răspunsului către server și tAdjust [45]

În Figura 6-5 se prezintă intervalele de timp măsurate cu osciloscopul pentru transmiterea și recepția caracterelor slot-ului. Astfel, pentru un slot care are 255 de caractere de transmis și 255 de caractere de primit, intervalul de timp este de 546,9 μs. Pentru un slot care are 64 de caractere de transmis și 64 de caractere de recepționat, intervalul de timp este de 141,3 μs. Pentru un slot care are 64 de caractere de transmis și 32 de caractere de recepționat, intervalul de timp este de 109 μs. Pentru un slot care are 64 de caractere de transmis și 16 caractere de recepționat, intervalul de timp este de 93,34 μs. Pentru un slot care are 32 de caractere de transmis și 8 caractere de recepționat, intervalul de timp este de 48,94 μs. Pentru un slot care are 16 caractere de transmis și 4 caractere de recepționat, intervalul de timp este de 26,49 μs. Datorită naturii deterministe a ciclului de achiziție (CA), valorile citite sunt întotdeauna aceleași (plus, minus), și nu s-au obținut diferențe semnificative între acești timpi [45].

La testarea pe sistemul experimental cu un ciclu de achiziție cu 10 slot-uri, duratele de timp pentru slot-uri au fost calculate matematic. Dar cu aceste valori pentru duratele slot-urilor, ciclul de achiziție nu a început să funcționeze corect. Pe lanțul de măsurare pe care l-am folosit empiric, am constatat că modelul matematic necesita un timp de ajustare (tAdjust) dat de anumite întârzieri ale microcontrolerului pe care nu le-am putut vizualiza. Apoi, pentru diferite valori ale lui tAdjust, am obținut un comportament determinist pentru ciclul de achiziție. tAdjust a fost determinat prin valori experimentale până când ciclul de achiziție a funcționat corect. Pentru slot-urile 0 și 1, tAdjust = 4 și pentru celelalte slot-uri tAdjus = 20 (determinare empirică). Acest tAdjust nu depinde de numărul de noduri. În cazul altor platforme, tAdjust trebuie determinat din nou. Există un algoritm prin care tAdjust poate fi determinat automat, dar din cauza memoriei reduse din PRU, acesta nu a putut fi implementat pe această platformă.

În viitor, se dorește crearea de software pentru generarea automată a acestor timpi. Acest tAdjust a realizat ajustarea ciclului de achiziție astfel încât să nu provoace erori. Perioada de timp dintre transmisie și recepție pentru un slot cu 510 caractere (Figura 6-6), de exemplu, depinde de timpii de procesare ai stațiilor server, cum ar fi: timpul de calcul al sumei CRC pentru a verifica corectitudinea mesajului primit, timpul pentru mutarea mesajului din buffer-ul de recepție în buffer-ul firului de execuție dacă cererea primită este de a scrie către server, timpul necesar formării răspunsului dacă cererea primită este de citit de la stația server. Acești timpi sunt mai greu de estimat și sunt preluați în tAdjust. Când stația client primește răspunsul, apare un timp pentru calculul sumei CRC pentru a verifica integritatea mesajului primit de la stația server. În Figura 6-6 semnalul de pe osciloscop de culoare roșie reprezintă indicatorul pentru direcția mesajului (transmisie sau recepție), semnalul de culoare albastru reprezintă mesajul transmis de pe stația client către stația server, semnalul de culoare verde reprezină mesajul transmis de pe stația server către stația client.

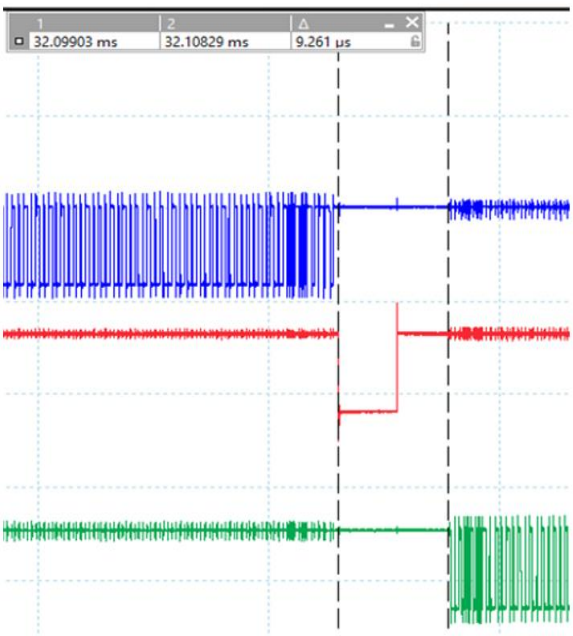


Figura 6-6 Perioada de timp dintre transmisie și recepție pentru un interval de 510 de caractere [45]

6.2 Modelarea canalului de comunicație a ciclului de achiziție (CA)

Modelarea canalului de comunicație a ciclului de achiziție Ciclul de achiziție din cadrul extensiei ModbusE constă din slot-uri. Astfel slot-urile au asociate câte o structură de date ce conține informații despre control, stare și date. După cum s-a mai menționat în lucrare, minimul de slot-uri ale unui ciclu de achiziție este de 3. Extensia ModbusE utilizează după cum s-a menționat în lucrare 2 tipuri de obiecte: PDO utilizat pentru transferul datelor de proces și SDO utilizat pentru configurare, întreținere și testare [32]. Aceste obiecte (PDO și SDO) pot fi transportate prin rețea folosind mesaje (mai multe mesaje alcătuiesc tranzacții). Extensia ModbusE folosește trei tipuri de mesaje: SDA - Send Data with Acknowledge, SDN-Send Data with No Acknowledge și SRD - Send and Request Data [32]. Un cadru Modbus RTU/MBE pentru date de 8 biți are 11 (1 bit de start, 8 biți de date, 2 biți de stop) biți [39]. Un mesaj Modbus RTU are maximum 256 de octeți.

Un cadru este format din [46]:

Ecuația 6-1

1f = 1 bit de start + 8 biți de date + 1 bit de stop = 2HT + 8 biți de date = 10 biți

Unde: f – frame(cadru), HT - Header Trailer(Header - adresa slot-ului , Trailer - CRC). Un slot este format din:

Ecuația 6-2

1 slot = 1 mesaj de scriere + 1 mesaj de citit.

Ecuația 6-3

1mesaj = 1frameheader (adresă slot) + ni f(data) + 2f(CRC) = 3f(HT) + ni f(data)

Folosim un ciclu de achiziție cu 10 slot-uri.

Ecuația 6-4

Slot 0 = 1 mesaj de scriere = 3f(HT)

Ecuația 6-5

Slot 1 = 1 mesaj de scriere = 3f(HT) + 1 caracter de control

Ecuația 6-6

Slot 2 = 1 mesaj de scriere + 1 mesaj de citit = (3f(HT) + 13f(data)) + (3f(HT) + 1f(data))

Ecuația 6-7

Slot 3 = 1 mesaj de scriere + 1 mesaj de citit = (3f(HT) + 29f(data)) + (3f(HT) + 5f(data))

Ecuația 6-8

Slot 4 = 1 mesaj de scriere + 1 mesaj de citit = (3f(HT) + 61f(data)) + (3f(HT) + 13f(data))

Ecuația 6-9

Slot 5 = 1 mesaj de scriere + 1 mesaj de citit = (3f(HT) + 61f(data)) + (3f(HT) + 29f(data))

Ecuația 6-10

Slot 6 = 1 mesaj de scriere + 1 mesaj de citit = (3f(HT) + 252f(data)) +(3f(HT) +252f(data))

Ecuația 6-11

Slot 7 = 1 mesaj de scriere + 1 mesaj de citit = (3f(HT) + 61f(data)) + (3f(HT) + 61f(data))

Ecuația 6-12

Slot 8 = 1 mesaj de scriere + 1 mesaj de citit = (3f(HT) + 61f(data)) + (3f(HT) + 61f(data))

Ecuația 6-13

Slot 9 = 1 mesaj de scriere + 1 mesaj de citit = (3f(HT) + 61f(data)) + (3f(HT) + 61f(data))

NfAC reprezintă numărul de cadre pe ciclu de achiziție (CA).

Ecuația 6-14

NfAC = 3f + 4f + 20f + 40f + 80f + 96f + 510f + 128f + 128f + 128f = 1137f

Max -> 11370 biți/CA (10 biți / caracter)

1bit la 12 Mb/s - >83,19 ns (măsurat pe osciloscop), din calcul 83,33 ns

Perioada unui ciclu de achiziție cu 10 slot-uri este de 1.349 ms

Ecuația 6-15

1349 us/0,08319 us = 16210 b/CA, care pot fi trimiși pe durata CA

Din împărțirea numărului de biți trimiși pe CA și numărul total posibili de trimis pe CA

Ecuația 6-16

11370 / 16210 = 0,701

rezultă un mesaj de 70,1% (cu adresă slot, cu biți de start și stop și cu suma de control CRC).

Dacă numărul de cadre care trebuie transmise pe ciclu de achiziție de 10 slot-uri este redus cu caracterele pentru adresa slot-ului și suma de control

Ecuația 6-17

1137 - 55 = 1082

Unde: 55 reprezintă caracterele pentru adresa de slot și suma de control CRC pentru 10 slot-uri transmise și 8 slot-uri recepționate pe ciclu de achiziție plus 1 caracter de control pentru slot1.

Din împărțire

Ecuația 6-18

1082 / 1621 = 0,667

rezultă un mesaj util de 66,7%.

Din formula

Ecuația 6-19

(1082*8) / (1621*10) = 0,533 (biții de start și stop au fost eliminați)

rezultă o sarcină utilă de date de 53,3% (fără adresă de pornire, fără sumă de control CRC, fără biți de pornire și oprire și fără caractere care nu sunt date utile).

6.3 Fluxul de mesaje de comunicație între ciclul de achiziție (pe PRU0), dispathcer (pe ARM Cortex A8 cu Linux), server OPC UA (pe ARM Cortex A8 cu Linux), client OPC UA (pe Windows).

După cum se poate vedea în Figura 4-1, pe procesorul ARM Cortex A8 poate rula serverul OPC UA. Pentru aceasta, am portat biblioteca open62541 pe procesorul ARM Cortex A8. Biblioteca open62541 este open source și este o implementare gratuită a protocolului OPC UA scris în limbaj de programare C++. Serverul OPC UA care rulează pe procesorul ARM Cortex A8 se numește ServerOPCUAModBusE. Aplicația ServerOPCUAModBusE se folosește la rândul ei ca și client pentru comunicația cu dispatcher-ul (BBB_ARM_A8_MBE_IOT_GATEWAY). Comunicația se realizează prin intermediul socket-urilor TCP/IP. Aplicația ServerOPCUAModBusE este utilizată ca aplicație server pentru comunicația cu clientul OPC UA (clientul OPC_UA care rulează pe un computer folosind Windows ca sistem de operare). Figura 6-7 prezintă fluxul de mesaje de comunicație între clientul OPC UA/ModbusE (care rulează pe Windows),serverul OPC UA (care rulează pe ARM Cortex A8) , dispatcher-ul (care rulează pe ARM Cortex A8),și ciclul de achiziție (care rulează pe PRU0). Prin urmare, clientul OPC UA care rulează pe un PC trimite comenzi ModbusE prin socket-uri TCP/IP către serverul OPC UA. Serverul OPC UA trimite comenzile

MosbusE către dispatcher tot prin intermediul socket-urilor TCP/IP. Rolul dispatcher-ului este de a interpreta și media comenzile MobusE (mesaje) între serverul OPC UA (sau clienții ModubusE) și ciclul de achiziție. Apoi, dispatcher-ul trimite comenzi ModbusE ciclului de achiziție (CA) prin intermediul memoriei partajate (shared memory). Ciclul de achiziție (CA) dezactivează slotul pentru care a fost primită comanda (astfel încât ciclul de achiziție să nu suprascrie zona de memorie pentru răspunsul la comandă, simulare mutex). Ciclul de achiziție notifică (prin intermediul unui flag) dispatcher-ul că, comanda a fost primită și răspunde la comanda ModbusE. Când dispatcher-ul primește răspunsul pentru comanda ModbusE de la ciclul de achizitie, activează slotul pentru care a primit răspunsul și trimite apoi acest răspuns fie către clienții ModbusE (dacă au trimis o comandă), fie către serverul OPC UA. Serverul OPC UA trimite răspunsul către clientul OPC UA care rulează pe un PC. Clientul OPC UA utilizat este **UaExpert** de la Unified Automation și rulează pe un sistem de operare Windows, așa cum s-a menționat mai sus. În spațiul de adrese a serverului ServerOPCUAModBusE, am introdus două variabile de tip String pentru comanda și răspunsul ModBusE.

În Figura 6-7 se prezintă următorii timpi:

• tSOPCUAR_DTOUT—timpul dintre solicitarea serverului OPC UA și timeout-ul pe dispacher.

• tDTOUT_DR — timpul dintre timeout-ul pe dispatcher și solicitarea dispatcher-ului către ciclul de achiziție.

• tDR_CA—timpul dintre solicitarea dispatcher-ului și recepția comenzii în ciclului de achiziție.

• tCA_DA—timpul dintre răspunsul ciclului de achiziție și primirea răspunsului la comandă de către dispatcher.

• tDA_tSOPCUAA—timpul între trimiterea de către dispatcher a răspunsului la comanda (clientului OPC UA) și recepția răspunsului de către serverul OPC UA.

• tSOPCUAR_SOPCUAA—timpul dintre cererea serverului OPC UA și recepția răspunsului de către serverul OPC UA.

• tSOPCUAR_SOPCUAR—timpul dintre două cereri consecutive de server OPC UA.

Clientul OPC UA rulează pe un computer cu Windows ca sistem de operare. În clientul OPC UA s-au definit două obiecte de tip String, după cum urmează:

• Comanda ModBusE, unde utilizatorul introduce comanda ModbusE și apasă enter.

• Response ModBusE, unde utilizatorul primește răspunsul la comandă.

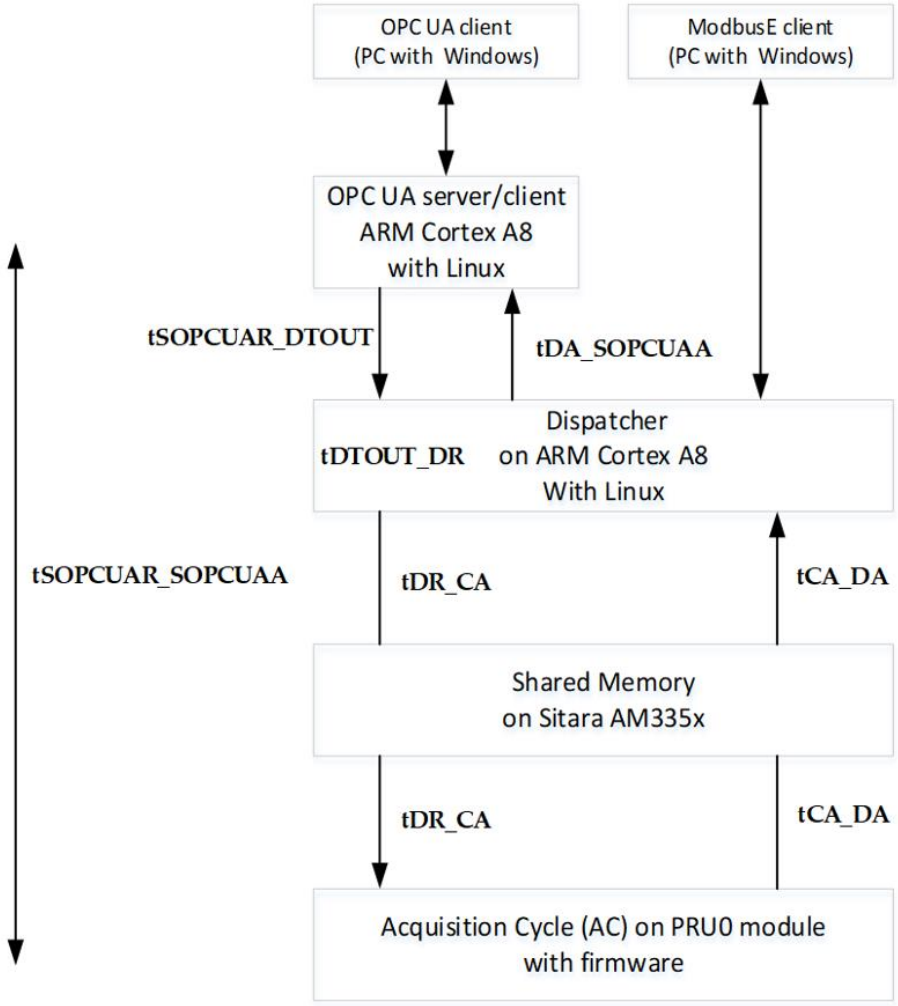


Figura 6-7 Comunicația fluxului de mesaje între clientul OPC UA/ ModbusE, dispatcher și ciclul de achiziție (CA) [46].

6.4 Analiza performanței comunicațiilor de date gateway IIoT, server OPC UA

În acest subcapitol începem cu analiza timpului necesar pentru ca informația cerută de serverul OPC UA (ServerOPCUAModBusE) să parcurgă următoarea cale: server OPC UA (ServerOPCUAModBusE)—dispatcher (BBB_ARM_A8_MBE_IOT_GATEWAY ciclu de achiziție (CA)—dispatcher—server OPC UA. În Figura 6-8, spike-urile numerotate reprezintă următoarele: 1 (culoare muștar) reprezintă lansarea comenzii de către serverul OPC UA care rulează pe procesorul ARM Cortex A8; 2 (culoare muștar) reprezintă răspunsul comenzii primite de serverul OPC UA ; 3 (culoare roșie) reprezintă momentul în care dispatcher-ul care rulează pe procesorul ARM Cortex A8 testează dacă există o comandă de la serverul OPC UA; 4 (culoare verde) reprezintă momentul în care dispatcher-ul trimite comanda la ciclul de achiziție (CA — care rulează pe nucleul PRU0); 5 culoare verde reprezintă momentul în care dispatcher-ul (după ce ciclul de achiziție a executat comanda recepționată) a terminat operația de citire sau scriere în zona de date partajată și a reactivat slot-ul din care comanda a fost trimisă; iar 6 (culoare albastru) reprezintă momentul în care cererea de la dispatcher este

preluată de către ciclul de achiziție (CA) la sfârșitul analizei slot-ului curent, slot-ul pentru care a fost trimisă cererea este dezactivat. Timpul tSOPCUAR_DTOUT (între frontul crescător al spike-ului 1 și frontul crescător al spike-ului 3) reprezintă timpul dintre momentul în care serverul OPC UA (ServerOPCUAModBusE — care rulează pe procesorul ARM Cortex A8) a trimis cererea către dispatcher (BBB_ARM_A8_MBE_IOT_GATEWAY — rulând pe procesorul ARM Cortex A8) și momentul în care dispatcher-ul testează dacă este o cerere de la serverul OPC UA. Timpul tDTOUT_DR(request) [între frontul descrescător al spike-ului 3 și frontul descrescător al spike-ului 4] reprezintă timpul între momentul în care dispatcher-ul a testat dacă a existat o cerere de la serverul OPC UA și timpul în care a trimis cererea mai departe către ciclul de achiziție. Timpul tDR(request)_CA (între frontul descrescător al spike-ului 4 și frontul crescător al spike-ului 6) reprezintă timpul între momentul în care dispatcher-ul a trimis cererea către ciclul de achiziție (CA) și momentul când ciclul de achiziție a preluat cererea de la dispatcher. Timpul tCA_DA(answer) [între frontul descrescător al spike-ului 6 și frontul crescător al spike-ului 5] reprezintă timpul dintre momentul în care ciclul de achiziție a preluat cererea de la dispatcher, a dezactivat slot-ul pentru care a venit cererea, a executat cererea și momentul în care dispatcher-ul a finalizat operația de citire sau scriere în zona de date partajate (datele slot-ului din ciclul de achiziție) și a reactivat slot-ul de la care solicitarea a fost trimisă. Timpul tDA_SOPCUAA (între frontul descrescător al spike-ului 5 și frontul crescător al spike-ului 2) este timpul între momentul când dispatcher-ul, după ce a pregătit răspunsul la cerere, a trimis răspunsul către serverul OPC UA și timpul când serverul OPC UA a preluat răspunsul de la dispatcher. Timpul tSOPCUAR_SOPCUAA (între frontul crescător al spike-ului 1 și frontul crescător al spike-ului 2) este durata dintre momentul în care serverul OPC UA trimite cererea (comanda) către dispatcher și momentul în care serverul primește răspunsul la cerere de la dispatcher. Timpul tSOPCUAR_SOPCUAR (între frontul crescător al spike-ului 1 prima comandă și frontul crescător al spike-ului 1 pentru comanda următoare) este timpul dintre momentul în care serverul OPC UA lansează o cerere pentru dispatcher și momentul în care serverul OPC UA lansează următoarea cerere către dispatcher (timpul dintre două cereri consecutive). Excluderea mutuală la nivel de slot se face prin software, în sensul că operațiile de citire scriere a datelor pe slot se fac la sfârșitul analizei slot-ului când nu sunt operații de citire scriere specifice ciclului de achiziție.

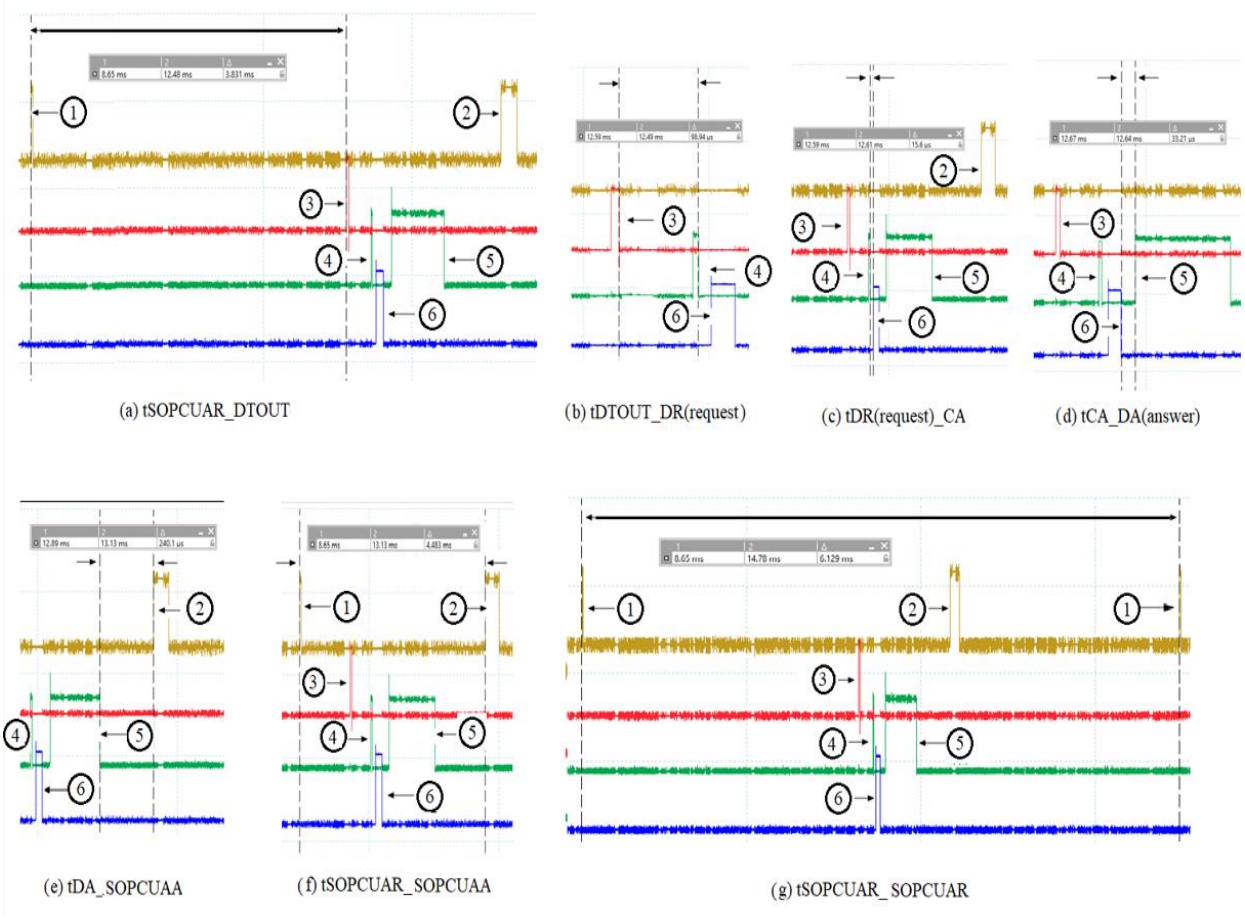


Figura 6-8 Timpi măsurați necesari pentru ca informațiile solicitate de serverul OPC UA să ajungă la ciclu de achiziție și retur [46].

Tabel 6-1 Valorile timpilor *tSOPCUAR_DTOUT, tDTOUT_DR, tDR_CA, tCA_DA, tDA_SOPCUAA, tSOPCUAR_SOPCUAA, tSOPCUAR_SOPCUAR* pentru 10 măsurători cu osciloscop.

No.	tSOPCUAR_DTOUT	tDTOUT_DR	tDR_CA	tCA_DA	tDA_SOPCUAA	tSOPCUAR_SOPCUAA	tSOPCUAR_SOPCUAR
1	3.831 ms	98.94 µs	15.6 µs	33.21 µs	240.1 µs	4.483 ms	6.129 ms
2	3.67 ms	155.8 µs	42.47 µs	71.22 µs	224.2 µs	4.73 ms	6.706 ms
3	2.961 ms	102.6 µs	159 µs	96.13 µs	231.9 µs	3.775 ms	5.402 ms
4	3.559 ms	102.7 µs	153.7 µs	68.9 µs	221.4 µs	4.328 ms	6.28 ms
5	3.352 ms	399.2 µs	34.21µs	72.52 µs	222.9 µs	4.224 ms	5.872 ms
6	3.41 ms	142.7 µs	37.56 µs	40.54 µs	232.1 µs	4.038 ms	5.766 ms
7	3.672 ms	143.8 µs	242.8 µs	64.37 µs	206.4 µs	4.515 ms	6.21 ms
8	3.508 ms	101.8 µs	244.4 µs	39.31 µs	266.2 µs	4.347 ms	5.953 ms
9	3.569 ms	140.3 µs	79.64 µs	71.35 µs	230.6 µs	4.266 ms	6.228 ms
10	3.319 ms	102 µs	88.02 µs	118.9 µs	226 µs	4.084 ms	5,716 ms

În Tabel 6-1 se prezintă valorile *tSOPCUAR_DTOUT, tDTOUT_DR, tDR_CA, tCA_DA, tDA_SOPCUAA, tSOPCUAR_SOPCUAA, tSOPCUAR_SOPCUAR* pentru 10 măsurători cu osciloscop. În Figura 6-9 a și b, spike-urile numerotate reprezintă următoarele: 7 (culoarea muștar) reprezintă momentul în care serverul OPC UA (care rulează pe procesorul ARM Cortex A8) primește cererea prin socket-uri de la clientul OPC UA (care rulează pe un computer folosind sistemul de operare Windows) și o trimite prin socket la dispatcher; 8 (culoarea muștar) reprezintă momentul în care serverul OPC UA primește răspunsul la cerere de la dispatcher; și fronturile crescătoare numerotate 3, 4, 5 și 6 sunt aceleași ca în Figura 6-8 și, prin urmare, sunt descrise în descrierea pentru Figura 6-8. Timpul între frontul descrescător 7 și frontul descrescător 8 notat T7_8 (Figura 6-9 a – s-a ales culoarea muștar pentru acest semnal pe osciloscop) este timpul dintre momentul în care serverul OPC UA se pregătește să trimită mai departe cererea de la clientul OPC UA către dispatcher și momentul în care serverul OPC UA primește răspunsul de la dispatcher și se pregătește pentru a-l transmite către clientul OPC UA. Timpul între frontul descrescător 7 și frontul descrescător 7 T7_ 7 (Figura 6-9 b - s-a ales culoarea muștar pentru acest semnal pe osciloscop) este timpul dintre două solicitări consecutive pe care serverul OPC UA le face.

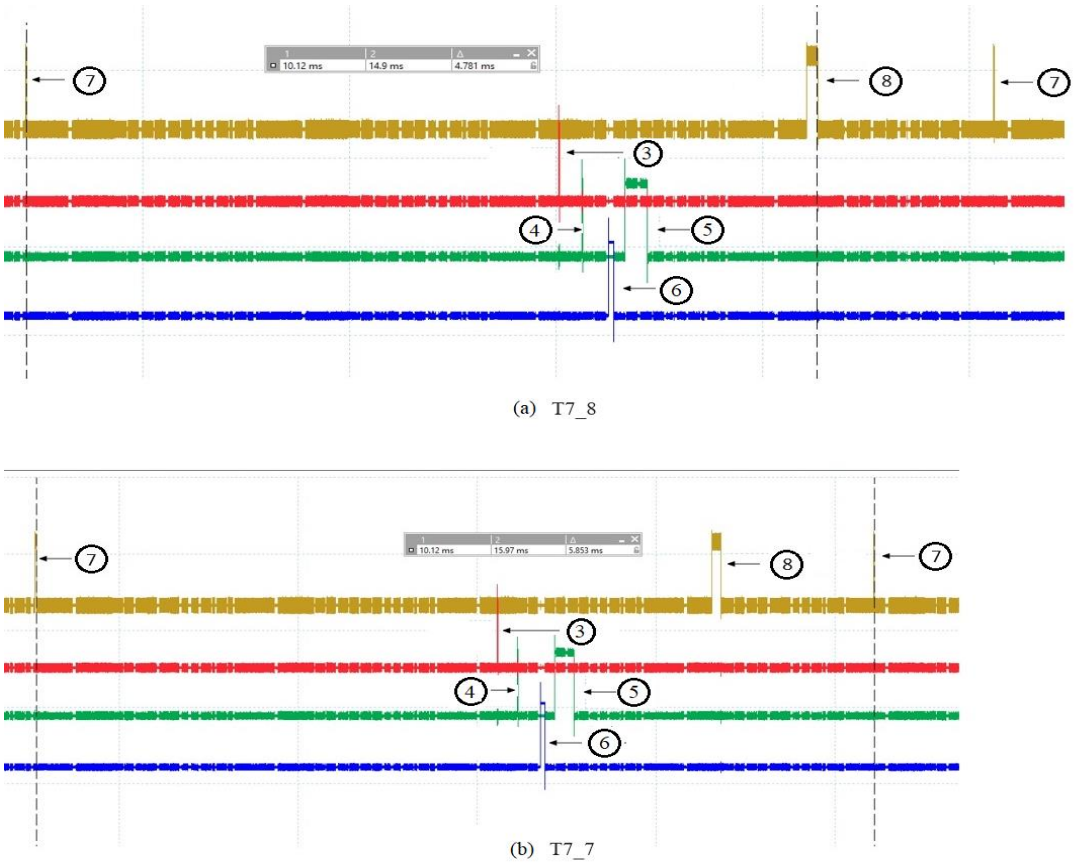


Figura 6-9 Timpul măsurat necesar pentru ca informațiile primite pe serverul OPC UA de la clientul OPC UA să ajungă la ciclul de achiziție și să revină (Figura 6-9 a – T7_8) și timpul măsurat pentru două solicitări consecutive de la clientul OPC UA (Figura 6-9 b – T7_7) [46].

În Tabel 6-2 se prezintă valorile pentru timpii T7_8 și T7_7 prezentați în Figura 6-9, obținute prin 10 măsurători cu osciloscopul.

Tabel 6-2 Valorile timpilor T7_8 și T7_7 pentru 10 măsurători efectuate cu osciloscopul.

No. de măsurători	T7_8	T7_7
1	4.781 ms	5.853 ms
2	4.798 ms	5.9 ms
3	4.913 ms	5.996 ms
4	5.381 ms	6.482 ms
5	4.504 ms	5.577 ms
6	4.774 ms	6.081 ms
7	4.783 ms	5.848 ms
8	5.605 ms	6.689 ms
9	4.374 ms	5.438 ms
10	4.919 ms	5.894 ms

Măsurătorile semnalelor au fost făcute cu osciloscopul PicoScope 640B cu 4 canale și frecvența maximă de 500 Mhz vezi Figura 6-10.

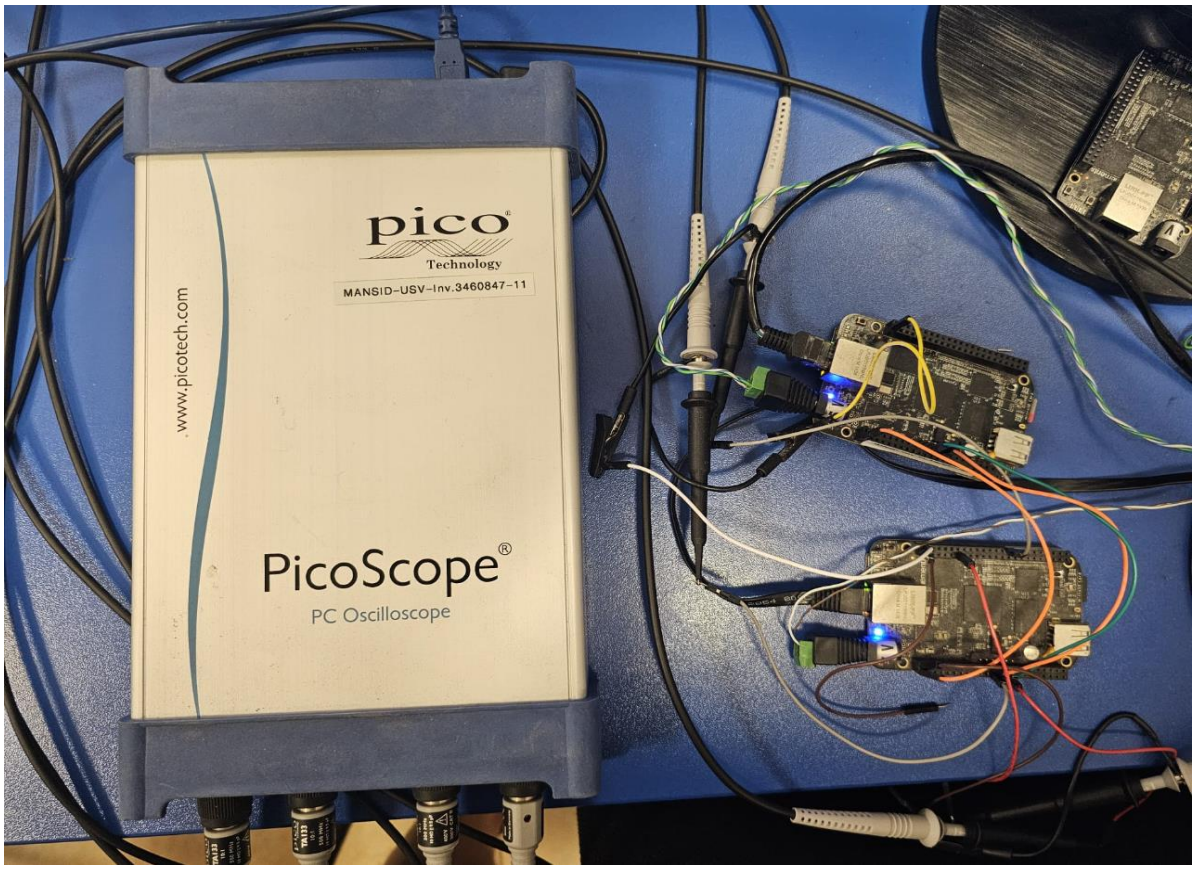


Figura 6-10 Schema de test din laborator

În Figura 6-10 este prezentată schema de test din laborator formată din osciloscop și ce 2 plăci BeagleBone Black (client și serverul ModbusE).

7 Concluzii, contribuții teoretice și practice

7.1 Concluzii

Obiectivele propuse pentru această lucrare au fost îndeplinite in totalitate prin definirea arhitecturii gateway-ului IIoT — ModbusE, dezvoltarea și testarea componentelor software și analiza performanțelor. Arhitectura cuprinde 3 module de bază și anume clientul ModbusE, dispatcher-ul și serverul OPC UA. Pentru testarea funcționalității s-a folosit o aplicație client OPC UA pe un calculator cu Windows 10. Ca suport hardware s-a ales procesorul Sitara AM335X care conține 2 module PRU și un procesor ARM Cortex A8. Modulul client ModbusE se ocupă de comunicația serială cu stațiile server pentru transferul de date din proces și cu implementarea ciclului de achiziție. Ciclul de achiziție este alcătuit din slot-uri, mărimea de date a slot-ului și implicit durata acestuia fiind dictată de aplicație. Fiecare slot este definit de o structură care conține printre altele o ștampilă de timp, contor de mesaje recepționate eronat, contor de mesaje recepționate corect, pointer la mesajele pentru citire și scriere, lungimea acestor mesaje, un cuvânt de stare a slot-ului, durata slot-ului, etc. Buffer-ul de date de la nivelul fiecărui slot reprezintă zona de memorie care asigură transferul de date între ciclul de achiziție și dispatcher. Ciclul de achiziție asigură un comportament determinist în scrierea și citirea mesajelor pe magistrala RS485. Pentru fiecare slot este definit un timp de lucru care va fi întotdeauna respectat. Chiar dacă mesajul curent depășește durata slot-ului acesta este suspendat. Din calcul CRC se poate vedea dacă mesajul este corect sau eronat. Accesul la acest slot se va relua în ciclul următor. Modulul dispatcher asigură operații atomice de citire/ scriere cu buffer-ele de date asociate fiecărui slot. Datorită faptului ca dispatcher-ul rulează pe Arm Cortex A8 sub Linux, operațiile de citire/scriere se realizează prin intermediul zonei de memorie comune (PRU – Cortex A8). Dispatcher-ul comunică cu serverul OPC UA utilând socket-uri. Pentru implementarea serverului OPC UA s-a implementat un fir de execuție care utilizează biblioteca open62541. Acest fir de execuție poate transfera comenzi de la, la un client OPC UA și dispatcher.

Lucrările anterioare care au folosit extensia ModbusE pe arhitecturile Cortex Mx au atins performanțe de aproximativ 49,6% date utile la o viteză a portului serial de 10,5 Mb/s cu STM32F407, 36% date utile pe portul serial de 27 Mb/s cu STM32F746 și 58,9% date utile la o viteză a portului serial de 11,5 Mb/s cu LPC4357 care are 2 procesoare Cortex M0 și M4. Dar, procesoarele Cortex M0 și M4 nu au suficiente resurse pentru a implementa serverul OPC UA de înaltă performanță. Cu Sitara AM335x a fost obținut un procent de 53,3% date utile la o viteză a portului serial de 12 Mb/s. După cum a fost prezentat, Sitara AM335x este format din procesorul ARM Cortex A8 și nucleele în timp real PRU0 și PRU1. Deci Sitara Am335x are resursele necesare pentru a putea implementa serverul OPC UA (pe procesorul ARM Cortex A8). Anterior s-a prezentat că perioada unui ciclu de achiziție cu 10 slot-uri este de 1,349 ms, astfel încât ciclul de achiziție permite utilizarea acestei viteze pentru dispozitivele ModbusE conectate la fieldbus. În Tabel 6-1 putem observa că timpul tSOPCUAR_SOPCUAA (timpul dintre momentul în care serverul OPC UA solicită o comandă ModbusE și momentul în care serverul OPC UA primește răspunsul) este de 6.706 ms, mai mare decât perioada unui ciclu de achiziție (CA). Valorile măsurate sunt mai mari la nivelul superior deoarece clientul OPC UA, serverul OPC UA și dispatcher-ul folosesc Ethernet prin sisteme de operare Windows și Linux care aduc timpi de întârziere. Stațiile server (slave) ModbusE pot comunica

date utile între ele la viteze mari, fără a fi nevoie ca aceste date să ajungă la clientul OPC UA. Doar valorile care trebuie afișate ajung la clientul OPC UA. Dispozitivele conectate la nivelul inferior pe ModbusE pot comunica folosind modelul publisher-subscriber direct prin ciclul de achiziție (CA) la o viteză de 12 Mb/s (care este și viteza maximă pentru protocolul Profibus). Întrucât la nivelul nucleului PRU0 nu s-a putut depana software-ul pentru a vedea dacă ciclul de achiziție (CA) rulează corect, am folosit nucleul PRU1 pentru a putea citi anumite variabile din memorie și în acest fel am putut testa dacă software-ul funcționează corect. Am folosit nucleul PRU1 pentru a lăsa nucleul PRU0 să ruleze doar ciclul de achiziție. Totuși, în ciuda determinismului foarte bun nucleul PRU are o limitare serioasă privind resursele hardware în sensul că nu se pot defini multe slot-uri. Pentru aplicație de complexitate mică, medie. Pentru exemplul cu 10 slot-uri utilizat în lucrare ciclul de achiziție are 1082 de caractere, ceea ce ar putea reprezenta 541 de regiștri Modbus. Toate testele au fost realizate în condiții de laborator, dar nu într-un proces industrial propriu-zis.

7.2 Contribuții

În urma unor mai multor activități de cercetare industrială concretizate prin analize , studii, teste, s-a realizat implementarea gateway-ului IoT utilizând extensia ModbusE a protocolului Modbus. Motivul alegerii procesorului Sitara AM335x a fost acela de a îmbunătăți utilizarea canalului de date prin folosirea nucleului de timp real PRU0 și de a obține o conexiune performantă la IoT cu procesorului ARM Cortex A8.

7.2.1 Contributii teoretice

Contribuțiile teoretice sunt următoarele:

- S-a realizat un studiu privind tehnologiile aferente implicate de Industry 4.0 și IoT.
- S-a realizat un studiu privind protocoalele de comunicație IoT, domenii și aplicații IoT, arhitecturi IoT, gateway-uri IIoT, rețele industriale locale (RIL).
- S-a realizat o sinteză a principalelor caracteristici ale protocolului Modbus și a extensiei MosbusE necesare implementării extensiei ModbusE a protocolului Modbus. Extensia ModbusE păstrează în anumite condiții compatibilitatea cu Mosbus.
- S-a realizat o analiză bibliografică referitoare la protocolul Modbus și extensia ModbusE și Modbus TCP/IP.
- S-a propus arhitectura principală a sistemului experimental pentru gateway IIoT – ModbusE, prezentându-se aspectele principale legate de modulul BeagleBone Black, procesorul Sitara AM335x cu subsistemul PRU-ICSS (Programmable Real-time Unit and Industrial Communication Sub-System), procesorul ARM Cortex A8 și arhitectura unificată OPC UA.
- S-a propus pentru ciclul de achiziție (CA) organigrama algoritmului pentru implementarea clientului ModbusE (master) pe procesorul PRU0.
- Pentru transferul datelor de la clientul ModbusE (care implementează ciclul de achiziție) s-a propus sub formă de organigramă un algoritm denumit dispatcher care are rolul de a transfera datele către serverul OPC UA / client TCP/IP.
- S-a realizat analiza performanțelor extensiei ModbusE pentru payload și timpii de comunicație cu referire la nucleul PRU cu ciclu de achiziție (CA), fluxul de mesaje de

comunicație între ciclul de achiziție (pe PRU0), dispatcher (pe ARM Cortex A8 cu Linux), server OPC UA (pe ARM Cortex A8 cu Linux), client OPC UA (pe Windows).

- S-a propus un model matematic pentru ciclul de achiziție pentru evaluarea payload-ului pentru datele utile transmise pe slot-urile din ciclul de achiziție.

7.2.2 Contribuții practice

Contribuțiile practice pentru această lucrare sunt:

- Implementarea practică a clientului ModbusE incluzând ciclul de achiziție (CA) și a comunicației pe USART cu specificațiile Modbus, utilizând procesorul PRU0 cu perifericele USART și IEP (se folosesc 2 comparatoare ale acestui timer, un comparator pentru durata slot-ului și un comparator pentru durata de 3.5 caractere între 2 mesaje consecutive).
- S-a implementat pe PRU1 un program cu ajutorul căruia să se depaneze la nivelul Linux aplicația de pe PRU0.
- Pentru comunicația între clientul ModbusE care rulează pe PRU0 și serverul OPC UA s-a dezvoltat o aplicație denumită dispatcher, care intermediază datele între ciclul de achiziție și serverul OPC UA folosind socket-uri TCP/IP.
- S-a realizat o aplicație client utilă pentru depanarea aplicație de pe IIoT ModbusE gateway.
- S-a testat funcționalitatea IIoT ModbusE gateway cu un server OPC UA conectat la dispatcher și cu aplicația client UaExpert OPC UA care se execută pe un PC ce utilizează sistemul de operare Windows 10.
- S-au definit date și structuri pentru structurarea slot-urilor din ciclul de achiziție (CA).

7.3 Cercetări viitoare

Cercetările și dezvoltările viitoare sunt prezentate în continuare:

- Construcția unui instrument software puternic care să permită configurarea automată a rețelei.
- Construcția unui instrument software care să permită configurarea automată a lui tAdjust.
- Testarea cu mai multe stații server (cel puțin 10 stații) ce implementează extensia ModbusE.
- Testarea schimbării direcției de linie la transmisia și recepția mesajelor folosind drivere hardware de linie RS485.
- Testarea OPC UA pentru conexiune publisher-subscriber folosind protocolul MQTT.
- Implementarea pentru crearea de istorice de date a unui datalogger.

7.4 Diseminarea rezultatelor

Contribuțiile proprii care au rezultat în urma cercetărilor din cadrul prezentei lucrări au fost publicate în volumele unor conferințe internaționale de specialitate după cum urmează:

	Publicații în volumele unor conferințe internaționale
1	**C. Ventuneac,** V. G. Găitan, “Implementation of an IIoT Access Gateway for the ModBusE – Modbus Extension using BeagleBone Black”, INTERNATIONAL CONFERENCE European Integration - Realities and Perspectives 16 th Edition, EIRP 2021, Danubius University, May 2021. (BDI)
2	**C. Ventuneac,** V. G. Gaitan and C. Lupu, "Improving ModBus Extension performance using PRU unit from Sitara AM335x," 2022 International Conference on Development and Application Systems (DAS), Suceava, Romania, 2022, pp. 73-77, doi: 10.1109/DAS54948.2022.9786150. (BDI)
3	**Ventuneac, C**., Gaitan, V.G. Industrial Internet of Things Gateway with OPC UA Based on Sitara AM335X with ModbusE Acquisition Cycle Performance Analysis. Sensors 2024, 24, 2072. https://doi.org/10.3390/s24072072 (ISI, F.I.: 3.4 - Q2)
4	**C. Ventuneac,** V. G. Gaitan, "Performance Evaluation of the Acquisition Cycle for an Original Modbus Extension IIoT Gateway Based on Sitara AM335x Processor," Advances in Electrical and Computer Engineering, vol.24, no.2, pp.41-48, 2024, doi:10.4316/AECE.2024.02005 (ISI, F.I.: 0.7)
5	LUPU Catalin**,** TURCU, Corneliu-Octavian; **VENTUNEAC Cornel**, GĂITAN Vasile-Gheorghiță, The Vaccine’s QR Code is in Your Eyes, IJCSNS journal, DOI: 10.22937/IJCSNS.2022.22.4.70, Vol. 22 no. 4, p. 595-602, 2022 (ISI)

8 Bibliografie

[1] A. Rayes, S. Salam, Internet of Things From Hype to Reality,https://doi.org/10.1007/978-3-319-99516-8.

[2] J. Lee, H.A. Kao, S. Yang, Service innovation and smart analytics for industry 4.0 and big data environment. Procedia CIRP. 16, 3–8 (2014)

[3] Isabel Castelo-Branco, Frederico Cruz-Jesus, Tiago Oliveira,Assessing Industry 4.0 readiness in manufacturing: Evidence for the European Union,Computers in Industry,Volume 107, 2019, Pages 22-32, ISSN 0166-3615, https://doi.org/10.1016/j.compind.2019.01.007.

[4] Oztemel, E.; Gursev, S. Literature review of Industry 4.0 and related technologies. J. Intell. Manuf. 2018, 31, 127–182, doi:10.1007/s10845-018-1433-8.

[5] Ali Turkyilmaz, Dinara Dikhanbayeva, Zhanybek Suleiman, Sabit Shaikholla, Essam Shehab, Industry 4.0: Challenges and opportunities for Kazakhstan SMEs, Procedia CIRP, Volume 96, 2021, Pages 213-218, ISSN 2212-8271, https://doi.org/10.1016/j.procir.2021.01.077.

[6] Kagermann, H., Wahlster.W. and Johannes, H. Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0. Forschungsunion, 2013.

[7] Tay, Shu & Te Chuan, Lee & Aziati, A. & Ahmad, Ahmad Nur Aizat. (2018). An Overview of Industry 4.0: Definition, Components, and Government Initiatives. Journal of Advanced Research in Dynamical and Control Systems. 10. 14.

[8] Internet of Things Global Standards Initiative. https://www.itu.int/en/ITU-T/gsi/iot/Pages/default.aspx.

[9] Seetharaman, A., Patwa, N., Saravanan, A. S., & Sharma, A. (2019). *Customer expectation from Industrial Internet of Things (IIOT). Journal of Manufacturing Technology Management.* doi:10.1108/jmtm-08-2018-0278.

[10] Industrial Internet Reference Architecture. Available online: http://www.iiconsortium.org/IIRA.htm.

[11] https://www.st.com/resource/en/datasheet/stm32mp157c.pdf

[12] https://www.st.com/resource/en/datasheet/stm32h747ai.pdf

[13] Vasile Gheorghiță GĂITAN, Ionel ZAGAN, Ioan UNGUREAN Andy Cristian TĂNASE. (2020). Rețele industriale locale – Modbus Extins Ediția a 2-a

[14] FOUNDATION FIELDBUS EXPLAINED https://www.fieldcommgroup.org/technologies/foundation-fieldbus/foundation-technology-overview

[15] Fei, C., Ding, F., & Zhao, X. (2012). *Network Partition of Switched Industrial Ethernet by Using Novel Particle Swarm Optimization. Physics Procedia, 24, 1493–1499.* doi:10.1016/j.phpro.2012.02.221

[16] Modbus.org. (2006). MODBUS Messaging on TCP/IP Implementation Guide V1.0b. Retrieved from https://modbus.org/.

[17] Modbus.org. (2006). MODBUS APPLICATION PROTOCOL SPECIFICATION V1.1b. Retrieved from https://modbus.org/

[18] Y. Wang, V. Gaspes, “A compositional implementation of Modbus in Protege”, *6th IEEE International Symposium on Industrial and Embedded Systems*, Vasteras, 2011, pg. 123-131. doi: 10.1109/SIES.2011.5953654.

[19] G. Cena, M. Cereia, I. Cibrario Bertolotti, S. Scanzio, “A MODBUS extension for inexpensive distributed embedded systems”, 2010 IEEE International Workshop on Factory Communication Systems Proceedings, Nancy, Franța, 2010, pg. 251-260. doi: 10.1109/WFCS.2010.5548625.

[20] A. Lemay, J. M. Fernandez, S. Knight, “A Modbus command and control channel”, 2016 Annual IEEE Systems Conference (SysCon), Orlando, FL, 2016, pg. 1-6. doi: 10.1109/SYSCON.2016.7490631.

[21] L. Jean, “Python Software Foundation - modbus_tk 0.5.8”, (https://pypi.python.org/pypi/modbus_tk)

[22] V. G. Găitan, I. Ungurean, N. C. Găitan, V. Popa, “Utilizarea specificaţiilor OPC DA pentru implementarea aplicaţiilor distribuite de tip SCADA”, Editura DrukArt Cernăuţi, ISBN 978-966-2021-69-1, 2013.

[23] T. Hu, I. C. Bertolotti, “Overhead and ACK-induced jitter in Modbus TCP communication”, 2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a better tomorrow (RTSI), Torino, Italia, 2015, pg. 392-397. doi: 10.1109/RTSI.2015.7325130.

[24] LPC2468 Product data sheet, rev. 4, NXP B.V., Oct. 2008

[25] T. Nguyen, T. Huynh, “Design and implementation of modbus slave based on ARM platform and FreeRTOS environment”, 2015 International Conference on Advanced Technologies for Communications (ATC), Ho Chi Minh City, 2015, pg. 462-467. doi: 10.1109/ATC.2015.7388372.

[26] C. Urrea, C. Morales, R. Muñoz, “Design and implementation of an error detection and correction method compatible with MODBUS-RTU by means of systematic codes”, Measurement, vol. 91, 2016, pg. 266-275, ISSN 0263-2241, https://doi.org/10.1016/j.measurement.2016.05.055.

[27] G. Cena, I. C. Bertolotti, T. Hu, A. Valenzano, “Design, verification, and performance of a MODBUS-CAN adaptation layer”, 2014 10th IEEE Workshop on Factory Communication Systems (WFCS 2014), Toulouse, Franța, 2014, pg. 1-10. doi: 10.1109/WFCS.2014.6837605.

[28] G. B. M. Guarese, F. G. Sieben, T. Webber, M. R. Dillenburg, C. Marcon, “Exploiting Modbus Protocol in Wired and Wireless Multilevel Communication Architecture”, 2012 Brazilian Symposium on Computing System Engineering, Natal, Brazilia, 2012, pg. 13-18. doi: 10.1109/SBESC.2012.12.

[29] G. Künzel, M. A. Corrêa Ribeiro, C. E. Pereira, “A tool for response time and schedulability analysis in modbus serial communications”, 2014 12th IEEE International Conference on Industrial Informatics (INDIN), Porto Alegre, Brazilia, 2014, pg. 446-451. doi: 10.1109/INDIN.2014.6945554.

[30] K. Wang, D. Peng, L. Song, H. Zhang, “Implementation of Modbus Communication Protocol based on ARM Coretx-M0”, 2014 IEEE International Conference on System Science and Engineering (ICSSE), Shanghai, China, 2014, pg. 69-73. doi: 10.1109/ICSSE.2014.6887907.

[31] C. Urrea, C. Morales, J. Kern, "Implementation of error detection and correction in the Modbus-RTU serial protocol", International Journal of Critical Infrastructure Protection, vol. 15, 2016, pg. 27-37, ISSN 1874-5482, https://doi.org/10.1016/j.ijcip.2016.07.001.
[32] GĂITAN, V. G., & ZAGAN, I. (2019). *Rețele industriale locale – Modbus Extins/Local industrial networks – Modbus Extension*. Suceava: Editura Universității "Ștefan cel Mare".
[33] Z. Xiang-Li, "Study on communication scheduling of fieldbus", *Controland decision conference, CCDC '09*, China, 2009, pg. 565-70. http://dx.doi.org/10.1109/CCDC.2009.5194952.
[34] V. G. Găitan, N. C. Găitan, I. Ungurean, "A flexible acquisition cycle for incompletely defined fieldbus protocols", *ISA Transactions*, vol. 53, nr. 3, 2014, pg. 776-786, ISSN 0019-0578, https://doi.org/10.1016/j.isatra.2014.02.006.
[35] D. Pavkovic, S. Polak, D. Zorc, "PID controller auto-tuning based on process step response and damping optimum criterion", *ISA Trans* 2014, 53(1), pg. 85-96, http://dx.doi.org/10.1016/j.isatra.2013.08.011.
[36] DANIEL FLOW PRODUCTS: "Modbus communications model 2500"; Part Number: 3-9000-545 REVISION D, Noiembrie, 1992, (https://www.emerson.com/documents/automation/daniel-modbus-communications-model-2500-manual-en-43890.pdf).
[37] Z. Wang, X. Shen, J. Chen, Y. Song, T. Wang, Y. Sun, "Real-time performance evaluation of urgent aperiodic messages in FF communication and its improvement", *Comput Stand Interfaces 2005*; 27(2), pg. 105-115, http://dx.doi.org/10.1016/j.csi.2004.05.001.
[38] A. Malinowski, H. Yu, "Comparison of embedded system design for industrial applications", *IEEE Trans Ind Inf 2011*; 7(2), pg. 244-254, http://dx.doi.org/10.1109/TII.2011.2124466.
[39] Gaitan, V. G., & Zagan, I. (2021). *Experimental Implementation and Performance Evaluation of an IoT Access Gateway for the Modbus Extension*. Retrieved from https://doi.org/10.3390/s21010246.
[40] BeagleBone Black https://beagleboard.org/black.
[41] BeagleBoneBlack, https://elinux.org/Beagleboard:BeagleBoneBlack
[42] Referat 3 "Cercetări experimetale privind performanțele gateway-ului IoT bazat pe protocolul ModBus Extins."
[43] Cornel Ventuneac, Vasile Gheorghiță Găitan. (2021). Implementation of an IIoT Access Gateway for the ModBusE – Modbus Extension using BeagleBone Black. EIRP Proceedings. Vol. 16 No. 1, pg 344-348.
[44] Referat 1 Stadiul actual privind conectarea senzorilor și a elementelor de execuție la un sistem IoT utilizând porți de acces (gateway)
[45] C. Ventuneac**,** V. G. Gaitan, "Performance Evaluation of the Acquisition Cycle for an Original Modbus Extension IIoT Gateway Based on Sitara AM335x Processor," Advances in Electrical and Computer Engineering, vol.24, no.2, pp.41-48, 2024, doi:10.4316/AECE.2024.02005
[46] Ventuneac, C., Gaitan, V.G. Industrial Internet of Things Gateway with OPC UA Based on Sitara AM335X with ModbusE Acquisition Cycle Performance Analysis. Sensors 2024, 24, 2072. https://doi.org/10.3390/s24072072