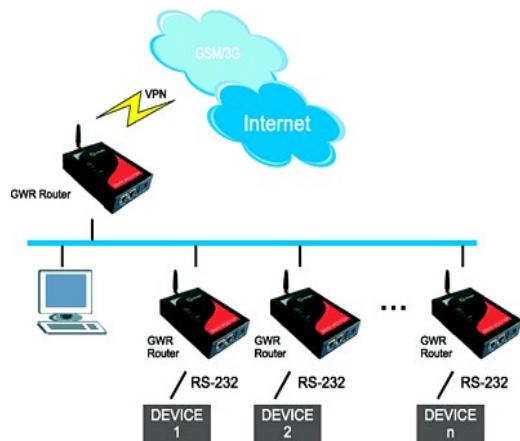


Convertor Serial2Ethernet pentru SCADA



Cuprins:

1. Descrierea problemei de proiectare si dezvoltare a aplicatiei.
2. Descrierea cadrului de interes pentru aplicatia dezvoltata.
 1. Descrierea modelului generic al comunicatiilor si arhitecturilor de control in mediul industrial
 2. SCADA, control supervisor si achizitii de date
 3. Comunicatiile in sisteme SCADA
 4. Componentele generice ale sistemului de comunicatii SCADA
 5. Tendinte in evolutia sistemelor SCADA
3. Descrierea modulelor hardware si software ale aplicatiei dezvoltate.
 1. Prezentarea modulului de dezvoltare si a MCU AVR Atmega16
 2. Prezentarea modulului de dezvoltare NGW100 cu MPU AVR32
 3. Descrierea softwareului de conversie si de test pentru aplicatia dezvoltata
 4. Detalii de implementare si codul aplicatiei modulului IO cu ATMega16
 5. Detalii de implementare si codul aplicatiei modulului NGW100 cu AVR32
 6. Pasi executie
 7. Anexe

Nota

Aplicatia **Convertor Serial2Ethernet pentru SCADA** a fost dezvoltata in cadrul proiectului de semestru pentru materia SCADA din cadrul programului de master IACA, FACIEE, UGAL.

Echipa :

**Cristian AXENIE
Aurelian ZANOSCHI
Bogdan ARTENE**

1. Descrierea problemei de proiectare si dezvoltare a aplicatiei

Aplicatia dezvoltata propune o solutie **ieftina si robusta** de implementare a unui **convertor** de informatie de pe **linia seriala RS-232 la o linie TCP/IP Ethernet**, destinata uzului industrial pentru aplicatii **SCADA**. Intrucat se folosesc elemente hardware specifice existente in structura de baza SCADA (Ethernet gateway si module IO cu comunicatie seriala RS-232) aplicatia vine ca o solutie care sa inlocuiasca modulele complexe de IO si ofera extensibilitate si scalabilitate. **Modulul gateway** este un modul de comunicatie avansat ce permite multitasking la nivelul unui sistem de operare preemptiv si ofera conectivitate multipla prin interfete Ethernet la linii de comunicatie TCP/IP sau UDP/IP si interfatare pe linii de comunicatie seriala RS-232.

Datorita faptului ca modulul gateway este implementat sub forma unui platforme embedded (**AVR32 NGW100**) consumul de putere este redus si flexibilitatea si extensibilitatea platformei pot determina extinderea functionalitatilor. Pentru exemplificare s-a utilizat si un modul IO distribuit (**ATMega16 EvalBoard**) care permite interfatare cu traductoare si elemente de executie avand o gama larga de periferice disponibile pentru interfatarea in domeniul semnalului digital dar si semnalului analogic.

Cele doua module vor comunica pe linia seriala RS-232 si informatia primita de la modulul IO va fi apoi incapsulata si transmisa in pachete UDP/IP de catre gateway catre noduri client, care preiau informatii pentru logare sau pentru sinteza unor regului / comenzi de supravizor.

2. Descrierea cadrului de interes pentru aplicatia dezvoltata

1. Descrierea modelului generic al comunicatiilor si arhitecturilor de control in mediul industrial

Sistemele de conducere distribuită sunt utilizate din ce în ce mai mult, atât în sistemele de fabricație, cât și în automatizarea proceselor continue. Cu ajutorul acestora, un task de conducere complex poate fi împărțit, de regulă pe criterii funcționale, în mai multe subtaskuri, mai ușor de gestionat. Drept urmare, este necesară o comunicație eficientă între subtaskurile implementate pe sisteme distribuite geografic.

Distribuirea funcțiilor de conducere are numeroase avantaje, dintre care se pot menționa:

- funcționarea independentă sau simultană a unor portiuni distincte ale sistemului condus, sub controlul unor unități distincte;
- realizarea unor programe de conducere mai simple și mai clare;
- prelucrarea paralelă prin distribuirea sarcinilor pe mai multe unități de prelucrare.

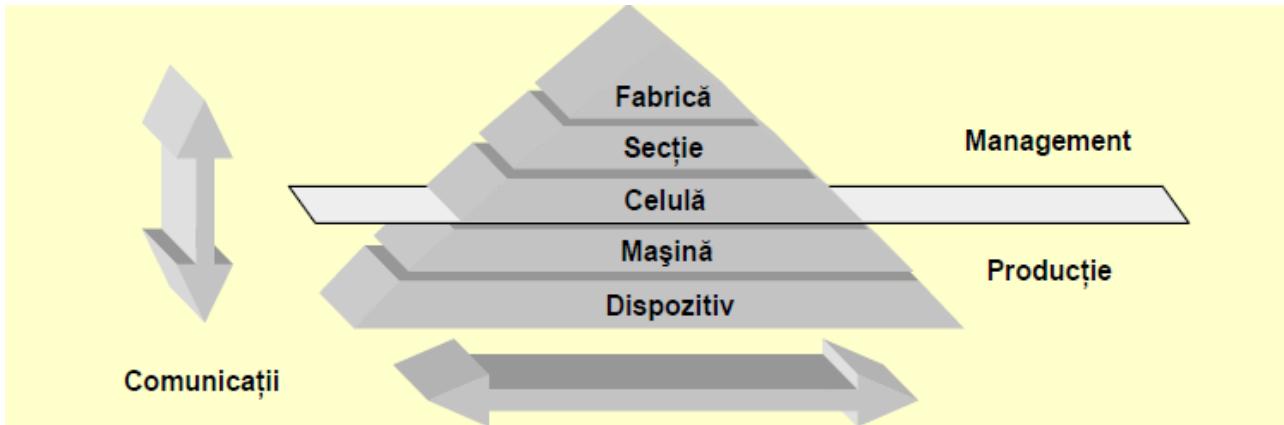
Aceste avantaje conduc la creșterea performanțelor sistemului de conducere distribuită prin:

- timpi mai mici de reacție la schimbările survenite în sistemul condus;
 - reducerea încărcării unităților de conducere individuale;
 - structuri compuse la nivel de sistem pentru realizarea unor operații de diagnoză și înregistrare;
- creșterea disponibilității sistemului, întrucât dacă o unitate se defectează, celelalte pot continua operarea.

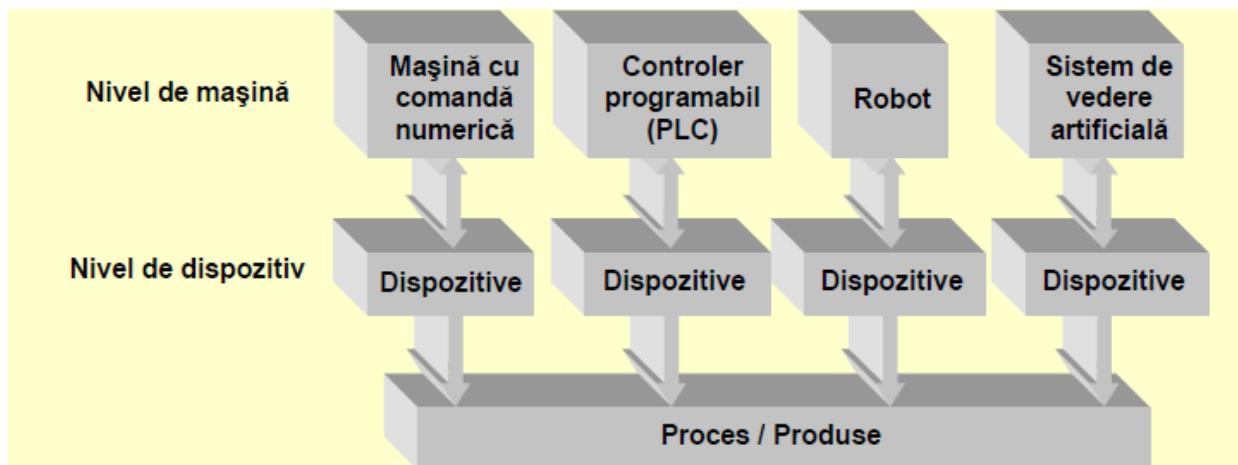
O primă împărțire funcțională, naturală, a sarcinilor în cadrul unei întreprinderi conduce la o structură ierarhizată a comunicațiilor, pe două niveluri de bază:

- nivelul producției nemijlocite;
- nivelul de management al producției.

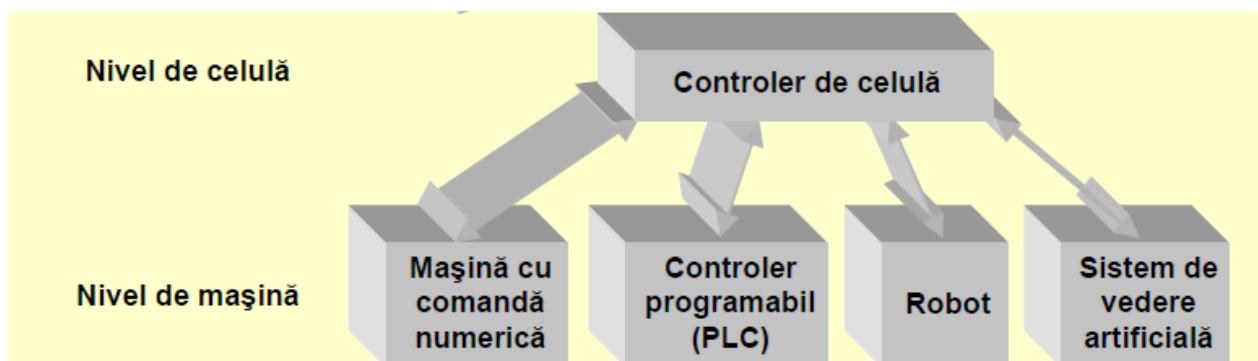
O rafinare suplimentară a acestei ierarhii, pe baza complexității și naturii sarcinilor întâlnite la cele două niveluri principale, conduce la un model ierarhizat pe 5 niveluri, ca în figura urmatoare :



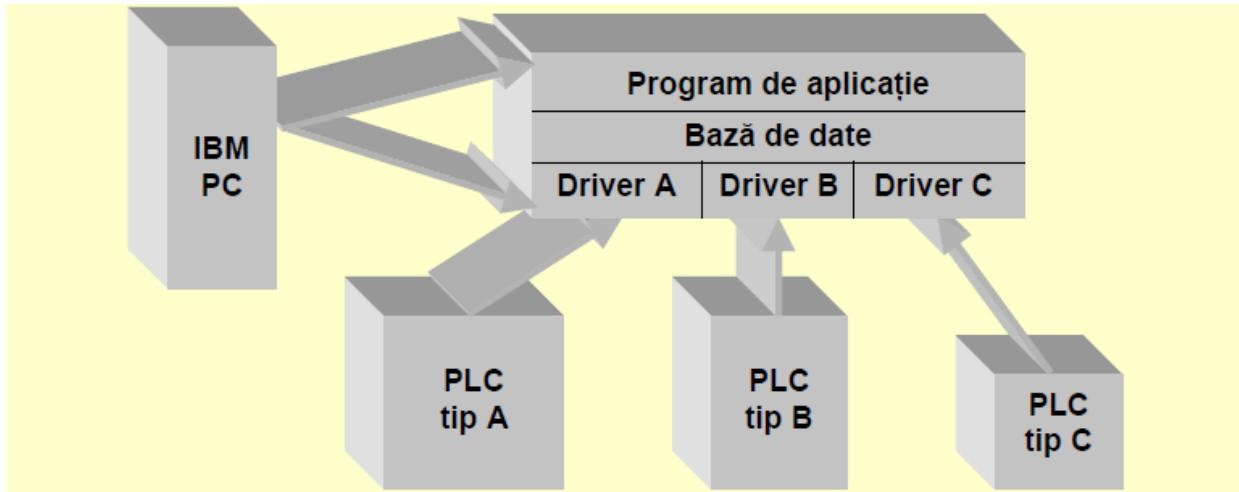
Fiecare mașină produce date utile în timpul funcționării. Controlerelor mașinilor (machine controllers) cunosc numărul produselor transportate/ prelucrate, cât durează un ciclu de produs, cât durează operațiile de pornire și oprire etc. Dacă sunt utilizate (ceea ce se întâmplă în puține cazuri), atunci aceste date pot crește productivitatea muncii în mod semnificativ.



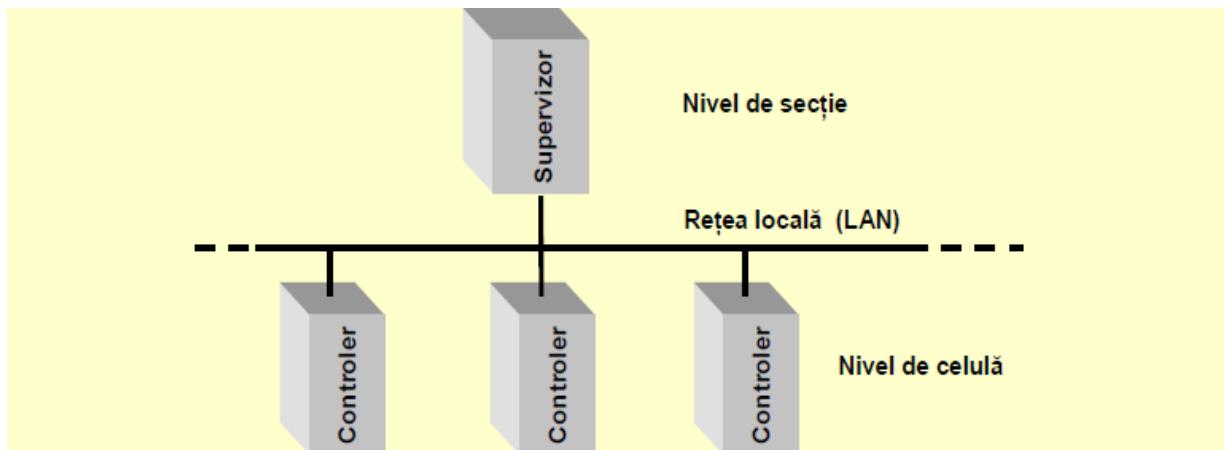
O celulă este un grup format din mașini de tipuri diferite care sunt utilizate pentru realizarea unuia sau mai multor produse similare. Fiecare mașină are de regulă un tip de program aparte și propriul său protocol de comunicație, ceea ce le împiedică să comunice direct între ele.



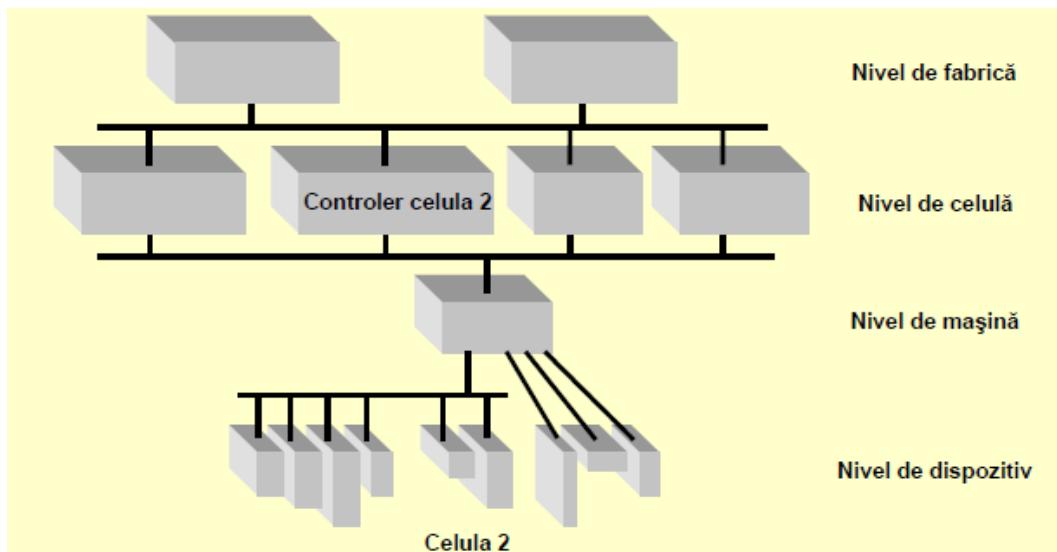
Comunicația la nivelul celulei este realizată de obicei prin intermediul unui pachet software de tip **SCADA** (**S**upervisory **C**ontrol and **D**ata **A**cquisition). Acesta este un pachet de programe care, prin intermediul unor drivere de comunicație adecvate, permite dialogul cu o mare varietate de dispozitive după cum este figurat în figura urmatoare :



La nivel superior controlerile de zonă (area controllers) au funcția de supervizare. Ele primesc comenzi de la nivelul ierarhic superior, apoi dă de lucru celulelor pentru realizarea sarcinilor. De asemenea, ele comunică cu controlerile altor secții pentru a sincroniza producția. Controlerle de secție folosesc comunicații sincrone prin rețelele locale (**LAN – Local Area Network**) după cum s poate observa în figura urmatoare :



Progresele înregistrate în ultimul timp în domeniul calculatoarelor și al rețelelor de comunicații a determinat mutații importante în ierarhia comunicațiilor industriale. Două dintre aceste schimbări majore sunt ilustrate în figura următoare :



Datorită creșterii puterii de calcul și a facilităților de comunicație al microcalculatoarelor, nivelul de secție a dispărut, fiind absorbit de nivelul de fabrică. Atât nivelul de fabrică, cât și nivelul de celulă se conectează la aceeași ete de uz general: **Ethernet**, ARCNet, FDDI, IBM Token Ring, MAP.

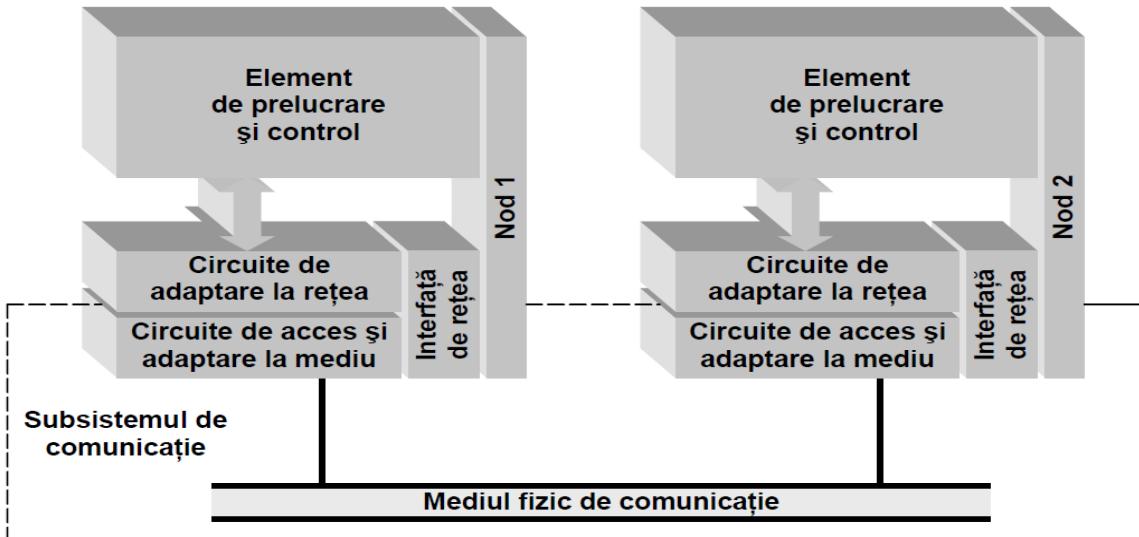
Rețelele industriale de control au o arhitectură ce presupune noduri interconectate într-un mediu de comunicație. Nodul este element de prelucrare și control dotat cu o interfață prin intermediul căreia se conectează în rețea (interfață de rețea). Elementele de prelucrare și control sunt sisteme de conducere cu microprocesoare/microcontrolere de uz general sau cu procesoare specializate. Interfața de rețea poate fi realizată fizic distinct sau se poate regăsi integrată în structura elementului de prelucrare și control.

Subsistemul de comunicație al rețelei:

- Segmente de rețea
- Repetor
- Switch
- **Gateway**.

In continuare se consideră descrierea fiecarui element constitutiv din structura hardware a unei rețele industriale.

Elementele de prelucrare și control se află la nivelul mașinilor și al grupurilor de mașini (celule) și pot utiliza Controlere de tipul **PLC** (Programmable Logic Controller), **IPC** (Industrial PC), **Controlere de automatizare** industrială sau **Sisteme de tip SBC** (Single Board Computer) care pot fi încorporate în structura mașinilor și a instalațiilor industriale (embedded control systems).



Structura hardware a unei rețele industriale

Controlerile de automatizare industrială oferă facilități tradiționale de control discret și continuu, la fel ca și echipamentele de tip PLC. Poseda deosemenea și capabilități de control în buclă închisă al servomotoarelor, o tehnică care tinde să se generalizeze în industrie și pe care PLC-urile nu o pot aborda datorită simplității constructive și vitezei relativ reduse de procesare. Pentru a se integra în structura de conducere ierarhizată la nivelul întreprinderii, oricare dintre cele 4 variante de controlere mai sus menționate are nevoie de cel puțin o interfață de rețea fie ea integrată în structura elementului de prelucrare sau disponibilă sub forma unui modul de extensie. Interfața de rețea redă arhitectura stratificată a interfeței de rețea trebuie implementată printr-o combinație hardware/firmware sau hardware/software. Interfața de rețea implementează, de obicei în hardware, funcțiile de comunicație de la nivelurile fizic și al legăturii de date, oferind executivului un set de servicii de bază pentru livrarea bidirectională a datelor între nodurile rețelei. Interfața de rețea este formată din două straturi distincte și anume o parte orientată către rețea, care asigură conectarea la mediul de comunicație și o a doua parte orientată către elementul de prelucrare și control, specifică structurii I/E a acestuia.

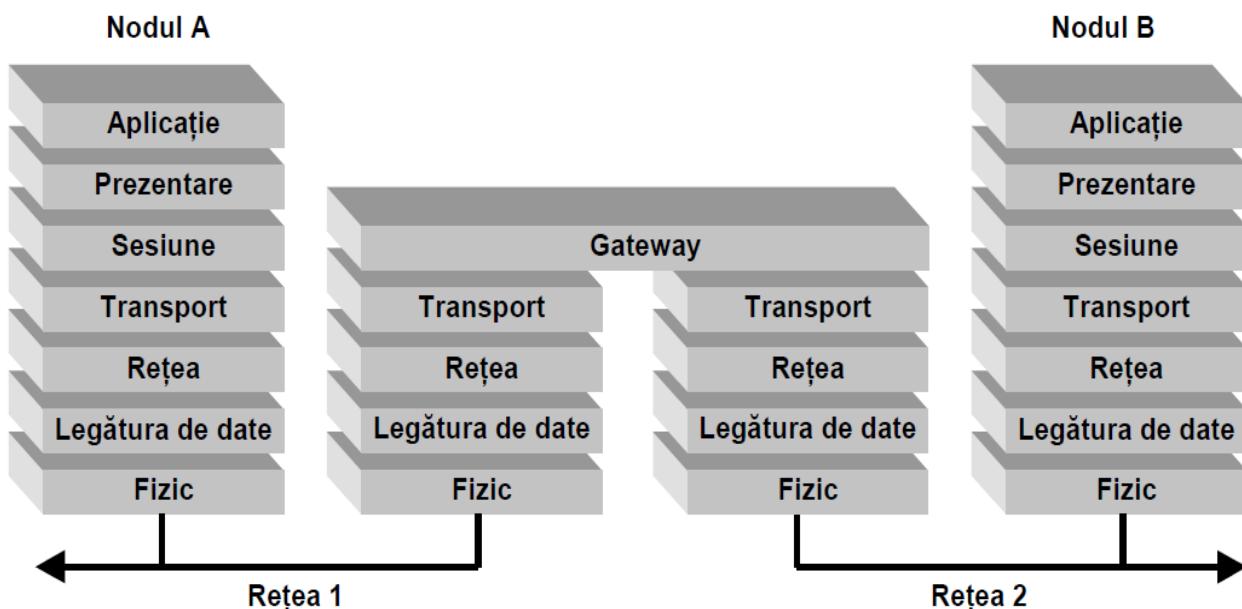
Portiunea orientată către rețea este din punct de vedere funcțional aceeași pentru toate nodurile rețelei și este formată în general din două componente distincte:

- **adaptorul de comunicație**, destinat conectării mecanice și electrice la mediul fizic de comunicație și transmisiei/recepției primare de biți de date;
- **unitatea de acces** – care se ocupă de tratarea pachetelor de date și de

protocolul de acces asociat.

Cealaltă porțiune a interfeței de rețea, orientată către elementul de prelucrare și control - adaptor de rețea, dă fiind rolul său de adaptare a structurii de I/E specifice elementului de prelucrare și control la caracteristicile rețelei. Adaptorul de comunicație asigură o interfațare serială pe bit de transmisie/recepție date între unitatea de acces și mediul fizic de comunicație. Acesta furnizează alte informații, cum ar fi de exemplu "detectare coliziune" pentru tehniciile de acces la mediu bazate pe competiție și implementează în principal funcțiile nivelului fizic specificat în standardul rețelelor industriale. Pentru topologia de tip magistrală, acesta mai este denumit și transceiver (transmitter/receiver), în timp ce pentru topologiile în inel poartă numele de repetor. Unitatea de acces realizează o legătură de date serială la nivel de bit cu adaptorul de comunicație și implementează în hardware sau firmware un protocol de acces specific. Mediul fizic de comunicare este independent de schema de conectare a nodurilor în rețea. Există 4 tipuri principale de medii de comunicație întâlnite în rețelele industriale de control și anume, cablu torsadat (twisted-pair), cablu coaxial, fibră optică și unde radio. Cablul torsadat asigură comunicația în banda de bază și este format din două fire răscute pe întreaga lungime, ceea ce face să crească imunitatea la zgomote (electrice). S-au standardizat două tipuri de cabluri, ecranate (STP – Shielded Twisted-Pair) și neecranate (UTP – Unshielded Twisted-Pair).

Intrucât retelele industriale pot fi **eterogene** se impune folosirea în cat mai multe designuri ale gateway-urilor. Poarta (gateways) folosite pentru interconectarea unor rețele cu arhitecturi complet diferite (ex.: Ethernet – EtherCAT, Modbus – Profinet).



2. SCADA, control supervisor și achiziții de date

SCADA este tehnologia care oferă utilizatorului posibilitatea să colecteze date de la unul sau mai multe echipamente aflate la distanță și să transmită un set limitat de instrucțiuni de comandă acestor echipamente. SCADA face să nu mai fie necesar ca un operator să stea sau să inspecteze frecvent acele locații telecomandate în cazul în care acestea funcționează normal.

SCADA include interfața operator și manipularea datelor concrete ale aplicației dar nu se limitează numai la asta. Unii producători oferă pachete software pe care le denumesc SCADA și desigură legături de comunicare cu alte echipamente necesare, ele nu sunt un sistem SCADA complet. SCADA este acronimul format din primele litere ale termenilor: „supervisory control and data acquisition” (control de supraveghere și achiziție de date).

Desi termenii de bază nu se referă la termenul distantă, acesta este comun tuturor sistemelor SCADA. Un astfel de sistem permite unui operator să realizeze schimbarea punctului de funcționare al unui regulator aflat la

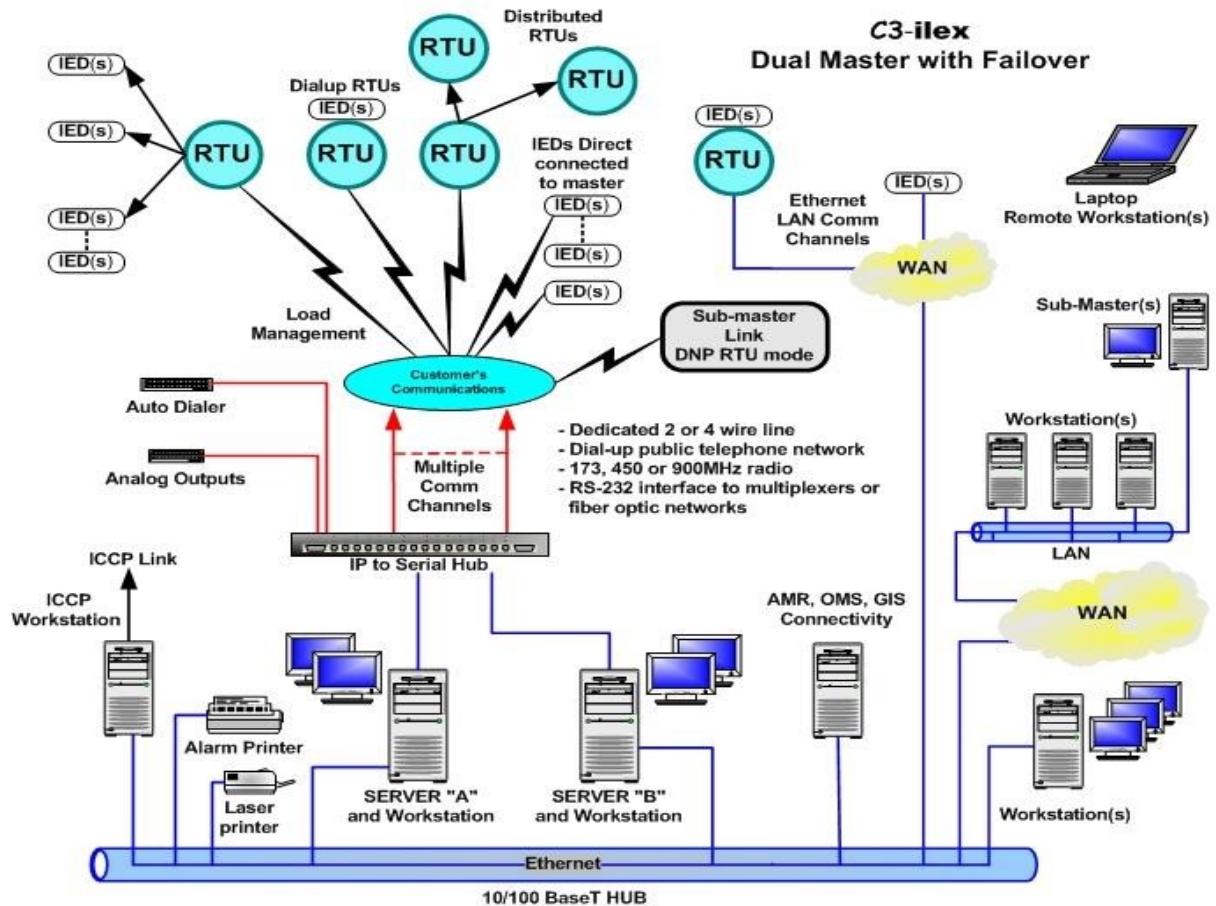
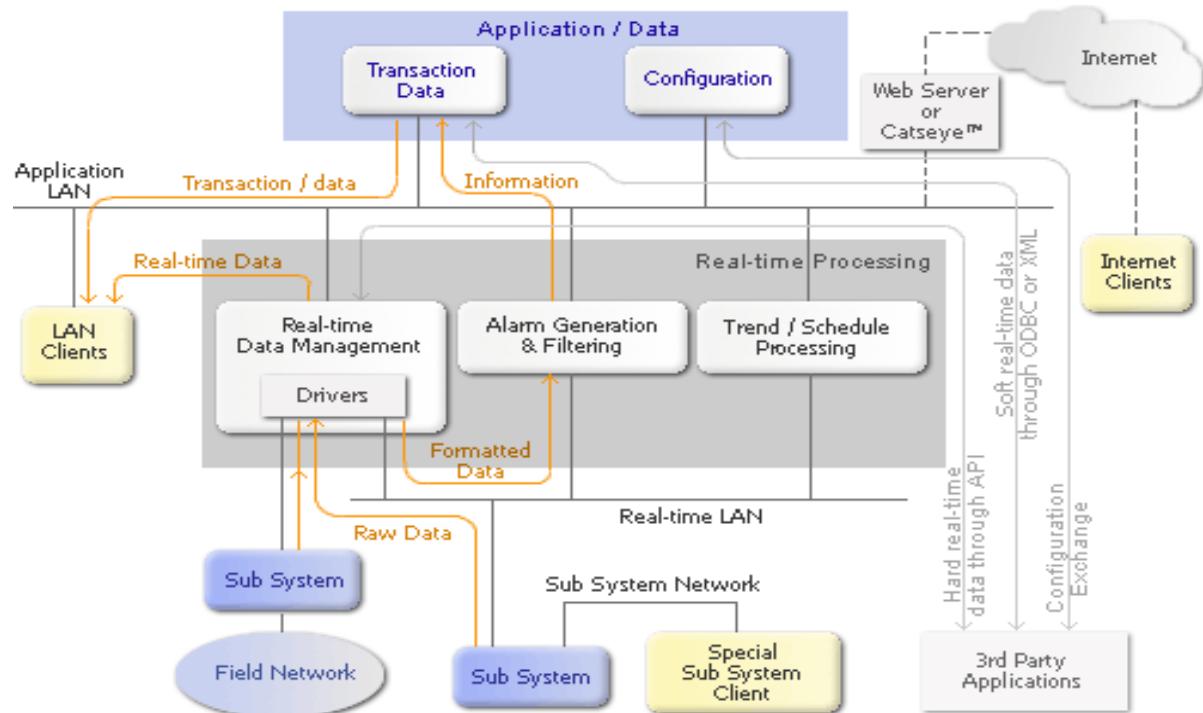
distantă, să închidă sau să deschidă vane sau întreruptoare, să monitorizeze alarme și să distribuie informații despre măsurători de la o unitate centrală la o instalatie distribuită în spatiu. Când dimensiunile unui proces devin foarte mari (sute , mii km), se pot aprecia beneficiile SCADA, în sensul reducerii costurilor vizitelor de rutină pentru urmărirea funcționării echipamentelor. Tehnologia SCADA se aplică cel mai bine proceselor care sunt distribuite pe suprafețe largi, sunt relativ simple de controlat și urmărit și care necesită intervenții frecvente, regulate sau imediate.

Exemple de astfel de procese pot fi următoarele:

1. Grupuri de stații hidroenergetice mici care sunt pornite și opuse ca răspuns al cererii de energie ai clientilor și care sunt localizate, în general, în locuri îndepărtate. Acestea pot fi comandate prin închiderea sau deschiderea unor vane a turbinei, trebuie să fie urmărite continuu și să răspundă rapid cererilor dispecerului energetic;
2. Zone de extractie petrol și gaze, incluzând sonde, sisteme de colectare, echipament de măsurare a debitelor și pompe. Sunt în general distribuite pe arii largi, necesită comenzi relativ simple, cum ar fi pornirea sau oprirea motoarelor, necesită centralizarea informațiilor metrologice în mod regulat și trebuie să răspundă rapid la condițiile restului exploatarii;
3. Retele de distribuție gaz, petrol, produse chimice, apă, care au elemente ce sunt localizate la diferite distanțe de la un punct central de control. Acestea pot fi comandate prin închiderea sau deschiderea unor vane sau pornirea și oprirea unor pompe și trebuie să răspundă rapid condițiilor pietei și pierderilor de materiale toxice sau periculoase;
4. Sisteme de transmisie electrice, care acoperă mii de km², pot fi comandate prin închiderea sau deschiderea întreruptoarelor și trebuie să răspundă aproape instantaneu de sarcina de pe linie;
5. Sisteme de irigații care acoperă sute de km², pot fi comandate prin închiderea sau deschiderea unor simple vane și necesită centralizarea informațiilor metrologice asupra cantităților de apă furnizate consumatorilor.

În afara acestor exemple de comenzi relativ simple, SCADA poate fi utilizată și pentru comenzi mai complexe, datorită evoluției tehnologiei. Pe liniile de transmisie, între unitățile îndepărtate și unitatea centrală SCADA, pot fi transmise bidirectional, atât informații binare cât și analogice (comenzi de pornire/oprire, comenzi pentru schimbarea pozitiei vanelor).

Un model functional generic pentru SCADA este redat în continuare, împreună cu structura hardware și topologia rețelei de comunicație industriale.



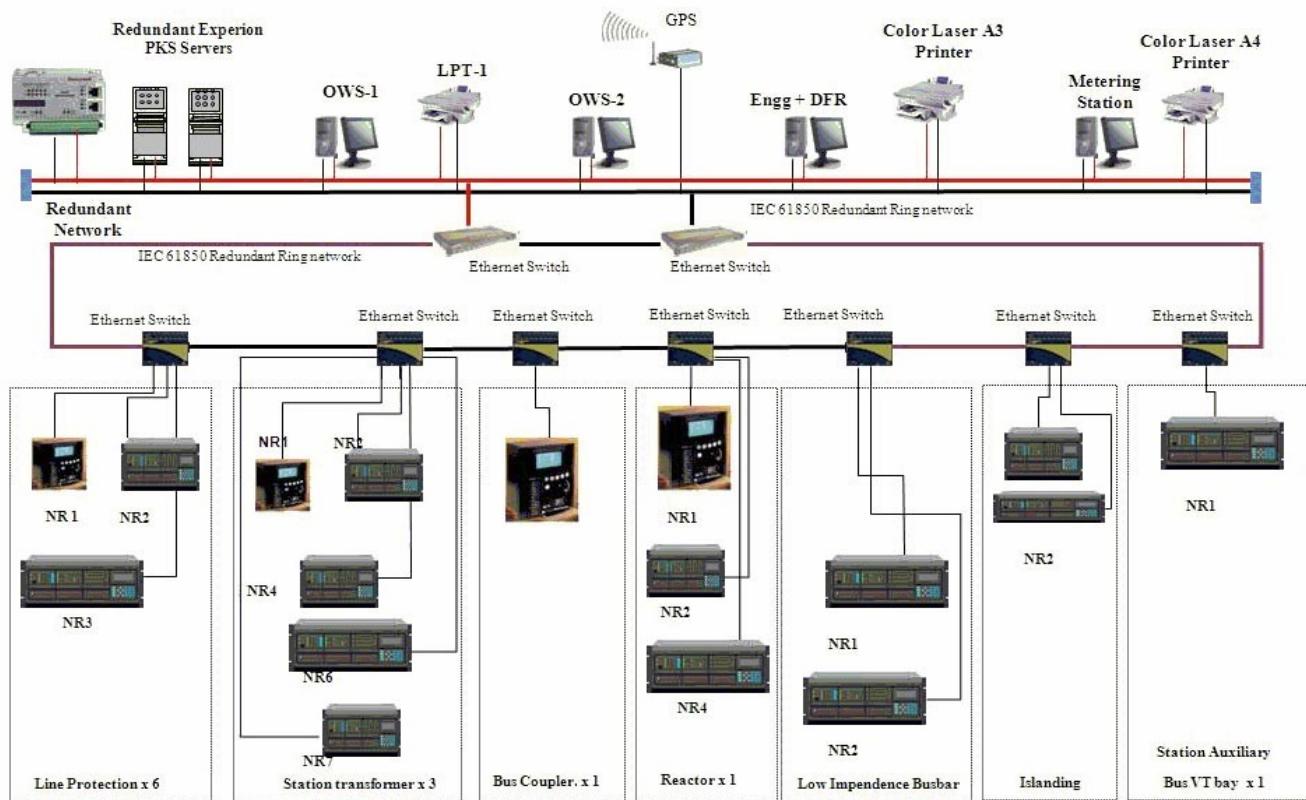
Se pot observa în figurile anterioare elementele constitutive ale unei arhitecturi **SCADA**. În centrul oricărei arhitecturi SCADA este un operator care accesează sistemul prin intermediul unui echipament interfăță numit "consola operator". Aceasta funcționează ca o fereastră a operatorului către proces. Consola operator este alcătuită din două elemente: o unitate display (VDU- Video Display Unit), care afisează date în timp real despre proces și o tastatură pentru a introduce comenzi operatorului sau mesaje către proces.

Accesul operatorului în sistem se face prin **MTU (Master Terminal Unit)** care este controlerul sistemului. Unele industrii folosesc termenul de „host computer”. Unitatea MTU este totdeauna bazată pe un calculator. Ea poate monitoriza și comanda instalatia chiar și când operatorul nu este prezent. Aceasta se poate realiza prin intermediul unor agende ce pot fi programate să repete anumite instrucțiuni, la intervale prestabilite. De exemplu, se poate programa să se ceară informații periodice de la **RTU (Remote Terminal Unit)**, ce sunt localizate departe de stația centrală. Un sistem SCADA poate avea de la câteva RTU la câteva sute. Comunicarea se poate face prin două modalități obisnuite: fie cu liniile terestre (cabluri de fibre optice sau electrice), care pot fi proprietatea firmei care utilizează sistemul SCADA sau pot fi închiriate de la companii telefonice și prin radio. Sistemele mari pot utiliza combinații de sisteme de comunicație radio și telefonice. În ambele cazuri, este necesar un **MODEM (Modularează și DEModulează semnale pe o portătoare)**.

În general, sistemele SCADA controlează procese simple, cantitatea de informații este mică și deci vitezele de transmisie sunt mici. Uzual, vitezele sunt de ordinul 300 bps (biti/sec). Sistemele electrice necesită viteze mai mari (2400bps) dar care sunt încă acceptabile pentru liniile telefonice obisnuite. În mod normal, unitățile MTU au echipamente auxiliare: imprimante, memorii tampon (backup memories). Acestea sunt considerate ca făcând parte din MTU. MTU poate comunica către nivele ierarhice superioare transmitând informații despre starea sistemului prin intermediul unor liniilor speciale sau, mai nou, prin retele **LAN (Local Area Network)**. În cele mai multe sisteme SCADA, MTU poate primi informații de la alte calculatoare. În felul acesta, programe care rulează pe alte calculatoare pot asigura un control supraveghetor al sistemului SCADA. Desi comenzi ce sunt date într-un sistem SCADA sunt, în general, limitate, această facilitate este caracteristica distinctivă a sistemelor SCADA. SCADA este un sistem bidirectional care permite nu numai să se monitorizeze o instalatie dar să se si actioneze asupra ei. Partea de control supraveghetor a unui sistem SCADA are această functie. Mult timp SCADA a functionat independent de alte sisteme numerice și faptul că era un sistem în timp real, nu era important. Din ce în ce mai mult apar sisteme SCADA care funcționează pe bază de program prestabilit sau pe bază de cerere. În prezent SCADA îmbină atât elemente în timp real cât și funcționare preprogramată. Pentru sistemele SCADA funcționarea în timp real înseamnă actualizarea comenziilor în funcție de schimbările din proces. În sens strict, **comanda în timp real** este aceea care **nu introduce timp mort** între receptia măsurătorilor din proces și semnalele de comandă. În realitate, toate sistemele de comandă introduc o oarecare întârziere. Acelea care introduc întârzieri fără nici un efect măsurabil sunt denumite, în general, sisteme în timp real. În opozitie cu acestea sunt sistemele preprogramabile (secventiale). Majoritatea sistemelor care controlează procese continui funcționează în timp real. De cele mai multe ori inertiale specifice sistemului sunt mult mai mari decât timpul de răspuns al sistemului de comandă, ceea ce face să se poată considera că aceasta se face în timp real. Un alt element important în proiectarea sistemelor SCADA îl reprezintă alegerea **perioadei de scanare** în funcție de necesitățile procesului și care trebuie făcută de specialisti care cunosc efectele întârzierilor din sistem. Viteza de reacție depinde de caracteristicile sistemului controlat putând fi de la ordinul secundelor, în cazul sistemelor energetice, până la ordinul orelor, în cazul câmpurilor de extractie petrolieră. Practic, timp real pentru sisteme înseamnă că, timpul de întârziere nu este atât de mare încât să introducă efecte evidente în control. Din acest motiv majoritatea sistemelor SCADA sunt considerate sisteme de comandă în timp real chiar dacă pot fi puse în evidență anumite întârzieri. Atât pentru MTU cât și pentru RTU, având la bază un sistem numeric și trebuind să comunice între ele, este importantă selectarea protocolului de comunicatie.

În sistemele SCADA cea mai frecventă metodă de comunicare este astăzi numita **MASTER-SLAVE**. În acest protocol o singura mașină (MTU) este capabilă să inițieze comunicare. MTU apelează un RTU, îi dă instrucțiuni, cere informații și comandă RTU să răspundă. Apoi MTU așteaptă răspunsul. RTU răspunde imediat ce MTU a terminat comunicare, apoi se opreste și așteaptă noi comenzi. MTU comută apoi la alt RTU și trece prin aceeași procedură. RTU nu poate iniția un mesaj; el poate trimite un mesaj doar dacă MTU i-a ordonat să facă asta. **Procesul de interogare** al fiecărui RTU în ordine, și apoi refăptărcerea la primul RTU este numit proces de scanare. Timpul de scanare nu trebuie să introducă întârzieri în comandă. În același timp există limitări din punct de vedere al vitezei de transmisie a datelor. Rezultă un interval optim de scanare a fiecărei unități RTU care depinde de numărul de RTU, ce vor fi scanate și de cantitatea de informații ce va fi vehiculată la fiecare scanare. **Cantitatea de informații depinde de gradul de independență în decizie al echipamentelor** și de complexitatea acestora. În continuare este redată o imagine amanuntita a unei configurații de rețea SCADA.

Typical SAS Architecture



În funcție de gradul de independentă în decizie al echipamentelor și de complexitatea configurației arhitecturii SCADA, datele pot fi de la un simplu semnal de stare (1, 2 biti) până la sute de asemenea semnale sau zeci de semnale analogice (8-16 biti). Pentru alegerea perioadei de scanare se tine seama de unitatea RTU care furnizează cea mai mare cantitate de informații. Dar și **comunicatia este bidirectională** și există intervale în care MTU cere fiecărui RTU informații, sau transmite informații fiecărui RTU. Al treilea factor de care depinde intervalul de scanare este viteza de transmisie. Aceasta poate fi redusă sau crescută pentru a se ajunge la un optim. Se pot defini două grupe de viteză de transmisie: prima, care este utilizată pentru liniile telefonice obisnuite și pentru **comunicatia radio UHF**, în gama 300,2400 bps; a doua, care se aplică la mediile de comunicatie specială: 19200,10mil. Bps. Un alt factor care determină intervalul de scanare este randamentul comunicatiei, care reprezintă raportul dintre timpul necesar transmiterii informației utile și timpul total de comunicatie. Diferențele dintre cei doi timpi sunt de cele mai multe ori evidente, cum ar fi adresele RTU care sunt continute în fiecare mesaj și care nu sunt purtătoare de informații. Alte componente sunt mai puțin evidente, cum ar fi codurile de detectare a erorilor sau timpii de pornire a elementelor de emisie radio, care pot lua mai mult timp decât mesajul. În cazul în care perioada de scanare determină funcționarea la limită a unor RTU, soluția este creșterea vitezei de transfer (de la 1200bps la 2400bps). Dublarea ratei de transfer nu va reduce la jumătate perioada de scanare deoarece randamentul comunicatiilor este o funcție neliniară de rata de transfer. Dacă este necesară reducerea mai drastică a timpului de răspuns se poate revizui metoda de comunicatie. Dacă se cunosc atributile fizice ale procesului ce trebuie controlat, inclusiv frecvența naturală maximă, atunci se poate determina frecvența de aliasing. Parametrii procesului trebuie esantionati la o frecvență mai mare decât cea de aliasing. Dacă frecvența de scanare este mai mare decât frecvența de aliasing a procesului, RTU va esantiona, va transmite datele către MTU, care va realiza calculele, iar rezultatele vor fi trimise înapoi către RTU și procesate. Pentru acele procese pentru care frecvența de aliasing este mai mare decât frecvența de scanare calculele trebuie făcute de către RTU, înainte de scanare. În funcție de valorile frecvenței de aliasing și de scanare se stabilește locația unde se vor face calculele (RTU sau MTU). Rezultă astfel și necesitățile hardware pentru RTU. În prezent, datorită reducerii dramatice a prețurilor sistemelor numerice de calcul, tendința este de a se utiliza în locațiile RTU calculatoare suficiente de puternice care să realizeze toate calculele, instrucțiunile către senzori și memorarea istoriei, în mod local.

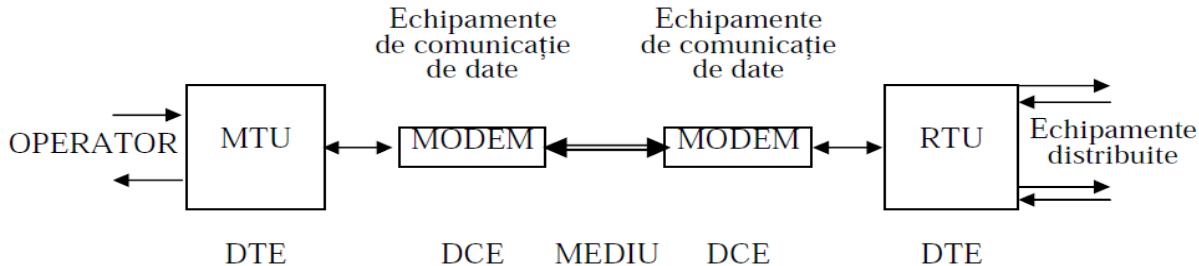
3. Comunicatiile in sisteme SCADA

Comunicarea este transferarea datelor (informatiilor) de la o locatie la alta. Pentru ca aceasta să aibă loc trebuie să fie îndeplinite trei conditii si anume, să existe calea de comunicare, să existe un echipament la capătul de emisie al căii de comunicare, ce să conditioneze datele si să le pună într-o formă transmisibilă, prin modul ales, să existe un echipament la capătul de receptie al căii de comunicare ce să extragă mesajul transmis si să-l utilizeze in luarea deciziilor.

Comunicatiile joacă un rol vital în functionarea sistemelor SCADA. Instalarea sistemelor SCADA este justificată în cazul locatiilor greu accesibile, sau accesibile cu costuri mari. În alte cazuri, se justifică dacă prezenta unei persoane la o anumită locatie este periculoasă, dăunătoare sănătății sau neplăcută. De cele mai multe ori este prea scump pentru a avea un operator la o anumită locatie, în mod permanent. Dacă se poate stabili o comunicatie între o locatie îndepărtată si o locatie centrală sau master, datele pot fi vehiculate. Dacă nu se poate stabili o linie de comunicatie, nu se poate dezvolta un sistem SCADA. Toate datele vehiculate între MTU si RTU sunt date binare. Ele pot fi sub această formă în mod natural (starea unui contact, închis/deschis) sau pot fi aduse în această formă în urma unei conversii din formă analogică.

4. Componentele generice ale sistemului de comunicatii SCADA

Urmatoarea figura evidențiază un sistem **SCADA** simplu format dintr-un MTU si un RTU si echipamentul de comunicatie corespunzător. În domeniul telecomunicatiilor MTU si RTU se numesc echipamente terminale de date (DTE). Ele pot comunica între ele prin intermediul MODEM-urilor care sunt numite echipamente de comunicatie de date (DCE). Acestea sunt capabile să adapteze informatie primită de la DTE-uri si să o transmită pe linia de comunicatie. La celălalt capăt al liniei de comunicatie DCE-ul reconstituie informatie primită si o transformă într-o formă compatibilă DTE-ului.



Într-un sistem SCADA mediul de comunicare este determinat de două criterii si anume, **viteza de transmitere si costul**. În cadrul sistemului SCADA pentru care sunt necesare intervale de scanare mici, datele trebuie să fie transmise cu viteza mare, peste 5000bps. În acest caz se utilizează ca medii de comunicatie cabluri de fibră optică, radio în microunde sau sisteme UHF mai sofisticate. Liniile telefonice închiriate pot fi de asemenea o posibilitate. Costurile sunt destul de mari însă pentru industriile care necesită un volum si viteză ridicată de date există această alternativă. Dacă intervalele de scanare sunt acceptabile si pot fi asigurate cu viteză de transmisie între 300, 4800bps variantele sunt mai numeroase. Pe lângă soluțiile amintite anterior, pot fi utilizate liniile telefonice obisnuite, care sunt proiectate să funcționeze în domeniul 300, 3400Hz sau echipamente radio ieftine destinate uzului comercial. În general cablurile telefonice au fost mediul de comunicatie preferat. Ele sunt atractive din punct de vedere al costului propriu-zis si al instalării tinând cont de faptul ca în prezent se realizează cabluri telefonice bine protejate împotriva umidității si a rozătoarelor. Ele au însă si dezavantaje, transmisia fiind influențată de liniile de înaltă tensiune (situate în paralel cu liniile de transmisie) si de efectele magnetice de joasă frecvență ale activității solare. Pentru locații îndepărtate, unde companiile telefonice nu au interesul să-si instaleze propriile liniilor, este posibil ca utilizatorul să plătească un pret ridicat pentru instalarea unei liniilor telefonice ce ar fi ulterior închiriată. În alte cazuri este posibil ca utilizatorul să-si instaleze propriul cablu. Fibrele optice au devenit competitive, din punct de vedere al costului, cu cablurile obisnuite de cupru, chiar si pentru legături care nu necesită viteză mare de transmisie. Toate cablurile au însă dezavantajul neflexibilității structurii sistemului SCADA. Pentru multe aplicații aceasta nu este o problemă. Echipamentele radio UHF au fost dezvoltate în special pentru SCADA. Ele oferă flexibilitate ridicată, cost scăzut si fiabilitate ridicată. Comunicatiile sunt cheia sistemelor SCADA. Ele depind, mai mult decât orice altă componentă a sistemului SCADA, de condițiile concrete în care este instalat sistemul. Rezultă

o atenție sporită acestui aspect în etapele initiale ale proiectării unui sistem SCADA. Deoarece înaltă frecvență UHF se transmite esențial în linie dreapta, este important să existe o vizibilitate directă între transmitător și receptor. Amplasarea antenelor se face în urma unui studiu al căii de comunicație, pe baza hărtilor topografice. Activitatea solară cu periodicitate de 11 ani, caracterizată prin furtuni și explozii solare, reprezintă cea mai puternică sursă de perturbatii radio. Frecvențele UHF și microondele (utilizate pentru viteze de transmisie foarte mare) sunt mai puțin influențate de aceasta. Cel mai bun mijloc de a minimiza efectele radio ale activității solare este asigurarea de echipamente în funcțiune și calibrate. Dezvoltările în electronică au determinat îmbunătățiri semnificative în tehnologia radio. Acestea se reflectă nu numai în gabaritul și consumul scăzut dar și în fiabilitatea și menținabilitatea sistemului. Noile echipamente sunt realizate pentru a compensa efectele variației temperaturii și pentru a avea o imunitate mult mai bună la vibrații și la variații ale sursei de alimentare. Totuși rămâne necesitatea efectuării întreținerii echipamentelor de comunicație radio, cum ar fi, realinierea antenelor deplasate de vânt, întreținerea acumulatorilor din componenta UPS (Uninterruptible power supplies = Surse neîntrerupte de energie), recalibrarea stațiilor radio. Sateliții geostationari reprezintă o importanță facilitate pentru realizarea comunicațiilor sistemelor SCADA. Dacă pentru unele aplicații poate fi doar o variantă, pentru altele, cum ar fi retelele de conducte sau retelele electrice în special în zonele îndepărtate sau slab dezvoltate, comunicația prin sateliți poate reprezenta cea mai ieftină metodă. În ceea ce privește costurile unei astfel de comunicații, ponderea cea mai mare o reprezintă antenele și echipamentul special radio. Există și o taxă lunară pentru utilizarea acestui serviciu. Nivelul ei este comparabil cu cel al oricărui alt mijloc de comunicație. Desi sistemele radio par să aibă multe dezavantaje ele oferă suficiente avantaje în comparație cu liniile terestre pentru a fi alese ca mediu de comunicare.

5. Tendinte în evoluția sistemelor SCADA

Tendințele de evoluție ale sistemelor SCADA se referă la cele trei componente principale, **comunicații, RTU și MTU**. În ceea ce privește comunicațiile dezvoltările din domeniul electronic vor permite realizarea unor echipamente mai mici și mai economice din punct de vedere al consumului de energie făcând posibilă integrarea lor, împreună cu modem-ul, chiar în RTU. Echipamentele radio pot fi completate cu **subsisteme numerice** care să realizeze **auto calibrarea** acestora rezultând o reducere a timpului de pornire a emițătoarelor până la ordinul sec și deci a perioadei de scanare. Referitor la calea de comunicație două sunt tendințele de evoluție. Una se referă la utilizarea sateliților geostationari pentru sisteme SCADA care sunt extinse pe suprafețe mari. Pot fi realizate stații de emisie portabile al căror preț va fi comparabil cu cel al telefoanelor mobile. A doua tendință se referă la **utilizarea comunicațiilor cu fibră optică**, ce oferă avantajele unei **viteze mari de transmisie, a securității și confidentialității sporite**. Acest mediu de transmisie este bine adaptat necesităților de comunicații din domeniul energetic. Evoluția unităților RTU va permite creșterea flexibilității acestora, configurarea pentru o anumită cerință făcându-se pe bază software. **Miniaturizarea și reducerea prețurilor calculatoarelor** personale au făcut ca acestea să devină comparabile cu stații RTU dedicate simple. Aceasta determină realizarea de **RTU** bazate pe calculatoare personale cu funcționalități variate, cum ar fi **regulatoare, totalizatoare, strategii de control** etc. Dezvoltările ulterioare ale unităților MTU sunt focalizate pe trei planuri: **îmbunătățirea interfetei operator** (interfete grafice, mediu windows, obiecte orientate, reprezentări grafice); **creșterea autonomiei de comandă** automată (sisteme inteligente autoinstruite); îmbunătățirea comunicației masină-masină (dezvoltarea retelelor LAN).

3. Descrierea modulelor hardware și software ale aplicatiei dezvoltate

Aplicatia dezvoltata isi propune implementarea unui convertor serial la Ethernet utilizand o platforma embedded NGW100 cu procesor AVR32. Pentru testare este necesar si un modul de IO care sa comunice pe linia seriala RS-232 informatie despre starea unei intrari analogice. Acest modul de test este o platforma cu MCU ATMega16. Prima parte a descrierii se refera la modulul IO distribuit cu ATMega16, urmand apoi descrierea detaliata a platformei gateway NGW100 impreuna cu functionalitatea aplicatiei dezvoltate.

1. Prezentarea modulului de dezvoltare și a MCU AVR ATmega16

Modulul de dezvoltare ales se bazează pe MCU AVR ATmega16 al producătorului Atmel. Acesta este un MCU de 8 biți bazat pe o arhitectură RISC avansată. Printre caracteristicile de bază ale MCU amintim un set puternic de 131 de instrucțiuni care se execută într-un singur ciclu de ceas (majoritatea), 32 de registre de uz general, 64 de registre dedicate pentru modulele I/O și o putere de calcul de 16 MIPS la 16MHz.

Spațiul de memorie se împarte în două, un spațiu de memorie program (16 KB memorie Flash In-System Programming On-Chip, cu 10000 cicluri W/E) și un spațiu de memorie de date (1KB SDRAM și 512 K memorie EEPROM cu 100.000 cicluri W/E).

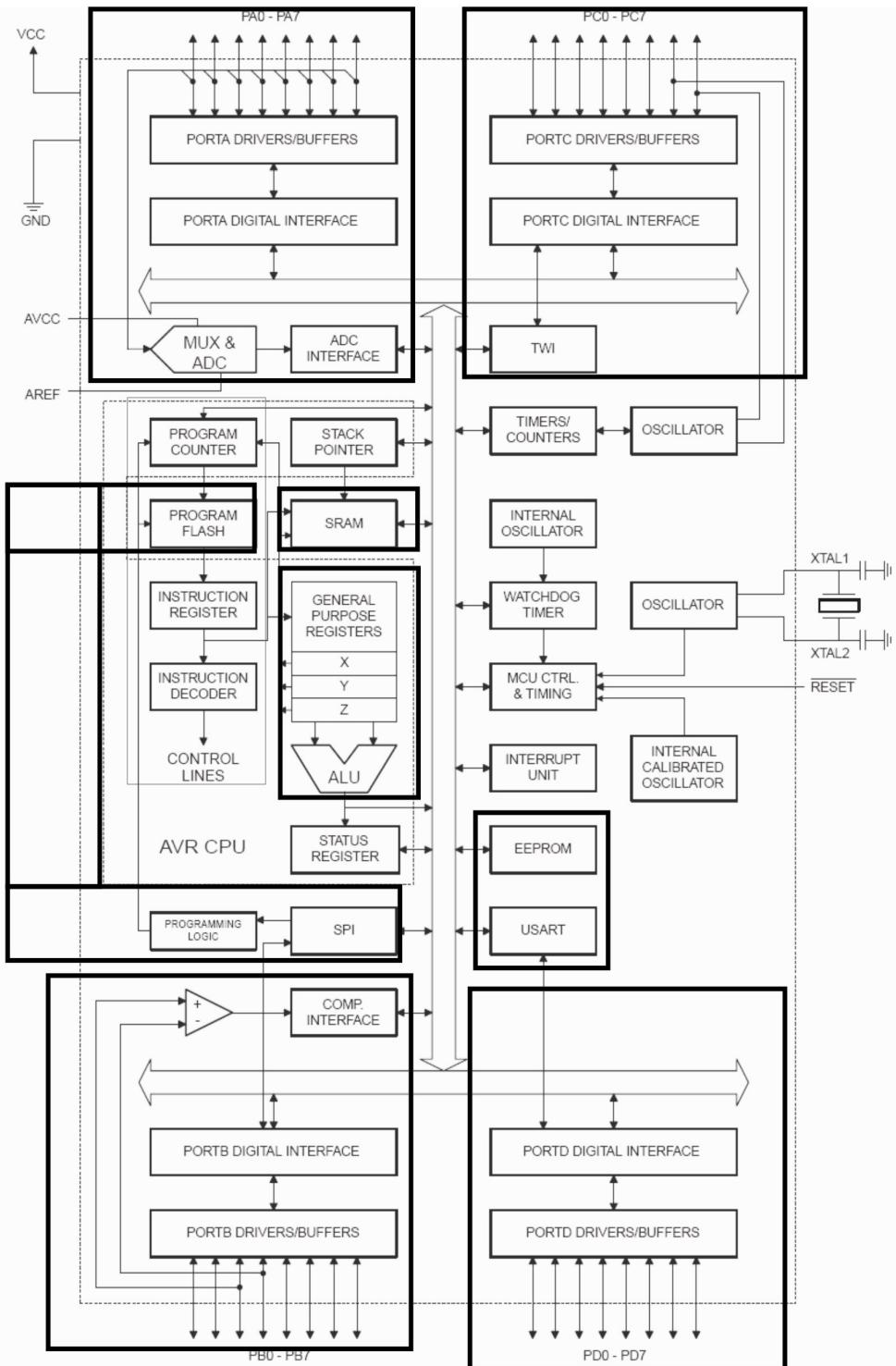
Din punctul de vedere al caracteristicilor perifericelor amintim prezența a două timere de 8 biți cu unități de prescaler și comparare separate și un timer de 16 biți cu moduri de comparare și captură, un counter real-time, 4 canale PWM, un CAN de 10 biți cu intrări multiple, interfață serială programabilă USART, timer watchdog, oscilator și comparator analogic on-chip. MCU ATmega16 prezintă 32 de linii de I/O digitale grupate în 4 porturi de 8 biți, PORTA, PORTB, PORTC, PORTD. Cele 4 porturi sunt conectate la o magistrală generală de date alături de registrele de uz general, ALU, SRAM, EEPROM, logica de întrerupere, timere și o interfață SPI, fiecare cu interfețe specifice. O caracteristică demnă de luat în seamă este prezența unei magistrale de date dedicată/separată între memoria Flash, unitatea de extragere instrucțiuni, unitatea de decodificare și cea de execuție, specifice arhitecturilor Harvard.

Funcționalitatea porturilor este descrisă în continuare întrucât este importantă în tema de dezvoltare propusă. PORTA deservește ca intrări analogice pentru CAN de 10 biți cu intrări multiple. În mod general, când nu se utilizează CAN deservește un port bi-directional generic de 8 biți, optional prevăzut cu rezistori pull-up. PORTB deservește generic un port bi-directional de 8 biți și este utilizat pentru funcții speciale ale MCU. Deasemenea este utilizat la modul de lucru output compare al timerului, întrucât este conectat la un comparator analogic care interfețează cu magistrala generală de date. PORTC deservește un port bi-directional de 8 biți și un buffer cu amplificare fiind utilizat în implementarea funcțiilor interfeței JTAG și a altor funcții speciale ale MCU. PORTD are același specific de funcționare ca port de 8 biți bi-directional ca și celelalte porturi, însă spre deosebire de celelalte logica de interfațare cu magistrala generală de date, comunică și cu modulul USART de comunicație serială. Modulul de dezvoltare prezintă circuite de interfață pentru liniile I/O digitale bazate pe ULN2803S (interfață pentru liniile de ieșire, pentru curenti mari, conține 8 drivere Darlington) și două optocuploare CNY74-4 (ce asigură izolarea masei MCU de masele circuitelor de intrare).

Circuitele de interfață pentru liniile analogice se bazează pe TL084 (care conține 4 AO), ICL7660 (convertește tensiuni pozitive în negative, conține un regulator DC, un oscilator RC, un translator de nivel de voltaj) și stabilizatorul UA7805. Partea de comunicație a modulului este destinată ambelor standarde RS232 și RS485 fiind asigurată de existența circuitelor MAX232 (pentru RS232) și SN75176 (pentru RS485).

Instrumentul ales pentru programarea și dezvoltarea sistemului este compilatorul CodeVision AVR de la HPIInfoTech(Windows) sau Kontrollerlab (Linux), întrucât oferă o interfață accesibilă de programare, precum și instrumente puternice pentru dezvoltarea de aplicații pentru Atmega16 precum și pentru întreaga familie de MCU AVR.

Descrierea detaliată a modulului de dezvoltare și a MCU este redată în continuare, urmărind elementele arhitecturale ale MCU și circuitele de interfațare, necesare în proiectarea sistemelor de control.



MCU AVR Atmega16 prezinta 4 moduri de lucru auxiliare pe langa cel activ :

Modul power-down : salveaza continutul regisrelor, blocheaza oscilatorul si celelalte functii pana la aparitia urmatoarei intreruperi externe sau RESET hardware

Modul power-save : timerul asincron isi continua functionarea, pentru mentinerea unei baze de timp in timp ce dispozitivul este oprit

Modul standby : oscilatorul functioneaza in timp ce restul dispozitivului este inactiv, obtinandu-se un consum redus de energie

Modul extended standby : cand atat oscilatorul cat si timerul continua sa functioneze

Atmega16 e un MCU RISC care prezinta o arhitectura flexibila si rapida pentru dezvoltarea de aplicatii diverse.

Specificul RISC rezulta din faptul ca memoria Flash de program impreuna cu blocurile de instruction fetch, decoding si execution comunica printr-un bus propriu separat de bus-ul general de date.

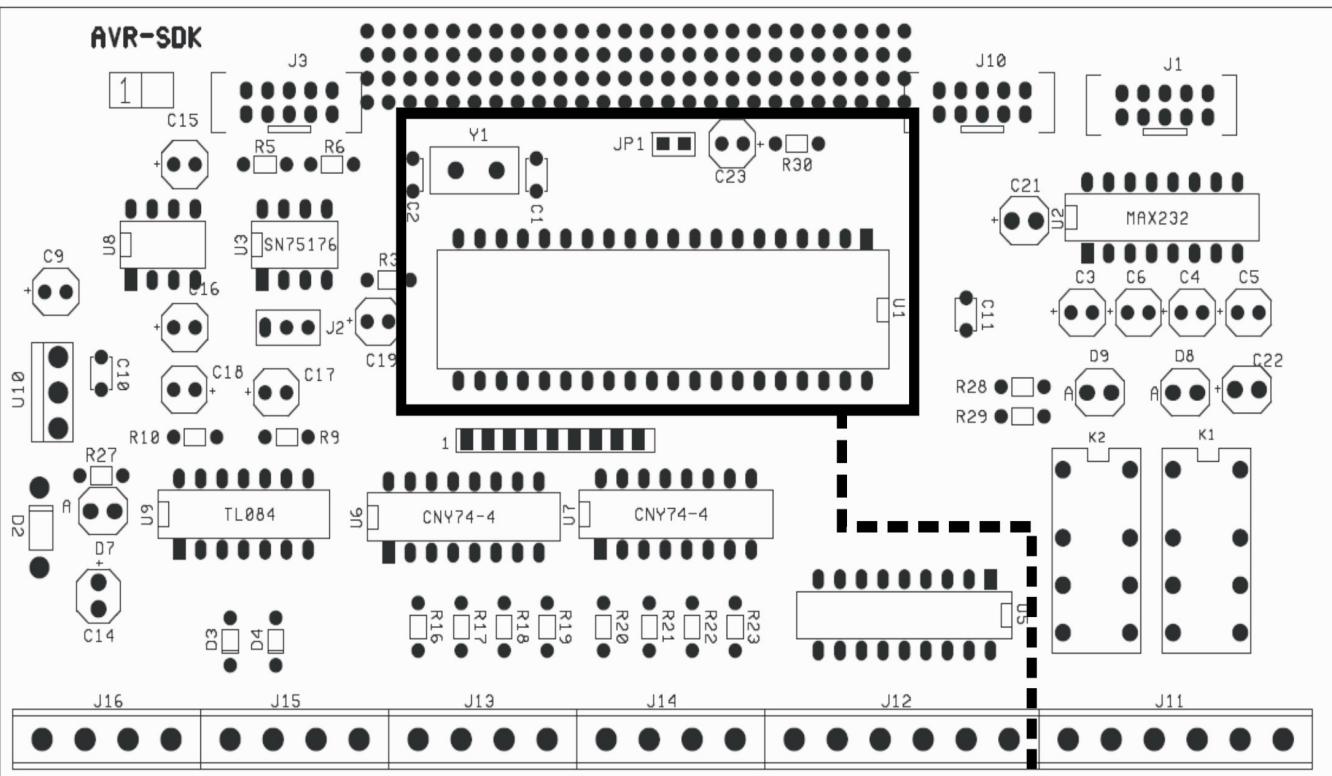
In componenta sa distingem urmatoarele subsisteme:

- **16KB memorie Flash pentru stocarea programelor (programata pe serial prin SPI sau printr-un program de boot on-chip)**
- **1KB de memorie RAM**
- **512B memorie EEPROM**
- **doua timere de 8 biti (cu counter, prescaler si modul de output compare incluse)**
- **un timer de 16 biti (counter, prescaler si module de output compare si input capture incluse)**
- **un CAN de 10 biti cu intrari multiple**
- **un comparator analogic**
- **un modul USART pentru comunicatie seriala**
- **dispune de un cronometru cu oscilator intern**

- 32 de linii i/O organizate in 4 porturi PA, PB, PC, PD

- Prezinta o magistrala generala de date la care se conecteaza mai multe module:

module: ALU, cele 32 de registre de uz general, memoria RAM si EEPROM, liniile de intrare si celelalte blocuri de I/O care sunt controlate de un set special de registre, fiecare modul avand un numar specific de registre (in total 64)

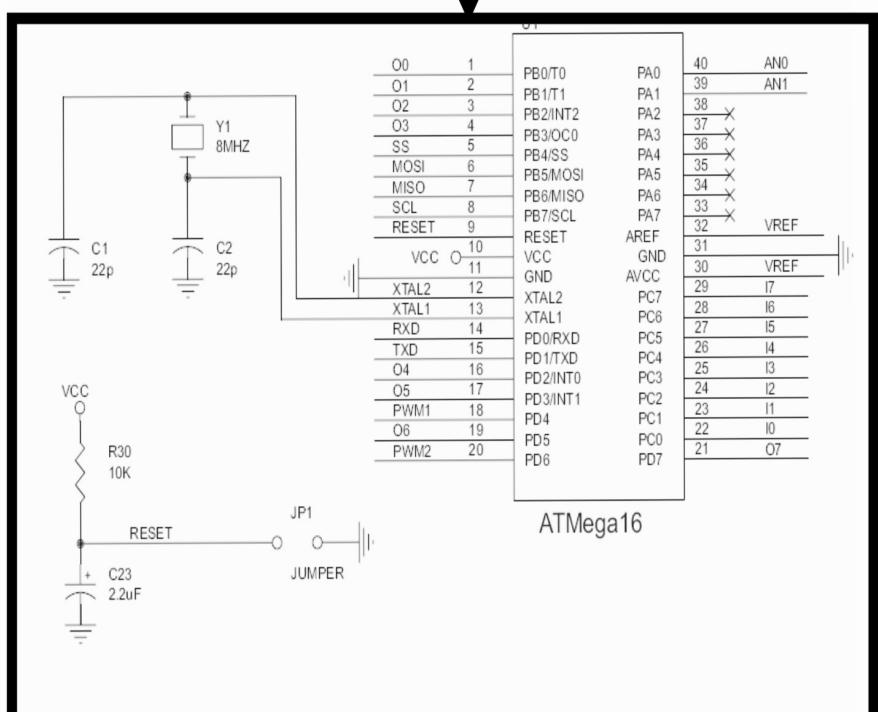


Prezentarea de ansamblu a modulului de dezvoltare cu MCU AVR Atmega 16, reprezentarea MCU.

Circuitul de RESET este alcătuit dintr-un resistor pull-up (R30) un condensator (C23) și un jumper pentru scurtcircuit.

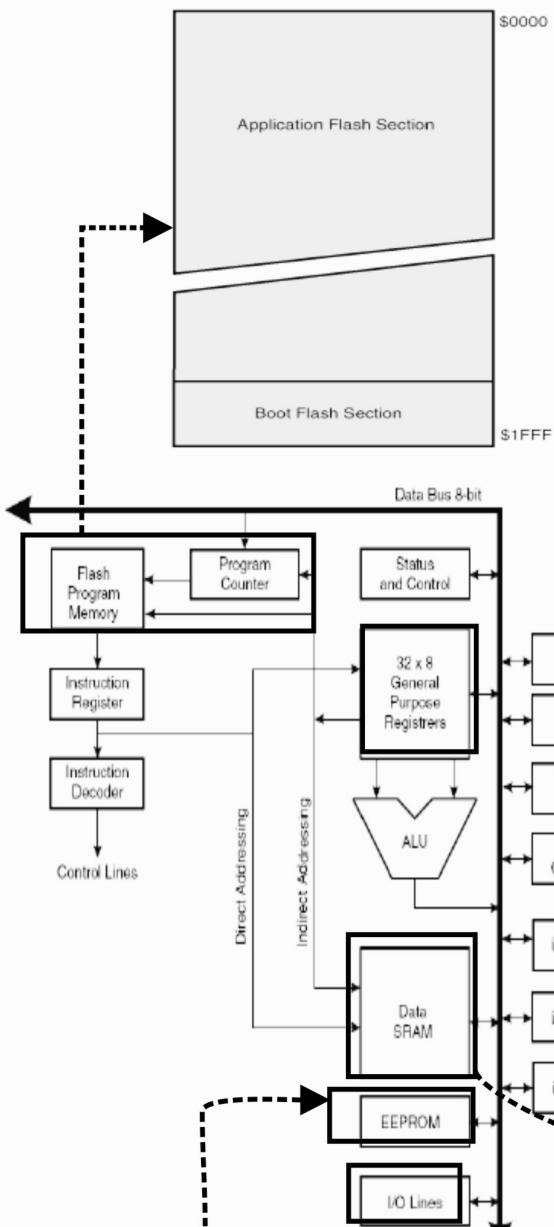
Pinul de RESET(activ L)este conectat in interiorul MCU la subsistemul MCU CTRL & TIMING, care este conectat la magistrala generala de date.

Circuitul de generare a CLK este alcătuit dintr-un oscilator de 16MHz (Y1) și două condensatoare (C1, C2).



Organizarea Memoriei

Program Memory Map



Memoria EEPROM are o dimensiune de 512B cu o capacitate de 100.000 cicluri citire/scriere, fiind organizata ca spatiu separat de date in care pot fi scrisi si cititi biti individuali. Registrii de acces EEPROM sunt in spatiu I/O, la citirea EEPROM CPU intra in halt pentru 4 perioade de CLK inainte de executia comenzi urmatoare, iar la scrierea in EEPROM CPU este oprit timp de 2 perioade de CLK. Cronometrarea acceselor EEPROM este realizata de oscilatorul calibrat.

Spatiul de memorie I/O

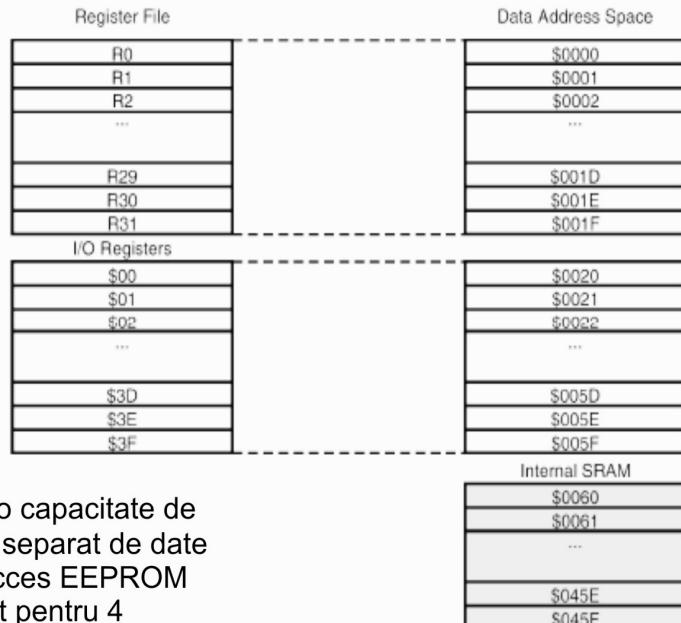
Toate I/O-urile de la ATmega 16 si perifericele sunt plasate in spatiu I/O. Locatiile I/O sunt accesate de catre comenzi IN si OUT , transferand datele dintre cei 32 de registrii de lucru si spatiu I/O. Registrii I/O cuprinsi intre valorile adreselor \$00 - \$1F sunt direct accesabile. La aceste registrii valoarea bitilor unici poate fi verificata.Cand se utilizeaza instructiunile specific IN si OUT , trebuie folosite adresele I/O din zona \$00 - \$3F. Cand se adreseaza registrii I/O ca spatiu de date cu instructiunile LD si ST , trebuie adaugat la aceste adrese, \$20, adica salt peste zona regisrelor de uz general.

MCU Atmega16 are doua spatii de memorie principala, un spatiu pentru memoria de date si un spatiu pentru memoria de program. Memorarea datelor se face pe suportul oferit de o memorie nevolatila EEPROM. Cele trei tipuri de memorie sunt cu adresare liniara.

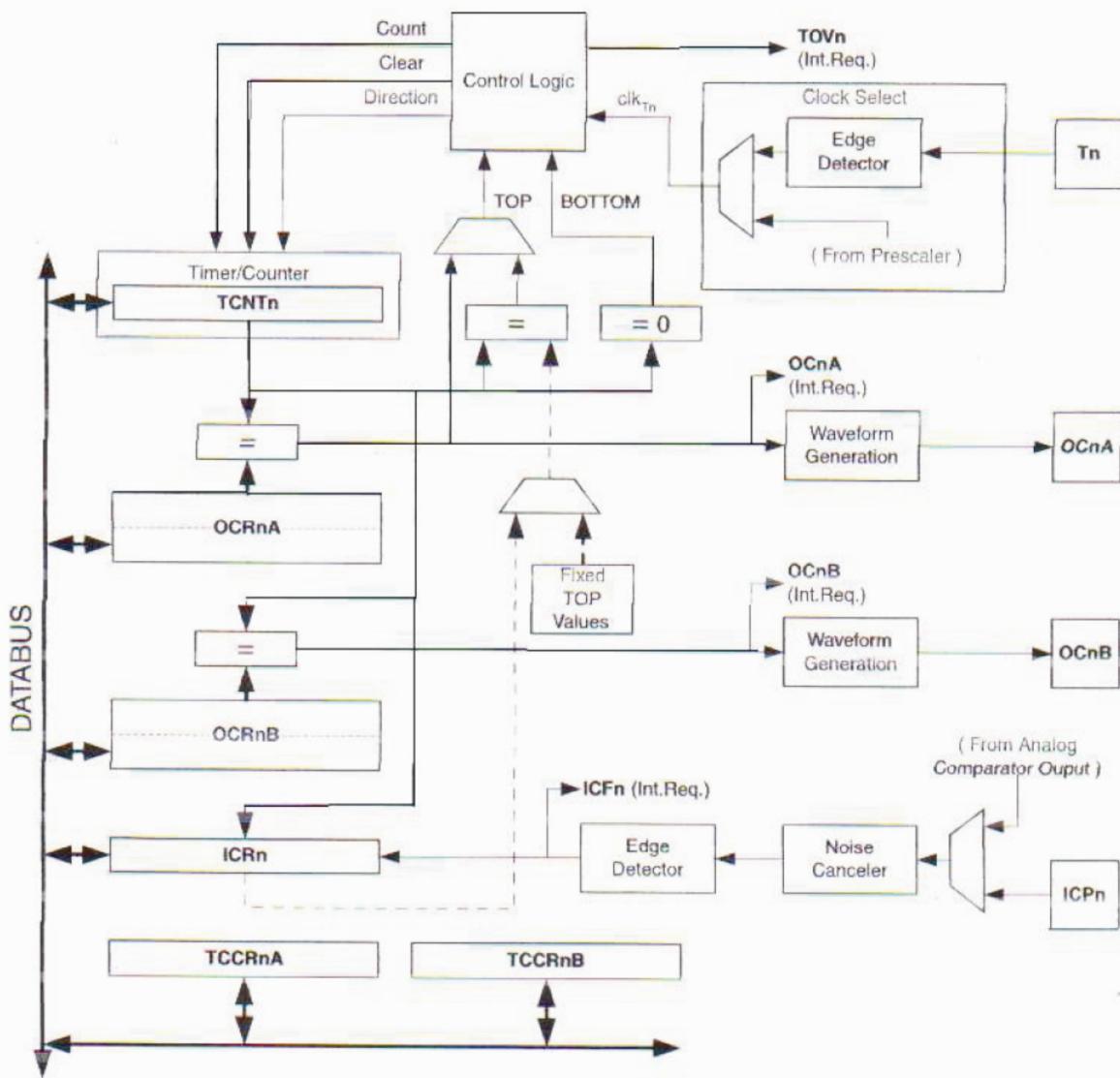
Memoria Flash reprogramabila (In-System On-Chip Self-Programmable Flash) de 16 KB este organizata ca 8Kx16, deoarece toate comenziile pentru AVR sunt de 16 si 32 de biti. Pentru securitatea softwareului spatiul de memorie de program Flash e impartit in doua sectiuni: sectiunea de program aplicatie si sectiunea de program boot, care contine biti de lock independenti si cu capacitatea programarii in-system prin executia de operatii Read-While-Write. Memoria Flash are un numar de 10.000 de cicluri de citire/scriere asigurate.

Modurile de adresare specifice modulelor MCU sunt prezentate mai jos, caile de date fiind usor de urmarit.

Memoria de date SDRAM este organizata astfel incat primele 96 de locatii determina fisierul de registre, iar urmatoarele 1024 de locatii se refera la datele interne ale SDRAM.



Unitatea Timer de 16 biti



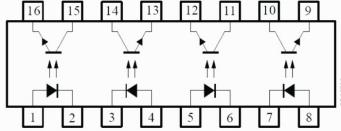
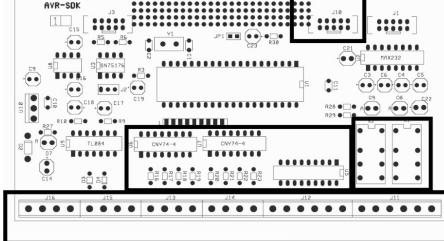
Timerul de 16 biti face posibila cronometrarea precisa a executiei programului (managementul evenimentelor, masura duratei sau numararea evenimentelor asincrone exterioare), generarea de forme de unda , masurarea timpului semnalelor. In cea mai mare parte MCU sunt prevazute cu timere de uz general pentru a indeplini functii ca generarea ceasului real-time, generarea semnalelor PWM sau functia de watchdog.

Subsistemu timer generic prezinta 4 moduri de operare:

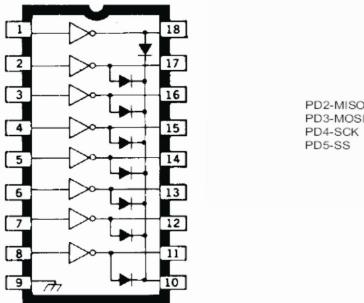
- + Timer Overflow - evenimentul specific fiind ajungerea TCNT la valoarea maxima de numarare si resetarea in urmatorul ciclu de ceas; se genereaza o IRQ catre CPU.
- + Input Capture - continutul TCNT este transferat in ICR(input capture reg) la aparitia unui eveniment exterior definit de un front al semnelului de intrare, lansandu-se o IRQ; ofera posibilitatea determinarii intervalului de timp dintre doua evenimente.
- + Output Compare - continutul TCNT este comparat continuu cu continutul OCR(output compare reg), cu ajutorul unui comparator digital, la potrivirea continutului se genereaza o IRQ si se poate modifica starea unor linii de iesire.
- + External Event Count - TCNT este conectat la un pin al MCU, numarand pulsurile asociate evenimentelor externe, oferind posibilitatea de a determina numarul de evenimente exterioare.

Caracteristicile Timerului de 16 biti:

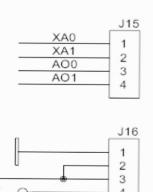
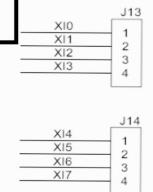
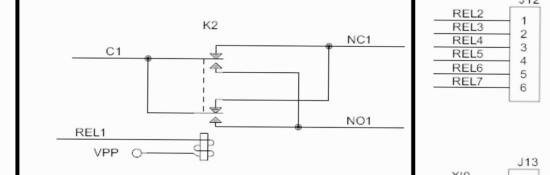
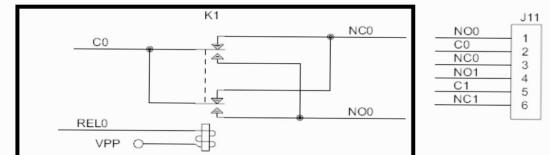
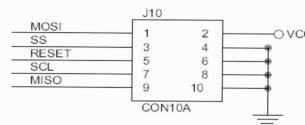
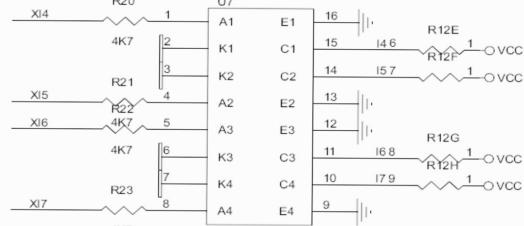
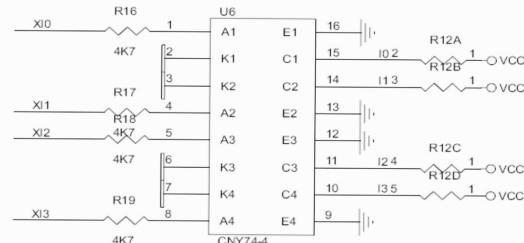
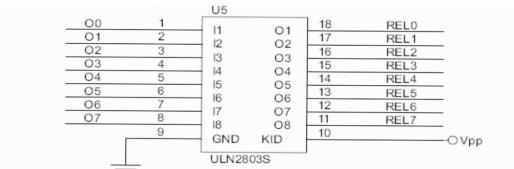
- posibilitatea de a genera semnale PWM de 16 biti (Design True 16 bit)
- prezinta doua unitati independente de output compare si o unitate input capture
- anulator de zgomot pe intrare, auto-reloading, 4 surse de interruperi independente
- generator de frecventa, puls de corectare a fazelor cu modulator (PWM), perioada PWM variabila



CNY74-4 - asigură izolarea masei MCU de masele circuitelor de intrare.

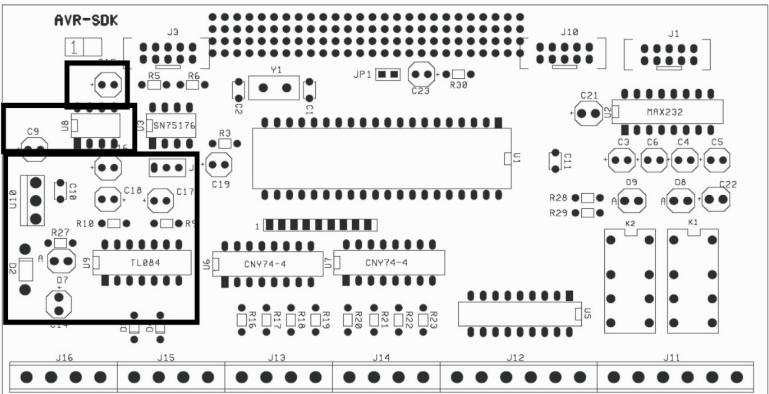


ULN2803S - interfață pentru liniile de ieșire, pentru curenti mari, conține 8 drivere Darlington.

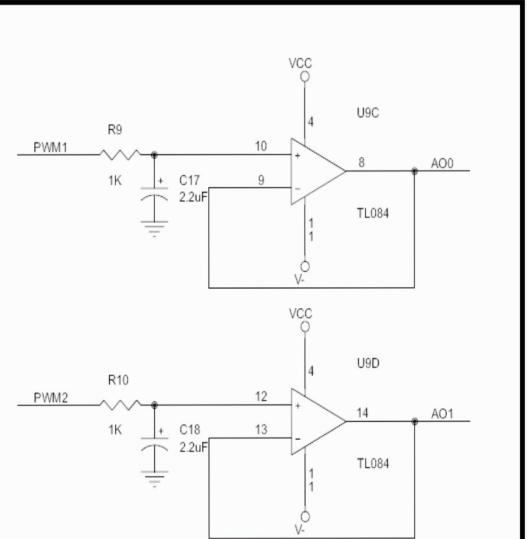
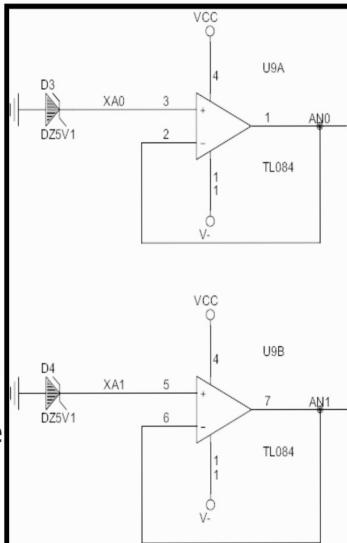
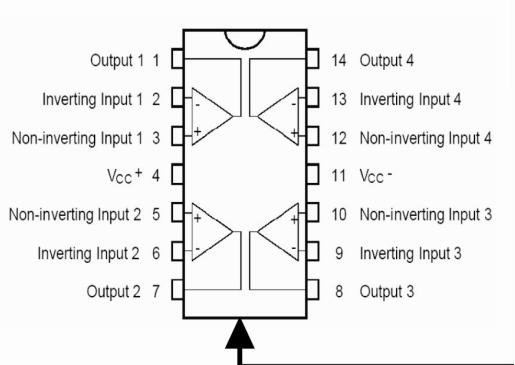


Releele (K1, K2) sunt conectate la liniile de ieșire pentru a asigura protecție. Intrucat la RESET toate liniile de intrare potentialul la intrarea circuitului de comandă a releeului este nedeterminat și astfel determină activarea acestuia. Dacă curentul sau tensiunea sarcinii sunt mai mari decât posibilitatele MCU se impune utilizarea unor circuite de buffer între circuitul de control și sarcina.

Circuitele de interfata pentru liniile I/O digitale

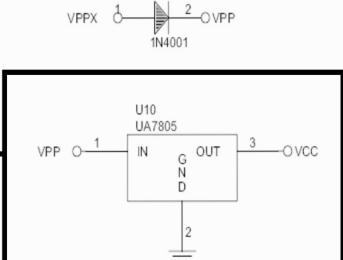
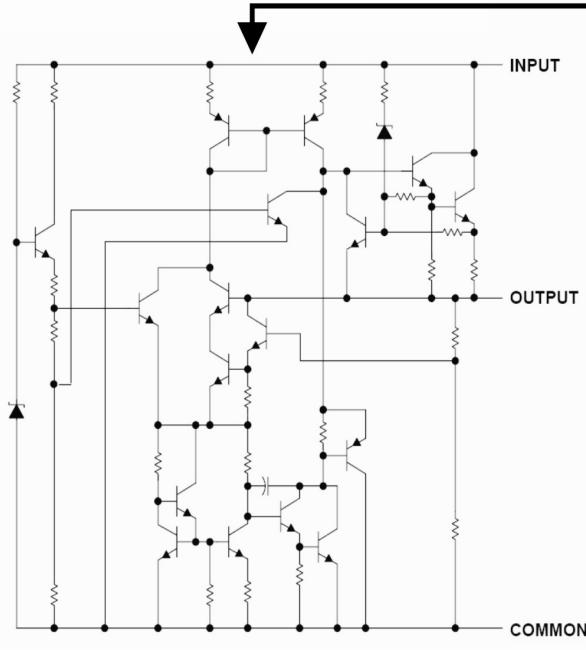


Circuitele de interfatare pentru liniile analogice

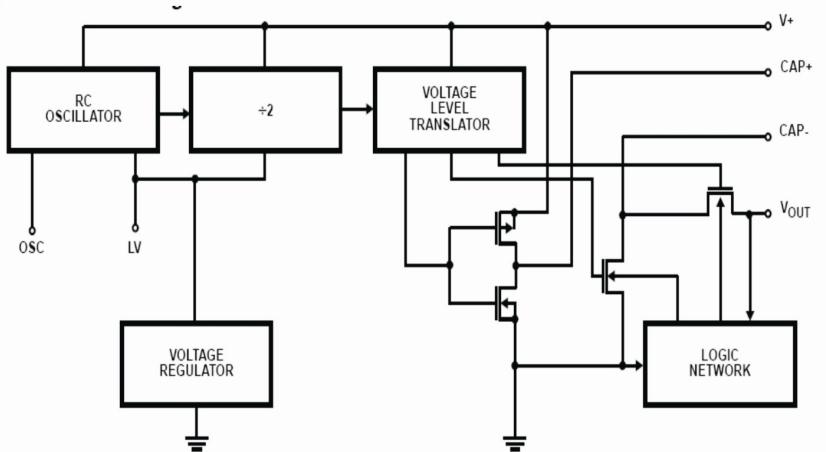


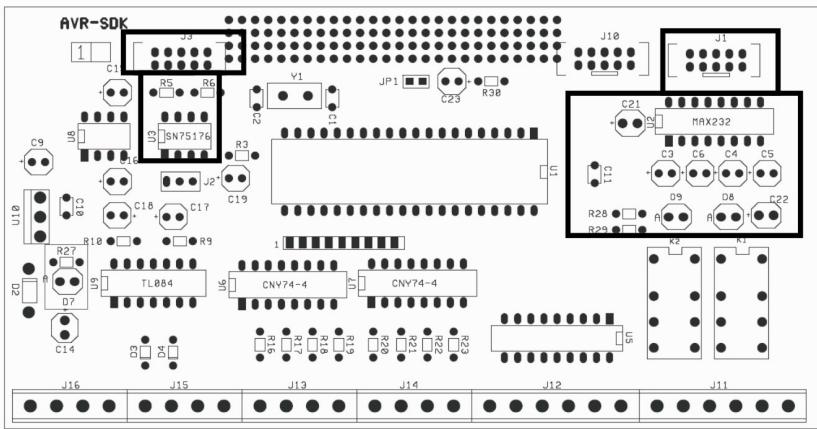
TI084 - circuit integrat monolithic care este alcătuit din 4 AO J-FET care contin transzistoare J-FET si bipolare de tensiuni mari, oferind o panta scazuta a semnalului la intrare si curent de offset scazut.

UA7805 - este un stabilizator de tensiune utilizat pentru eliminarea zgomotului si a problemelor de distributie din semnalul util.

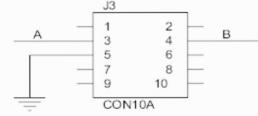
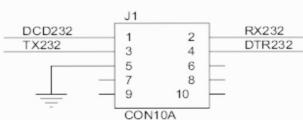
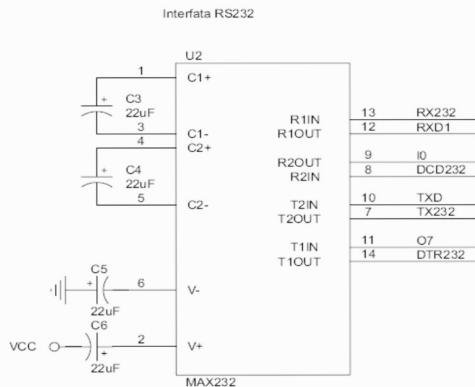


ICL 7660 MTV - este un convertor de tensiune CMOS



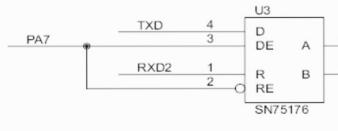
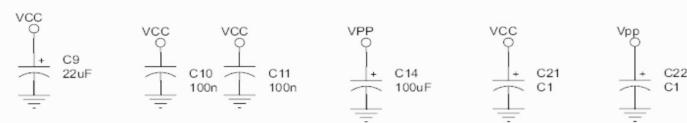


Circuitele de interfata pentru comunicatii RS 232 si RS 485

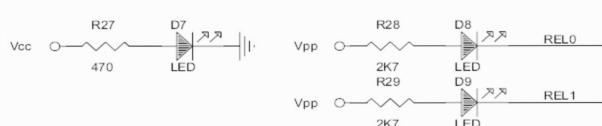
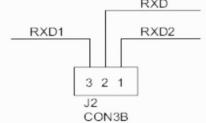


Conector RS232

Conector RS485



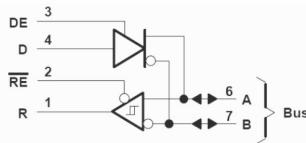
Interfata RS485



Circuitul MAX 232 realizeaza interfata hardware pentru comunicatia seriala pe distante relativ mari in medii industriale ostile fiind proiectat sa suporte comunicatii prin protocolul RS 232 si V.28. Circuitul contine doua transmitatoare si doua receptoare RS 232 in aceeasi capsula, circuitul realizand corespondenta intre tensiunile de la intrare intre 0-5V si cele de la iesire +/- 12V. Tensiunea de iesire de +/- 12 V este generata prin utilizarea unui oscilator intern si a 4 condensatoare externe pornind de la tensiuni de +5V.

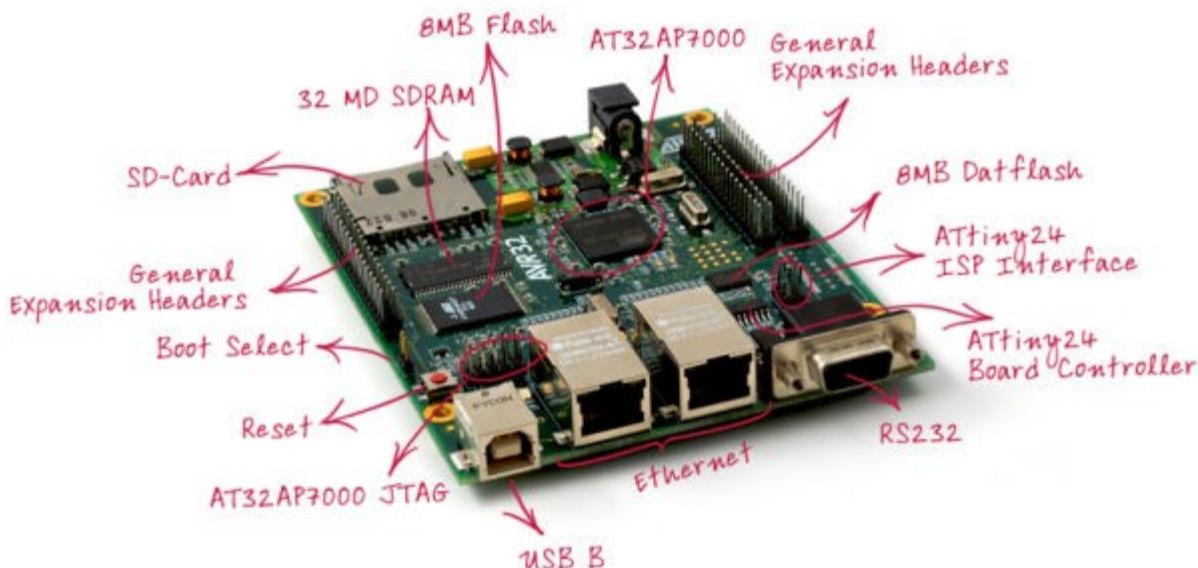


Circuitul SN75176 realizeaza interfata hardware pentru comunicatia seriala diferentiala prin protocolul RS485; acesta permite o comunicatie multipoint, half-duplex. Circuitul contine in aceeasi capsula un transmitator diferential si un receptor diferential, avand intrari de activare individuale.



2. Prezentarea modulului de dezvoltare NGW100 cu MPU AVR32

NGW100 sau Network Gateway 100 este un sistem de calcul embedded complet functional care demonstreaza capabilitatile procesorului pe 32 de biti AP7000 din familia AVR32 de la Atmel. Placa este dotata cu o memorie flash de 16MB si SDRAM de 32MB. Memoria poate fi extinsa prin conectarea unui card SD/MMC intr-un slot alocat. Conectarea la retea este posibila prin oricare dintre cele doua porturi Ethernet sau prin portul USB B (device). O consola este disponibila pe portul RS232 de pe placa. Prin cele 3 header-e de expansiune ale placii se pot interfata mai multe periferice ale procesorului, cum ar fi: UART, PS/2, ABDAC (convertor digital/analogic), ISI (Image Sensor Interface - pentru camera video), LCDC (periferic pentru LCD), EBI (magistrala externa), si multe altele. Bootloader-ul folosit in acest sistem (U-Boot) ofera posibilitatea incarcarii kernel/sistemului de pe retea, prin linia seriala, de pe flash sau de pe cardul MMC. Configurari ale bootloader-ului se pot face atat la programarea initiala cat si printr-o consola (pe RS232). Odata incarcat, sistemul de operare Linux de pe NGW demonstreaza capabilitatile complete ale procesorului in materie de comunicatii la viteze inalte.



Nucleul AVR32 pe 32 de biți de la Atmel a fost introdus prin intermediul microcontrolerelor UC3 bazate pe tehnologia Flash și al procesoarelor de semnal AP7. Nucleul AVR32 este caracterizat în primul rand prin cel mai bun raport performanță/consum de energie din gama tehnologiilor pe 32 de biți existente. Arhitectura lor se bazează pe o memorie Flash de până la 512 KB cu multe interfețe de comunicații, precum și pe procesoare de înaltă performanță cu unități de gestionare a memoriei și cu cash-uri speciale pentru sisteme de operare Linux dedicate.

Familiile SAM-ARM de la Atmel acoperă o gamă largă de aplicații cu suport pentru Windows CE (versiune Windows pt sistemele dedicate multimedia : PDA, telefoane mobile) și Linux dedicat. Un set bogat de periferice de comunicații, multe implementări inteligente și diferite instrumente de dezvoltare și sisteme de operare disponibile pe piață fac posibila utilizarea familiei SAM-ARM în aplicații precum : terminale POS (point of sale), sisteme de securitate, control industrial, PC-uri, telefoane 3G, PDA-uri, aplicații medicale etc.

Acesta familia face trecerea de la microcontrolere catre procesoarele specializate și sistemele on Chip folosite pentru sistemele dedicate.

De obicei, producătorii de chip-uri maresc puterea de procesare prin mărirea vitezei procesoarelor. Aceasta este o problemă reală pentru dispozitivele portabile pentru că o mărire a frecvenței de tact mărește în mod direct consumul de energie și reduce durată de viață a bateriei. Strategia folosită de Atmel cu AVR32 este de a mări cantitatea de operații pe care procesorul le poate face intern și de a micșora frecvența de tact.

Arhitectura nucleului AVR32 este optimizată pentru a avea cea mai mare capacitate de calcul în raport cu un

consum redus de energie. Majoritatea arhitecturilor RISC risipesc cicli de procesor pentru operații non-productive cum ar fi:

- încărcarea, stocarea sau mutarea de date,
- încărcarea de date care nu sunt în cache
- încărcarea de date care așteaptă să se termine o instrucțiune multi-ciclu.

Aceste operații nu contribuie la execuția efectiva a aplicației.

Caracteristicile nucleului AVR32

Au fost făcute multe îmbunătățiri pentru a crește performanțele în ansamblu ale nucleului:

- Accumulator-Cache, o nouă metodă patentată de Atmel, a fost dezvoltată o procedură care însumează și transfera datele într-un singur ciclu de ceas, fără a utiliza un registru adițional. Accumulator-Cache este utilizat pentru a stoca valoarea care trebuie adăugată la rezultatul multiplicării.
- Pipeline-ul mai suportă și „Data Forwarding”. Toate instrucțiunile completate vor fi trimise la începutul pipeline-ului pentru a fi utilizate pentru instrucțiunile în așteptare fără a fi nevoie de cicli adiționali.
- Cu instrucțiunile SIMD (o singură instrucțiune/date multiple) procesarea datelor de către unii algoritmi PDS poate fi accelerată considerabil.
- În astfel de arhitecturi, tehnica „predictiei saltului” este folosită cu succes. Avantajul pipeline-urilor adânci este clar la frecvențe mai mari, dar acestea pot pierde din performanțe când o instrucțiune de salt este executată, iar pipeline-ul trebuie reîncărcat. În special în cazurile în care buclele mici și izolate trebuie executate, eficiența unui pipeline este redusă dramatic.

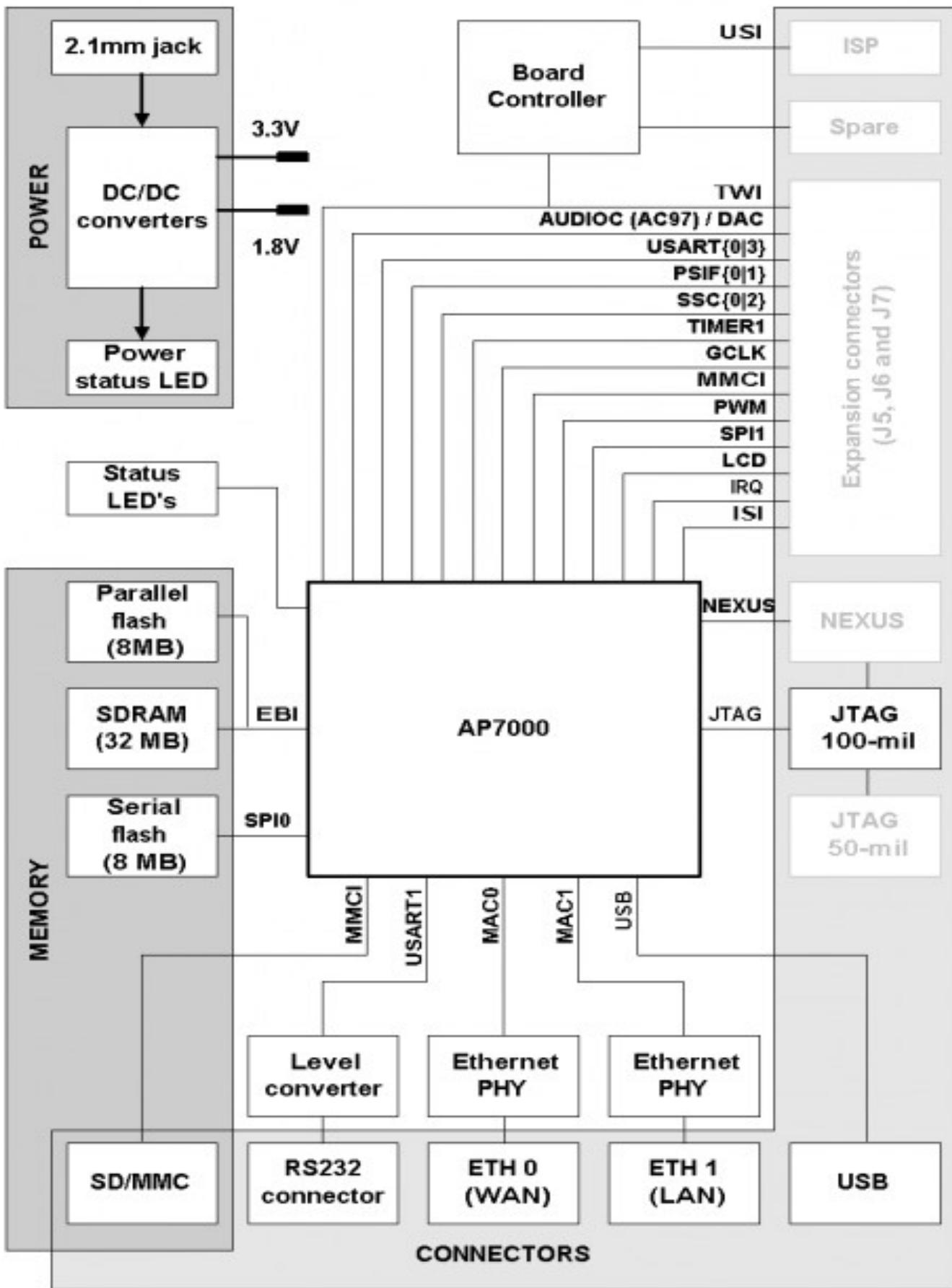
Revenind la modulul NGW100 se poate aminti ca acesta utilizează microcontrolerul AT32AP7000 care combină noul procesor digital de semnale AVR32 cu o selecție variată de interfețe de comunicații.

NGW100 poate fi folosit aplicatii in rețea și oferă următoarele caracteristici:

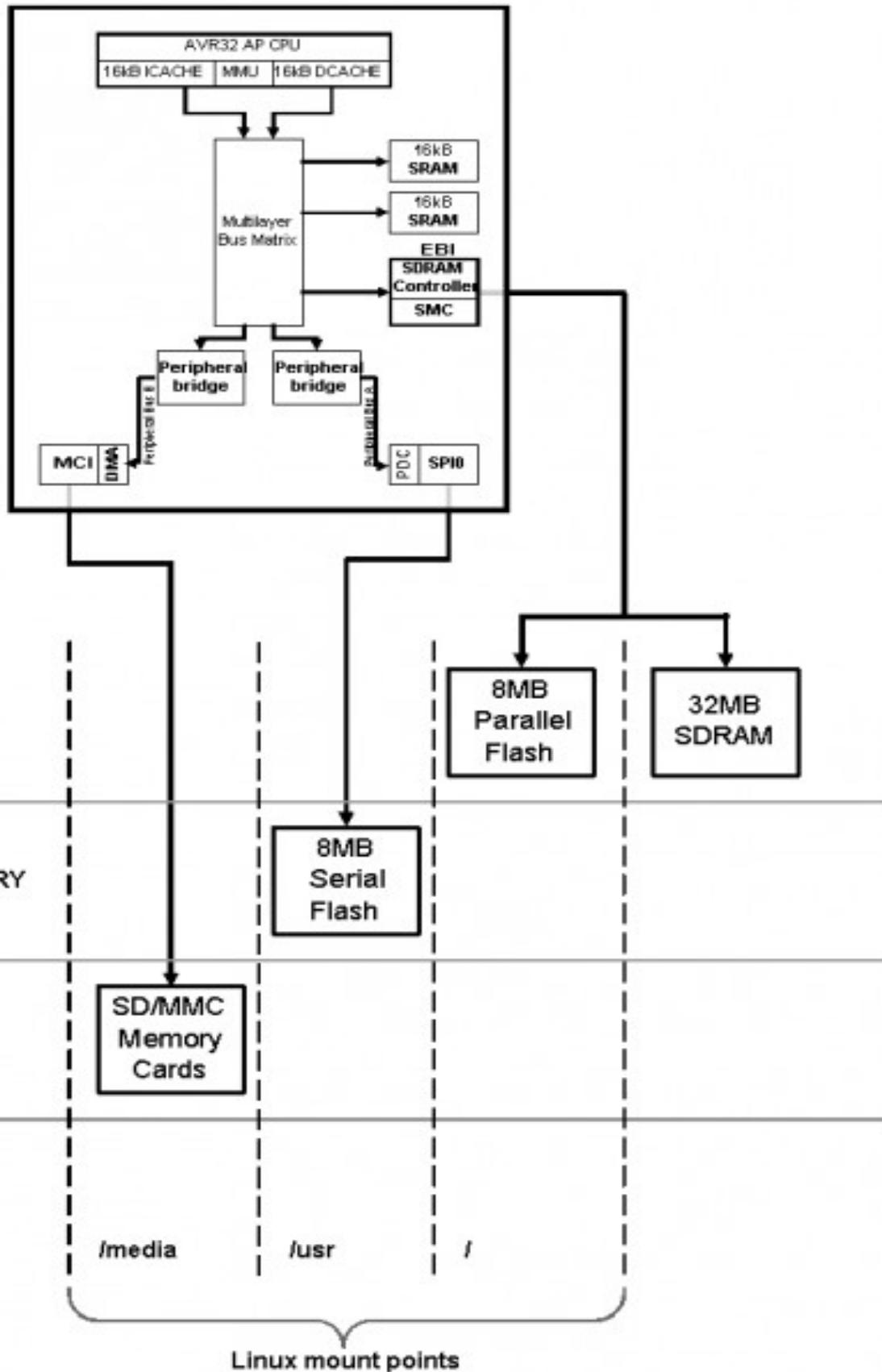
- Doi conectori Ethernet;
- 32 MB SDRAM;
- 16 MB memorie Flash on-board;
- Memorie extensibilă prin carduri de memorie SD sau MMC;
- Conector USB;
- Conector JTAG pentru debugging sau programarea memoriei Flash;
- Conectori de expansiune cu porturi I/O de uz general sau module periferice de la AP7000;
- LED-uri de stare, sistem și alimentare;
- Două LED-uri controlabile de utilizator;
- Spațiu pentru conector micotor-38 (utilizat de emulatorul NEXUS).

NGW100 este de asemenea un kit de dezvoltare ideal pentru AT32AP7000. Toate resursele sunt disponibile, și suportă comunicații pe oricare dintre interfețele de comunicații ale dispozitivului. Kitul este încărcat în prealabil cu Linux și livrat cu drivere de interfață I/O care pot fi apelate de codul scris de utilizator

Arhitectura platformei de dezvoltare NGW100 cu AVR32 este redată în continuare împreună cu distribuția memoriei sistemului (memoriei program și memoriei de date) pe cele trei nivele ierarhice și punctele de interfata cu sistemul de operare Linux.



AP7000



SDRAM

Memoria SDRAM de pe Network Gateway este de 256Mbit (4M x 16biti x 4bancuri) sau 32MB si este legata de AP7000 printr-un bus adresa/date de 16 biti.

Flash Paralel

Memoria de boot are o capacitate de 8MB si este adresata pe 16biti de catre AP7000. Ea vine pre-incarcata cu U-Boot (softul bootloader) si un sistem de operare Linux Embedded.

Flash Serial

Memoria flash seriala (conectata la interfata SPI) este pre-incarcata cu sistemul de fisiere (filesystem) Linux.

3. Descrierea softwareului de conversie si de test pentru aplicatia dezvoltata

In continuare este prezentat codul si descrierea aplicatiei modulul IO distribuit ce ruleaza pe platforma cu ATMega16, aplicatia de conversie serial la ethernet, converter client, ce ruleaza pe platforma NGW100 cu AVR32, si aplicatia server ce ruleaza pe un PC host (sau poate fi vazut ca un calculator industrial), sintetizand elemente de design ale softwareului si prezentarea instrumentelor specifice utilizate in dezvoltare.

Astfel pe modulul IO distribuit cu ATMega16 ruleaza o aplicatie care citeste periodic (temporizare timer 1s) intrarea analogica 0 si transmite valoarea

preluata pe linia seriala RS232 si daca valoarea citita este mai mare THRES_UP LED0 este comandat , iar daca valoarea citita este mai mica decat THRES_DOWN este comandat LED1.

Codul aplicatiei a fost dezvoltat sub Linux cu ajutorul mediului de programare Kontrollelab.

```
/*
 * main.c
 *
 *
 *      Created on: Jan 16, 2011
 *
 *      Author: Cristian Axenie
 *
 *          Aplicatie care citeste periodic (temporizare timer 1s)intrarea analogica 0 si
transmite valoarea
 *
 *          preluata pe linia seriala RS232 si daca valoarea citita este mai mare THRES_UP LED0
este comandat
 *
 *          iar daca valoarea citita este mai mica decat THRES_DOWN este comandat LED1.
 */

*/
#include <avr/io.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

/* functie de initializare a subsistemelor mcu */
void init();

/* functie de initializare a convertorului adc */
```

```

extern void init_adc(void);

/* functie de citire a unui canal al adc */

extern unsigned char read_adc(unsigned char);

/* functie de initializare a modulului usart */

extern void init_usart(void);

/* functie de transmitere a unui caracter pe linia seriala */

extern void send_char(char);

/* functie de conversie binar la ascii */

extern void convert_to_ascii(unsigned char);

/* functie care implementeaza un soft timer pt temporizare */

extern void soft_timer(void);

/* functie de initializare a timerului hardware */

extern void init_timer(void);

/* functie care decrementeaza timerul cu 10ms */

extern void decrement_10ms(void);

/* functie care converteste 2 octeti hex in binar */

extern unsigned char convert_to_bin(unsigned char, unsigned char);

extern int txflag;// flag receptie

extern unsigned char txdata; // date de transmis

extern unsigned char lonib, hinib; // nibble superior si inferior a datei de transmis

extern unsigned char timer_table[8];// tabele timer

extern unsigned int OCR1;// valoarea care se va scrie in registrul output compare a Timer1

extern short qtoc;// flag de intrerupere

static unsigned char THRES_UP = 0xD3; // prag superior la 4 V

static unsigned char THRES_DOWN = 0x35; // prag inferior la 1 V

int main(){

    unsigned char adc_val;// valoarea citita de la convertorul ADC

    init();// initializare

    init_usart();// init USART

```

```
init_timer(); // init timer

init_adc(); // init ADC

timer_table[0] = 100; // seteaza temporizarea de 1s

// bucla infinita

while(1){

    soft_timer(); // se porneste timerul software

    adc_val = read_adc(0); // se citeste intrarea analogica 0

    if(timer_table[1]==0){ //daca timerul a expirat

        timer_table[0] = 100; // reinitializare timer

        convert_to_ascii(adc_val); // se converteste in caractere ASCII pentru TX pe seriala

        send_char('r');

        send_char('e');

        send_char('z');

        send_char(0x20);

        send_char('A');

        send_char('D');

        send_char('C');

        send_char(':');

        // se transmite caracterul pentru nibbleul superior

        send_char(hinib);

        // se transmite caracterul pentru nibbleul inferior

        send_char(lonib);

        send_char('h');

        send_char('|');

        // se converteste valoarea in binar pentru a extrage valoarea ternsiunii

        // corespunzatoare pentru valoarea hex citita

        // U[V] = adc_val[hex]/33[hex] (interpolare liniara)

        convert_to_ascii(convert_to_bin(hinib, lonib)/0x33);

        send_char('v');

        send_char('a');

        send_char('l');

        send_char(0x20);
}
```

```

    send_char('U');

    send_char(':');

    // valoare zecimala a tensiunii corespunzatoare

    send_char(lonib);

    send_char('V');

    send_char(0xA); // delimitator

}

// se testeaza prejurile

if(adc_val >= THRES_UP) PORTB|=(1<<0); // se comanda LED0

else PORTB&=~(1<<0); // LED0 stins

if(adc_val <= THRES_DOWN) PORTB|=(1<<1); // se comanda LED1

else PORTB&=~(1<<1); // LED1 stins

};

return 0;
}

void init(void)

{

    // Initializarea porturilor IO

    // Code Generated by the Configuration Wizard

    PORTA=0x00;

    DDRA=0x00;

    PORTB=0x00;

    DDRB=0xFF;

    PORTC=0x00;

    DDRC=0x00;

    PORTD=0x00;

    DDRD=0x00;
}

```

```
// Timer/Counter 0 initialization
// Clock source: System Clock
// Clock value: Timer 0 Stopped
// Mode: Normal top=FFh
// OC0 output: Disconnected
TCCR0=0x00;
TCNT0=0x00;
OCR0=0x00;
// Timer/Counter 2 initialization
// Clock source: System Clock
// Clock value: Timer 2 Stopped
// Mode: Normal top=FFh
// OC2 output: Disconnected
ASSR=0x00;
TCCR2=0x00;
TCNT2=0x00;
OCR2=0x00;
// External Interrupt(s) initialization
// INT0: Off
// INT1: Off
// INT2: Off
MCUCR=0x00;
MCUCSR=0x00;
// Timer(s)/Counter(s) Interrupt(s) initialization
TIMSK=0x10;
// Analog Comparator initialization
// Analog Comparator: Off
// Analog Comparator Input Capture by Timer/Counter 1: Off
ACSR=0x80;
SFIOR=0x00;
}
```

```

/*
 * adc.c
 *
 *      Created on: Jan 16, 2011
 *      Author: Cristian Axenie
 *      Fisier cu functii de configurare si de preluare a datelor specifice ADC
 */

#include <avr/io.h>
#include <util/delay.h>

// 0x60 = 01100000
#define ADC_VREF_TYPE 0x60

void init_adc(void);
unsigned char read_adc(unsigned char);

void init_adc(void)
{
    // ADC initialization
    // ADC Clock frequency: 1000.000 kHz
    // ADC Voltage Reference: AVCC pin
    // Only the 8 most significant bits of
    // the AD conversion result are used
    ADMUX=ADC_VREF_TYPE & 0xff;// 01100000 & 11111111 = 01100000

    // 0 1 1 ... AVCC with external capacitor at AREF pin (bits 7 and 6) and ADC Left Adjust
    Result (bit 5)

    ADCSRA=0x84;// 10000100 --> ADC Enable (bit 7) and Division Factor 16 (bits 2 1 0
combination)

}

// Read the 8 most significant bits (ADCH register)
// of the AD conversion result
unsigned char read_adc(unsigned char adc_input)
{

```

```

ADMUX = adc_input | (ADC_VREF_TYPE & 0xff);

/* Delay needed for the stabilization of the ADC input voltage*/

_delay_us(10.0);

// Start the AD conversion

ADCSRA|=0x40;// 01000000 --> ADSC bit set = AD start conversion

// Wait for the AD conversion to complete

while ((ADCSRA & 0x10)==0);// Test the ADIF bit (AD interrupt flag) for AD conversion
finish

ADCSRA|=0x10;

return ADCH;// return the conversion result ADCH

}

/*
* usart.c

*
*      Created on: Jan 16, 2011
*
*      Author: Cristian Axenie
*
*      Functii specifice pentru lucrul cu modulul USART
*/

```

```

#include <avr/io.h>

#include <avr/interrupt.h>

// macrouri

#define RXB8 1

#define TXB8 0

#define UPE 2

#define OVR 3

#define FE 4

#define UDRE 5

#define RXC 7

// prototipuri functii

// initializare modul USART

void init_usart(void);

```

```

// functie pentru transmiterea unui caracter pe linia seriala
void send_char(char);

// functie de conversie binar la ascii
void convert_to_ascii(unsigned char);

// functie care converteste 2 octeti hex in binar
unsigned char convert_to_bin(unsigned char, unsigned char);

// variabile globale
int txflag;//flag semnalizare TX
unsigned char txdata; // datele pentru transmisie
unsigned char lonib, hinib; // nibble low si high (1/2byte)

// rutina de tratare a intreruperilor pe transmisia USART
ISR(USART_TxC_vect){

    char status;
    status = UCSRA; // se preia statusul modului din registrul de stare UCSRA
    UDR = txdata;// se scrie data de transmis in registrul de date
    txflag = 1;// seteaza flagul de transmisie
}

// functie de conversie numar binar la 2 caractere ASCII hinib si lonib
void convert_to_ascii(unsigned char ch){

    hinib = (ch & 0xf0)>>4; // primul caracter
    lonib = (ch & 0x0f);      // al doilea caracter

    if(hinib>9)hinib = hinib + '7';// daca este litera in baza 16
    else hinib = hinib + '0';      // daca este cifra in baza 16

    if(lonib>9)lonib = lonib + '7';// daca este litera in baza 16
    else lonib = lonib + '0';// daca este cifra in baza 16
}

```

```

// functie care converteste doua caractere ASCII presupuse a fi digitii hex valizi intr-un
// octet binar

unsigned char convert_to_bin(unsigned char ch1, unsigned char ch2)
{

    unsigned char ch;

    if(ch1 <= '9') ch1=ch1&0x0f;

    else ch1= (ch1-7) & 0x0f;

    if(ch2 <= '9') ch2=ch2&0x0f;

    else ch2= (ch2-7) & 0x0f;

    ch=(ch1<<4)|ch2;

    return(ch);

}

// functie pentru transmiterea unui caracter pe linia seriala RS 232

void send_char(char val){

    UDR = val;// se scrie valoare pentru a fi scrisa

    while(!(UCSRA & (1<<UDRE)));// cat timp sunt date in bufferul de TX

}

void init_usart(){

    // Code Generated by the Configuration Wizard

    // USART initialization

    UCSRA = 0x00;// USART control and status register A set to 0

    //USART control and status register B

    // Bit 6 – TXCIE: TX Complete Interrupt Enable is set

    // Bit 3 – TXEN: Transmitter Enable is set

    UCSRB = 0x48;// 01001000

    //USART control and status register C

```

```

// Bit 7 - URSEL: Register Select is set
// Bits 2:1 - UCSZ1:0: Character Size are set
UCSRC = 0x86;// 10000110

// USART baud rate register
UBRRH = 0x00;

// Trigger an immediate update of the baud rate prescaler
UBRRL = 0x67;// 01100111

}

/*
 * timer.c
 *
 *      Created on: Jan 16, 2011
 *      Author: Cristian Axenie
 *      Functiile specifice lucrului cu Timer1 (16-bit)
 */

```

```

#include <avr/io.h>
#include <avr/interrupt.h>

unsigned char timer_table[8];
unsigned int OCR1;
short qtoc;
//functii specifice
// implementare soft timer
void soft_timer(void);
// initializare timer1
void init_timer(void);
// functie decrementare 10ms
void decrement_10ms(void);
// rutina de tratare a intreruperilor de timer
// in mod de operare output compare
ISR(TIMER1_COMPA_vect)
{

```

```

// se aduna continutului registrului OCR valoarea 20000 pt a genera intrerupere la
20000*0.5us = 10ms

qtoc = 1;// generare intrerupere

OCR1 = (OCR1AH*256+OCR1AL)+20000;

OCR1AH = (OCR1 & 0xFF00)>>8;

OCR1AL = (OCR1 & 0xFF);

}

// implementeaza functionalitatea de timer software

void soft_timer(){

if(qtoc==0) return;//dezactiveaza intreruperile

qtoc=0;

decrement_10ms();

}

// decrementeaza valoarea din tabela timerului

void decrement_10ms(){

unsigned short i;

for(i=0;i<8;i++){

if(timer_table[i]!=0)

timer_table[i]--;// decrementeaza timerul


}

// Functie de initializare a timerului

void init_timer(void)

{

// Code Generated by the Configuration Wizard

// Timer/Counter 1 initialization

```

```

// Clock source: System Clock

// Clock value: 2000.000 kHz

// Mode: Normal top=FFFFh

// OC1A output: Discon.

// OC1B output: Discon.

// Noise Canceler: Off

// Input Capture on Falling Edge

// Timer 1 Overflow Interrupt: Off

// Input Capture Interrupt: Off

// Compare A Match Interrupt: On

// Compare B Match Interrupt: Off

TCCR1A=0x00;// timer / counter control register A

// timer / counter control register B,

// the bit CS11 is set so CS12 CS11 CS10 = 0 1 0 and clkI/O/8 (From prescaler)

TCCR1B=0x02;

// nothing written in the counter

TCNT1H=0x00;

TCNT1L=0x00;

// input capture disabled

ICR1H=0x00;

ICR1L=0x00;

// output compare units A and B initially set to 0

OCR1AH=0x00;

OCR1AL=0x00;

OCR1BH=0x00;

OCR1BL=0x00;

}

}

```

Astfel, pe modulul NGW100 distribuit cu AVR32 ruleaza o aplicatie care preia informatia de pe linia seriala RS-232, o despacheteaza si o impacheteaza in UDP/IP pe interfata Ethernet Eth0. Aplicatia de conversie serial la Ethernet utilizeaza doua instante. Primul modul este clientul/convertorul care va opera pe platforma cu AVR32 si va comunica cu modulul IO distribuit cu ATMega16. A doua componenta, serverul/testerul, va rula pe un PC si va avea rolul de a prelua pachetele UDP/IP primite de la modulul gateway NGW100 care incapsuleaza informatia de la senzorii conectati la modulul IO si de a o afisa pe display.

Observatii:

- Aplicatiile fiind dezvoltate pentru operarea sub sistemul Linux vor utiliza librarii de functii specifice care vor fi descrise punctual in momentul in care se descrie functionalitatea implementata.
- Intrucat aplicatia poate fi utilizata si pe o alta platforma hardware decat AVR2 am ales implementarea ambelor componente, client si server, pentru doua platforme si anume, x86 si AVR32. Diferenta consta doar in compilatorul folosit pentru transformarea codului de limbaj C in cod executabil pe masina respectiva. In demonstratia propusa aplicatia server este compilata si ruleaza pe platforma cu AVR32 iar aplicatia client este rulata pe un sistem x86 cu Linux OS Ubuntu.

Aplicatia client care ruleaza pe modulul NGW100 cu AVR32 este descrisa in continuare. Functionalitatea serverului este implementata in jurul unei interfete de linie comanda in care utilizatorul poate seta adresa IP serverului la care se poate conecta si care asculta, portul de comunicatie UDP, dispozitivul serial de unde preia informatia seriala si optiuni de viteza si parametrii de comunicatie seriala. Linia de comanda are urmatorul format general.

"Utilizare: s2e-converter -a <addr> <port> -d <dev> <opt> [--verbose | v] \n"

"unde <addr> si <dev> reprezinta argumente pentru:\n"

" -a <addr> <port> adresa serverului si port UDP\n"

" -d <dev> <opt> dispozitivul serial local si parametri de comunicatie\n"

" <opt> este de forma sss-dps\n"

" sss este viteza de comunicatie seriala (50,...,230400)\n"

" d numarul de biti per caracter (5,6,7,8)\n"

" p tipul de paritate (N,E,O)\n"

" s numarul de biti de stop (1,2)\n";

In programul principal se creeaza un socket pentru trimitera informatiei sub forma de datagrame, apoi se aloca memorie pentru membrii structurii IP si seteaza valori de initializare. In continuare se seteaza familia de socket (AF_INET) si se realizeaza conversia valorii din linia de comanda pentru port si se converteste in formatul acceptat. Se preia spori adresa IP si se converteste intr-un format acceptat. Urmatoarea etapa este accesarea portului serial pentru care se realizeaza niste setari speciale. Pentru setarea conexiunii seriale se initializeaza instanta de configurare la valorile specific OS, apoi se initializeaza instanta de configurare la valorile specific OS. Dupa aceasta se seteaza baudrate-ul si se scrie in instanta structurii termios cfg valoarea de configurare pentru viteza pe RX din variabila baudrate.

Apoi se scrie in instanta structurii de configurare valoarea de configurare pentru viteza pe TX din variabila baudrate, dupa care se seteaza tipul de paritate si numarul de biti / char la reprezentarea interna. Se seteaza valorile de configurare primite la intrare si setate in structura termios si se revine din functia de setup pentru seriala. Dupa deschiderea si setarea parametrilor de comunicatie se preiau, in bucla infinita, octetii de pe linia seriala si se transmit pe suportul UDP/IP Ethernet.

In acelasi timp aplicatia server ruleaza pe o masina x86 cu Linux OS instalat. Aplicatia server initializeaza comunicarea valorilor de la clientul NGW100 si poate capta semnale de intrare si ieșire. In programul principal se deschide un socket pentru ascultarea pe conexiunea UDP/IP, se aloca memorie pentru membrii structurii IP si seteaza valori de initializare, se seteaza portul, adresa IP si se activeaza suportul de ascultare pentru orice IP care se conecteaza. Dupa crearea socketului se realizeaza legatura cu conexiunea (bindingul) si se primesc pachete UDP/IP care sunt apoi pasate si campul de date este extras si formatat corespunzator pentru iesire.

Codul aplicatiei a fost dezvoltat sub Linux cu ajutorul mediului de build Buildroot.

Codul serverului : s2e-tester

```
/*
 *  s2e-tester.c
 *
 *  Created on: Jan 16, 2011
 *      Author: Cristian Axenie
 *
 *  Aplicatie de transport(conversie) a informatiei de pe interfata seriala pe o interfata
 *  UDP/IP Ethernet
 *
 *  destinata comunicatiei cu modulele distribuite de control intr-un sistem SCADA.
 *
 *  Componenta SERVER - comunica cu modulele de comunicatie si primeste informatie
 *  impachetata UDP/IP pe suport
 *
 *  Ethernet ce contine informatie de stare sau date de la senzorii procesului conectati la
 *  modulele IO.
 *
 *  Aplicatia ruleaza pe unitatea centrala de monitorizare a ierarhiei SCADA.
 *
 */
#include "../inc/s2e-tester.h"

/* Functie responsabila cu logarea erorilor, extragerea mesajelor de eroare din structurile
 * aferente */
void s2eLogError(char *s)
{
    perror(s);
    exit(EXIT_FAILURE);
}

/* Functie responsabila pentru captarea semnalelor de tip SIGINT pentru oprirea serverului */

void s2eStopComm(int x)
{
```

```

s2eLogMessage
    ("s2eTester : Serverul care capteaza pachete UDP/IP a primit semnal de stop!%s",
     "\n");
close(s);
s2eLogMessage("s2eTester : Conexiunea UDP/IP a fost inchisa !%s", "\n");
s2eLogMessage
    ("s2eTester : ======%s",
     "\n");
}

int main(void)
{
    struct sockaddr_in si_me, si_other; // socketi pentru retinearea adreselor UDP
    int slen = sizeof(si_other); // dimensiunea adresei
    char buf[BUFSIZE]; // dimensiunea bufferelor RX/TX (2 char)
    char *data;
    s2eLogMessage
        ("s2eTester : ======%s",
         "\n");
    s2eLogMessage
        ("s2eTester : Convertorul Serial la Ethernet a fost lansat !%s",
         "\n");
    s2eLogMessage
        ("s2eTester : Serverul care capteaza pachete UDP/IP a fost lansat !%s",
         "\n");
    s2eLogMessage
        ("s2eTester : Se primesc valori de la modulul IO distribuit : ....%s",
         "\n");
    // se creeaza un socket pentru ascultarea pe conexiunea UDP/IP
    if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
        s2eLogError("socket");
    // aloca memorie pentru membrii structurii IP si seteaza valori de initializare

```

```

memset((char *) &si_me, 0, sizeof(si_me));

si_me.sin_family = AF_INET;
si_me.sin_port = htons(PORT);
si_me.sin_addr.s_addr = htonl(INADDR_ANY);

// se realizeaza bindingul la conexiune

if (bind(s, (struct sockaddr *) &si_me, sizeof(si_me)) == -1)
    s2eLogError("bind");

// se primesc pachetele

while (1) {

    if (recvfrom

        (s, buf, sizeof(buf), 0, (struct sockaddr *) &si_other,
         (socklen_t *) & slen) == -1)

        exit(EXIT_FAILURE);

    // se afiseaza informatia primita de la nodul distribuit de comunicatie cu modulul IO
    // ( tensiunea corespunzatoare de la iesire senzorului (intrarea modului IO distribuit))

    data = strtok(buf, "\n");

    // filtru pentru a filtra pachetele care contin doar informatie partiala

    // informatie partiala apare datorita faptului ca citirea din bufer este asincrona

    if ((strstr(data, "rez") != NULL) && (strstr(data, "ADC") != NULL)
        && (strstr(data, "U") != NULL) && (strstr(data, "V") != NULL)
        && (strstr(data, "|") != NULL) && (strstr(data, "h") != NULL)
        && (strstr(data, "val") != NULL)) {

        s2eLogMessage

            ("s2eServer : Pachetele sunt receptionate de la modulul distribuit cu adresa
%s:\n",
             inet_ntoa(si_other.sin_addr));

        s2eLogMessage("s2eServer : Campul de date din pachet contine : %s\n",
                      data);
    }

    // sectiune in care se parseaza informatia primita si se determina tensiunea
    // corespunzatoare de la iesirea

    // senzorului (intrarea modului IO distribuit)

    signal(SIGINT, s2eStopComm);

    signal(SIGKILL, s2eStopComm);
}

```

```

    signal(SIGHUP, s2eStopComm);

}

// se inchide conexiunea

close(s);

return 0;

}

/*
*   s2e-tester.h

*
*   Created on: Jan 16, 2011
*       Author: Cristian Axenie
*
* Aplicatie de transport(conversie) a informatiei de pe interfata seriala pe o interfata
UDP/IP Ethernet
* destinata comunicatiei cu modulele distribuite de control intr-un sistem SCADA.
*
* Componenta SERVER - comunica cu modulele de comunicatie si primeste informatie
impachetata UDP/IP pe suport
* Ethernet ce contine informatie de stare sau date de la senzorii procesului conectati la
modulele IO.
* Aplicatia ruleaza pe unitatea centrala de monitorizare a ierarhiei SCADA.
*
*
* Fisier header
*
*/

```

```

#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>

```

```

#include <string.h>
#include <syslog.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <termios.h>
#include <unistd.h>
#include <time.h>

/* Macrodefinitii si variabile simbolice */
#define PORT 3000
#define BUFLEN 60

// macro pentru logging
#define s2eLogMessage(format,args...) \
    do{ \
        fprintf(stdout,format,args); \
    } while(0);

// descriptor conexiune
int s;
/* Functie responsabila cu logarea erorilor, extragerea mesajelor de eroare din structurile
afereante */

extern void s2eLogError(char *s);
/* Functie responsabila pentru captarea semnalelor de tip SIGINT pentru oprirea serverului */
extern void s2eStopComm(int x);

```

Codul clientului : s2e-converter

```

/*
 * s2e-converter.c
 *
 * Created on: Jan 16, 2011
 *      Author: Cristian Axenie

```

```

/*
 * Aplicatie de transport(conversie) a informatiei de pe interfata seriala pe o interfata
 * UDP/IP Ethernet

 * destinata comunicatiei cu modulele distribuite de control intr-un sistem SCADA.

 *

 * Componenta CLIENT - comunica cu modulele IO si trimite informatie impachetata UDP/IP pe
 * suport Ethernet

 * unitatii centrale de monitorizare a ierarhiei SCADA

 *

 */

#include "../inc/s2e-converter.h"

/*
Functie care este responsabila cu interfata de linie comanda utila la apelarea aplicatiei de
conversie RS232 la ethernet.

*/
void s2eCommandLine()
{
    fprintf(stdout,
            "Utilizare: s2e-converter -a <addr> <port> -d <dev> <opt> [--verbose | v] \n"
            "unde <addr> si <dev> reprezinta argumente pentru:\n"
            "  -a <addr> <port>           adresa serverului si port UDP\n"
            "  -d <dev> <opt>             dispozitivul serial local si parametri de
comunicatie\n"
            "  <opt> este de forma sss-dps\n"
            "  sss este viteza de comunicatie seriala (50,...,230400)\n"
            "  d numarul de biti per caracter (5,6,7,8)\n"
            "  p tipul de paritate (N,E,O)\n"
            "  s numarul de biti de stop (1,2);\n"
    return;
}

/*
 * Functie responsabila cu deschiderea portului serial pentru comunicatia RS232.

```

```

* Returneaza file descriptorul asociat dispozitivului tty sau -1 in cazul unei erori.
*/
int s2eOpenSerialPort(char *port)
{
    int fd;           // file descriptor pentru port
    // se deschide portul ttyS0 cu urmatoarele moduri de acces
    // O_RDWR - deschidere pentru citire si scriere
    // O_NOCTTY - dispozitivul terminal nu va controla procesrul curent
    // O_NDELAY - parametru intarziere
    fd = open(port, O_RDWR | O_NOCTTY | O_NDELAY);
    if (fd == -1) {
        s2eLogMessage("s2eOpenSerialPort: Nu se poate deschide %s !\n", port);
    } else
        // functie file control
        // seteaza flagurile de stare ale fisierului dat de un file descriptor
        fcntl(fd, F_SETFL, 0);

    return (fd);
}

/* Functie responsabila cu logarea erorilor, extragerea mesajelor de eroare din structurile
   aferente. */
void s2eLogError(char *s)
{
    perror(s);
    exit(EXIT_FAILURE);
}

/*
 * Functie care implementeaza comportamentul la finalizarea comunicatiei.

```

```

* - se elibereaza buferele de TX si RX si se elibereaza conexiunea seriala.

*/
void s2eStopComm(int s)
{
    // elibereaza datele scrise in bufferul de TX dar care nu au fost transmise
    // si datele care au fost receptionate in bufferul de RX dar nu au fost citite
    // pentru conexiunea 0
    tcflush(s, TCIOFLUSH);
    // se inchide conexiunea seriala
    close(s);

    // redirecteaza mesajele catre fluxul setat
    s2eLogMessage("s2eStopComm: Conexiunea %d seriala fost inchisa !\n", s);
    exit(EXIT_SUCCESS);
}

/*
Functie care este responsabila cu setarea parametrilor de comunicatie pe linia seriala RS232

- seteaza viteza de comunicatie la transmisie si receptie
- seteaza reprezentarea caracterelor in informatia transmisa (5/6/7/8 biti / caracter)
- seteaza paritatea (N - none, E - even, O - odd) utila in detectia erorilor de transmisie
- seteaza numarul de biti de stop

```

Informatia privind parametrii de transmisie se seteaza intr-o instanta a structurii struct termios

unde se incapsuleaza informatie privind parametrii pentru intrare, iesire, moduri de control, moduri locale si caractere de control.

```

int s2ePortSetup(int s, struct termios *cfg, int speed, int data,
                 unsigned char parity, int stopb)
{

```

```

// se initializeaza instanta de configurare la valorile specifice OS
tcgetattr(s, cfg);

s2eLogMessage
    ("s2ePortSetup : Atributele de configurare au fost setate pentru conexiunea %d ...\\n",
     s);

// seteaza instanta de configurare a portului serial la valori default ale atributelor
cfmakeraw(cfg);

s2eLogMessage
    ("s2ePortSetup : S-a creat obiectul de configurare pentru conexiunea %d ...\\n",
     s);

// retine baudrate-ul

speed_t baudrate;

switch (speed) {           // preia viteza

case 9600:
    baudrate = B9600;
    s2eLogMessage("s2ePortSetup : S-a setat baudarate %d ...\\n", speed);
    break;

case 19200:
    baudrate = B19200;
    s2eLogMessage("s2ePortSetup : S-a setat baudarate %d ...\\n", speed);
    break;

case 38400:
    baudrate = B38400;
    s2eLogMessage("s2ePortSetup : S-a setat baudarate %d ...\\n", speed);
    break;

case 57600:
    baudrate = B57600;
    s2eLogMessage("s2ePortSetup : S-a setat baudarate %d ...\\n", speed);
    break;

case 115200:
    baudrate = B115200;
    s2eLogMessage("s2ePortSetup : S-a setat baudarate %d ...\\n", speed);
}

```

```

        break;

default:

    s2eLogMessage("s2ePortSetup : Viteza %d baud nu este suportata !\n",
                  speed);

    break;

}

// scrie in instanta structurii termios cfg valoarea de configurare
// pentru viteza pe RX din variabila baudrate
cfsetispeed(cfg, baudrate);

s2eLogMessage

    ("s2ePortSetup : S-a configurat conexiunea pentru RX pentru baudarate %d ...%\n",
     speed);

// scrie in instanta structurii termios cfg valoarea de configurare
// pentru viteza pe TX din variabila baudrate
cfsetospeed(cfg, baudrate);

s2eLogMessage

    ("s2ePortSetup : S-a configurat conexiunea pentru TX pentru baudarate %d ...%\n",
     speed);

// preia paritatea si seteaza campul corespunzator din structura de configurare

switch (parity | 32) {

case 'n':           //  fara valoare de paritate

    // dezactiveaza generarea valorii de paritate la TX
    // si verificarea paritării la RX
    cfg->c_cflag &= ~PARENB;

    s2eLogMessage

        ("s2ePortSetup : Nu se genereaza bit de paritate pentru conexiunea %d\n",
         s);

    break;

}

case 'e':           // valoare de paritate para

```

```

{

    // activeaza generarea valorii de paritate la TX si verificare la RX
    cfg->c_cflag |= PARENB;

    // activeaza verificarea paritatii la TX si RX pt valoare para
    cfg->c_cflag &= ~PARODD;

    s2eLogMessage

        ("s2ePortSetup : Activeaza verificarea paritatii la TX si RX pt valoare para pentru
conexiunea %d\n",
        s);

    break;
}

case 'o':           // valoare de paritate impara
{
    // activeaza generarea valorii de paritate la TX si verificare la RX
    cfg->c_cflag |= PARENB;

    // activeaza verificarea paritatii la TX si RX pt valoare impara
    cfg->c_cflag |= PARODD;

    s2eLogMessage

        ("s2ePortSetup : Activeaza verificarea paritatii la TX si RX pt valoare impara
pentru conexiunea %d\n",
        s);

    break;
}

// masca pentru setarea dimensiunii unui caracter in informatia transmisa
cfg->c_cflag &= ~CSIZE;

// se extrage capacitatea de reprezentare a unui caracter

switch (data) {

    // seteaza campul corespunzator reprezentarii
    // unui caracter cu macroul corespunzator

case 5:

{
    cfg->c_cflag |= CS5;// 5 biti / char
}

```

```

s2eLogMessage

    ("s2ePortSetup : Seteaza %d biti / char pentru conexiunea %d\n",
     data, s);

    break;
}

case 6:

{
    cfg->c_cflag |= CS6;// 6 biti / char

    s2eLogMessage

    ("s2ePortSetup : Seteaza %d biti / char pentru conexiunea %d\n",
     data, s);

    break;
}

case 7:

{
    cfg->c_cflag |= CS7;// 7 biti / char

    s2eLogMessage

    ("s2ePortSetup : Seteaza %d biti / char pentru conexiunea %d\n",
     data, s);

    break;
}

case 8:

{
    cfg->c_cflag |= CS8;// 8 biti / char

    s2eLogMessage

    ("s2ePortSetup : Seteaza %d biti / char pentru conexiunea %d\n",
     data, s);

    break;
}

}

// se extrage numerul de biti de stop si ajusteaza valoare de configurare
if (stopb == 1) {

    cfg->c_cflag &= ~CSTOPB;      // seteaza un bit de STOP
}

```

```

s2eLogMessage

("s2ePortSetup : Seteaza %d biti biti de stop pentru conexiunea %d\n",
stopb, s);

} else {

cfg->c_cflag |= CSTOPB;      // seteaza 2 biti de STOP

s2eLogMessage

("s2ePortSetup : Seteaza 2 biti de stop pentru conexiunea %d\n",
s);

}

// seteaza valorile de configurare primite la intrare si setate in structura termios
// pentru portul serial considerat in s
// setarea are loc imediat si apoi se revine cu un cod de return intreg

return tcsetattr(s, TCSANOW, cfg);
}

```

```

int main(int argc, char **argv)
{

struct sockaddr_in si_other;    // structura care implementeaza lucrul cu adrese IP
int s, i;                      // variabile auxiliare
socklen_t slen = sizeof(si_other); // dimensiunea adresei IP
char buf[BUFSIZE];             // buffer date RX/TX
int ser_dev;                   // identificator dispozitiv serial
struct termios cfg;            // structura cu date de configurare a portului serial
char *bufptr;                  // caracterul curent in buffer
int nbytes;                    // numarul de octeti cititi
int cnt;                       // contor
int ret;                        // cod de return
int n_pack;                    // numarul de pachete transmise
unsigned char parity;          // paritate

```

```

int speed, data, stopb; // baudrate, byte/char si numar biti de stop

if (argc != 7) {           // daca numarul de parametri nu este corespunzator
    // afiseaza help
    s2eCommandLine();
    return 1;
}

s2eLogMessage
(
    "s2eConverter : ======%s",
    "\n");
s2eLogMessage
(
    "s2eConverter : Convertorul Serial la Ethernet a fost lansat !%s",
    "\n");

// se creeaza un socket pentru trimitera informatiei sub forma de datagrame
if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    s2eLogError("socket");

// aloca memorie pentru membrii structurii IP si seteaza valori de initializare
memset((char *) &si_other, 0, sizeof(si_other));

// seteaza familia de socket
si_other.sin_family = AF_INET;

// se preia portul de comunicatie in format specific
si_other.sin_port = htons(atoi(argv[3]));

// conversia adresei IP din format generic string in struct addr_in
if (inet_aton(argv[2], &si_other.sin_addr) == 0) {
    s2eLogMessage("s2eConverter : Conversia a esuat pentru %s\n", argv[2]);
    exit(EXIT_FAILURE);
}

// deschide portul serial
if ((ser_dev = s2eOpenSerialPort(argv[5])) == -1)
    s2eLogMessage("s2eConverter : Portul serial %d nu poate fi deschis!\n",
                  ser_dev);

```

```

// parametrizeaza conexiunea seriala

sscanf(argv[6], "%d-%d%c%d", &speed, &data, &parity, &stopb);

ret = s2ePortSetup(ser_dev, &cfg, speed, data, parity, stopb);

if (ret < 0) {

    s2eLogMessage("s2eConverter : Portul serial %s nu se poate configura!",
                  argv[5]);

    exit(EXIT_FAILURE);

}

// se testeaza daca se doreste afisarea mesajelor de stare in timpul executiei

if ((0 == strcmp(argv[8], "--verbose"))

    || (0 == strcmp(argv[8], "-v")))

    verbose = 1;

while (1) {

    s2eLogMessage("s2eConverter : Se transmite pachetul %d\n", i);

    // se initializeaza pointerul in buffer la primul element

    bufptr = buf;

    // se citesc octeti de pe linia seriala

    while ((nbytes =

            read(ser_dev, bufptr, buf + sizeof(buf) - bufptr - 1)) > 0) {

        bufptr += nbytes;

        cnt++;

        // dimensionam dimensiunea datelor pentru afisare

        if (cnt == BUflen / 4)

            break;

    }

    printf("%s\n", buf);

    // se impacheteaza si se transmite informatia de pe seriala pe linkul UDP/IP

```

```

    if (sendto(s, buf, BUFLEN, 0, (struct sockaddr *) &si_other, slen) ==
        -1)
        s2eLogError("sendto()");
}

// inchide conexiunea seriala
close(s);

s2eLogMessage
(
    "s2eConverter : Aplicatia a revenit dupa transmiterea a %d pachete !\n",
    n_pack);
s2eLogMessage
(
    "s2eConverter : ======%s",
    "\n");
}

return 0;
}

/*
 *  s2e-converter.h
 *
 *  Created on: Jan 16, 2011
 *      Author: Cristian Axenie
 *
 *  Aplicatie de transport(conversie) a informatiei de pe interfata seriala pe o interfata
 *  UDP/IP Ethernet
 *
 *  destinata comunicatiei cu modulele distribuite de control intr-un sistem SCADA.
 *
 *  Componenta CLIENT - comunica cu modulele IO si trimite informatie impachetata UDP/IP pe
 *  suport Ethernet
 *
 *  unitatii centrale de monitorizare a ierarhiei SCADA
 *
 *  Fisier header
 *
 */
#include <arpa/inet.h>

```

```

#include <errno.h>
#include <fcntl.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <termios.h>
#include <unistd.h>

/* Macrodefinitii si variabile simbolice */
// dimensiunea maxima a bufferului
#define BUFSIZE 512
#define NPACK 100

// macro pentru loggingul mesajelor de stare
unsigned char verbose = 0;
#define s2eLogMessage(format,args...) \
    do{ \
        if(verbose) \
            fprintf(stdout,format,args); \
    } while(0);

/* Prototipurile functiilor */
/* Functie responsabila cu logarea erorilor, extragerea mesajelor de eroare din structurile
aferente */
extern void s2eLogError(char *s);
/* Functie care este responsabila cu setarea parametrilor de comunicatie pe linia seriala

```

```

RS232 */

extern int s2ePortSetup(int s, struct termios *cfg, int speed, int data,
                       unsigned char parity, int stopb);

/* Functie responsabila cu deschiderea portului serial pentru comunicatia RS232 */

extern int s2eOpenSerialPort(char *port);

/* Functie care implementeaza interfata de linie comanda pentru executia aplicatiei */

extern void s2eCommandLine();

/* Functie care implementeaza comportamentul la finalizarea comunicatiei. */

void s2eStopComm(int s);

```

Pasi executie

Pe platforma embedded NGW100 cu AVR32 unde ruleaza clientul se executa :

```

~ # ./prepare_SCADA_comm_node

Se seteaza IP ul corespunzator ...

Modul AVR 32 IP : 192.168.0.104

Se transfera binarul s2e-converter

Connecting to 192.168.0.103 [192.168.0.103:80]

s2e-converter      100% |*****| 14912      --:-- ETA

Transferul binarului a fost finalizat !

Aplicatia de conversie Serial la Ethernet poate fi lansata !

Pentru lansare utilizati comanda ./s2e-converter <optiuni>

~ #

~ # ./run_SCADA_comm_node -a 192.168.0.103 3000 -d /dev/ttyS0 9600-8n1

```

Pe sistemul PC unde ruleaza serverul se executa :

```

haustiq@MultiOsSystem:~/development/avr32_scada_dev/serial2eth_x86/s2e-tester$ ./s2e-tester

s2eTester : =====

s2eTester : Convertorul Serial la Ethernet a fost lansat !

s2eTester : Serverul care capteaza pachete UDP/IP a fost lansat !

s2eTester : Se primesc valori de la modulul IO distribuit : ....

s2eServer : Pachetele sunt receptionate de la modulul distribuit cu adresa 192.168.0.104:

s2eServer : Campul de date din pachet contine : rez ADC:00h|val U:0V

s2eServer : Pachetele sunt receptionate de la modulul distribuit cu adresa 192.168.0.104:

```

```
s2eServer : Campul de date din pachet contine : rez ADC:00h|val U:0V
s2eServer : Pachetele sunt receptionate de la modulul distribuit cu adresa 192.168.0.104:
s2eServer : Campul de date din pachet contine : rez ADC:00h|val U:0V
s2eServer : Pachetele sunt receptionate de la modulul distribuit cu adresa 192.168.0.104:
s2eServer : Campul de date din pachet contine : rez ADC:00h|val U:0V
s2eServer : Pachetele sunt receptionate de la modulul distribuit cu adresa 192.168.0.104:
s2eServer : Campul de date din pachet contine : rez ADC:00h|val U:0V
s2eServer : Pachetele sunt receptionate de la modulul distribuit cu adresa 192.168.0.104:
s2eServer : Campul de date din pachet contine : rez ADC:92h|val U:2V
s2eServer : Pachetele sunt receptionate de la modulul distribuit cu adresa 192.168.0.104:
s2eServer : Campul de date din pachet contine : rez ADC:91h|val U:2V
^Cs2eTester : Serverul care capteaza pachete UDP/IP a primit semnal de stop!
s2eTester : Conexiunea UDP/IP a fost inchisa !
s2eTester : =====
```

Se revine din executie prin emiterea unui semnal, adica prin combinatii de taste (Ctrl-C, Ctrl-X, Ctrl-Z) si toate resursele sunt dezalocate si conexiunea este inchisa.

Observatii:

Pentru dezvoltarea aplicatiei s-a utilizat o masina virtuala VirtualBOX, de tip Linux Ubuntu 8.04 in care am instalat instrumentele necesare pentru cross-compilarea aplicatiei pentru platforma AVR32. Aceasta abordare se bazeaza pe faptul ca toolchainurile pentru compilare au restrictii de versiune si compatibilitate. Pentru compilarea kernelului sistemului Linux de pe platforma embedded s-a utilizat ecosistemul de build Buildroot. Acesta a pus la dispozitie un framework de creare, configurare si compilare a unui kernel pentru platforma cu AVR32, de creare a unui filesystem cu aplicatii cross-compilate pentru AVR32 si a unui toolchain cu compilator, linker, assembler si archiver pentru cross-compilarea aplicatiilor pentru platforma embedded.

Anexe

Scripturi utile pe platforma NGW100 :

Pe host PC trebuie sa avem instalat un server http pentru a putea prelua fisier prin download utilizand adresa IP a serverului.

```
~ # cat ./prepare_SCADA_comm_node
#!/bin/sh
echo 'Se seteaza IP ul corespunzator ...'
ifconfig eth0 192.168.0.104
echo 'Modul AVR 32 IP : 192.168.0.104'
```

```

echo 'Se transfera binarul s2e-converter'
cd ../s2eConverter_bin && rm -fr * && wget http://192.168.0.103/s2e-converter
chmod +x s2e-converter
echo 'Transferul binarului a fost finalizat !'
cd ../s2eConverter_bin
echo 'Aplicatia de conversie Serial la Ethernet poate fi lansata !'
echo 'Pentru lansare utilizati comanda ./s2e-converter <optiuni>'

~# ./run_SCADA_comm_node -a 192.168.0.103 3000 -d /dev/ttyS0 9600-8n1

```

Fisiere Makefile de compilare pentru cele doua versiuni de convertor Serial la Ethernet (x86 si AVR32)

Makefile converter

```

CC = /<BUILD_ROOT_INSTALL_DIR>/buildroot-2010.11/output/staging/usr/bin/avr32-linux-gcc
CFLAGS = -g -Wall
INCLUDE=-I/inc/
SOURCES=src/s2e-converter.c
s2e-converter_OBJECTS= s2e-converter.o
OBJECTS= $(s2e-converter_OBJECTS)
PROGRAMS= s2e-converter
all: $(PROGRAMS)
s2e-converter:
    $(CC) $(CFLAGS) $(INCLUDE) -o $(PROGRAMS) $(SOURCES)
clean:
    rm -fr *.o s2e-converter

```

Makefile tester

```

CC = gcc
CFLAGS = -g -Wall
INCLUDE=-I/inc/
SOURCES=src/s2e-tester.c
s2e-tester_OBJECTS= s2e-tester.o
OBJECTS= $(s2e-tester_OBJECTS)
PROGRAMS= s2e-tester
all: $(PROGRAMS)
s2e-tester:
    $(CC) $(CFLAGS) $(INCLUDE) -o $(PROGRAMS) $(SOURCES)
clean:
    rm -fr *.o s2e-tester

```

Buildroot

Buildroot este un set de Makefileuri si patchuri care permite generarea usoara a unui toolchain de cross-compilare, a unui root filesystem si a unui kernel Linux pentru targetul embedded. Un toolchain este un set de instrumente pentru compilarea codului pentru un sistem embedded si este alcătuit din compilator (gcc), assembler si linker (binutils), librarii standard de C (GNU Libc, uCLibc, dietlibc). Buildroot poate fi obtinut de la adresa <http://buildroot.net/downloads/>.

Modificari aduse in buildroot la momentul configurarii mediului de build pentru generarea toolchainului, kernelului Linux si a filesystemului sub forma unor patchuri.

```
--- linux-2.6.36.1.orig/include/linux/kprobes.h>2010-12-30 23:41:46.030461016 +0200
```

```
+++ linux-2.6.36.1/include/linux/kprobes.h<---->2010-12-30 23:42:11.550774390 +0200
@@ -205,7 +205,7 @@
.

#ifndef CONFIG_KPROBES
DECLARE_PER_CPU(struct kprobe *, current_kprobe);
-DECLARE_PER_CPU(struct kprobe_ctlblk, kprobe_ctlblk);
+DECLARE_PER_CPU(struct kprobe_ctlblk *, kprobe_ctlblk);

--- linux-2.6.36.1.orig/drivers/usb/gadget/atmel_usba_udc.c<---->2010-12-30
23:54:37.910470155 +0200
+++ linux-2.6.36.1/drivers/usb/gadget/atmel_usba_udc.c<>2010-12-30 23:55:52.102731918 +0200
@@ -2016,7 +2016,7 @@
        }
    } else {
        /* gpio_request fail so use -EINVAL for gpio_is_valid */
-        ubc->vbus_pin = -EINVAL;
+        udc->vbus_pin = -EINVAL;
    }
}

--- linux-2.6.36.1.orig/arch/avr32/include/asm/syscalls.h<---->2010-12-30
12:36:02.631428197 +0200
+++ linux-2.6.36.1/arch/avr32/include/asm/syscalls.h<-->2010-12-30 12:36:30.371458714 +0200
@@ -21,8 +21,10 @@
    unsigned long, unsigned long,
    struct pt_regs *);
asmlinkage int sys_vfork(struct pt_regs *);
-asmlinkage int sys_execve(const char __user *, char __user * __user *,
- char __user * __user *, struct pt_regs *);
+asmlinkage int sys_execve(const char __user *ufilename,
+ const char __user *const __user *uargv,
+ const char __user *const __user *uenvp,
+ struct pt_regs *regs);
.

/* kernel/signal.c */
asmlinkage int sys_sigaltstack(const stack_t __user *, stack_t __user *,
```