EFICIENTIZAREA CONSUMULUI DE ENERGIE ÎN SISTEME TIMP-REAL

Teză destinată obținerii
titlului științific de doctor inginer
la
Universitatea Politehnica Timișoara
în domeniul CALCULATOARE ȘI TEHNOLOGIA INFORMAȚIEI

ing. Cristina-Sorina Certejan

Conducător științific:	prof. univ. dr. ing. Vladimir Ioan Crețu
Referenți științifici:	prof. univ. dr. ing. Nicolae Țăpuș
	prof. univ. dr. ing. Sergiu Nedevschi
	prof. univ. dr. habil. ing. Mihai V. Micea

Ziua susţinerii tezei: 19.06.2015

Seriile Teze de doctorat ale UPT sunt:

1. Automatică
2. Chimie
3. Energetică
4. Ingineria Chimică
5. Inginerie Civilă
6. Inginerie Electrică
7. Inginerie Electronică și Telecomunicații
8. Inginerie Industrială
9. Inginerie Mecanică
10. Știința Calculatoarelor
11. Știința și Ingineria Materialelor
12. Ingineria sistemelor
13. Inginerie energetică
14. Calculatoare și tehnologia informației
15. Ingineria materialelor
16. Inginerie și Management
17. Arhitectură
18. Inginerie civilă și instalații

Universitatea Politehnica Timișoara a inițiat seriile de mai sus în scopul diseminării expertizei, cunoștințelor și rezultatelor cercetărilor întreprinse în cadrul Școlii doctorale a universității. Seriile conțin, potrivit H.B.Ex.S Nr. 14 / 14.07.2006, tezele de doctorat susținute în universitate începând cu 1 octombrie 2006.

România, 300159 Timișoara, Bd. Republicii 9,
Tel./fax 0256 403823
e-mail: editura@edipol.upt.ro

Cuvânt înainte

Această lucrare marchează un punct important în activitatea mea de cercetare și astfel a-și dori să-i mulțumesc în primul rând lui Dumnezeu pentru că am ajuns aici și pentru persoanele deosebite alături de care am parcurs acest drum.

Astfel, îi sunt recunoscătoare și îi mulțumesc în mod deosebit conducătorului meu de doctorat, care a fost mereu alături de noi, și ne-a sprijinit pe parcursul cercetării doctorale și nu numai, domnului prof. dr. ing. Vladimir I. Crețu. Mulțumiri deosebite datorez celui fără de care probabil nu aș fi pornit pe acest drum, prof. dr. habil. ing. Mihai V. Micea, cel care m-a primit acum în urmă cu aproape 8 ani în echipa de cercetare a laboratorului DSPLabs și m-a inițiat, sprijinit și îndrumat pe acest drum al cercetării, descoperindu-mi frumusețea domeniului sistemelor timp-real și a cercetării în general. Mulțumiri datorez întregii echipe de la DSPLabs (îndeosebi as. dr. ing. Dan Chiciudean, ș.l. Răzvan Cioargă, apoi și colegilor ing. Lucian Ungurean, ing. Gabriel Cârstoiu, ing. Andrei Stancovici, ing. Ramona Tîrnovan), pentru cunoștințele împărtășite, pentru suport și pentru prietenia oferită pe parcursul acestor ani.

Mulțumesc domnilor prof. dr. ing Mircea Stratulat, prof. ing Ionel Jian, conf. ing. Marius Marcu și ș.l. ing. Sebastian Fuicu, pentru întrumarea și colaborarea oferită pe parcursul acestei activități de cercetare doctorală și nu numai.

Mulțumiri deosebite, aș dori să adresez și domnului prof. dr. ing. Toma L. Dragomir pentru șansa acordată prin colaborarea în cadrul grantului POSDRU-ATRACTING.

Mulțumiri deosebite datorez și doamnei ing. Claudia Micea, de asemenea și pr. conf. univ. dr. Constantin Jinga pentru suportul moral și tot ceea ce au făcut pentru mine.

Nu în ultimul rând datorez mulțumiri soțului și partenerului meu pe durata acestui drum, încărcat de împliniri, dar și de greutăți, as. drd. ing. Valentin Stângaciu, părinților mei Sorina și Marius Certejan, care m-au crescut și m-au susținut în procesul lung al formării mele educaționale; copiilor mei, fratelui, bunicilor, prietenilor și apropiaților mei care m-au sprijinit și îndurat mai ales în această perioadă, nu tocmai facilă.

De asemenea îndrept un gând de mulțumire tuturor dascălilor și profesorilor pe care i-am avut și care m-au format.

Timişoara, iunie, 2015 Cristina-Sorina Certejan

Familiei mele.

Această lucrare a fost finanțată parțial din proiectul strategic POSDRU/159/1.5/S/137070 (2014) al Ministerului Educației Naționale, Romania, co-finanțat din Fondul Social European – Investește în oameni, în cadrul Programului Operațional Sectorial Dezvoltarea Resurselor Umane 2007-2013

Certejan, Cristina-Sorina

Eficientizarea consumului de energie în sisteme timp-real

Teze de doctorat ale UPT, Seria 14, Nr. 26, Editura Politehnica, 2015, 120 pagini, 41 figuri, 11 tabele.

ISSN: 2069-8216

ISSN-L: 2069-8216

ISBN: 978-606-554-963-0

Cuvinte cheie: Sisteme timp-real, Algoritmi de planificare, Aplicații critice

Rezumat, Lucrarea de față abordează domeniul sistemelor timp-real incorporate, punând accent pe cele critice. Problema vizată este cea a reducerii consumului de energie în aceste sisteme, având în vedere particularitățile acestora.

În vederea soluționării acestei probleme, autoarea face o analiză a principalelor metode de reducere a energiei în sisteme timp real încorporate, referind peste 90 de lucrări și făcând o comparație între principalele tipuri de metode, având în vedere impactul asupra sistemelor critice.

Pe baza pricipalelor caracteristici date pe de o parte de sistemul țintă și pe de altă parte de aplicațiile timp-real este propus un model de comportament energetic și temporal al sistemului și al aplicațiilor la nivel de task.

Pentru determinarea parametrilor din modelul propus și pentru analiza aplicației și a sistemului este propus un sistem cadru modular.

Soluția de eficientizare a consumului datorat rulării unei aplicații pe sistemul țintă este oferită de către un algoritm de planificare a taskurilor unei aplicații timp real, cu funcție de eficientizare a consumului.

Acest algoritm și modelul energetic și temporal aferent au fost validate și analizat prin implementarea pe două platforme țintă și prin măsurarea perfomanțelor sale, obținându-se reduceri între 50-66% pentru studiile de caz prezentate.

Cuprins

Listă acronime

Listă tabele

Lista de figuri

1 Introducere

1.1 Domeniile și tema cercetării

Teza de față abordează o serie de domenii conexe, dintre care principalele două sunt reprezentate de către:

- *cel al sistemelor timp-real*, un domeniu vast, complex, cu o vechime de peste patruzeci de ani, a cărui dezvoltare a fost strâns legată de cea a unui alt domeniu și anume:
- cel al *sistemelor încorporate,* de asemenea un domeniu vast, ce cunoaște o dezvoltare continuă și o aplicabilitate din ce în ce mai mare în, practic, toate domeniile de activitate ale omului;

Pornind de la aceste domenii principale, cercetarea se axează pe alte subdomenii și anume:

- cel al rețelelor de senzori și dispozitive inteligente
- cel al tehnicilor de planificare pentru aplicații timp-real și
- cel al consumului de energie electrică în sisteme timp-real încorporate, un domeniu relativ nou, de un interes deosebit, fapt dovedit mai ales prin numărul mare de publicații recente în domeniu, număr aflat într-o continuă creștere.

Tema cercetării cuprinse în această teză este cea *a cadrelor suport și a metodologiilor de reducere a energiei electrice în sistemele timp-real încorporate*.

1.2 Motivația cercetării

În ultimii zece ani a existat o puternică dezvoltare tehnologică în următoarele direcții: rețele digitale, comunicații fără fir și dezvoltarea de dispozitive încorporate tot mai mici si mai ieftine. Aceste progrese sporesc potențialul de dezvoltare al unui domeniu mult mai complex și anume: domeniul rețelelor de senzori fără fir și dispozitive inteligente. Acest domeniu bucurându-se de un interes major din partea comunității academice, industriale și chiar al organismelor economice internaționale [1], fapt datorat vastei aplicabilități a acestor sisteme. Practic acest tip de rețele se integrează în aproape toate domeniile activității umane. Cu greu ne mai putem imagina în ziua de astăzi un automobil modern fără computer de bord, senzori și alte dispozitive digitale încorporate. Rețelele de senzori sunt prezente de la aplicații militare, ca deminare, explorarea terenului ostil, aplicații în domeniul prevenirii pericolelor și menținerii siguranței populației, recuperări după dezastre, monitorizarea zonelor cu potențial pericol: păduri, clădiri, poduri și diferite alte structuri, monitorizarea poluării în diferite medii, aplicații medicale de tipul monitorizării stării pacienților, aplicații industriale diverse, până la agricultură și aplicații domestice. Un exemplu de aplicație, dezvoltat în cadrul departamentului de Calculatoare al Universității *Politehnica* Timișoara este un sistem de monitorizare, analiză, control și predicție a consumului de energie pentru diferite dispozitive electrice [2, 3]. Acest sistem este bazat pe o rețea de senzori ce comunică cu un server, oferind și o interfață web pentru utilizator, prin intermediul căreia se pot analiza datele colectate, care sunt interpretate sub diferite forme,

incluzând grafice. Pe baza acestor date se pot lua decizii și se pot comanda dispozitivele în timp real.

O categorie specială de aplicații destinate rețelelor de senzori sunt cele critice, acele aplicații timp real, având constrângeri stricte de timp.

Prin sistem timp-real înțelegem ceea ce „A Dictionary of Computing" definește ca fiind "*orice tip de sistem pentru care timpul de răspuns este un parametru esențial. Acest lucru se datorează faptului că intrările acestuia corespund unor variații din mediul fizic înconjurător în așa fel încât ieșirile sistemului trebuie să corespundă la rândul lor, acelorași variații. Decalajul dintre timpii de intrare și cei de ieșire trebuie sa fie suficient de mic, relativ la tipul aplicațiilor implementate*" [4].

Funcționarea corectă a aplicațiilor critice implică pe lângă corectitudinea rezultatelor furnizate și respectarea strictă a constrângerilor de timp [5]. Astfel de exemple sunt: supravegherea și prevenirea incendiilor, aplicații medicale, siguranță, etc.

Legat de acest tip de aplicații, literatura de specialitate menționează următoarele probleme, încă de actualitate [6]:

- *Problema constrângerilor de timp și a predictibilității*: datorită naturii aplicațiilor și a interacțiunii acestora cu mediul înconjurător, se impune ca sistemul să reacționeze la evenimentele externe într-un interval de timp bine delimitat, într-un mod predictibil.
- *Problema energiei electrice*: aceste dispozitive sunt alimentate de obicei de o sursa cu capacitate finită (baterie). Se pune problema prelungirii timpului de funcționare al acestor dispozitive, având grijă totodată de a nu periclita corectitudinea informației furnizate, cât și garantarea în continuare a respectării condițiilor de timp.
- *Problema resurselor limitate*: datorită limitărilor date de dimensiunile și costurile cât mai mici ale acestor dispozitive, avem de a face cu unități de procesare cu putere computațională și memorie limitată. Astfel, se impune o gestionare cât mai eficientă a resurselor.

Activitatea de cercetare, sintetizată prin lucrarea de față, abordează problema energiei electrice și propune ameliorarea acesteia prin folosirea unei metodologii și al unui cadru complet de analiză, estimare și eficientizare a consumului de energie electrică la nivel de platformă țintă.

1.3 Obiective propuse

Principala țintă a tezei este aceea de a propune un cadru suport și o metodologie pentru analiza consumului unei aplicații timp-real critice pe un sistem țintă (încorporat) și de a oferi o soluție de reducere a acestui consum. Pentru atingerea acestei ținte au fost fixate următoarele obiective:

1. *Conceperea și dezvoltarea unui model de consum de energie la nivel de sistem pentru o serie de clase de dispozitive timp-real încorporate, definite în prealabil.*
2. *Conceperea și dezvoltarea unui cadru, reprezentând un mediu hardware și software și a unei metodologii aferente, pentru măsurarea și determinarea parametrilor modelului consumului de energie de la punctul precedent.*
3. *Dezvoltarea unui mecanism de planificare, pe mai multe niveluri, pentru sisteme timp-real încorporate, cu funcție de eficientizare a consumului de energie electrică.*

4. *Validarea mecanismului de planificare, prin execuția aplicației pe sistemul țintă și realizarea unei analize a comportării aplicației timp-real direct pe sistem.*

1.4 Contextul de realizare

Teza de față prezintă contribuțiile proprii și sintetizează activitatea de cercetare, cuprinsă în programul de doctorat cu tema "*Planificarea sistemelor de timp real cu funcție de eficientizare a consumului*", pe care am întreprins-o sub conducerea științifică a domnului prof. dr. ing. Vladimir I. CREȚU, în cadrul laboratorului de prelucrare numerică a semnalelor DSPLabs, parte componentă Centrului de Cercetare în Calculatoare și Tehnologia Informației (CCCTI) al facultății de Automatică și Calculatoare, departamentul Calculatoare, Universitatea *Politehnica* Timișoara.

Această cercetare a fost finanțată parțial din proiectul strategic POSDRU/159/1.5/S/137070 (2014) al Ministerului Educației Naționale, Romania, co-finanțat din Fondul Social European – Investește în oameni, în cadrul Programului Operațional Sectorial Dezvoltarea Resurselor Umane 2007-2013

Activitatea este o continuare a preocupărilor de cercetare în domeniile vizate, începute din timpul programului de studii de licență și continuate pe parcursul studiilor de master.

1.5 Structura lucrării

Lucrarea de față este structurată în șapte capitole, după cum urmează:

- Primul capitol constituie o scurtă introducere în domeniul cercetării, motivând alegerea temei și prezentând obiectivele cercetării și contextul de realizare.
- Al doilea capitol definește și prezintă pe scurt principalele noțiuni teoretice utilizate în această lucrare.
- Al treilea capitol reprezintă starea domeniului de cercetare și prezintă totodată o analiză a problematicii vizate și a principalelor metode de abordare prezente în literatura de specialitate la momentul scrierii tezei, evidențiind aspectele care încă nu și-au găsit o rezolvare deplină.
- În al patrulea capitol, este propus un sistem cadru, numit TEEARTS. Acesta este un sistem modular, pentru analiza și planificarea aplicațiilor timp real, pe o platformă țintă, cu funcție de eficientizare a consumului de energie electrică. Pentru fiecare principal modul component al sistemului cadru este dedicat un subcapitol separat.
- Al cincilea capitol prezintă atingerea principalului obiectiv al cercetării de față și anume planificarea aplicațiilor timp-real (critice), pe sistemul țintă, cu funcție de eficientizare a consumului. În acest capitol este descris pe larg algoritmul propus în acest scop.
- Al șaselea capitol prezintă validarea, prin implementarea pe două platforme țintă și prin analiza unor studii de caz, de asemenea, prezintă analiza performanțelor acestui algoritm și implicit validarea sistemului cadru.
- În ultimul capitol sunt prezentate concluziile acestei cercetări, principalele contribuții și perspective de dezvoltare.

2 Noțiuni teoretice

În capitolul de față sunt prezentate o serie de noțiuni teoretice ce constituie baza și punctul de pornire al cercetării curente. Principalul scop al acestui capitol este cel de a defini termenii utilizați pe parcursul acestei lucrări.

În cele ce urmează vor fi definite, descrise pe scurt și clasificate sistemul și aplicațiile timp-real. De asemenea se vor prezenta modele ale acestor aplicații, ale taskurilor acestora; iar la final vor fi prezentați și clasificații principalii algoritmi de planificare consacrați aferenți acestor taskuri.

2.1 Sisteme timp-real

După cum a mai fost menționat, un sistem de timp real este un sistem în care timpul de producere a rezultatelor este la fel de important ca și rezultatele în sine. Această condiție este dată de interacțiunea sistemului cu mediul înconjurător (fie el natural sau artificial) de la care primesc informații prin intermediul senzorilor și pe care îl controlează. Răspunsul la anumite evenimente trebuie să apară într-un interval bine determinat (ex. acționarea airbag-ului la mașină după detecția unui impact), încălcarea acestui interval putând avea consecințe catastrofale.

Aplicabilitatea acestor sisteme este din ce în ce mai mare, fie că sistemele sunt de sine stătătoare, fie că sunt integrate în rețele (o situație din ce în ce mai întâlnită în ziua de astăzi).

În funcție de strictețea aplicațiilor ce rulează pe acestea, sistemele se împart în [7]:

- *Critice*: În aceste sisteme, respectarea constrângerilor de timp este o problemă critică. Astfel de sisteme se pot întâlni la controlul reactoarelor unei centrale nucleare, la controlul zborului unui avion. Pentru aceste aplicații, nerespectarea limitelor de timp are efecte catastrofale.
- *Stricte*: În cazul acestor sisteme, respectarea constrângerilor de timp este necesară pentru o comportare corectă a sistemului. Nerespectarea termenelor în astfel de sisteme duce la furnizarea de rezultate greșite și/sau poate duce la pagube materiale (ex. funcționarea eronată a unui ciclu de producție, ce poate duce la anularea unei serii întregi de produse).
- *Lejere*: Sunt acele sisteme, în care depășirea termenelor de timp duce la o scădere a calității serviciului furnizat. Astfel de sisteme se găsesc cu precădere în domeniul multimedia (ex. depășirea termenelor în cazul unei transmisii audio/ video duce la o slabă calitate a acestei transmisii prin pierderea de cadre).

La rândul lor constrângerile de timp se împart în constrângeri "*hard*" (în sensul de critice), "*firm*" (stricte) și "*soft*" (lejere) [8].

În privința sistemelor timp-real sunt semnalate unele concepții greșite [9]. Se poate crede că problemele legate de constrângerile de timp pot fi rezolvate prin folosirea unor procesoare foarte puternice și rapide. Într-adevăr o viteză de computație mare ajută sistemul să ofere un răspuns cât mai rapid, dar se pune problema predictibilității, iar aceste procesoare foarte performante de obicei nu sunt predictibile. Acest lucru poate fi datorat arhitecturii lor complexe (paralelizărilor interne, folosirii

memoriei cache și a altor mecanisme impredictibile). De aceea în dezvoltarea unor astfel de sisteme suntem constrânși la anumite tipuri de arhitecturi.

Pentru astfel de sisteme fizice s-au dezvoltat sisteme de operare specifice și anume sistemele de operare timp-real (*RTOS, Real Time Operating Systems*). Caracteristicile unor astfel de sisteme de operare, dintre care putem aminti LynxOS și QNX sunt [8, 10]:

- *Multitasking:* în primul rând aceste sisteme trebuie să ofere suport pentru rularea în timp real a mai multor taskuri.
- *Nivele de priorități*: trebuie să ofere suport pentru definirea de nivele de priorități diferite pentru diferite taskuri.
- *Planificare:* un sistem de operare în timp real oferă cel puțin 32 niveluri de priorități, conform standardului POSIX.
- *Viteză și eficiență:* timpi de deservire a rutinelor de întrerupere mici, schimbări de context puține și rapide.
- *Apeluri sistem*: părțile din nucleu care sunt non-preemptive sunt cât mai scurte și deterministe.
- *Rutine de deservire a întreruperilor:* partea non-preemptivă a rutinelor de întrerupere este scurtă.
- *Controlul inversiunilor de prioritate*: există mecanism de control, dar poate fi dezactivat pentru a reduce timpul de răspuns.
- *Mecanisme de comunicare inter task*: pentru a asigura integritatea datelor, sistemele trebuie să pună la dispoziție mecanisme de sincronizare și comunicare inter task fiabile.
- *Rezoluția de timp și ceas:* rezoluția de ceas poate fi de ordinul nanosecundelor, dar rezoluția cu care este reprezentat timpul este de ordinul unei microsecunde datorită timpului de deservire a întreruperilor de ceas și a modului de reprezentare al timpului.
- *Modularitate și scalabilitate:* nucleul este mic și sistemul de operare configurabil.
- *Managementul memoriei:* sistemul poate oferi mapare virtuală sau fizică a adreselor de memorie, dar fără paginare.
- *Rețelistică*: sistemul poate fi configurat să suporte comunicare de tip TCP/IP.

Există și anumite concepții greșite asupra sistemelor de operare timp real [10] dintre care menționăm ideea despre viteză. Se poate presupune că un sistem de operare timp real trebuie să fie rapid. De fapt trăsătura lui de bază este determinismul și nu viteza de procesare, care este totdeauna raportată la constrângerile de timp.

În concluzie un sistem de timp real este un sistem al cărui răspuns este determinist și suficient de rapid pentru a garanta respectarea termenelor de timp, în cazul cel mai defavorabil de operare [11].

2.2 Modelarea temporală a aplicațiilor timp-real

Tipurile de aplicații vizate în această lucrare sunt cele critice. Și anume, acele aplicații destinate sistemelor pentru care respectarea constrângerilor de timp este o problemă vitală. Exemple de astfel de sisteme se pot întâlni la controlul sistemelor de siguranță ale reactoarelor unei centrale nucleare, ale unui avion, autovehicul etc.

Aplicațiile dedicate sistemelor timp-real se împart în taskuri (secvențe de instrucțiuni, executate de sistem). Printr-un task înțelegem ceea ce s-a definit în [9]

ca fiind *"o entitate executabilă, care în mod minimal este caracterizată printr-un timp de execuție pentru cel mai defavorabil caz și o constrângere de timp".*

Taskurile pot fi clasificate în:

- *periodice* - taskuri ce sunt activate regulat la intervale constant de timp, numite perioade și pot fi executate doar o singură dată per perioadă;
- *aperiodice* – taskuri ce sunt activate neregulat, fără a respecta anumite condiții;
- *sporadice* - taskuri ce sunt activate neregulat, dar între două activări succesive există un interval minim de timp [9].

Pentru toate tipurile de taskuri enumerate mai sus constrângerile de timp pot fi termenul pentru finalizarea execuției unei instanțe *(deadline-ul)* și/sau momentul de lansare al taskului în execuție.

În această lucrare s-a luat în considerare modelul taskurilor critice independente. Parametrii de timp considerați fiind:

- Perioada taskului *i* (T_i), definită ca *"intervalul de timp maxim dintre invocările taskului* i"[12].
- Timpul de execuție al taskului *i* (C_i), definit ca *"intervalul de timp maxim necesar pentru a executa complet task-ul* i *pe un procesor dedicat*"[12].
- Termenul (*deadline*) absolut al taskului *i* (d_i) definit ca *"un moment în timp la care instanța curentă a taskului* i *trebuie să fi fost executată complet*"
- Termenul (*deadline*) relativ al taskului *i* (D_i) definit ca *"deadline-ul calculat relativ la momentul activării taskului (începutul perioadei)"*
- Prima invocare a taskului *(release time) (I_i)*– momentul de timp la care taskul devine disponibil pentru execuție, tratarea lui se va face doar după acest moment [7].
- Timpul de răspuns *(response time)* (R_i)– intervalul de timp măsurat între apariția unei instanțe a unui task și momentul la care execuția ei este completă [8].

Un caz particular de task este așa numitul ModX (*Modul Executabil*), un task periodic critic, complet specificat de parametrii săi temporali, planificat și executat într-un mediu nonpreemptiv [11].

Planificarea taskurilor se referă la găsirea de soluții fiabile pentru asignarea procesorului, pentru fiecare în parte, astfel încât să nu existe suprapuneri ale execuției lor pe durata operării sistemului [11].

O aplicație pentru sisteme de timp real, poate fi modelată printr-un graf direcționat, aciclic cu următoarele componente [7]:

$$G \equiv < \theta, V > \tag{2-1}$$

unde θ reprezintă mulțimea nodurilor grafului (taskurile aplicației), iar V reprezintă mulțimea arcelor ordonate:

$$V \equiv \{(\theta_i, \theta_j) \,|\theta_i, \theta_j \in \theta, 1 \le i \le n\} \tag{2-2}$$

unde n reprezintă numărul de taskuri.

Un task este specificat la rândul său printr-o serie de parametri:

$$\theta_i \equiv < T, P, I, D > \tag{2-3}$$

unde ***T*** reprezintă setul de parametrii temporali, ce modelează comportarea în timp a taskului, ***P*** reprezintă setul de parametrii de intrare, ieșire și variabile globale ai

taskului, ***I*** reprezintă setul de instrucțiuni al taskului, iar ***D*** reprezintă setul de dispozitive hardware accesate de task.

Pentru specificarea acestor parametri temporali, trebuie bine definit, în primul rând un sistem al coordonatelor de timp. În sistemele de timp real, timpul reprezintă o coordonată esențială atât în dezvoltarea cât și în execuția aplicației.

2.3 Reprezentarea timpului în sisteme timp-real

Presupunem modelul unui sistem clasic, sincron, monoprocesor, în care frecvența cu care datele sunt procesate este dată de frecvența procesorului. În acest tip de sistem dacă avem o frecvență de lucru constantă pentru procesor putem defini timpul sistem ca fiind o reprezentare a timpului numerică (discretă), de o granularitate egală cu perioada tactului procesor:

$$\tau_{CLK} = \frac{1}{f_{CLK}} \tag{2-4}$$

unde τ_{CLK} reprezintă perioada tactului procesor, iar f_{CLK} frecvența de lucru a acestuia.

Pentru un procesor cu frecvență variabilă, trebuie aleasă o frecvență de referință, pe care o vom numi în continuare *frecvența nominală*.

$$\tau_{nom} = \frac{1}{f_{nom}} \tag{2-5}$$

Această frecvență nominală a procesorului, va reprezenta frecvența de referință a întregului sistem. Astfel modelul temporal considerat se bazează pe modelul clasic al sistemelor de timp real prezentat în [7], la care se adaugă anumite proprietăți de tranziție dintr-un sistem de timp în altul, datorat frecvențelor variabile.

În continuare vom prezenta timpul sistem ca o funcție t_{STR} cu următoarele proprietăți:

1. Funcția de timp este definită pe un domeniu pozitiv:

$$t_{STR} : \mathbf{R}^{+} \rightarrow \mathbf{R}^{+} \tag{2-6}$$

2. Funcția este strict crescătoare:

$$x_1 < x_2 \Rightarrow t_{STR}(x_1) < t_{STR}(x_2) \tag{2-7}$$

unde x_1, x_2 reprezintă instanțe de timp, iar $t_{STR}(x1)$ sunt valorile de timp corespunzătoare acelor instanțe exprimate în cicluri de tact. În continuare se vor utiliza următoarele notații:

$$t(x_n) = t_n \tag{2-8}$$

Considerăm că valoarea pentru prima instanță de timp, corespunzătoare momentului când sistemul este pornit și gata să lanseze în execuție aplicații, este egală cu 0.

$$t_0 = 0 \tag{2-9}$$

Corespondența dintre timpul sistem și timpul absolut este dată de relația:

$$t_{abs}(x_i) = t_{abs}(x_0) + [t_{STR}(x_i) - t_{STR}(x_0)] \cdot \tau_{nom} \quad (2\text{-}10)$$

2.4 Modele temporale de taskuri

Modelul de taskuri vizat în această lucrare este cel al taskurilor periodice, independente. Pentru deservirea unor evenimente asincrone, vom utiliza anumite taskuri speciale, numite *servere de deservire a evenimentelor,* care sunt tot taskuri periodice, a căror perioadă este condiționată de tipul evenimentului deservit. Taskurile periodice pot fi specificate într-un mod minimal prin parametrii lor de timp:

$$T_i \equiv \{C_i, T_i, D_i, \varphi_i\} \quad (2\text{-}11)$$

- C_i – este timpul de execuție al taskului *i* în cazul cel mai defavorabil, altfel spus timpul maxim de execuție sau WCET (W*orst Case Execution Time*). Pentru un sistem de timp real, acest timp de execuție maxim trebuie să fie bine determinat. În practică, acest parametru reprezintă suma timpilor de execuție ce cuprind calea cea mai lungă de execuție a programului, în condițiile de operare cele mai defavorabile.
- T_i – este perioada taskului *i.* Pentru modelul taskurilor periodice această perioadă este constantă pe durata execuției aplicației, nu diferă de la o execuție a unui task la alta.
- D_i – este intervalul limită pentru planificarea și execuția taskului *i* în cadrul perioadei acestuia.
- φ_i– reprezintă defazajul sau timpul de start al execuției taskului *i* în cadrul perioadei acestuia și este dat de algoritmul de planificare.

Între parametrii de timp se stabilesc următoarele relații:

$$0 < C_i \le T_i \quad (2\text{-}12)$$

$$D_i = T_i \quad (2\text{-}13)$$

Prin aceste relații este definit un task, periodic de perioadă T_i, cu un timp de execuție nenul și pentru care termenul limită *(deadline)* este egal cu perioada sa.

Primii trei parametri menționați anterior sunt esențiali în procesul de planificare a taskurilor, iar φ_i, al patrulea parametru, este specificat în urma planificării și este dependent de algoritmul de planificare.

Scopul determinării acestor parametrii de timp este dublu. Pe de o parte se dorește specificarea cât mai exactă a comportării taskurilor în timp, comportare dată de cerințele aplicației, pe altă parte se dorește obținerea de seturi de taskuri fezabile din punct de vedere al planificării acestora.

Spunem că setul de taskuri θ_i este fezabil din punct de vedere al planificării, dacă există un anumit algoritm capabil să planifice pentru execuție toate taskurile setului, astfel încât ele să-și respecte specificațiile de operare temporală, pe toată durata operării sistemului [7].

Astfel, pe de o parte avem cerințele aplicației, care se pot împărți în două mari categorii:

- *Cerințe funcționale* – sunt acele cerințe care se referă la funcționalitatea sistemului. Pe baza acestor cerințe se dezvoltă codul fiecărui task, astfel

acestea determină practic setul de instrucțiuni ***I***, parametrii de intrare și ieșire ***P*** și setul dispozitivelor hardware accesate ***D*** după cum apar în modelul (2-3).

- *Cerințele temporale* - sunt acele cerințe care se referă la comportarea în timp a sistemului. Aceste cerințe sunt specificate prin condiționarea directă a parametrilor temporali ai taskurilor: pentru perioada de execuție T_i se poate specifica o valoare maximă, dată de frecvența cu care trebuie executat taskul *i*, iar în cazul serverelor (periodice) de deservire a evenimentelor valoarea maximă este dată de timpul maxim de răspuns tolerat de sistem.

$$0 < T_i \le T_{i\_\max} \tag{2-14}$$

În ceea ce privește timpul de execuție C_i, acesta este determinat în mod direct de setul de instrucțiuni ***I*** și restricționat de relația (2-12), ca fiind mai mic decât timpul de execuție. Astfel, acesta este condiționat atât de cerințele funcționale, cât și de cele temporale.

Cei doi parametrii de timp principali C_i și T_i pot fi variați pentru obținerea unor seturi fezabile de taskuri, între anumite limite.

Astfel, pentru obținerea unui timp de execuție mai mic se pune problema divizării taskurilor: un task să fie împărțit în două sau mai multe taskuri. Problema divizării taskurilor este echivalentă cu problema găsirii unui număr finit de puncte în care un task poate fi întrerupt, fără a crea probleme prin schimbările de context efectuate sau de a transforma un set fezabil de taskuri într-unul nefezabil prin adăugarea unor timpi suplimentari datorați lansării în execuție a unui număr superior de taskuri. Această problemă a fost studiată și soluționată în [13], unde este oferit un algoritm de determinare a unui număr optim de puncte de întrerupere în cod pentru planificarea cu doi dintre algoritmii consacrați (EDF și FP).

Obținerea unei îmbunătățiri în ceea ce privește fezabilitatea setului de taskuri se poate realiza și prin variația perioadei taskurilor, în limita valorilor impuse de cerințele aplicației. Combinând relațiile (2-12) și (2-14) obținem limitele de variație a perioadei unui task T_i.

$$C_i < T_i \le T_{i\_\max} \tag{2-15}$$

În practică, valorile acestor perioade sunt alese astfel încât să se obțină, dacă este posibil, un set fezabil de taskuri [14].

2.5 Algoritmi de planificare

Planificarea taskurilor se referă la găsirea de soluții fiabile pentru asignarea procesorului, pentru fiecare în parte, astfel încât să nu existe suprapuneri ale execuției lor pe durata operării sistemului [11].

Algoritmii de planificare se pot clasifica în [7]:

- După numărul de procesoare sistem:
  - Algoritmi monoprocesor;
  - Algoritmi multiprocesor.
- După momentul de generare al planificării:
  - Algoritmi statici (planificare generată offline);
  - Algoritmi dinamici (planificare generată online, în timpul operării sistemului).

- În funcție de acceptarea sau nu a întreruperilor:
  - Algoritmi preemptivi (se admite întreruperea task-ului curent de către un alt task cu prioritate mai mare);
  - Algoritmi non-preemptivi (nu se admit întreruperi ale task-urilor).

În ceea ce privește tehnicile de planificare există mai multe direcții de abordare între care amintim [8]:

- Abordarea bazată pe mecanismul de timp (*Clock-Driven*)

Este o abordare bazată pe algoritmi statici de planificare, deciziile privind ce task se va executa la care moment de timp se iau a priori, înainte ca sistemul să-și înceapă execuția. Parametrii de timp pentru taskurile cu restricții critice de timp sunt ficși și cunoscuți. Se face o planificare a taskurilor, conform acestor parametrii, la momente exacte de timp, înainte de a începe execuția sistemului, apoi la momente bine definite de timp planificarea se reia. O practică frecventă este ca planificarea să se repete la intervale regulate, planificarea făcându-se pe cicluri de timp (cicluri de planificare). Lansarea planificatorului se face pe întreruperi periodice de ceas.

- Abordarea bazată pe mecanisme de tip coadă (*Round-Robin*)

Este o abordare folosită în mod curent pentru planificarea aplicațiilor ce partajează timpul. Taskurile care sunt gata de execuție sunt puse într-o structură FIFO (*First In First Out* – primul intrat primul ieșit). Din coadă se scoate prima aplicație pentru a fi rulată pentru un interval de timp alocat (*time slice*), dacă taskul nu este gata la finalul intervalului de timp el este întrerupt și pus la sfârșitul cozii, apoi se scoate din coadă taskul din vârf și operațiunea se repetă.

- Abordarea bazată pe priorități (*Priority-Driven*)

Se referă la o clasă largă de algoritmi de planificare care nu lasă nici o resursă inactivă în mod intenționat. Altfel spus o resursă este neutilizată doar dacă nu este nici un task activ care să o solicite. Decizii de planificare se fac atunci când apar evenimente ca lansarea unei noi instanțe a unui task sau terminarea execuției acesteia. Fiecărui task i se asignează un nivel de prioritate, taskurile care sunt gata să fie lansate se pun într-o coadă care este ordonată în funcție de priorități. Exemple de astfel de algoritmi sunt EDF (*Earliest Deadline First*), MLF (*Minimum Laxity First*), SETF (*Shortest Execution Time First*), RM (*Rate Monotonic*), DM (*Deadline Monotonic*) etc.

După cum s-a menționat, algoritmii pot fi dinamici sau statici. Într-un sistem dinamic, taskurile se alocă în mod dinamic procesorului. Astfel, o abordare bazată pe priorități este una de tip dinamic. Decizia, ce task va fi lansat la un moment dat, se ia în timpul rulării [8].

Algoritmii dinamici sunt singura soluție într-un sistem în care încărcarea sistemului este nepredictibilă, dar fără a ști dinainte despre viitoarele taskuri, planificatorul nu poate lua decizii optime.

Algoritmii de planificare statică se bazează pe o analiză offline a planificabilității aplicației pe acel sistem. Aceasta presupune pe de o parte analiza condițiilor în care planificarea este fezabilă (examinarea fezabilității condițiilor de timp pentru fiecare task), iar pe de altă parte stabilirea timpilor de lansare a fiecărei instanțe a unui task sau ordinea de lansare a acestora (planificarea efectivă). După cum am menționat, această planificare se poate relua în mod ciclic [11].

Avantajele tehnicilor statice sunt că ele oferă predictibilități mari ale operării sistemului. Pentru aceste sisteme există tehnici consacrate de validare a constrângerilor de timp. De asemenea complexitatea implementării unor astfel de sisteme este mai mică, de aceea ele pot fi implementate cu ușurință pe microcontrolere.

Pe de altă parte aceste tehnici oferă și mari dezavantaje. În primul rând utilizarea procesor de către aplicații este mică, ceea ce restrânge drastic cazurile de planificare

cu succes a acestora. Se face resimțită și lipsa de flexibilitate (modificarea unui parametru duce la refacerea analizei de planificabilitate offline) [11]. Această abordare este posibilă doar atunci când sistemul este determinist, însemnând că sistemul furnizează un set de task-uri fix, și că timpul de lansare și timpul necesar unei execuții complete este cunoscut și nu variază sau variază puțin (în practică se folosește ca parametru WCET (Worst Case Execution Time) al unei instanțe, adică timpul maxim de utilizare procesor pe care îl poate solicita o instanță) [8].

În funcție dacă admit sau nu întreruperi, tehnicile de planificare se împart în *preemptive* (execuția oricărui task poate fi întreruptă de un alt task cu prioritate mai mare) și *non-preemptive* (execuția unui task nu poate fi întreruptă) [15].

Algoritmii preemptivi s-au bucurat de o mai mare atenție [16, 17] datorită faptului că nu generează probleme computaționale de ordin nepolinomial, asemenea algoritmilor non-preemptivi [11].

Algoritmii preemptivi oferă o serie de avantaje dintre care menționăm în primul rând faptul că algoritmii de genul EDF, LST (Least Slack Time) sunt optimi în varianta lor preemptivă, dar nu și într-o variantă non-preemptivă [8]. Înțelegem prin optimi, capacitatea de a oferi o soluție fezabilă de planificare pentru un set de taskuri, dacă aceasta există [15].

Un alt avantaj îl constituie valoarea mai mare a factorului de utilizare procesor, față de soluția non-preemptivă.

Dezavantajele oferite de acești algoritmi nu sunt nici ele neglijabile. În primul rând sunt mai greu de implementat, necesită mai bune resurse fizice, deoarece pot folosi structuri mai mult sau mai puțin complicate de stocare a taskurilor în așteptare (cozi, stive, arbori). Pe de altă parte este necesară refacerea contextului, a stării taskului întrerupt, ceea ce implică apariția unor timpi in plus de utilizare procesor. De asemenea în aceste sisteme pot apărea anomalii de planificare și alte probleme legate de blocarea resurselor, care duc la scăderea gradului de predictibilitate.

La rândul lor, algoritmii non-preemptivi oferă o serie de avantaje. Pentru multe probleme practice de planificare a task-urilor de timp real, ca de exemplu taskurile care interacționează cu semnale de intrare/ieșire ale sistemului, lucrul cu întreruperile nu este permis sau este prea costisitor.

Algoritmii de planificare non-preemptivi sunt mai ușor de implementat, și folosesc un număr de resurse mai mic pentru planificarea din timpul operării sistemului.

Comportamentul și resursele de timp și de calcul ale algoritmilor de planificare preemptivă sunt mai dificil de caracterizat și de modelat, spre deosebire de cei non-preemptivi. Mai mult, în cele mai frecvente abordări, resursele sistem ocupate în timpul planificării sunt ignorate, rezultând că implementarea unui algoritm non-preemptiv este mai apropiată de modelul formal.

Datorită faptului că taskurile au acces exclusiv la resursele partajate, se elimină necesitatea și costul implementării mecanismelor de sincronizare.

Există însă și o serie de dezavantaje ale algoritmilor non-preemptivi [11]. Un prim dezavantaj este problema granularității. Dacă se lucrează cu o granularitate prea mică (taskuri de dimensiuni prea mari), planificarea non-preemptivă devine extrem de dificilă. Dacă, dimpotrivă, se lucrează cu o granularitate prea mare (taskuri de dimensiune mică), se complică foarte mult mecanismele de transmitere a parametrilor și de salvare/ restaurare a contextului între taskurile aplicației.

Un alt dezavantaj este lipsa de flexibilitate. Odată cu eliminarea întreruperilor din sistem, se elimină și capacitatea sistemului de a face față unor configurații noi, asincrone, ale mediului în care operează și cu care interacționează.

3 Abordări curente privind reducerea consumului de energie electrică

Motto: *"O problemă fără soluție este o problemă greșit formulată."* ***Albert Einstein***

Acest capitol trece în revistă principalele abordări privind consumul de energie electrică pentru sisteme timp-real încorporate, prezentând totodată starea domeniului de cercetare.

Capitolul începe prin descrierea problematicii consumului de energie electrică în sistemele vizate, urmând să prezinte principalele direcții de abordare a acestei probleme. Apoi, sunt prezentate comparativ și analizate din punct de vedere al reducerii de energie și al posibilelor implicații asupra comportării temporale și a predictibilității sistemului timp-real critic, principalele soluții propuse din literatura de specialitate. Capitolul se încheie cu un tabel comparativ al acestor soluții.

3.1 Problematica consumului de energie electrică în sisteme timp-real încorporate

Problematica reducerii consumului de energie electrică în sistemele timp-real încorporate este una delicată, datorită cerințelor speciale ale acestor sisteme, dintre care cele mai importante fiind: respectarea constrângerilor de timp și menținerea predictibilității sistemului. La aceste cerințe se adaugă și limitările datorate utilizării unor resurse limitate, ca rezultat al dimensiunilor și costurilor cât mai mici a dispozitivelor folosite. De aceea, mecanismele dezvoltate pentru alte tipuri de sisteme cum ar fi serverele sau sistemele de calcul de uz general nu se pot aplica și în cazul curent, acest aspect ducând la deschiderea unei noi direcții de cercetare.

În analiza problematicii consumului de energie electrică pentru sistemele încorporate s-a început prin identificarea și clasificarea pe de o parte a principalilor consumatori și pe de altă parte a tipurilor de consum de energie electrică din aceste sisteme.

În ceea ce privește consumatorii din aceste tipuri de sisteme au fost identificați principalii doi: *modulul de comunicație* și *procesorul*. Astfel, au fost evidențiate două tipuri de puteri, specifice acestor consumatori: așa numita *putere de comunicație* respectiv *putere computațională* [18]. Pornind de la această clasificare s-au dezvoltat două mari direcții de reducere a consumului de energie electrică pentru sistemele în cauză: o direcție ce vizează reducerea energiei electrice prin reducerea puterii computaționale și o direcție ce vizează reducerea aceluiași consum, dar prin reducerea puterii de comunicație.

Prima direcție, cea care vizează reducerea puterii computaționale, a dus la dezvoltarea mecanismelor de reducere a consumului la nivel local, de procesor sau cel mult la nivel de placă. Din aceste mecanisme cea mai mare pondere o au cele bazate pe reducerea consumului prin folosirea unor tehnici de planificare a taskurilor care să țină cont de cerințele și capabilitățile energetice ale procesorului/ plăcii (așa numitele tehnici de planificare *"power-aware"* în limba engleză).

Cea de-a doua direcție, cea care vizează reducerea puterii de comunicație, a dus la dezvoltarea mecanismelor de reducere a consumului la nivel de sistem de dispozitive, în special la nivel de protocol de comunicație.

3.2 Puterea de comunicație

Pe partea de comunicații, reducerea consumului de energie a fost abordată la nivel de protocol. În [19] este prezentată o soluție pentru prelungirea duratei de funcționare a rețelei bazată pe clustere (HARP - Hierarchical Adaptive and Reliable Routing Protocol = "protocol de rutare ierarhic adaptiv și sigur"). Soluția reduce consumul de energie la nivel de comunicație, oferind în același timp și un mecanism de toleranță la defecte la nivel de legătură de comunicație, cu toate acestea nu este prezentată încărcarea suplimentară datorată acestui algoritm și algoritmul nu are pretenția de a oferi suport pentru aplicații timp-real.

În [20] este prezentată o abordare similară, ca suport pentru o aplicație de monitorizare a poluării bazată pe EDETA (Energy-efficient aDaptive hiErarchical and robusT Architecture = "*arhitectura eficientă din punct de vedere energetic, adaptivă, ierarhică și robustă*"). De data aceasta arhitectura de rutare este bazată atât pe clustere cât și pe arbori dinamici. Aceeași arhitectură apare în [21], în acest caz oferind suport pentru un sistem de detecție a incendiului. Chiar dacă etapele protocolului sunt delimitate temporal, nu există o restricție de timp la nivel de recepție a mesajului, astfel nu putem vorbi despre comunicare în timp real în sensul strict al cuvântului, nici despre aplicații critice.

Astfel, deși o abordare a problemei la nivel de protocol pare a fi cea mai potrivită abordare, pentru sisteme timp-real critice nu avem o soluție viabilă. Aceasta se datorează, probabil, în mare măsură problemelor de predictibilitate existente în comunicațiile timp-real fără fir [22].

3.3 Puterea de computație și energia consumată

Pe partea de computație, au fost identificate principalele tipuri de putere consumate de un procesor și anume: *puterea dinamică* și cea *statică*. În circuitele CMOS predominantă este puterea dinamică, ce apare în timpul comutării circuitului, necesară încărcării capacităților de intrare a porților comandate și a altor capacități parazite de la ieșirea circuitului integrat. Relația matematică ce modelează această putere este [23]:

$$P_D = C_P \cdot V_{dd}^2 \cdot f_c \tag{3-1}$$

unde C_P reprezintă capacitatea parazită echivalentă de la ieșirea circuitului integrat, V_{dd} – tensiunea de alimentare și respectiv f_c – frecvența de comutare.

Dacă în loc să ne referim la frecvența de comutare, ne referim direct la frecvența procesorului, ecuația de mai sus devine [24, 25]:

$$P_D = C_p \cdot N_c \cdot V_{dd}^2 \cdot f \tag{3-2}$$

unde N_c reprezintă numărul de comutații per ciclu de *clock*, iar f frecvența procesorului.

În literatura de specialitate mai întâlnim și notația C_{ef} pentru produsul dintre C_p și N_c, provenind de la denumirea în limba engleză: *effective switched capacitance*, forma cea mai întâlnită a relației ce definește puterea dinamică fiind [26-30]:

$$P_D = C_{ef} \cdot V_{dd}^2 \cdot f \tag{3-3}$$

După cum se poate vedea din ecuația de mai sus, prin reducerea valorii tensiunii de alimentare obținem o reducere pătratică a puterii dinamice. Acest aspect a devenit de interes o dată cu dezvoltarea de procesoare cu capacitatea de reglare în mod dinamic a tensiunii de alimentare, capacitate denumită în limba engleză DVS (*Dynamic Voltage Scaling*). O reducere mai puțin impresionantă a puterii se obține doar prin reducerea frecvenței de lucru a nucleului procesorului. Această metodă, denumită și DFS (*Dynamic Frequency Scaling*), este realizabilă într-o serie de procesoare relativ noi ce oferă capacitatea de a ajusta în mod dinamic (în timpul rulării), frecvența de lucru a nucleului. Exemple de astfel de procesoare, care fie acceptă ajustarea frecvenței, fie a tensiunii de alimentare, fie ambele, sunt [31-34]: AMD ELAN SC400, StrongARM1100, Hitachi SuperSH, Intel XScale, IBM PowerPC 405LP, IntelPXA 250, ADSP-BF537.

Prima metodă de reducere a puterii implică în mod automat și pe cea de a doua, datorită faptului că reducerea tensiunii de alimentare duce la reducerea frecvenței maxime de lucru a procesorului.

Timpul de răspuns al circuitului este dat de relația [24]:

$$\delta = \frac{C_p V_{dd}}{K(V_{dd} - V_T)^{\alpha}} \tag{3-4}$$

unde C_p reprezintă capacitatea parazită echivalentă de la ieșirea circuitului integrat, V_{dd} – tensiunea de alimentare, K- o constantă ce depinde de procesul de producție și de mărimea circuitului, V_T – tensiunea de prag și α – coeficientul vitezei de saturație și ia valori între 1 și 2.

Frecvența de lucru a procesorului este direct proporțională cu inversul timpului de răspuns al circuitului. De aici rezultă dependența dintre valoarea frecvenței maxime și valoarea tensiunii de alimentare.

Puterea statică, deși mai mică decât puterea dinamică, cunoaște o creștere continuă de la o generație de circuite la alta, datorită faptului că dimensiunile circuitelor sunt tot mai mici [27]. La tehnologiile mai mici de 65 nm, puterea statică nu mai poate fi neglijată, fiind o parte semnificativă a puterii totale [35]. Acest tip de putere poate fi aproximat cu relația [27, 35]:

$$P_s \cong I_{rez} \cdot V_{dd} + \left|V_{BS}\right| \cdot I_j \tag{3-5}$$

unde I_{rez} reprezintă curentul rezidual al tranzistorului CMOS blocat, V_{DD} – tensiunea de alimentare, V_{BS} – tensiunea bază-sursă, numita și *body biasing*, I_j – curentul rezidual pe joncțiune.

V_{BS} este o tensiune aplicată între sursa sau drena tranzistorului și substrat și are rolul de a schimba valoare tensiunii de prag V_{TH} [36], având ca efect direct modificarea curentului rezidual.

Curentul rezidual este direct proporțional cu:

$$K_3 \cdot e^{K_4 \cdot V_{dd}} \cdot e^{K_5 \cdot V_{BS}} \tag{3-6}$$

unde $K_{3\text{-}5}$ reprezintă constante dependente de tehnologia circuitului.

Energia consumată de un dispozitiv într-un interval de timp (t_1,t_2) este:

$$E(t_1,t_2)=\int_{t_1}^{t_2} P_i(f(t))dt \tag{3-7}$$

unde P_i este funcția de putere, iar f este frecvența de lucru a procesorului.

Dacă funcția P_i este continuă și constantă pe intervalul (t_1,t_2) atunci, conform teoremei de medie, ecuația de mai sus devine:

$$E(t_1,t_2)=P_i(f)(t_2-t_1) \tag{3-8}$$

Din cele prezentate mai sus, vedem efectele și limitele reducerii consumului de putere: cel mai important compromis făcându-se între consum și viteza de procesare. Un consum mic, implică o viteză de procesare mică. În sistemele de timp real, viteza de procesare trebuie tratată cu mare atenție, datorită constrângerilor de timp, această problemă delicată deschizând o întreagă direcție de cercetare.

3.4 Metode de reducere a puterii de computație la nivel de procesor

Metodele de reducere a puterii de computație, pot fi împărțite după componenta de putere pe care o vizează: în metode de reducere *a puterii dinamice, a puterii statice* sau *hibride.*

După nivelul la care se face această reduce ele pot fi clasificate în: metode *la nivel de componente*, metode *la nivel de placă* sau *la nivel de rețea*.

3.4.1 Metode de reducere a puterii dinamice

În lucrarea [29] este prezentată o metodă clasică de reducere a consumului de putere dinamică (Dynamic Voltage and Frequency Scaling). După cum a fost menționat în secțiunea precedentă, această metodă pornește de la formula (3-5) a puterii dinamice și folosindu-se de capacitatea unor procesoare de a-și modifica frecvența (și eventual și tensiunea de alimentare), încearcă să reducă consumul datorat acestei puteri. Metoda este valabilă pentru o variație convexă a puterii în funcție de frecvență.

Metoda poate fi descrisă în felul următor: pentru a reduce puterea dinamică se reduce frecvența de lucru și tensiunea de alimentare dacă este posibil, ceea ce duce în final tot la reducerea frecvenței de lucru a procesorului. Se pune problema cu cât se poate reduce frecvența pentru fiecare task, astfel încât nici o limită de timp (*deadline*) să nu fie depășită. Modelul de taskuri considerat este cel al taskurilor periodice cu limita de timp egală cu perioada.

O soluție, prezentată în articolul citat anterior, este împrumutată din acea metodă de planificare a taskurilor pentru sisteme de timp real, numită planificare bazată pe beneficiu *(reward based scheduling)*. În această metodă de planificare se consideră pentru fiecare task o fracțiune de execuție obligatorie și una opțională. Partea obligatorie trebuie executată în întregimea ei fără a depăși limita de timp specifică ei, partea opțională are atașată o funcție beneficiu și poate fi executată parțial, total sau deloc, în funcție de beneficiul total pe care îl aduce aceasta pentru întreaga aplicație.

Se caută să se execute din partea opțională a taskurilor atât cât este necesar pentru ca beneficiul total să fie maxim.

Pornind de la ecuația (3-8) prin metoda de eficientizare a consumului, menționată, se dorește o minimizare a ecuației pe un anumit interval fără ca să se depășească constrângerile de timp. Se consideră că toate instanțele unui anumit task vor rula la aceeași frecvență. Se va aborda problema minimizării pentru fiecare task, pe intervalul de timp determinat de rularea unei instanțe a acelui task. Astfel, o minimizare globală a energiei, este suma minimizării consumului pentru fiecare task în parte (taskurile fiind independente). Problema se reduce la a minimiza suma de mai jos [29]:

$$\sum_{i=1}^{n} \frac{C_i}{f_i} \cdot P_i(f_i) \tag{3-9}$$

unde C_i reprezintă timpul necesar computației taskului *i* la frecvența maximă, iar f_i – frecvența la care rulează taskul *i,* aceasta este limitată la intervalul (f_{min},f_{max}).

C_i / f_i reprezintă de fapt timpul procesor alocat pentru taskul *i,* pe care îl vom nota cu X_i.

Dacă facem substituția de variabilă în ecuaţia anterioară, aceasta devine:

$$\sum_{i=1}^{n} X_i \cdot P_i(\frac{C_i}{X_i}) \tag{3-10}$$

unde X_i este mărginit de valorile C_i / f_{max} și C_i / f_{min}.

Această problemă este asemănătoare din punct de vedere matematic cu cea din planificarea bazată pe beneficiu:

$$\sum_{i=1}^{n} B_i(t_i) \tag{3-11}$$

unde B_i este funcția de beneficiu și t_i este mărginit de valorile *0* și o_{maxim} (timpul maxim de computație al fracțiunii opționale).

Pentru a reduce consumul de energie electrică există mai multe abordări diferite. Metodele de reducere a consumului prin reducerea frecvenței de lucru a procesorului pot fi clasificate în felul următor [37]:

După momentul unde are loc variația frecvenței procesorului:

A. **Intra-task DVS** – frecvența este variată în interiorul unui task
   - **Metode bazate pe traseu (path based)** – în cadrul codului executat se alege un traseu de referință. Se ajustează frecvența procesorului în funcție de calea pe care o urmează execuția efectivă a task-ului (dacă este mai scurtă decât calea de referință se micșorează frecvența, iar dacă este semnificativ mai lungă se mărește).
   - **Metode stocastice** – se începe rularea cu o frecvență mică și se mărește frecvența după un timp în cazul în care task-ul nu s-a terminat.

B. **Inter-task DVS** – frecvența este variată între taskuri.

C. **Metode hibride** – o combinație între inter și intra-task DVS.

După metoda de determinare a timpului în care procesorul nu este ocupat (*slack time*):

A. **Metode statice** – determinarea se face înaintea lansării în execuție a aplicației *(offline)*

B. **Metode dinamice** – determinarea se face în timpul execuției aplicației *(online)*

C. **Metode agresive** – determinarea este bazată pe predicții

Metodele *intra-task*, au dezavantajul că necesită adesea inserții de marcaje în cod. Astfel de metode au fost prezentate în [38, 39]. O metodă stocastică este prezentată în [40].

În [29] este prezentată o *metodă statică* de determinare a frecvenței optime, folosită peste planificarea EDF.

"Dacă funcția de putere este identică pentru fiecare task, atunci frecvența optimă a procesorului, pentru a minimiza consumul total de energie, pentru un set de taskuri periodice critice independente, este constantă și egală cu:

$$f_{opt} = \max(f_{\min}, U), unde U = \sum_{i=1}^{n} \frac{C_i}{T_i}" \quad (3\text{-}12)$$

O *metodă statică* asemănătoare, dar puțin mai complexă din punct de vedere computațional, folosită de data aceasta pentru planificarea RM, și în general pentru planificări bazate pe priorități alocate static, este prezentată în [41]. În continuare este descrisă această metodă:

- Se determină un set de puncte numite *de planificare (scheduling points)* pe baza relației de mai jos și se ordonează crescător:

$$S_i = \{k \cdot T_j \mid j = 1,2,..i; k = 1,..., \left\lfloor \frac{T_i}{T_j} \right\rfloor\} \quad (3\text{-}13)$$

unde, T_i reprezintă perioada taskului *i*.

- Se determină factorii de scalare ai frecvenței:

$$n_{i,j} = \frac{\sum_{k=1}^{i} C_k \cdot \left\lceil \frac{S_{i,j}}{T_k} \right\rceil}{S_{i,j}} \quad (3\text{-}14)$$

unde, $S_{i,j}$ reprezintă al *j*-lea element din lista de puncte de planificare.

Factorul de scalare pentru taskul *i* este:

$$n_i = \min_j n_{i,j} \quad (3\text{-}15)$$

Factorul de scalare global pentru tot setul de taskuri este:

$$\eta_i = \min_j \eta_{i,j} \quad (3\text{-}16)$$

Astfel frecvența optimă este:

$$f_{opt} = \eta \cdot f_{\max} \quad (3\text{-}17)$$

Autorii articolului [42] propun un algoritm optim pentru alocarea statică a tensiunii de alimentare V_{dd}, când acesta ia valori discrete. În 1995 [43], fusese deja propus un algoritm optim, dar valabil doar pentru o funcție continuă a valorilor de alimentare, iar în 1998 [44] s-a demonstrat un algoritm optim pentru valori discrete ale V_{dd}, valabil doar pentru un singur task. În [26] a fost prezentat un algoritm de alocare statică a

frecvențelor optime pentru sisteme în care funcția de putere nu este aceeași pentru fiecare task.

Aceste metode statice au avantajul că determină o frecventă optimă, ceea ce reduce numărul de comutații intre diferite niveluri de tensiune și diferite valori ale frecvenței de lucru a procesorului, implicit reducând și surplusul de energie/timp datorat comutațiilor.

Dezavantajul acestor metode statice este că nu se folosesc decât de timpul cât procesorul este liber datorită faptului că nu există nici o solicitare, din partea niciunui task, nu se folosesc și de faptul că un task ar putea să-și termine execuția mai repede decât este preconizat și nu în ultimul rând aceste metode nu sunt disponibile decât pentru un set foarte restrâns de algoritmi de planificare. Mai mult, aceste metode iau în considerare doar utilizarea procesorului, fără a ține cont de momentele de start.

Metodele dinamice se folosesc în principal de faptul că în sistemele de timp real se lucrează cu parametrii pentru cel mai defavorabil caz. De cele mai multe ori aceste prezumții sunt pesimiste, și timpul de execuție efectiv al unui task este cu mult mai mic decât cel considerat pentru cel mai defavorabil caz. Astfel, diferența de timp între cel mai defavorabil caz si cel efectiv, poate fi folosită pentru a reduce și mai mult puterea consumată. Astfel de algoritmi au fost prezentați în [30, 33, 39, 45-49].

Metodele dinamice, implică de obicei structuri fizice mai complicate (de tipul cozilor), care introduc și întârzieri suplimentare în procesare [45].

Mergând mai departe, există și metode, așa numitele metode *agresive*, care fac predicții asupra viitoarelor încărcări din sistem (a timpilor de execuție viitori). Astfel de metode au fost prezentate în [24, 30, 38, 45, 46, 50, 51]. În [24] și [38] timpul de execuție efectiv este estimat printr-o analiză statistică în urma unor măsurători succesive. În [50] este prezentată o funcție euristică pentru determinarea timpului în care procesorul este liber.

O altă metodă de a determina încărcarea procesor în mod dinamic este cea bazată feedback. Se ajustează frecvența/voltajul, în funcție de răspunsul primit de la procesor în ceea ce privește încărcarea procesor pentru o perioadă anterioară. Prin extrapolare se presupune că și perioada imediat următoare va avea o încărcare procesor asemănătoare. Un astfel de algoritm a fost prezentat în [46], ca fiind implementat pe placa IBM PowerPC 405LP peste un sistem de operare MontaVista Embedded Linux. Problema cu acest sistem ar fi timpul mare de stabilizare a frecvenței procesor (până la 200ms), neadecvat aplicațiilor critice. Un alt algoritm, care vizează de data aceasta task-urile lejere, este prezentat în [52].

Aceste metode sunt problematice, după cum menționează însuși autorii [45], deoarece există riscul depășirii timpului de execuție pentru cel mai defavorabil caz.

După ce *slack time*-ul este determinat, printr-o metodă sau alta, acesta trebuie alocat unuia sau mai multor task-uri care își vor extinde timpul de execuție prin reducerea frecvenței de lucru a procesorului. Există mai multe abordări în funcție de metoda de alocare a acestui *slack time:* metode *Greedy* (lacome), care fie alocă acest interval de timp primului task disponibil, fie îl alocă, după o metrică stabilită, acelui task care ar presupune o reducere maximă de energie la momentul respectiv [53] sau metode care distribuie uniform acest interval de timp tuturor celorlalte task-uri [47] sau unui subset de task-uri [48].

În literatura de specialitate există și modele mai realiste de consum de putere la nivel de procesor, care iau în considerare surplusul de energie datorat tranzițiilor de la un nivel de tensiune la altul, și de asemenea rezoluția cu care se poate incrementa/decrementa tensiunea de alimentare. Astfel de modele au fost studiate în [32, 35, 54-56]. Experimental s-a arătat că în unele cazuri variația de tensiune mărește consumul de energie în loc să-l reducă [54]. În [55] au fost prezentați patru algoritmi

bazați pe tehnica de planificare DM, diferiți în funcție de surplusul de energie și timp datorat comutațiilor:

- *Sys-Clock* – un algoritm care determină o singură frecvență pentru toate task-urile și este recomandat atunci când surplusul de energie/ timp este prea mare pentru a face comutații dese între frecvențele de lucru ale procesorului. A fost implementat pe o platforma cu procesor Compaq iPAQ H3700. Această platformă a dovedit dezavantajul unei întârzieri relativ mare (20ms) datorate resincronizării procesorului cu SDRAM-ul ori de câte ori se schimbă frecvența procesorului.
- *PM-Clock* – un algoritm în care se stabilește pentru fiecare task frecvența lui de lucru și este recomandat atunci când întârzierile datorate comutării frecvenței sunt mici.
- *Opt-Clock* – un algoritm foarte complex din punct de vedere computațional, care determină în mod nelinear, frecvențele optime pentru fiecare task.
- *DPM-Clock* – un algoritm potrivit pentru cazurile când execuția efectivă este mult mai mică decât cel mai defavorabil caz.

Din articolele prezentate mai sus doar [46, 54-56] au prezentat implementări reale pe sisteme fizice, restul rezumându-se la simulări sau la modele pur teoretice. Dintre implementările fizice [54] a luat în considerare întârzierile datorate comutării frecvenței și tensiunii de alimentare, dar nu a luat în considerare surplusul de energie datorat acestor comutații, iar procesoarele folosite (AMD Athlon, Intel XScale si Transmeta Crusoe) sunt mai degrabă procesoare de uz general cu o putere de calcul mare, și o predictibilitate nu chiar atât de bună ca a celor dedicate sistemelor de timp real. În [55], se prezintă implementări pe Compaq iPAQ H3700, tot un procesor de uz general, al cărui întârziere datorată stabilizării frecvenței este de 20ms, o valoare prea mare pentru aplicații critice. Aceeași problemă se pune și pentru implementarea prezentată în [46], de data aceasta pe IBM PowerPC 405LP, cu o întârziere de până la 200ms. În [56] este prezentată o metodă implementată pe o platformă cu Intel StrongARM 1100, dar care vizează sisteme de operare de uz general. Tot pe sisteme de uz general s-a pus și problema sincronizării task-urilor cu acces la resurse comune [57].

În concluzie, tehnica de variație dinamică a frecvenței procesorului și/sau tensiunii de alimentare, a fost prezentată până acum mai mult teoretic, iar în ceea ce privește sistemele de timp real și aplicațiile critice, aceasta a fost abordată insuficient, neexistând o soluție completă.

3.4.2 Metode de reducere a puterii totale

Metodele prezentate în paragraful anterior vizau doar puterea dinamică. În ultimul deceniu și metodele de reducere a puterii statice s-au bucurat de o oarecare atenție. A fost studiată o metodă considerată complementară metodei DVS și numită ABB (*Adaptive Body Biasing*) [18, 27, 35, 36, 45, 58]. Metoda se bazează pe reducerea curentului rezidual prin schimbarea valorii tensiunii V_{BB} (v. (3-6)). Până la urmă compromisul se va face tot între consumul de energie și viteza de procesare. Un curent rezidual mic va implica și un răspuns lent al porților. În general valorile acestei tensiuni sunt fixe, și se stabilesc în faza de design a componentei hardware. Pot fi mai multe valori prestabilite pentru diferite moduri de funcționare ale dispozitivului, cum este și cazul procesorului Intel XScale [36]. Tot în această lucrare s-a propus o abordare dinamică, în sensul în care cu ajutorul unor dispozitive hardware suplimentare să se facă o schimbare dinamică a valorii V_{BB}. Metoda prezintă însă câteva dezavantaje: pe lângă faptul că necesită componente fizice suplimentare, valorile tensiunii V_{BB} ce pot fi

alese, nu pot fi stabilite prin calcul, ci doar în urma unor măsurători, după cum afirmă autorii.

Pe lângă influența tensiunii V_{BB} asupra curentului rezidual, în [59] a fost luată în considerare și influența temperaturii.

O abordare ușor diferită o constituie cea bazată pe modelul de descărcare al bateriilor. Se arată că durata de viața a bateriei nu depinde doar de valoarea energiei consumate, ci mai ales de profilul de consum și de profilul de descărcare al bateriei [31, 60]. Aceste articole însă sunt orientate pe telefoane mobile și pe laptop-uri/ PDA-uri și nu vizează aplicații critice. În [18] a fost prezentat un model de arhitectură de nod de senzori și un algoritm de reducere a energiei, care a luat în considerare și modelul descărcării bateriei din [60], dar acest model nu a fost însă implementat fizic, ci doar simulat.

O altă direcție de reducere a consumului energiei electrice la nivel de nucleu de procesor a fost propusă prin reducerea codului ce va fi executat [61] sau prin folosirea unor instrucțiuni echivalente, dar cu consum mai redus (capabilitate existentă pe unele procesoare – ex. arhitectura ARM prin instrucțiunile THUMB [62]). În [63] este prezentat chiar un algoritm genetic folosit pentru a determina o soluție de mix între rutine implementate prin instrucțiunile ARM clasice sau THUMB, care să reducă cât mai mult energia consumată.

O metodă clasică de reducere a puterii, care implică pe lângă reducerea puterii dinamice și reducerea celei statice, este *managementul dinamic al puterii* DPM (*Dynamic Power Management*). Aceasta vizează dispozitivele care au predefinite mai multe stări de funcționare cu consumuri de energie diferite (*activ, oprit, în așteptare* etc.).

Astfel de algoritmi au fost studiați în [58, 64-68]. În principal acești algoritmi presupun mărirea intervalului cât timp procesorul/dispozitivul nu este solicitat, pentru a obține un interval destul de mare, ce va fi folosit pentru a trece acel dispozitiv într-o stare inactivă [67]. În [58] a fost prezentată o metodă bazată pe DPM pentru diferite dispozitive, nu doar procesor. În [68] se ia în considerare și influența temperaturii asupra consumului de energie.

Dezavantajul acestor algoritmi este că timpul de comutație dintr-o stare în alta presupune un surplus relativ mare de energie și timp, de aceea aceste tranziții nu pot fi efectuate des, și nici nu se pretează pentru evenimente critice cu constrângeri de timp foarte stricte. De exemplu, nu se pot folosi (sau trebuie folosite cu mare atenție) pentru evenimente ce necesită un răspuns imediat ca declanșarea unui airbag, controlul unei centrale nucleare etc. O soluție mai bună, atât din punct de vedere al reducerii consumului de energie, cât și al predictibilității o constituie combinațiile între cele două metode (DVS și DPM). Astfel de soluții au fost prezentate în [64, 65]. Concluziile din [65] arată că DVS luat separat obține o reducere de energie mai bună față de DPM singur. În acest articol este propusă o soluție hibrid dinamică care face uz de învățarea automată, interesantă ca idee, dar nepotrivită pentru sisteme critice.

3.5 Metode de reducere a puterii de computație la nivel de placă

După cum s-a observat din paragrafele anterioare cele mai multe metode de reducere a energiei au fost dezvoltate la nivel de componente. Dintre aceste componente cel

care se bucură de departe de cea mai mare atenție este procesorul, totuși pe lângă el există și alte dispozitive care au fost luate în atenție.

Metodele principale de reducere a consumului de energie electrică rămân DVS și DPM, pe baza acestora dezvoltându-se însă și metode hibride, care fie le cuprind pe ambele, fie cuprind o serie de dispozitive (nu doar procesorul) pentru care se aplică algoritmii vizați.

În ceea ce privește DVS, aceasta a fost cu preponderență folosită în dezvoltarea mecanismelor de reducere a consumului energetic care vizează exclusiv procesorul. Există, însă, și excepții la faptul că acești algoritmi bazați pe DVS vizează doar procesorul. Un astfel de exemplu îl reprezintă o metodă bazată pe DVS agresiv (bazat pe predicții), care pe lângă procesor ia în considerare și consumul altor componente (în special memoria). Această metodă a fost implementată fizic pe Intel PXA255, un procesor puternic, destinat mai mult pentru PDA-uri și a fost prezentată în [69]. Metodele bazate pe predicție nu pot fi folosite în sisteme de timp real critice, cu toate acestea, modelul de consum prezentat în acest caz este unul mai realist decât cele prezentate până acum. Acest model ia în considerare faptul că timpul de execuție al unui task poate avea o parte dependentă de frecvența de lucru a procesorului și o parte independentă de aceasta (datorată de exemplu de timpii de așteptare după memorie sau alte dispozitive). Astfel, la rândul ei energia totală are o componentă dependentă de frecvența de lucru și o parte independentă de aceasta.

Pe de altă parte s-a încercat o modelare asemănătoare cu cea a procesorului pentru DVS, pentru modulul de comunicație, doar că pentru acest modul s-a variat fie modulația semnalului, fie rata de transmisie ca un parametru echivalent pentru frecvența de lucru a procesorului. Astfel de algoritmi sunt prezentați în [53, 70].

DPM, pe de altă parte, a avut mai mult succes în dezvoltarea de algoritmi de reducere a energiei electrice pentru alte componente decât procesorul, datorită faptului că mult mai multe dispozitive au implementate și definite diferite tipuri de regimuri de funcționare cu consum energetic diferit, așa cum necesită DPM; față de capabilități de adaptare dinamică a tensiunii de alimentare și/sau a frecvenței de lucru, așa cum necesită DVS.

În [58] a fost prezentată o metodă bazată pe DPM pentru dispozitive ce au trei stări de funcționare (*active, standby* și *sleep*). Soluția propusă presupune o parte fizică, reprezentată de un controler prevăzut cu un buffer, care controlează deservirea evenimentelor de către dispozitiv și o parte software, reprezentată de algoritmul de deservire a evenimentelor. Ideea centrală este ca evenimentele să fie deservite cât mai târziu posibil, pentru a mări timpul cât dispozitivul se află într-o stare inactivă (care implicit consumă mai puțin decât una activă). Sistemul este dezvoltat mai pe larg în [71], unde se propune și un model matematic pentru descrierea taskurilor și a surplusului de energie/timp datorat tranzițiilor dintr-o stare în alta: de data aceasta modelul nu mai este la nivel de task, ci la nivel de dispozitiv. Pentru fiecare dispozitiv sunt definite trei regimuri sau stări de funcționare, fiecare stare fiind caracterizată de puterea P. O tranziție dintr-o stare în alta necesită un cost de timp t_{sw} și un cost energetic E_{sw}.

Problema s-a pus și la nivel de dispozitive de intrare-ieșire. În [72] a fost prezentat un algoritm care identifică intervalele de timp când un astfel de dispozitiv este liber, încearcă să mărească sau chiar să creeze astfel de intervale pentru a reduce energia consumată prin trecerea dispozitivului într-o stare inactivă. Aici modelul taskului se extinde prin adăugarea ca parametru al unui task, pe lângă parametrii temporali (perioadă, timp de execuție, etc.), a unui vector ce conține dispozitivele folosite de acel task. În mod similar cu algoritmul precedent stările și tranzițiile sunt caracterizate de aceeași parametri (putere, costuri de timp și energie pentru tranziții). Algoritmi

asemănători sunt şi cei prezentaţi în [72-76] ce pot fi folosiţi atât pentru procesor cât şi pentru dispozitive de intrare/ieşire, diferenţele între aceştia fiind la nivel de politică de planificare (*preemptivă* sau non-preemptivă, vizează sau nu utilizarea de resurse *non-preemptive*, permite sau nu rearanjarea taskurilor, etc.) şi la nivel de număr de stări/regimuri de funcţionare ce sunt cuprinse în model.

S-a încercat de asemenea o unificare a celor două metode, prin dezvoltarea unor mecanisme de reducere a energiei electrice, hibride, bazate atât pe DVS cât şi pe DPM.

Un astfel de mecanism reprezentat de un algoritm hibrid şi de modelul său aferent a fost prezentat în [66]. În acest caz este prezentat un sistem compus dintr-un procesor şi *k* dispozitive (de gen periferice sau memorii). Pentru procesor sunt considerate mai multe stări active (date de frecvenţe de lucru şi tensiuni de alimentare diferite, plus o stare inactivă), iar pentru dispozitive sunt considerate stări predefinite (activă*/idle* şi *sleep*). Procesorul este tratat în mod diferit faţă de celelalte dispozitive, DVS apelându-se doar pentru procesor. Celorlalte dispozitive li se aplică un algoritm de DPM care ţine cont atât de costul de energie şi timp pentru tranziţia dintr-o stare în alta cât şi de dependenţele de precedenţă dintre taskuri. Modelul taskurilor este cel al unui set de taskuri periodice, în care există constrângeri de timp doar la nivel de set, nu şi la nivel de task.

Tot un algoritm hibrid (DVS+DPM) este prezentat şi în [77]. Acest algoritm este împărţit în două componente, una statică (*offline*) folosită pentru a calcula o frecvenţă de lucru minimă rentabilă pentru procesor şi pentru a determina anumite intervale de inactivitate pentru procesor şi dispozitive; şi una dinamică (*online*), care se foloseşte de variaţiile timpilor de execuţie efectivi faţă de cei teoretici (luaţi pentru cazul cel mai defavorabil). Acest algoritm se bazează pe încercarea de mărire a intervalului de inactivitate pentru dispozitive, definind un nou concept numit DFR (*Device Forbidden Regions*), ce presupune alegerea în avans a unor intervale de timp în care un anumit dispozitiv este inactiv.

Un model mai general, care poate fi aplicat pentru diferite dispozitive, şi care ia în considerare variaţia puterii în funcţie de frecvenţa de lucru a fost prezentat în [78]. Acest model exprimă faptul că puterea unui dispozitiv este o sumă de componente ce variază în funcţie de frecvenţa de lucru, normalizată (raportată la frecvenţa nominală), astfel:

- O componentă direct proporţională cu cubul frecvenţei de lucru, dată de acele dispozitive care îşi pot varia atât frecvenţa de lucru cât şi tensiunea de alimentare. Se consideră că ambele vor varia cu acelaşi factor, ceea ce este adevărat pentru o serie de procesoare particulare, dar nu poate fi generalizat la toate dispozitivele cuprinse în sistemele încorporate.
- O componentă direct proporţională cu frecvenţa de lucru, dată de acele dispozitive care îşi pot varia frecvenţa de lucru doar, fără a varia şi tensiunea de alimentare.
- O componentă constantă din punct de vedere al frecvenţei de lucru, în care se poate include şi puterea statică.

Astfel puterea unui dispozitiv oarecare devine o funcţie polinomială de ordinul trei:

$$P = a_3 \cdot s^3 + a_1 \cdot s + a_0 \qquad (3\text{-}18)$$

unde P reprezintă puterea, s reprezintă frecvenţa normalizată ($f/f_{nominal}$).

La acest model a fost adăugat un factor proporţional cu pătratul frecvenţei de lucru normalizate, datorat consumului convertoarelor de tensiune [79].

$$P = a_3 \cdot s^3 + a_2 \cdot s^2 + a_0 \qquad (3\text{-}19)$$

Acest model a fost propus atât pentru tehnici bazate pe DVS cât și DPM, în sensul în care stările inactive predefinite folosite în DPM pot fi modelate ca având o putere constantă față de frecvența de procesare. Cu toate acestea modelul nu oferă suport pentru precizarea costului de timp și energie datorat tranzițiilor din DPM.

În concluzie, deși au fost dezvoltate modele de consum la nivel de dispozitive, și algoritmi pentru reducerea acestui consum, în prezent nu există o abordare unitară la nivel de placă, suficient de complexă să poată fi aplicată pentru orice nod de rețea de senzori sau dispozitiv încorporat.

3.6 Metode existente de măsurare și estimare a parametrilor energetici

O etapă principală, în procesul de reducere a consumului de energie electrică, îl presupune măsurarea consumului efectiv și estimarea acestuia după diferite modele validate prin măsurători.

În [80] a fost efectuat un studiu asupra sistemelor ce au capabilitate de DVS. S-a încercat o clasificare a acestor sisteme în funcție dacă suportă sau nu execuție de task-uri în timpul tranziției de la un nivel de tensiune la altul, dacă schimbarea se poate face între orice valori ale frecvenței de lucru a procesorului sau există doar o listă de valori între care se poate alege. În funcție de aceste aspecte s-au definit patru modele (sisteme optimiste - acceptă execuție de taskuri în timpul tranziției de la o tensiune la alta; pesimiste – nu acceptă; ideale - tranziția se face instantaneu între orice frecvențe de lucru și multiple – unde tranziția se face instantaneu, dar este posibilă doar între anumite nivele de tensiune/ frecvențe).

În ceea ce privește definirea unui model de consum, există următoarele provocări [81]:

- Pentru a avea estimări cât mai precise, este nevoie de un model cât mai detaliat;
- Eventualele modificări ce survin în sistem, duc la modificarea modelului;
- Modelul trebuie validat prin compararea cu măsurătorile reale;
- Este foarte dificil sau chiar imposibil pentru un model să conțină toate detaliile de funcționare (condiții fizice de funcționare);
- Majoritatea producătorilor nu oferă toate detaliile dispozitivelor, din motive de competiție.

Autorii din [81] propun o metodă de măsurare a consumului la nivel de procesor, memorie, și dispozitive de intrare-ieșire prin folosirea unor surse de alimentare separate, dedicate celor trei niveluri și măsurarea curentului furnizat pentru fiecare. Pe baza acestor măsurători ei dezvoltă un model bazat pe stări. Din experimentele prezentate în acest articol se observă un consum aproximativ egal între memorie si procesor (~0.3 Wați). O metodă asemănătoare a fost prezentată în [82]. În acest caz nucleul procesorului este alimentat separat. Prin intermediul unui regulator de putere extern TPS62400 de la TexasInstruments nucleul primește diferite tensiuni de alimentare, iar consumul se măsoară cu ajutorul unui osciloscop cu sonde pentru curent și tensiune.

Un alt sistem, format de data aceasta din microcontrolerul Motorola HC908GP32, o memorie DRAM și un convertor analog-digital ADC0805, este prezentat în [83]. Din măsurătorile efectuate de autori se observă un consum mai puternic datorat memoriei

(aproximativ dublu decât cel datorat procesorului) și un consum relativ scăzut datorat convertorului.

Un alt sistem format din doua părți (una fixă și una reconfigurabilă), implementat cu FPGA și procesor cu capabilitate de DVS, a fost prezentat în [84]. Articolul este centrat pe prezentarea unui mecanism de reducere a energiei electrice prin efectuarea unor decizii, în mod dinamic, dacă un anumit task să ruleze pe procesor sau pe FPGA în funcție de consum/ timp de procesare.

Există și modele de consum pentru procesoare la nivel de instrucție. Un astfel de model este cel dezvoltat pentru arhitectura ARM7 în [85].

În [86], a fost propus un cadru pentru economisirea energiei în sisteme timp real. Acest cadru conține trei niveluri: un nivel fizic ce poate fi controlat din punct de vedere al puterii, un nivel software bazat pe DVS și unelte software de analiză a consumului. Abordarea a fost centrată pe software și s-au neglijat întârzierile datorate tranzițiilor, iar reperele folosite pentru măsurători nu au fost dedicate aplicațiilor timp real.

O altă problemă o constituie lipsa unor repere clare (*benchmark*) și a unor metrici pentru compararea eficienței din punct de vedere al consumului de energie electrică pentru dispozitivele dedicate sistemelor timp-real. În [87] au fost prezentate astfel de repere și metrici pentru procesoare și sisteme, dar nu pentru timp real sau încorporate. Este menționat doar un astfel de sistem de referință numit *EnergyBench*, dedicat procesoarelor pentru sisteme încorporate. Acesta clasifică procesoarele în funcție de raportul performanță / energie consumată. Cu toate acestea datele măsurate și publicate pe site-ul EEMBC nu sunt suficiente pentru o comparare realistă a procesoarelor de pe piață [88].

În ceea ce privește metricile clasice pentru sisteme timp-real putem menționa următoarele [9]: suma timpilor de finalizare a execuțiilor taskurilor, lungimea unui ciclu de planificare, timpul maxim de răspuns. Cu toate acestea nu există o metrică de referință cu care să fie comparate toate sistemele timp-real și mai mult, nu există o metrică universal acceptată, dezvoltată pentru aceste sisteme care să ia în considerare atât performanța cât și consumul de energie.

În concluzie, în prezent există prea puține și incomplete cadre de măsurare a performanțelor energetice la nivel de nod/dispozitiv. De asemenea în literatura de specialitate nu s-a întâlnit o descriere și o analiză a profilurilor de consum energetic în funcție de clasa de apartenență a acestor dispozitive inteligente așa cum se găsește de exemplu pentru aparatura de uz casnic în [89].

3.7 Concluzii

Din cele prezentate, vedem efectele și limitările reducerii consumului de putere: cel mai important compromis făcându-se între consum și viteza de procesare. Un consum mic, implică o viteză de procesare mică, iar în sistemele de timp real, viteza de procesare trebuie tratată cu mare atenție, din cauza constrângerilor de timp.

Pe de altă parte, deși au fost dezvoltate modele de consum la nivel de dispozitive și algoritmi pentru reducerea acestui consum, nu există modele și, implicit, nici algoritmi la nivel de placă, sau acestea sunt fie prea simple și, implicit, incomplete, fie modele particulare.

De asemenea în prezent există prea puține și incomplete cadre de măsurare a performanțelor energetice la nivel de nod/dispozitiv. În literatura de specialitate nu s-

a întâlnit o descriere și o analiză a profilurilor de consum energetic în funcție de clasa de apartenență a acestor dispozitive inteligente.

O analiză mai detaliată a metodelor prezentate în acest paragraf, din punct de vedere al gradului de aplicabilitate în sistemele de timp real critice, este prezentată în [90], de unde voi prelua un tabel cu rezultatele principale ale acestui studiu (Tabelul 1).

Tabelul 1. Comparații între principalele metode de planificare cu funcție de eficientizare a consumului electric

Metoda	**Implicații asupra descăr-cării bateriei**	**Implicații asupra mediului timp-real critic**	**Reduce-rea expecta-tivă a energiei**	**Validări**	**Avantaje**	**Dez-avantaje**
Intra-task DVFS	Consum de vârf	Risc mare de depășire *deadline*	Mai mare decât alte metode bazate pe DVFS	IntelXScale, SimDVS simulator	Detecția timpurie a timpului procesor nefolosit	Inserții de marcaje în cod, schimbări de frecvență numeroa-se
Inter-task DVFS	Consum de vârf	Dependente de algoritm, surplus (*overhead*) de timp	Mare, depen-dentă de algoritm	AMD Athlon, Intel XScale, Transmeta Crusoe, simulatoare	Dependente de algoritm	*Overhead* de timp
DVFS Static	Descărcare lentă	Predictibilă, potrivită	Mărginită de utilizarea procesor	Simulatoare	Schimbări de frecvență reduse, predictibilitat e	Disponibil ă doar pentru metode statice de planificare
DVFS Dinamic	Consum de vârf	Dependente de algoritm, surplus (*overhead*) de timp	Mare, depen-dentă de algoritm	AMD Athlon, Intel XScale, Transmeta Crusoe, simulatoare	Potrivite pentru orice algoritm de planificare	*Overhead* de timp mai mare decât la metodele statice
DVFS Agresiv	Consum de vârf	Fără garanții de timp, nepotrivită	Mult mai mare decât DVFS static	Simulatoare IBM PowerPC 405LP	Reduceri semnificative de energie	Nepotrivit e pentru sisteme critice
ABB	Reducerea curenților reziduali	Dependentă de algoritm	Mică	Simulatoare (e.g. 0.07 and 0.18 um)	Reducerea puterii statice	Hardware supli-mentar
DPM	Tipuri diferite de descărcare și tranziții	Surplus relativ mare de timp	Depen-dentă de algoritm și de hardware	Compaq iPAQ Intel XScale, StrongARM	Reducerea puterii totale	*Overhead* de timp în funcție de algoritm si de hardware

4 Sistem cadru pentru analiza, estimarea și eficientizarea consumului de energie la nivel de sistem încorporat-TEEARTS

Motto: *"Adu-ți aminte, ceea ce mentalizezi alimentezi cu energie. Și ceea ce alimentezi cu energie se va realiza".* ***Herbert Harris***

În capitolul de față este propus și prezentat un sistem cadru pentru analiza, estimarea și eficientizarea consumului de energie pentru o anumită aplicație ce rulează pe o platformă dată, numită și platforma sau sistemul *țintă* (în particular această platformă va fi un nod de rețea de senzori). Sistemul cadru propus se numește TEEARTS (*Time and Energy Efficiency Analysis for Real Time Systems*).

Dintr-o analiză a aspectelor teoretice prezentate în Capitolul 3 , putem identifica principalii factori de influență ai consumului de energie datorat execuției unei anumite aplicații pe o platformă dată: pe de o parte avem sistemul fizic, cu particularitățile sale energetice; pe de alta avem cerințele aplicației [90]. Atât sistemul fizic cât și particularitățile aplicației modelează consumul de energie prin câte un set de factori de influență, ce pot fi împărțiți în două mari categorii: factori energetici (cu o influență directă asupra consumului de energie) și factori temporali (cu o influență indirectă asupra consumului de energie). Fiecare set de factori de influență este determinat, specificat și devine un set de parametri măsurabili și controlabili, integrați într-un model unitar.

Pentru fiecare tip de parametru se va descrie, în cele ce urmează, cel puțin un exemplu de cadru de determinare și măsurare a acestuia. Acest cadru va cuprinde o descriere a uneltelor folosite, o metodologie de determinare a parametrilor și a valorilor acestora și cel puțin un exemplu de utilizare.

După ce parametrii au fost determinați, avem un model al consumului de energie al unei aplicații menite să ruleze pe o platformă țintă. Acest model va fi analizat cu ajutorul unui sistem de analiză și planificare a execuției, se vor face ajustări ale parametrilor modelului (acolo unde este posibil) și se va genera un model de execuție al aplicației pe platforma țintă astfel încât consumul energetic să fie minim sau cât mai apropiat de valoarea minimă posibilă pentru scenariul dat.

4.1 Principalele funcționalități urmărite

Sistemul cadru de analiză, estimare și eficientizare a consumului de energie la nivel de sistem timp-real încorporat sintetizează totalitatea obiectivelor cercetării de doctorat, astfel principalele funcționalității ale sistemului corespund cu aceste obiective:

1. *Conceperea și dezvoltarea unui model de consum de energie la nivel de sistem pentru o platformă țintă dată, ce face parte dintr-o serie de clase de dispozitive timp-real încorporate, definite în prealabil.*
2. *Măsurarea și determinarea parametrilor modelului consumului de energie de la punctul precedent, pentru o anumită aplicație dată.*

3. *Planificarea aplicației cu ajutorul unui mecanism de planificare, pe mai multe niveluri, pe platforma țintă, cu funcție de eficientizare a consumului de energie electrică.*
4. *Execuția aplicației pe sistemul țintă și realizarea unei analize a comportării aplicației timp-real direct pe sistem.*

4.2 Descrierea generală a sistemului

Structura de bază a sistemului propus este ilustrată în figura următoare, unde fiecare bloc reprezintă o componentă majoră a sistemului, pentru a cărei descriere este dedicat câte un subcapitol din cele ce urmează. Aceste componente pot fi împărțite în componente *hardware* (reprezentate prin dreptunghiuri cu colțuri ascuțite), componente *software* (reprezentate prin dreptunghiuri cu colțuri rotunjite) și modele (reprezentate prin elipse).

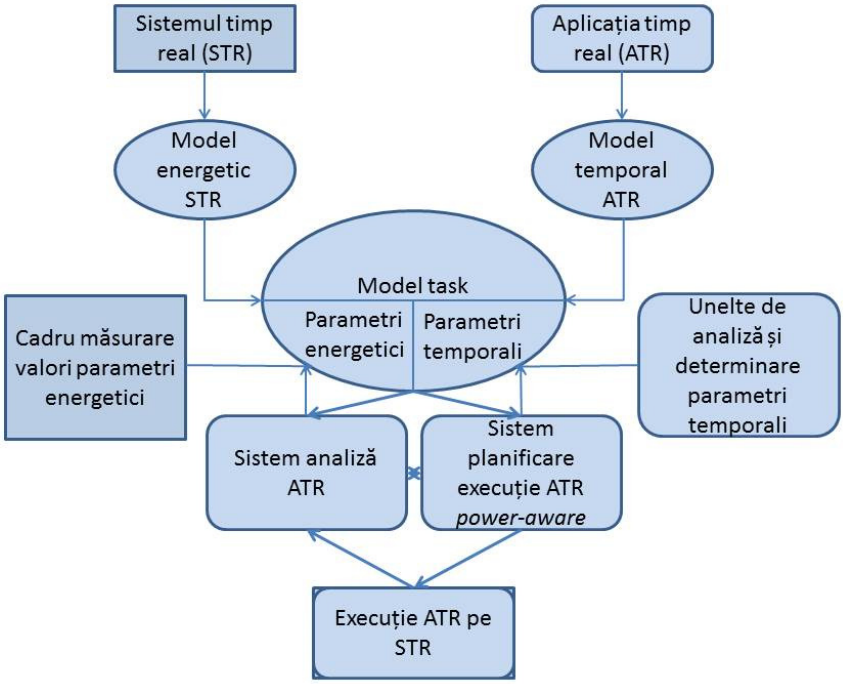


Figura 1. Sistemul cadru de analiză, estimare și eficientizare a consumului de energie

Sistemul timp-real (țintă) va fi specificat în subcapitolul 4.3 , unde pe baza specificațiilor este dezvoltat și prezentat un model de consum energetic, la nivel de componentă și la nivel de sistem. În subcapitolul 2.2 au fost modelate din punct de

vedere temporal aplicaţiile ce pot rula pe acest tip de sistem, iar în subcapitolul 4.4 este propus un model integrat din punct de vedere atât al comportării temporale cât şi a comportării energetice pentru aplicaţiile timp-real ce rulează pe platforma ţintă dată. Subcapitolul 4.5 prezintă o serie de unelte şi metodologii necesare determinării parametrilor energetici din modelul prezentat în subcapitolul 4.4 , iar subcapitolul 4.6 prezintă, în mod asemănător, o serie de unelte şi metodologii necesare determinării parametrilor de timp.

Sistemul de analiză al aplicaţiilor timp-real şi sistemul de planificare al acestora sunt prezentate în subcapitolele 4.7 şi respectiv 4.8 .

Validarea mecanismului de planificare prin execuţia aplicaţiei direct pe sistemul ţintă va fi prezentată într-un capitol ulterior.

4.3 Sistemul timp-real ţintă

4.3.1 Descrierea sistemului timp-real ţintă

După cum a fost deja menţionat, sistemul timp-real este acel sistem pentru care funcţionarea corectă nu implică doar ca rezultatele furnizate de sistem să fie corecte, ci implică şi respectarea constrângerilor de timp. Astfel, sistemele de timp real pot fi implementate doar pe acele platforme hardware care oferă suport fizic pentru îndeplinirea cerinţelor specifice de funcţionare a acestor sisteme dintre care enumerăm:

- *Predictibilitatea*: - Este definită ca fiind capacitatea de a demonstra că cerinţele de operare a sistemului sunt îndeplinite, atâta timp cât ipoteza de lucru e îndeplinită (ex. utilizarea procesor este sub o anumită valoare) [91].
- *Interacţiunea cu mediul*: - Capacitatea sistemului de a interacţiona cu mediul este o altă trăsătură fundamentală a sistemului de timp real.
- *Coordonata de timp*: - Analiza, planificarea şi execuţia aplicaţiei de timp real are la bază coordonata de timp.

Suportul pentru implementarea unui sistem predictibil este dat de o arhitectură hardware predictibilă (ex. o arhitectură ce conferă o operare predictibilă la nivel de instrucţiune). Aceasta înseamnă şi că mecanismele speculative de tip predicţia salturilor ş.a. şi a mecanismelor de accelerare a execuţiei ce au un caracter imprevizibil din punct de vedere a comportamentului temporal să fie evitate; iar benzile de asamblare (*pipeline*s în limba engleză) şi alte mecanisme de îmbunătăţire a performanţei să fie folosite doar în cazul în care acestea pot oferi o limită superioară de timp garantată (i.e. comportarea lor temporală este mărginită superior de o funcţie dată). Astfel mulţimea platformelor posibile este restrânsă. Exemplificarea sistemului nostru se va face pentru diferite tipuri de noduri de senzori de reţea, arătând totodată că sistemul poate fi folosit şi pentru alte tipuri de platforme fizice.

Datorită necesităţii sistemului de a interacţiona cu mediul, platforma fizică trebuie să ofere un suport pentru întreruperi.

Suportul pentru implementarea coordonatei de timp este dat în primul rând de temporizatoare (*timers*), de diferite surse de tact (ce pot fi sincronizate) şi alte mecanisme de contorizare si control a timpului sistemului. Într-un sistem de timp real strict, coordonatele de timp şi întreruperile sunt strâns legate.

Pe de altă parte, sistemul vizat trebuie să aibă capabilităţi de reducere a consumului de energie electrică prin control software. Dintre mecanismele de reducere

a consumului de energie electrică descrise în primul capitol, vizăm cele mai semnificative două tipuri: cele bazate pe DVS si cele bazate pe DPM. Din cele menționate în capitolul respectiv, pentru implementarea acestor mecanisme de reducere a consumului, platforma trebuie să ofere un anumit suport fizic. Exemplele cele mai semnificative fiind, capacitatea de modificare a valorii tensiunii de alimentare și/sau de modificarea a frecvenței de lucru în mod dinamic, capacitatea de a trece dintr-un regim de funcționare într-un alt regim cu consum diferit de energie, tranziția făcându-se tot prin mecanisme software. Dintre mecanismele de suport pentru reducerea consumului de energie electrică, menționate mai sus, platforma țintă trebuie să ofere cel puțin unul, pentru a se încadra în clasa de sisteme vizate în această lucrare.

De asemenea sistemul fizic conține și o sursă de alimentare, care însă nu este modelată și inclusă în modelul final.

O schemă a platformei fizice țintă este ilustrată în figura următoare:

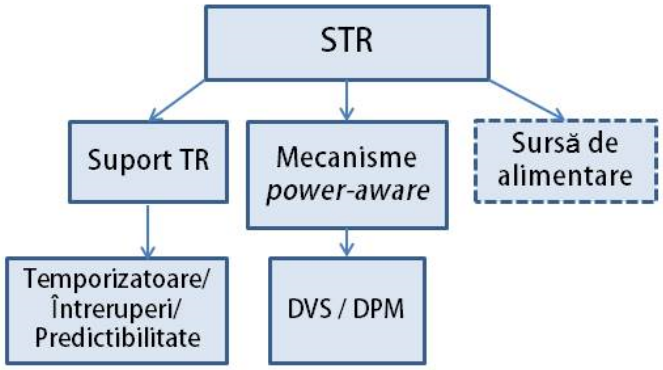


Figura 2. Sistemul timp-real țintă

4.3.2 Modelarea energetică a sistemului timp-real țintă

Un exemplu elocvent de tip de platforme vizate este nodul de rețea de senzori. Aceste noduri sunt componentele rețelelor de senzori fără fir, care îndeplinesc în mod principal funcții senzoriale, de procesare și de comunicare. Pe lângă aceste funcții, ele pot avea capabilități de mobilitate sau de acționare, în acest caz putem vorbi deja de roboți sau noduri dotate cu componente robotice. Pornind de la funcționalitățile de bază ale unui nod obișnuit, putem vedea nodul de rețea de senzori ca un ansamblu de componente ce îndeplinesc fiecare o anumită funcție. Astfel, principalele componente ale unui nod de rețea de senzori sunt [92-95]:

- *un subsistem de senzori și traductoare* pentru achiziții de date din mediul înconjurător. Acesta are rolul de a detecta și măsura anumite schimbări din mediu (ex. temperatura).
- *un subsistem de procesare* si stocare locală a datelor. Acesta are principalul rol de a procesa datele primite de la subsistemul de senzori.
- *un subsistem de comunicație* pentru transmisia datelor. Rolurile principale al acestuia sunt de a transmite datele culese din mediu către alte noduri sau către un punct de acces pentru utilizator, de a recepționa comenzi și de a asigura o cale de comunicație prin rețeaua de senzori.

La care se adaugă:

- *un subsistem de alimentare* cu rolul de a furniza tensiune și curent pentru fiecare subsistem din nod.

În Figura 3. sunt reprezentate aceste subsisteme, precum și interacțiunile dintre ele. Fiecare subsistem conține anumite componente de bază, la care se pot adaugă alte componente facultative. Astfel, unitatea senzorială, poate conține unul sau mai mulți senzori analogici sau numerici. Configurația unei unități sau a unui dispozitiv depinde și de configurațiile celorlalte dispozitive. De exemplu, în cazul în care senzorii sunt numerici, convertorul analog numeric este integrat în senzor și nu mai trebuie să constituie o componentă a unității de procesare.

Unitatea de procesare și stocare a datelor conține în mod obligatoriu o unitate de procesare, ce poate fi reprezentată de un microcontroler, un procesor pentru semnale digitale DSP (*Digital Signal Processor*) un procesor de uz general sau chiar o matrice programabilă FPGA (*Field Programmable Gate Array*). În ceea ce privește suportul de memorie fizică (necesară atât în procesul de prelucrare a datelor cât și pentru stocarea locală a acestora), aceasta poate fi un cip separat sau integrată în procesor.

Unitatea de comunicație poate fi constituită dintr-un receptor, transmițător sau transceiver, de obicei radio, iar unitatea de alimentare conține pe lângă bateria propriu zisă și un regulator de tensiune.

Pe lângă unitățile enumerate mai sus, mai pot fi întâlnite și unități opționale, ca: unități de comandă, ce conțin diverse actuatoare, prin care se poate acționa asupra mediului înconjurător sau unități de mobilitate, în cazul senzorilor mobili. Aceste unități apar în Figura 4.

Astfel, prin dispozitiv ne putem referi fie la o *unitate principală* (ex. unitatea senzorială), fie la o componentă a acestuia (un senzor, o parte componentă dintr-un senzor mai complex, etc.).

Se pune problema determinării acelor dispozitive, numite în continuare *dispozitive elementare*, care pot fi controlate din punct de vedere al consumului de putere prin software.

Definiția 1: Prin ***dispozitiv elementar*** înțelegem *acel modul hardware care nu este divizat în alte submodule, care poate fi trecut prin metode software în diferite regimuri de funcționare cu consum energetic diferit și pentru care se poate determina în mod direct energia consumată în aceste regimuri.*

Abordăm problema de la un nivel superior spre cele inferioare. Vom porni de la nivelul nodului de senzori. Factorii ce influențează consumul la nivel de placă și care pot fi controlați prin software sunt: *numărul și tipul componentelor active* și *regimul de funcționare al fiecărei componente.* Nu putem vorbi, în prezent, despre o variație dinamică a tensiunii de alimentare sau a frecvenței de lucru la nivel de placă. Astfel, pentru a controla consumul energetic, va trebui sa împărțim placa în alte subsisteme ce pot fi controlate energetic prin software, iar cele care nu pot fi controlate prin metode software să fie considerate ca subsisteme separate, astfel încât suma valorilor consumului la nivel de subsisteme componente să fie valoarea consumului la nivel de placă [83].

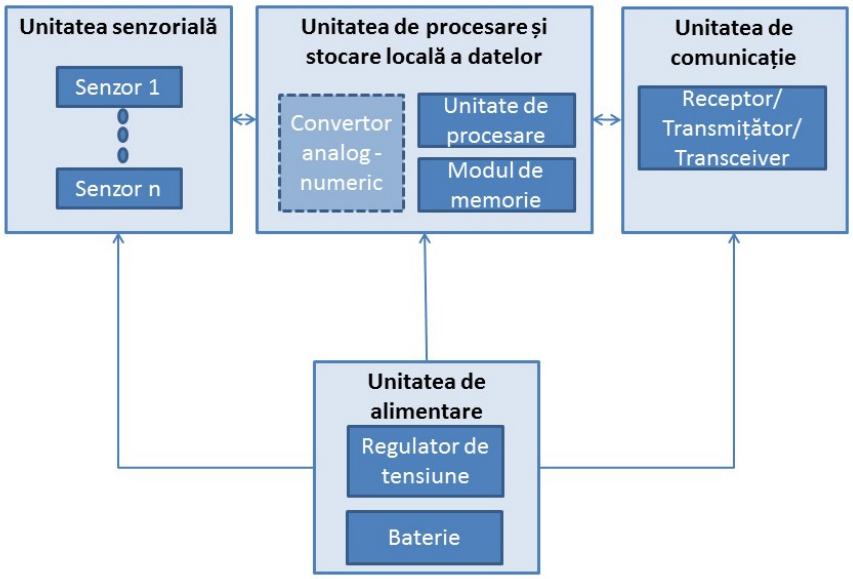


Figura 3. Nodul de rețea de senzori clasic

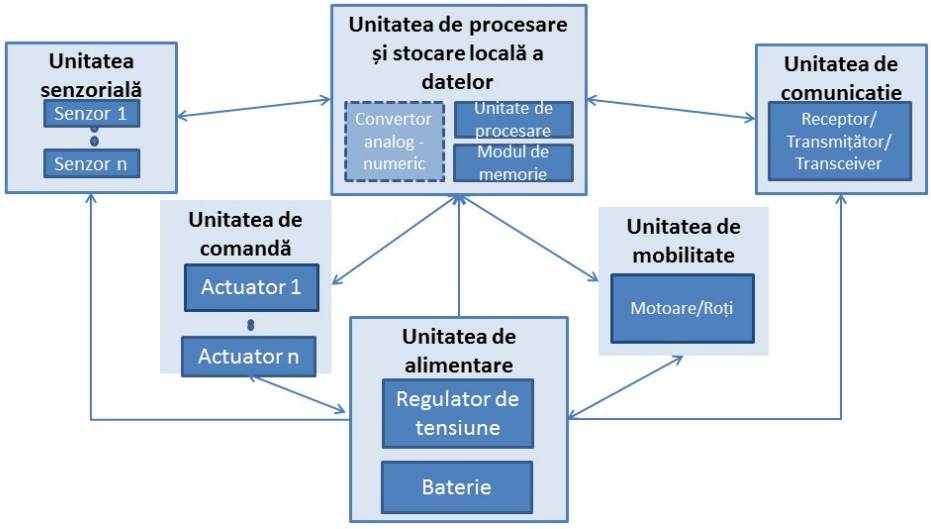


Figura 4. Nodul de rețea de senzori extins

La rândul lor aceste subsisteme pot fi împărţite în alte submodule, până se ajunge la ceea ce am numit *dispozitive elementare*. Condiția este ca suma valorilor consumului fiecărui dispozitiv elementar, plus eventual, a altor dispozitive neelementare, să fie egală cu valoarea consumului la nivel de placă.

$$E_{i\_tot} = \sum_{1}^{m} E_{i,j} + E_{ct} \tag{4-1}$$

unde, E_{i_tot} reprezintă consumul la nivel de placă datorat execuției unui taskul θ_i, pe durata unei instanțe a acestuia și este o sumă a valorilor consumului total al fiecărui dispozitiv elementar pe această durată, iar E_{ct} reprezintă consumul acelor dispozitive (neelementare) ce nu pot fi controlate din punct de vedere al regimului de funcționare prin metode *software*.

Altfel spus, consumul datorat instanței unui task, calculat la nivel de placă, variază în funcție de dispozitivele utilizate, de numărul acestora *m,* de consumul de energie al fiecărui dispozitiv în parte, care la rândul lui depinde de configurația $c_{i,j}$ a dispozitivului *j* pe durata rulării taskului *i*:

$$E_{i\_tot} = f(m, E_{i,j}(c_{i,j})) \tag{4-2}$$

În continuare, vom merge mai departe, de la nivelul plăcii la nivel de subsisteme sau *dispozitive*, menționate la începutul acestui capitol și prezentate în Figura 3.

4.3.2.1 Unitatea de senzori și traductoare

Unitatea de senzori și traductoare este alcătuit, dintr-unul sau mai multe traductoare echipate cu senzori prin intermediul cărora se extrag date asupra mediului înconjurător. Prin *traductoare* înțelegem întreg dispozitivul sau sistemul *"care stabilește o corespondență între valorile unei mărimi specifice acestui sistem și valorile unei mărimi de altă natură, specifice altui sistem, utilizat în tehnică, electricitate și telecomunicații"*[96]. Altfel spus, traductoarele fac o conversie dintr-o anumită mărime fizică neelectrică într-una electrică. Acestea conțin un senzor sau *"dispozitiv (ultrasensibil), care sesizează un anumit fenomen"* [96], elementele de legătură și transmisie, adaptoare electronice, și eventual convertoare analog numerice. În practică noțiunile de traductor și senzor se confundă, mai ales prin apariția noțiunii de *senzor inteligent (smart sensor)*, prin care înțelegem un sistem integrat bazat pe un micro-senzor, care conține și circuite adaptoare, de interfațare, (poate conține circuite de conversie analog-digitală și chiar microcontrolere), capabil să facă anumite operații asupra datelor și să comunice cu un microprocesor.

În cele ce urmează, vom folosi noțiunea de senzori cu sensul de senzori inteligenți și ne vom referi la întreg circuitul integrat reprezentat de aceștia.

După mărimea fizică de la intrare, senzorii pot fi: de temperatură, umiditate, luminozitate, masă, forță, radiații, densitate etc. După forma semnalului electric de ieșire senzorii se împart în două mari categorii: analogici și numerici (digitali).

În ceea ce privește modelarea acestora din punct de vedere energetic (fie că ne referim la senzorii analogici sau cei numerici), identificăm atât parametrii care influențează consumul de energie și pot fi controlați prin software cât și implicațiile variației acestor parametri:

- *Tensiunea de alimentare* – poate fi variată în cazul unor senzori având ca implicații directe scăderea puterii dar și scăderea performanței (ex. accelerometrul ADXL320. Aceasta poate fi variată între 2,4 și 5,25 V având ca implicații, în acest caz, scăderea curentului absorbit de circuit, o dată cu

scăderea tensiunii de alimentare, dar și scăderea preciziei de la 312mV/g la o alimentare de 5V la 135mV/g la o alimentare de 2,4V [97]). O altă implicație este variația nivelelor logice o dată cu variația tensiunii de alimentare. Luând în considerare această implicație, se vor elimina acele valori de tensiune, pentru care nivelele logice devin incompatibile pentru interfațarea cu celelalte dispozitive din nodul de rețea de senzori cu care comunică. Capacitatea de variație a tensiunii de alimentare pare a fi tot mai întâlnită în senzorii inteligenți de ultimă generație.

- *Stări de funcționare predefinite* – pentru unii senzori există diferite regimuri de funcționare predefinite, între care se poate comuta, în mod dinamic, prin programare, și care sunt definite prin valori ale puterii diferite, dar și prin nivele de performanță diferite (ex. accelerometrul LSM303DLH suportă diferite regimuri de funcționare pentru care energia consumată și rata de furnizare a datelor la ieșire este diferită [98]). Mai mult, pentru anumiți senzori se pot dezactiva anumite dispozitive (altele decât cel de comunicare/interfațare) prin trecerea într-un regim de funcționare inactiv, denumit de obicei *power-down.*

$$E_{i,j} = f(c_{i,j}) \tag{4-3}$$

Energia senzorului *j* pe durata rulării taskului *i* variază în funcție de configurația $c_{i,j}$ a senzorului care poate fi reprezentată de un regim de funcționare predefinit sau poate să varieze în funcție de tensiunea de alimentare:

$$c_{i,j} = f(U_{a\lim}) \tag{4-4}$$

Pentru senzorii care nu au capabilități de management al consumului (și implicit pentru orice dispozitiv elementar care nu are această capabilitate), se poate considera că au doar un singur regim de funcționare (o singură configurație disponibilă), pentru care determinăm puterea.

Dacă senzorul este analogic, atunci vom avea nevoie de un convertor analog-numeric, pentru interfațarea acestuia cu microprocesorul. Convertorul analog numeric, dacă este circuit separat de senzorul inteligent, poate fi văzut și el ca un dispozitiv elementar. Factorii care influențează consumul de energie sunt tot tensiunea de alimentare și modurile de operare, dacă există. Astfel, de exemplu convertoarele ADS1113, de la firma Texas Instruments, acceptă valori de alimentare între 2 și 5,5V și două moduri de lucru (mod continuu - în care se fac conversii cu o anumită rată și mod conversie unică - în care se face doar o conversie apoi convertorul este trecut într-un mod inactiv).

4.3.2.2 Unitatea de procesare și stocare locală a datelor

Unitatea de procesare și stocare locală a datelor este alcătuită dintr-o unitate de procesare și unul sau mai multe module de memorie. Opțional, aceasta poate conține și convertoare analog-numerice, ca dispozitive de sine stătătoare, sau integrate în procesor.

După cum s-a menționat anterior unitatea de procesare poate fi reprezentată de diferite dispozitive de procesare.

Procesoarele cele mai întâlnite în nodurile de senzori le putem clasifica în:

- *procesoare de uz general*, dezvoltate în general pentru dispozitive portabile și/sau telefoane mobile. Acestea au o putere de computație mare și sunt capabile de tehnici de management al puterii complexe, datorită arhitecturii

lor (Intel Xscale PXA27X, StrongARM SA-1100, DragonBall MC9328 MX1, etc.);

- *microcontrolere cu putere de procesare relativ mare* și capabile de diferite tehnici de management al puterii (familiile bazate pe arhitectura ARM7 sau ARM-Cortex, microcontrolerul MSP430, etc.);
- *microcontrolere cu putere de procesare redusă* și cu consum redus de energie (familiile ATmega8/16/128, bazate pe arhitectura 80C51, Motorola HC etc.);

Unitatea de stocare a datelor poate fi reprezentată de diferite tipuri de memorie externă sau chiar de memoria internă a procesorului. Un exemplu de astfel de memorie este *memoria EEPROM* (Electically Eraseble Programable Read Only Memory), o memorie non-volatilă, ce poate fi programată și reprogramată direct în circuit – poate să fie reprezentată de un cip de sine stătător sau poate fi integrată în cipul procesorului. Parametrii care ne interesează pentru acest dispozitive sunt: starea sau stările de funcționare, consumul în fiecare stare, timpul de tranziție dintr-o stare în alta și energia consumată prin tranziția respectivă. De exemplu cipul 25AA1024 al firmei Microchip are o stare activă (*active* - cu un consum de 5-7 mA), una de așteptare (*standby* - în care se intră atunci când cipul nu este selectat și are un consum de 1uA) și una inactivă (*deep power down* - cu un consum tot de 1uA). Intrarea în starea inactivă se face prin trimiterea unei comenzi de un octet. Timpul de tranziție din modul activ în această stare este reprezentat de suma dintre timpul necesar transmiterii unui octet pe interfața serială SPI către cip și un timp de întârziere specificat în paginile de catalog ca fiind 100 μs. Același timp îl necesită și tranziția din starea inactivă în cea de așteptare, neexistând o tranziție directă între starea inactivă și cea activă, iar tranziția din starea de așteptare în cea activă durează 100 μs (ne mai fiind nevoie de transmiterea unui cuvânt de comandă, aceasta făcându-se automat la selectarea cipului).

Astfel, cipul de memorie, poate fi văzut și modelat ca un dispozitiv elementar.

În plus, în unitatea de procesare și stocare locală a datelor se mai pot găsi și alte module, ca de exemplu convertoarele analog-numerice. Acestea pot fi la rândul lor dispozitive de sine stătătoare sau pot face parte din procesor.

Putem privi modulul de procesare și stocare locală a datelor ca un dispozitiv elementar sau ca o sumă de alte dispozitive elementare (procesor, memorie, ADC, etc.).

Dacă privim modulul de procesare ca un dispozitiv elementar, ne punem problema factorilor ce influențează consumul de energie, pentru a determina acele configurații energetice corespunzătoare modelului unui *dispozitiv elementar* (v. Definiția 1). Factorii principali luați în considerare sunt reprezentați în Figura 5.

Astfel, văzut ca un singur element, unitatea de procesare și stocare locală a datelor va avea un singur set de parametri. Frecvența dispozitivului considerat, va fi frecvența de lucru a procesorului, care trebuie să fie compatibilă cu frecvența de lucru a memoriei și a celorlalte circuite din unitate. Frecvența minimă a unității, va fi frecvența minimă a procesorului, pentru care și memoria și celelalte componente vor îndeplini cerințele de funcționare. De asemenea surplusul de timp datorat comutării dintr-o stare în alta va fi dat de timpul de stabilizare al întregului dispozitiv, iar parametrii de energie vor fi determinați tot la nivel de dispozitiv.

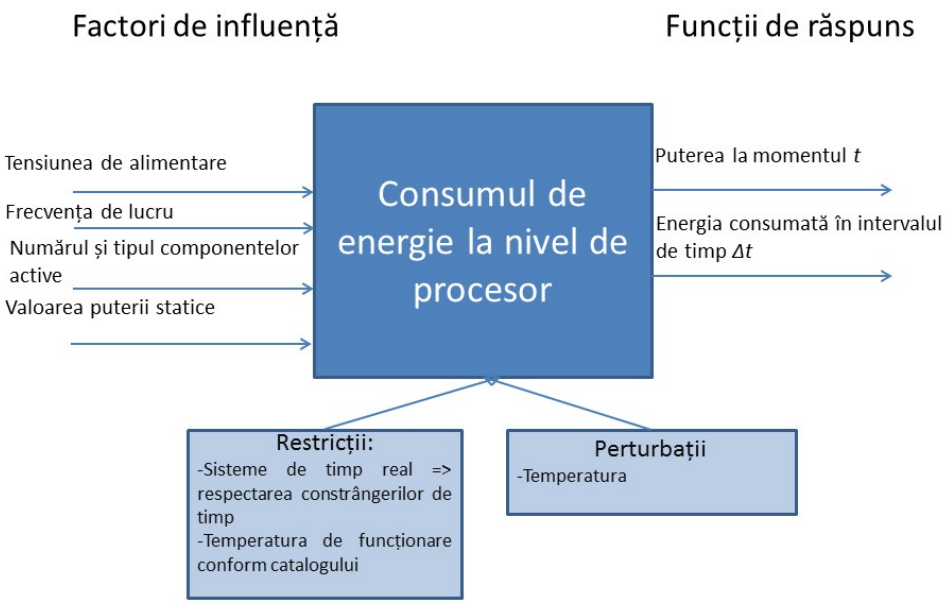


Figura 5. Consumul de energie la nivel de procesor

Dacă, în schimb, considerăm doar procesorul ca dispozitiv elementar, vom avea în vedere principalii factori de influență asupra consumului, discutați în Capitolul 3 .

După cum s-a menționat în capitolul respectiv, cea mai mare influență asupra consumului o are tensiunea de alimentare a procesorului. Variația puterii, funcție de tensiunea de alimentare și/ sau frecvența de lucru, a fost discutată în detaliu în primul capitol. Această variație este dată de variația componentei puterii dinamice, puterea totală fiind suma dintre puterea dinamică și cea statică și cea datorată curenților de scurtcircuit, care nu poate fi controlată prin software și având o valoare mică comparativă cu restul componentelor de putere, este neglijată. Puterea dinamică P_D este direct proporțională cu frecvența de lucru și proporțională cu pătratul valorii tensiunii de alimentare a procesorului conform ecuației (3-3).

Conform acestei ecuații, reducând tensiunea de alimentare sau frecvența de lucru a procesorului reducem puterea dinamică și implicit puterea totală a procesorului. Acest lucru nu este la fel de valabil însă si pentru energie. Reducând frecvența de lucru reducem puterea dinamică în mod liniar, dar creștem durata de execuție a taskului tot în mod liniar, după cum rezultă din ecuațiile ce urmează:

Conform ecuației următoare energia consumată de un dispozitiv *j* pe durata taskului *i* este: $E_{i,j} = P_j \cdot C_j$, iar

$$P_j = P_{j\_D} + P_{j\_S} \tag{4-5}$$

unde, P_{j_D} este puterea dinamică pentru dispozitivul *j*, în cazul acesta, dispozitivul este reprezentat de procesor, iar P_{j_S} este puterea statică a acestuia.

Din cele două relații matematice și relația (3-3) rezultă:

$$E_{i,j} = (P_{j\_D} + P_{j\_S}) \cdot C_i = P_{j\_D} \cdot C_i + P_{j\_S} \cdot C_i = C_{j\_ef} \cdot V_{j\_dd}^2 \cdot f_j \cdot C_i + P_{j\_S} \cdot C_i \tag{4-6}$$

P_{j_S} nu depinde de frecvența de lucru a dispozitivului (v. ecuația (3-5)).

Astfel, dacă scădem frecvența procesorului de la f_j la $\varepsilon \cdot f_j$, unde ε ia valori pe intervalul de numere reale (0;1], timpul de execuție se schimbă la rândul lui din C_i în $(1/\varepsilon)\cdot C_i$ ori, iar energia consumată devine:

$$E_{i,j}' = C_{j\_ef} \cdot V_{j\_dd}^2 \cdot \varepsilon \cdot f_j \cdot \frac{C_i}{\varepsilon} = C_{f\_ef} \cdot V_{j\_dd}^2 \cdot f_j \cdot C_i + P_{j\_S} \cdot \frac{C_i}{\varepsilon} \quad (4\text{-}7)$$

Ținând cont că ε ia valori pe intervalul de numere reale (0;1], C_i/ε este mai mare decât C_i, deci si $E_{i,j}' > E_{i,j}$. Astfel, energia consumată a crescut în loc să scadă.

Această relație este valabilă și pentru alte dispozitive care au capacitatea de a-și schimba frecvența de lucru.

Pe de altă parte scăderea tensiunii de alimentare conduce atât la reducerea puterii dinamice, cât și a celei statice și de asemenea la reducerea consumului de energie (conform ecuațiilor (3-3) și (3-5)). Altfel relațiile (4-3) și (4-4) rămân valabile și în cazul unității de procesare.

Așa cum consumul unei plăci poate fi văzut ca o sumă a valorilor de consum a componentelor sale, așa și procesorul poate fi văzut ca o sumă a valorilor de consum a componentelor sale, dintre care menționăm: nucleul procesorului, diferite periferice ca: porturi de intrare/ieșire, porturi de comunicație, contoare de timp, convertoare analog-numerice, etc.

Astfel, poate fi considerată ca *dispozitiv elementar* și modelată corespunzător întreaga unitate de procesare și stocare locală a datelor, procesorul, memoria sau alt modul al subsistemului de procesare și stocare locală a datelor, sau chiar un submodul al procesorului.

4.3.2.3 Unitatea de comunicație

În [99] este propus un model de consum pentru modulul de comunicație. Puterea consumată este împărțită în diferite componente, care fie sunt constante pentru un anumit dispozitiv, fie depind de distanța de transmisie. S-a luat ca studiu de caz și s-au făcut măsurători pe dispozitivele CC1000 și CC2420. Modelul este folosit pentru a calcula și compara consumul de putere în diferite cazuri de comunicație la diferite distanțe.

Modulul de comunicație este la rândul lui împărțit în componente și pentru fiecare componentă este apoi determinată funcția de putere. Astfel este prezentată o variantă de modelele pentru consumurile de putere pentru recepția, respectiv transmisia unui semnal:

$$\begin{cases} P_R = P_{RB} + P_{RRF} + P_L = P_{R0} \\ P_T(d) = P_{TB} + P_{TRF} + P_A(d) = P_{T0} + P_A(d) \end{cases} \quad (4\text{-}8)$$

unde, P_R reprezintă puterea totală consumată de modul la recepție, P_{RB} reprezintă puterea circuitului responsabil cu demodulația semnalului recepționat, P_{RRF} puterea circuitului responsabil cu recepția efectivă, P_L reprezintă puterea circuitului de amplificare și filtrare a semnalului recepționat. Deoarece nu depinde de distanța de comunicație, puterea de recepție poate fi considerată constantă pentru un anumit dispozitiv aflat într-o anumită configurație și este notată cu P_{R0}. În mod analog $P_T(d)$ reprezintă puterea totală consumată de modul la transmisie și este o funcție de distanța d maximă de transmisie, P_{TB} reprezintă puterea circuitului responsabil cu modulația semnalului transmis, P_{TRF} puterea circuitului responsabil cu transmisia efectivă, $P_A(d)$

reprezintă puterea circuitului responsabil cu amplificarea semnalului și este o funcție dependentă de distanța maximă de transmisie. Puterea de transmisie poate fi scrisă ca o sumă între o parte constantă din punct de vedere a distanței P_{T0} și o parte dependentă de această distanță $P_A(d)$.

Pornind de la acest model, putem identifica principalii factori de influență asupra consumului modulului de comunicație. Aceștia sunt reprezentați în Figura 6.

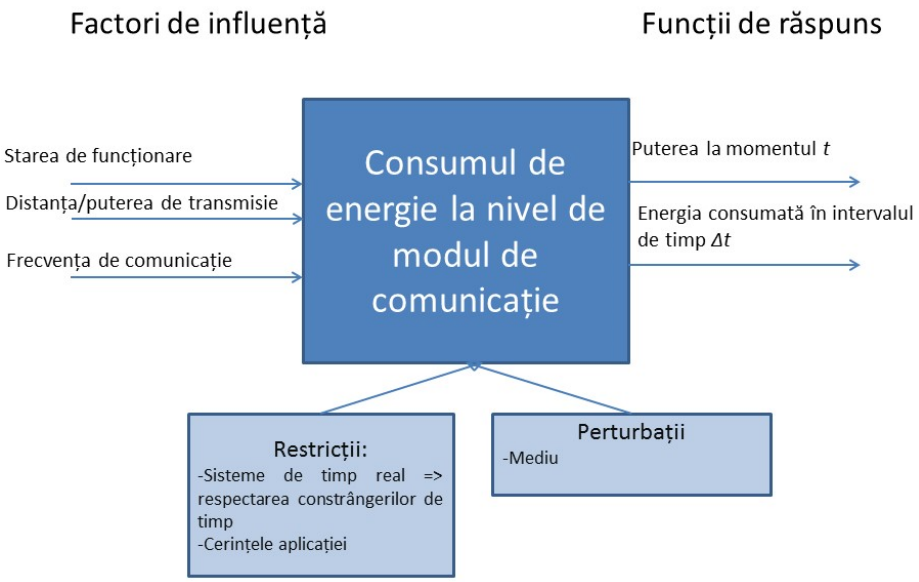


Figura 6. Consumul de energie la nivel de modul de comunicație

Studiind foile de catalog pentru dispozitivele de comunicație pe radio, identificăm următorii factori de influență ai consumului, ce pot fi controlați prin software:

- *Regimul de funcționare predefinit* – dat de numărul și tipul componentelor active. Astfel, de exemplu la modulul de comunicație CC1101 consumul de curent poate varia între 0,2 și 165uA pentru diferite stări inactive *(power-down)* sau între 1,4 și 39,3uA în diferite stări de așteptare [100].
- *Frecvența de comunicație* – influențează consumul atât la transmisie cât și la recepție.
- *Tipul de modulație și sensibilitatea* – influențează consumul doar pe partea de recepție.
- *Puterea de transmisie* – la o putere de transmisie mică avem un consum mai mic de energie, dar și distanța maximă de comunicație este redusă.
- *Tensiunea de alimentare* – există și module de comunicație care acceptă diferite valori pentru tensiunea de alimentare. Un exemplu este LMX3162 al firmei National Semiconductor care acceptă valori ale tensiunii de alimentare între 3 și 5,5V

Relația (4-3) rămâne valabilă și în cazul modulului de procesare. Iar relația (4-4), reprezentând factorii ce definesc o configurație (alta decât cele predefinite) poate fi extinsă în cazul dispozitivului/modulului de comunicație la:

$$c_{i,j} = f(f_{com}, P_t) \tag{4-9}$$

unde f_{com} reprezintă frecvența de comunicație și P_f reprezintă puterea de transmisie.

4.3.2.4 Unitatea de alimentare

Unitatea de alimentare nu poate fi utilizată pentru a scădea consumul de energie al nodului de senzori, astfel el nu poate face parte din vectorul de dispozitive ale unui task. Cu toate acestea, durata de viață a nodului de senzori este puternic influențată de tipul și capacitatea bateriei.

În [60] este prezentat un exemplu de patru scenarii diferite în care energia consumată este aceeași, dar modul de descărcare al bateriei este diferit și care duc la durate de viață diferite, pentru același tip de baterie. De asemenea se arată îmbunătățiri în cazul unui scenariu în care taskurile rulează la tensiuni de alimentare are procesorului în ordine descrescătoare. Ceea ce înseamnă că primul task necesită valoarea cea mai mare a tensiunii de alimentare, apoi valoarea este redusă sau păstrată de la un task la altul, dar niciodată crescută de la unul la altul.

Modelul de baterie utilizat este același cu cel din [18]:

$$\alpha = \int_0^L i(\tau)d\tau + 2\sum_{m=1}^{\infty}\int_0^L i(\tau)e^{-\beta^2 m^2 (L-\tau)d\tau} \tag{4-10}$$

unde α este capacitatea totală a bateriei, L este durata de viață a bateriei și β este un parametru specific bateriei, ce caracterizează profilul neliniar de descărcare al acesteia.

Pentru un curent de descărcare constant, ecuația devine:

$$\alpha = I[L + 2\sum_{m=1}^{\infty}\frac{1-e^{1-e^{-B^2 m^2 L}}}{\beta^2 m^2}] \tag{4-11}$$

Pentru acest model de baterie a fost elaborat și prezentat, tot în acest articol, și un algoritm de planificare *power-aware*, bazat pe modelul de baterie menționat anterior. Modelul taskurilor conține pe lângă parametrii temporali (timp de start, durata de execuție și deadline) și un parametru ce reprezintă consumul de curent (pentru aceeași tensiune de alimentare). Pentru fiecare set de taskuri planificat se calculează o funcție de consum σ care reprezintă valoarea cu care scade capacitatea bateriei după rularea acestor taskuri într-un timp T. Astfel, algoritmului de planificare i se poate asocia o funcție de cost $Q=\alpha-\sigma$, care trebuie maximizată. Pentru această funcție de cost au fost demonstrate următoarele proprietăți:

- Dacă nu se pot schimba valorile tensiunilor de alimentare pentru taskuri, aranjarea acestora în ordine descrescătoare a consumului de curent duce la o planificare optimă (dacă încărcarea procesor este constanta pe parcursul unui task).
- Pentru a reface din capacitatea bateriei pentru a face față cerințelor unui task k este mai bine să se scadă tensiunea de alimentare pentru acel task decât să se insereze perioade libere înainte de execuția acestui task.
- Pentru două taskuri cu aceeași cerință de consum, dacă există o perioadă liberă este mai eficient să fie alocată pentru a reduce tensiunea celui de-al doilea task, decât pentru primul.

Ținând cont de proprietățile enunțate mai sus, în articolul referit s-a dezvoltat un algoritm ce cuprinde două etape principale. În prima etapă se planifică taskurile cu

ajutorul algoritmului EDF, încercând să se rearanjeze taskurile, care se pot rearanja, în ordine descrescătoare a consumului, fără a depăși deadline-urile. În a doua etapă se folosește timpul cât procesorul nu e utilizat pentru DVS. Acesta este alocat pe cât se poate spre finalul ciclului de planificare.

În [18] peste acest algoritm este adăugată și o procedură de *adaptive body biasing*. Procedură prin care se modifică în mod dinamic tensiunea $V_{BS.}$ (3-6)

4.4 Modelarea taskurilor în sistemul TEEARTS

4.4.1 Aspecte generale

Lucrarea [94] definește conceptul de sistem "*power aware*" ca fiind *„un sistem care își scalează consumul de putere, prin modificări ale scenariului de operare, cu scopul de a-și maximiza eficiența energetică*". Altfel spus, acesta este un sistem care este conștient de consumul său de energie și este capabil de a ajusta acest consum.

Astfel, pornind de la modelele clasice de taskuri pentru sistemele de timp real, prezentate în capitolele precedente, s-au dezvoltat noi modele și algoritmi de planificare aferenți acestora, prin înglobarea caracteristicilor de consum pentru fiecare task în parte. Aceasta s-a realizat prin adăugarea de noi parametri, care să descrie consumul energetic realizat de un task.

În [73] este prezentat un model de taskuri care pe lângă parametrii temporali (perioada, timp de execuție) conține și o listă cu dispozitivele de intrare/ieșire folosite de acel task:

$$\begin{cases} T_i \equiv (P_i, C_i, Dev_i) \\ Dev_i \equiv \{\lambda_1, \lambda_2, ...\lambda_m\} \end{cases} \tag{4-12}$$

unde, T_i este task-ul *i,* P_i este perioada taskului, C_i este timpul de execuție și Dev_i este o listă a dispozitivelor folosite de task-ul *i.*

Modelul cuprinde dispozitive ce au două stări de funcționare. Fiecărui dispozitiv îi sunt asociați timpii de trecere dintr-o stare în alta și timpul minim necesar pentru ca dispozitivul să se afle în starea inactivă pentru a compensa costurile (energetice) datorate tranziției. Modelul este extins în [72] pentru a permite descrierea taskurilor ce utilizează în comun anumite resurse nonpreemptibile. Pe lângă parametrii menționați anterior apare și o listă a resurselor utilizate de un task. Un model asemănător este prezentat în [74] și [75]. Practic este același model pentru taskuri, iar lista de parametrii pentru un dispozitiv este următoarea:

$$\{ps_{l/h}, t_{wu}, t_{sd}, P_{wu}, P_{sd}, P_w, P_s\} \tag{4-13}$$

unde, $ps_{l/h}$ reprezintă starea de funcționare (activă/ inactivă), t_{wu} reprezintă timpul de tranziție din starea inactivă în cea activă, t_{sd} reprezintă timpul de tranziție din starea activă în cea inactivă, P_{wu} reprezintă puterea consumată în timpul tranziției din starea inactivă în cea activă, P_{sd} reprezintă puterea consumată în timpul tranziției din starea activă în cea inactivă, P_w reprezintă puterea consumată în starea activă și P_s reprezintă puterea consumată în starea inactivă.

În [75], acest model este extins și pentru dispozitive cu mai multe stări de funcționare (una activă și mai multe inactive). Astfel modelul devine:

$$\{ps_{u,1..m}, t_{wu}^{j}, t_{sd}^{j}, P_{wu}^{j}, P_{sd}^{j}, P_{w}, P_{s}^{j}\} \quad (4\text{-}14)$$

unde, $ps_{1..m}$ reprezintă starea de funcționare (reprezentată printr-un indice de la 1 la numărul total de stări inactive *m,* sau indicele *u* pentru starea activă), t^{j}_{wu} reprezintă timpul de tranziție din starea *j* în starea *j+1*, t^{j}_{sd} reprezintă timpul de tranziție din starea *j* în starea *j-1*, P^{j}_{wu} reprezintă puterea consumată în timpul tranziției din starea *j* în starea *j+1*, P^{j}_{sd} reprezintă puterea consumată în timpul tranziției din starea *j* în starea *j-1*, P_w reprezintă puterea consumată în starea activă și P^{j}_{s} reprezintă puterea consumată în starea inactivă *j*.

În [101] se arată că timpii și energia consumată de un procesor (sau alt dispozitiv), care poate funcționa la diferite frecvențe de lucru, diferă în funcție de această frecvență. Acest lucru face ca modelul de mai sus să nu fie potrivit pentru astfel de dispozitive. Astfel apare o altă serie de modele corespunde taskurilor ce rulează pe procesoare cu frecvența de lucru și/sau tensiunea de alimentare variabilă. Un astfel de exemplu se regăsește în [53]. Acest model conține pe lângă parametrii de timp pentru un task și un nivel de performanță notat cu p_α, care aici este reprezentat prin frecvența de lucru a procesorului. Astfel, parametrii de timp ai taskului nu mai sunt constanți, ci devin funcții de p_α. De asemenea consumul energetic pentru un task este o funcție de p_α.

Mai mult, autorii din [102] se folosesc de un model mai realist. Potrivit acestui model, timpii și energia consumați de un task conțin o parte ce variază o dată cu frecvența de lucru a procesorului și o parte independentă de aceasta (datorată de obicei accesului la magistrale externe). Astfel timpul de execuție C_i este exprimat ca fiind alcătuit din două componente, una variabilă cu frecvența c_i (exprimată în cicluri de tact) și una invariabilă m_i (exprimată în secunde):

$$C_i = c_i + \alpha \cdot m_i \quad (4\text{-}15)$$

unde α reprezintă frecvența procesorului în cicluri/secundă.

4.4.2 Modelul propus

După cum s-a arătat în capitolul anterior, taskurile ce rulează pe un sistem de timp real, conțin anumiți parametri ce caracterizează task-ul din punct de vedere temporal. Mai mult, s-a arătat necesitatea introducerii unor parametri suplimentari, pentru acele sisteme ce au și capabilități de management al consumului. Modelele propuse în acel paragraf reprezintă un fundament pentru modelarea taskurilor în astfel de sisteme, dar nu reprezintă un model universal valabil, după cum s-a menționat în secțiunea respectivă.

Modelul taskurilor de timp real propus în această lucrare, conține pe lângă parametrii de timp prezentați în primul capitol și parametri pentru descrierea consumului unui task:

$$\theta_i \equiv \{T_i, C_i, D_i, \varphi_i, \Psi_i\} \quad (4\text{-}16)$$

unde, θ_i este taskul *i* din set, definit de parametrii (T_i – perioada taskului, C_i – timpul de execuție (pentru cazul cel mai defavorabil),D_i reprezintă termenul (*deadline*) relativ al taskului *i,* φ_i – timpul de start sau faza, Ψ_i– un vector ce conține dispozitivele necesare taskului *i*).

4.4.2.1 Parametrii temporali

În ceea ce privește parametrii de timp, atât perioada taskului, timpul de execuție cât și timpul de start trebuie specificați în mod absolut. Fie că sunt specificați în unități de măsură în sistem internațional (submultipli ai secundelor), fie că sunt specificați în cicluri de tact ale procesorului, pentru o frecvență de lucru bine definită, aleasă ca reper și numită frecvența nominală. De obicei aceasta este frecvența maximă de lucru a procesorului, în condiții de funcționare bine stabilite. În continuare vom considera parametrii de timp ai taskului specificați în funcție de frecvența nominală, pe care o alegem să fie egală cu frecvența maximă, dacă nu este specificat altfel.

Durata de execuție a unui task este afectată de schimbarea frecvenței de lucru a procesorului. În continuare vom porni de la modelul (4-15) de variație a timpului de execuție în funcție de frecvență, deoarece acesta este un model mai realist, care ia în considerare faptul că nu toate instrucțiunile au durata direct proporțională cu durata ciclului de tact al procesorului. Există și instrucțiuni care sunt independente de aceasta, ca de exemplu cele care necesită accesul la diferite magistrale. Pornind de la acest model, vom merge mai departe afirmând că în execuția unui task există pe lângă componenta de timp dependentă de frecvența procesorului, componente de timp dependente de frecvențele dispozitivelor folosite de acel task și componente de timp constante din punct de vedere al frecvențelor de lucru. Dacă durata de execuție C_i^k pentru instanța *k* a taskului *i,* se va exprima în continuare în cicluri de tact date de frecvența nominală de lucru a dispozitivului vom avea:

$$C_i^k = \frac{C_{i\_v\_p}}{\varepsilon_0^k} + \frac{C_{i\_v}}{\varepsilon_{j\_\min}^k} + C_{i\_f} \tag{4-17}$$

unde $C_{i\_v\_p}$ reprezintă partea variabilă în funcție de frecvența procesorului a duratei de execuție, ε_0^k reprezintă raportul între frecvența de lucru a procesorului pe durata execuției instanței *k* a taskului *i*, C_{i_v} reprezintă partea variabilă în funcție de frecvența altor dispozitive, a duratei de execuție, $\varepsilon^0_{j\_min}$ reprezintă raportul între frecvența de lucru a celui mai încet dispozitiv și cea nominală pentru acel dispozitiv și C_{i_f} reprezintă componenta de timp cu durată fixă a execuției unui task, exprimată în aceeași unitate de măsură (cicluri de tact nominale). Cele trei componente însumate reprezintă chiar durata de execuție nominală a taskului C_i.

$$C_i = C_{i\_v\_p} + C_{i\_v} + C_{i\_f} \tag{4-18}$$

Astfel relația (4-16), pentru o instanța *k* a unui task se notează cu:

$$\theta_i^k \equiv \{T_i, C_i^k, D_i, \varphi_i^k, \psi_i^k\} \tag{4-19}$$

Perioada T_i și termenul relativ D_i ai taskului sunt aceiași pentru orice instanță a taskului *i* și nu depind de configurația specifică a dispozitivului pentru instanța *k,* în schimb durata de execuție C_i^k depinde de frecvențele dispozitivelor conform ecuației (4-17), faza φ_i^k este dată de algoritmul de planificare, iar vectorul de dispozitive Ψ_i^k este caracterizat de configurațiile specifice instanței *k.*

4.4.2.2 Parametrii energetici

În ceea ce privește parametrii energetici ai unui task, aceștia sunt dați în primul rând de setul de dispozitive elementare utilizate de întreg setul de taskuri al aplicației:

$$\Psi \equiv \{\psi_0,...,\psi_j,\psi_{j+1},...,\psi_m\} \quad (4\text{-}20)$$

unde, ψ_j este dispozitivul elementar *j.*

Observație: procesorul sau nucleul acestuia, dacă procesorul e la rândul său divizat în alte module, va avea indicele *j=0.*

Dispozitivele elementare definite în Capitolul 4.3 (v. Definiția 1) sunt caracterizate prin ceea ce vom numi în continuare ***configurații active și inactive minime necesare***.

$$\psi_j \equiv \{c_{j\_act}, c_{j\_inact}, f_{j\_nom}\} \quad (4\text{-}21)$$

unde c_{j_act} reprezintă un vector de configurații active minime necesare corespunzătoare dispozitivului *j,* c_{j_inact} reprezintă o configurație inactivă minimă necesară corespunzătoare dispozitivului *j* și f_{j_nom} reprezintă frecvența nominală a acestui dispozitiv.

Definiția 2: Definim ***configurațiile active minime necesare*** *(ale unui dispozitiv sau modul hardware)* ca fiind: *stările de funcționare a acelui dispozitiv elementar cu consumul minim de energie, determinate conform cerințelor fiecărui task care le utilizează.*

$$c_{j\_act} = \{c_j^i, i = 1,...,n\} \quad (4\text{-}22)$$

unde c_j^i reprezintă configurația minima a dispozitivului *j* pentru îndeplinirea cerințelor taskului *i.*

Definiția 3: Definim ***configurația inactivă minimă necesară*** *(a unui dispozitiv j)* ca fiind: *starea de repaos a acelui dispozitiv elementar cu consumul minim de energie, care să corespundă cerințelor aplicației.*

Definiția 4: Definim ***configurația activă specifică*** *(a unui dispozitiv sau modul hardware)* ca fiind: *starea de funcționare a acelui dispozitiv elementar cu consumul minim de energie, necesară pentru a îndeplini cerințele de funcționare ale tuturor taskurilor ce accesează acea componentă.*

$$c_{j\min} = c_j^{i`} \mid perf(c_j^{i`}) \geq \forall perf(c_j^i) \quad (4\text{-}23)$$

unde $c_{j\,min}$ reprezintă configurația activă minimă necesară a dispozitivului *j* și este dată de acea configurație $c_j^{i`}$ pentru care performanța dispozitivului, notată cu $perf(c_j^{i`})$, este maximă. Altfel spus, dintre configurațiile minime necesare, se va alege cea care din punct de vedere al performanței satisface cerințele tuturor taskurilor ce utilizează dispozitivul respectiv. Funcția de performanță este specifică fiecărui dispozitiv.

Noțiunea de *configurație activă specifică,* și implicit și cea de *configurație activă minimă necesară* este valabilă pentru orice dispozitiv, inclusiv procesorul. Pentru dispozitivul *j* această configurație $c_{j\min}$ este dată prin specificarea următorilor parametri:

$$c_{j\min} = \{f_j, o_{j\_\max}, P_{i,j}, \eta_{i,j\_\max}\} \quad (4\text{-}24)$$

unde f_j reprezintă frecvența la care rulează dispozitivul în configurația curentă, acest parametru este obligatoriu, daca frecvența respectivă diferă de frecvența nominală, altfel fiind opțional, o_{j_max} reprezintă intervalul de timp pentru cazul cel mai defavorabil datorat activării configurației (i.e. timpul maxim de tranziție dintr-o configurație inactivă în cea activă, pentru dispozitivul *j*), $P_{i,j}$ reprezintă puterea corespunzătoare

configurației *i* a dispozitivului *j,* iar η_{i,j_max} reprezintă energia maximă consumată în timpul tranziției în această configurație din oricare altă configurație a dispozitivului *j*.

Configurațiile pot fi predefinite sau definite de utilizator (ex. o configurație în care sunt alimentate doar anumite componente ale dispozitivului sau, pentru un dispozitiv de comunicație poate fi configurația necesară transmisiei pe distanța minimă necesară). Un caz particular îl reprezintă configurațiile predefinite de consum redus ale unui procesor (așa numitele *low-power states*).

Observație: Un dispozitiv *j* se va afla în configurația c_j^i pe durata execuției oricărei instanțe a taskului *i*.

Din motive practice, vom grupa doi dintre parametrii configurației, definiți anterior într-un nou parametru, numit stare de funcționare. Astfel, o stare de funcționare va fi definită ca:

$$S_j^k = \{P_j^k, f_j^k\} \tag{4-25}$$

unde S_j^k reprezintă starea *k* a modulului *j* , iar P_j^k și f_j^k reprezintă puterea și respectiv frecvența de lucru a modulului *j* în starea *k.*

Conform formulei energiei, energia maximă consumată de dispozitivul *j* pe durata C_i a execuției unei instațe ale unui task θ_i, este:

$$E_{i,j}(0, C_i) = \int_0^{C_i} P_i(t)dt \tag{4-26}$$

Dacă funcția de putere este constantă, atunci ecuația devine:

$$E_{i,j} = P_i \cdot C_i \tag{4-27}$$

La această energie consumată, se adaugă surplusul de energie η_j, datorat tranziției în configurația c_j^i , definită anterior. Astfel energia totală pentru un dispozitiv datorată execuției unei instanțe a taskului *i* devine:

$$E_{i,j\_tot} = E_{i,j} + \eta_{i,j\_\max} \tag{4-28}$$

Facem următoarea observație: un dispozitiv ψ_j își păstrează starea energetică pe durata unui task. Astfel, el poate fi activat, dezactivat, trecut dintr-o stare în alta doar la începutul și/sau la sfarșitul unui task. Nu se vor face tranziții dintr-o stare în alta în timpul rulării taskului.

În continuare notăm cu ε_j^i, raportul între frecvența f_j^i corespunzătoare configurației c_j^i și cea nominală (f_j^i /f_{jnom}), pentru un dispozitiv *j.* Atunci:

$$f_j^i = \varepsilon_j^i \cdot f_{j\_nom} \tag{4-29}$$

4.5 Modulul pentru măsurarea și determinarea valorilor parametrilor energetici ai taskurilor

4.5.1 Descrierea generală a modulului

Se pune problema determinării valorilor parametrilor din modelul unui task, definit anterior. Pentru aceasta, în primul rând vom împărţii parametrii în două mari tipuri: *parametrii energetici* şi *parametrii temporali*.

Pentru determinarea parametrilor energetici ai unui task *i*, în primul rând trebuie bine determinat şi specificat setul de dispozitive elementare utilizat de acel task. În determinarea acestora se va parcurge următorul algoritm:

- Se vor determina acele *unităţi principale* conţinute în sistem (conform clasificării ilustrate în Figura 3. şi Figura 4) şi utilizate de taskul *i* (i.e. care trebuie să rămână activ pe durata taskului).
- Pentru fiecare *unitate principală*, se vor determina acele *dispozitive elementare*, a căror consum energetic poate fi ajustat prin software.
- Pentru fiecare dispozitiv elementar ψ_j se vor determina acele regimuri de funcţionare sau stări active şi inactive specifice S_j^k din (4-25). Acestea vor fi reţinute printr-o valoare de cod.
- Pentru fiecare stare se va determina consumul specific reţinând următorii parametrii: frecvenţa de lucru a dispozitivului în acea stare de funcţionare f_k (care poate avea şi valoarea nulă, dacă dispozitivul este într-o stare inactivă) şi puterea consumată în acea stare P_k.

Astfel vectorul Ψ de dispozitive hardware, specificat în modelul taskului este specificat de dispozitivele elementare considerate.

Determinarea stărilor de funcţionare pentru fiecare dispozitiv elementar, se poate face conform foii de catalog, pe baza modurilor de funcţionare predefinite şi a valorilor între care poate varia frecvenţa de lucru şi tensiunea de alimentare şi/sau prin măsurare directă. Astfel, se vor specifica diferite stări de funcţionare, caracterizate minimal printr-un cod, o frecvenţă de funcţionare (în mod opţional) şi o putere specifică. Aceste stări sunt corespunzătoare configuraţiilor active şi inactive minime necesare definite anterior.

Exemple de astfel de stări, pentru un dispozitiv de comunicaţie şi pentru un microcontroler bazat pe arhitectura ARM-Cortex M3, sunt descrise în tabelele ce urmează:

Tabelul 2. Stări de funcţionare transceiver

Nume şi tip dispozitiv	Cod stare	Descriere stare	P[mW]
CC2420 2.4 GHz transceiver	PD	Power Down	70
	IDLE	Idle	1491
	R	Receive	68,95
	T0	Transmit 0dBm	60,9
	T1	Transmit -5dBm	49
	T2	Transmit -11dBm	38,5
	T3	Transmit -15dBm	34,65

Tabelul 3. Stări de funcționare procesor

Nume și tip dispozitiv	Cod stare	Descriere stare	Frecvența de lucru	P[mW]
STM32L152 microcontroler	A1.1	Activ-Vcore Range1 MSI range 6	4.194 MHz	3,0282
	A1.2	Activ-Vcore Range1 MSI range 5	2.097 MHz	1,4994
	A1.3	Activ-Vcore Range1 MSI range 4	1.048 MHz	0,8232
	A1.4	Activ-Vcore Range1 MSI range 3	524.288 KHz	0,4704
	A1.5	Activ-Vcore Range1 MSI range 2	262.144 KHz	0,294
	A1.6	Activ-Vcore Range1 MSI range 1	131.072 KHz	0,2058
	A2.1	Activ-Vcore Range2 MSI range 6	4.194 MHz	2,4696
	A2.2	Activ-Vcore Range2 MSI range 5	2.097 MHz	1,2936
	A2.3	Activ-Vcore Range2 MSI range 4	1.048 MHz	0,735
	A2.4	Activ-Vcore Range2 MSI range 3	524.288 KHz	0,4116
	A2.5	Activ-Vcore Range2 MSI range 2	262.144 KHz	0,2352
	A2.6	Activ-Vcore Range2 MSI range 1	131.072 KHz	0,1764
	A3.1	Activ-Vcore Range3 MSI range 6	4.194 MHz	2,058
	A3.1	Activ-Vcore Range3 MSI range 5	2.097 MHz	1,0878
	A3.1	Activ-Vcore Range3 MSI range 4	1.048 MHz	0,588
	A3.1	Activ-Vcore Range3 MSI range 3	524.288 KHz	0,3234
	A3.1	Activ-Vcore Range3 MSI range 2	262.144 KHz	0,2058
	A3.1	Activ-Vcore Range3 MSI range 1	131.072 KHz	0,1176
	S1.1	Sleep Vcore Range1 MSI range 6	4.194 MHz	0,684
	S1.2	Sleep Vcore Range1 MSI range 5	2.097 MHz	0,36
	S1.3	Sleep Vcore Range1 MSI range 4	1.048 MHz	0,216
	S1.4	Sleep Vcore Range1 MSI range 3	524.288 KHz	0,18
	S1.5	Sleep Vcore Range1	262.144 KHz	0,108

Nume și tip dispozitiv	Cod stare	Descriere stare	Frecvența de lucru	P[mW]
		MSI range 2		
	S1.6	Sleep Vcore Range1 MSI range 1	131.072 KHz	0,09
	S2.1	Sleep Vcore Range2 MSI range 6	4.194 MHz	0,558
	S2.2	Sleep Vcore Range2 MSI range 5	2.097 MHz	0,306
	S2.3	Sleep Vcore Range2 MSI range 4	1.048 MHz	0,18
	S2.4	Sleep Vcore Range2 MSI range 3	524.288 KHz	0,126
	S2.5	Sleep Vcore Range2 MSI range 2	262.144 KHz	0,09
	S2.6	Sleep Vcore Range2 MSI range 1	131.072 KHz	0,072

Dintre aceste stări se vor elimina acelea care nu corespund cerințelor de funcționare ale taskului *i.* Pentru fiecare stare se va determina puterea fie prin folosirea valorii de catalog, dacă există, fie prin măsurarea valorii tensiunii de alimentare pentru dispozitivul elementar și a consumului de curent, cu ajutorul unui cadru de măsurare.

Pe baza măsurătorilor se pot obține grafice de consum pentru fiecare dispozitiv. Exemple de astfel de grafice pentru procesorul STM32L152 se pot vedea în Figura 7. Puterea corespunzătoare diferitelor stări de funcționareFigura 7.

Un exemplu propus pentru modulul de măsurare al parametrilor energetici pentru diferite dispozitive, capabil să măsoare un consum de curenți între aproximativ 500nA și 10A, este descris în continuare:

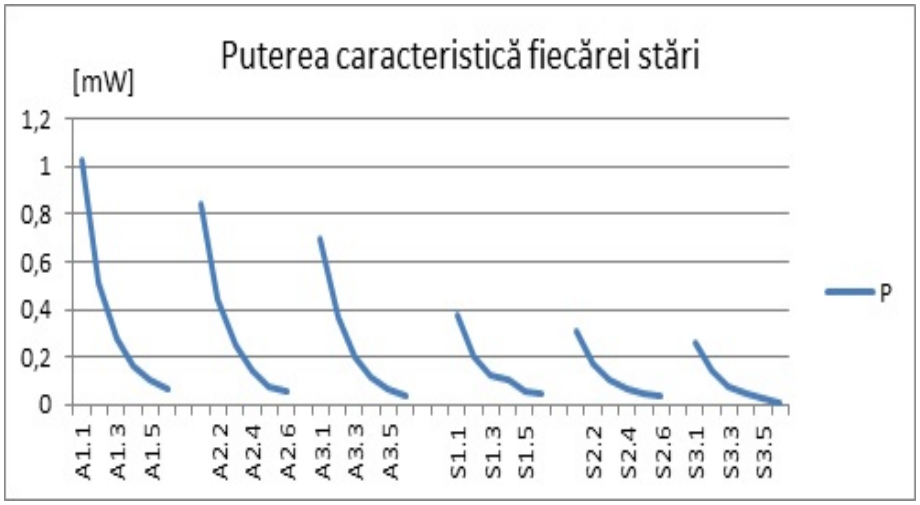


Figura 7. Puterea corespunzătoare diferitelor stări de funcționare

Sistemul este alcătuit din următoarele componente principale:

- *Ampermetru de mare precizie* MASTECH MS8050 [103], cu facilitate de achiziție a măsurătorilor și de comunicare printr-o interfață serială permițând astfel achiziționarea valorilor măsurate. Acesta are plajă de domenii de măsurare, pentru curenți, între 500μA și 10A.
- *Sursa de tensiune* HAMEG 7044 [104], reglabilă programatic. Aceasta este o sursă cu patru canale ce dispune de o interfață serială prin intermediul căreia se pot gestiona aceste canale.
- *Subiectul de măsurat* – sau sistemul țintă. Exemple: Microcontroler / DSP-uri cu plaja variabilă de alimentare ATMEGA16, LPC2294; procesoare cu capacitate de management al puterii evoluat (ex.ADSP-BF537); plăci la care se pot alimenta independent anumite dispozitive.

Din punct de vedere al uneltelor software sistemul conține o aplicație pentru PC care gestionează sursa de tensiune și ampermetrul, cu scopul de a putea modifica tensiunea de alimentare transmisă spre subiectul de măsurare și in același timp, măsurarea curentului consumat de aceasta.

Datele culese de la sursa de alimentare și de la ampermetru sunt stocate într-o baza de date ca mai apoi să se poate extrage de acolo informațiile dorite. În același timp există posibilitatea de a vizualiza aceste date în timp real, sub forma unui grafic, ce are o coordonată de timp și una de putere/curent consumat. Câmpurile principale din baza de date sunt: tensiunea, curentul, ștampila de timp (la nivel de milisecundă), un identificator al setului de măsurători care are o corespondență într-o altă tabelă cu descriptori (tipul procesorului, placa, condițiile de măsurare, observații, etc.).

4.5.2 Studiu de caz: energyAwareProfiler – Simplicity Studio

Recent, au fost dezvoltate cadre de măsurare conținând atât platforme fizice, cât și unelte pentru determinarea în timp real a consumului unui procesor sau dispozitiv. O astfel de unealtă software este Simplicity Studio energyAware Profiler [105] ce poate măsura curentul, tensiunea și momentul de timp la care se face citirea și poate determina în timp real consumul pentru o serie de dispozitive bazate pe microcontrolerele din familiile EFM32 GG, G, TG etc., bazate pe arhitectura ARM-Cortex (M0, M3, M4). Aceste platforme sunt dotate cu un sistem avansat de monitorizare a consumului care poate comunica în timp real cu PC-ul prin intermediul USB. Sistemul poate măsura curenți între 100nA și 50 mA cu o eroare de 1 μA pentru valori de sub 200μA, cu o rată de eșantionare de 60Hz și cu o eroare de 0,1 mA pentru valori între 200μA și 50mA, cu o rată de eșantionare de 120Hz.

Mai mult, cadrul de măsurare oferă posibilitatea măsurării consumului pentru orice platformă a utilizatorului, cu condiția ca aceasta să consume curenți în domeniul de măsurare specificat.

Astfel, pe post de modul pentru determinarea parametrilor energetici ai unui task, pot fi folosite oricare alte unelte și sisteme de măsură capabile de a determina puterea consumată de dispozitivele sistemului țintă, vizate, în fiecare din stările de funcționare considerate.

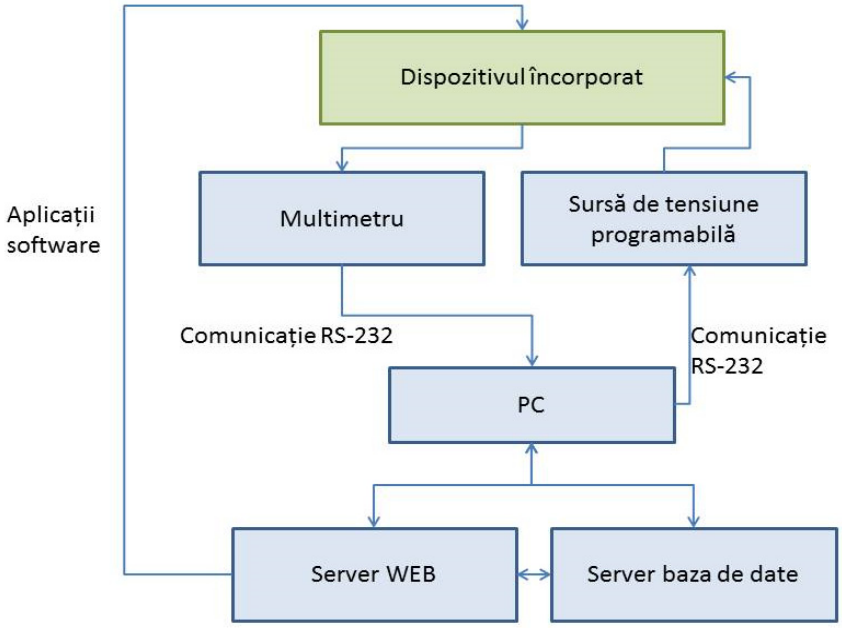


Figura 8. Cadrul de măsurare a parametrilor energetici – schema bloc

Metodologia de determinare a consumului specific fiecărei stări este:

- Se determină stările energetice care corespund cerințelor aplicației. Pentru fiecare stare determinată se reține valoarea frecvenței de lucru și se măsoară consumul, ținând celelalte dispozitive de pe placă oprite. Pentru stările cu consum relativ constant, se va reține valoare medie, iar pentru cele pentru care consumul variază, se va reține valoarea maximă.
- Se va trece procesorul într-o stare cu consum cunoscut, constant, relativ mic și se vor activa pe rând cate un dispozitiv care va fi trecut la rândul său prin toate stările corespunzătoare cerințelor aplicației.
- Pentru fiecare stare considerată se va reține frecvența de lucru a acestuia și valoarea puterii specifice acelei stări, având grijă să se extragă din valoarea măsurată acea valoare corespunzătoare stării procesorului.

Pe lângă determinarea puterii specifice fiecărei stări, trebuie determinate valorile energiei consumate în cazul tranzițiilor dintr-o stare în alta ($\eta^{k}_{j\_max}$). Această problemă este mai delicată decât prima. În primul rând puterea consumată în timpul tranziției nu este constantă, astfel energia consumată în timpul tranziției este (conform relației (3-7) unde intervalul de timp (t_2,t_1) este intervalul de timp necesar efectuării tranziției.

În figurile următoare, obținute cu ajutorul aplicației energyAware Profiler, se poate observa cum variază consumul de curent, în timpul unei tranziții dintr-o stare de funcționare predefinită în alta, pentru un procesor bazat pe arhitectura ARM-Cortex M3.

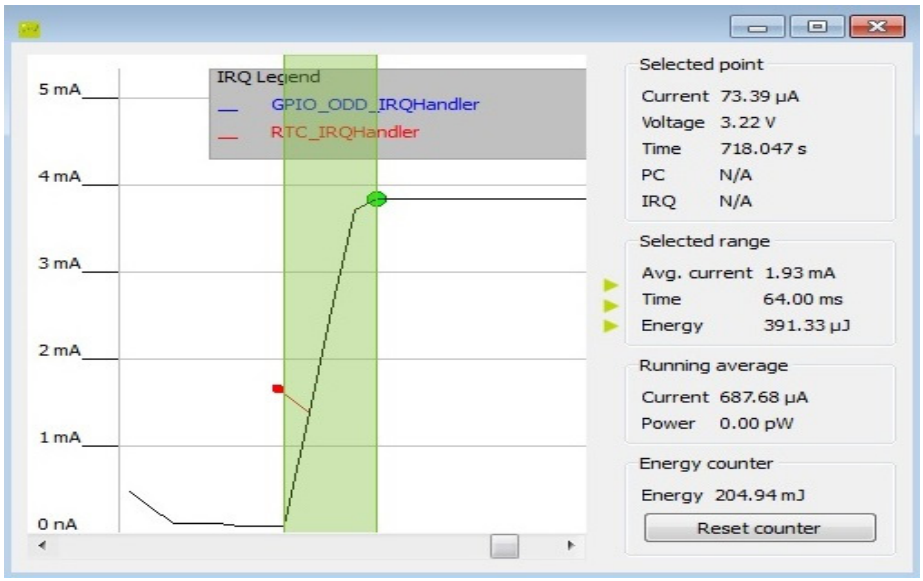


Figura 9. Tranziție din starea energetică predefinită EM3 în EM2 pentru procesorul EFM32G890F128

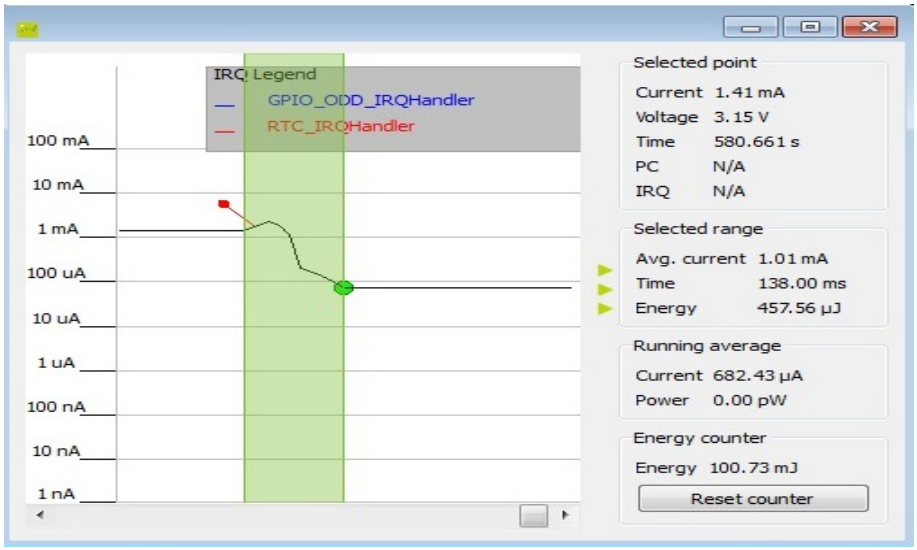


Figura 10. Tranziție din starea energetică predefinită EM1 în EM2 pentru procesorul EFM32G890F128

4.6 Modulul pentru analiza și determinarea valorilor parametrilor temporali ai taskurilor

4.6.1 Descrierea generală a modulului

Conform modelului pentru taskurile de timp real, definit în Capitolul 4.4 , parametrii de timp ce caracterizează un task sunt:

- T_i – perioada taskului
- C_i – timpul de execuție (pentru cazul cel mai defavorabil)
- D_i – termenul (*deadline*) relativ al taskului *i,*
- φ_i – timpul de start sau faza.

Ultimul parametru, timpul de start, este dat de algoritmul de planificare și se obține în ultimul pas al metodologiei, ca rezultat al planificării setului de taskuri cu un anumit algoritm.

Se pune problema determinării celorlalți parametri de timp, pentru specificarea completă a cerințelor temporale ale taskului. Aceste informații sunt indispensabile pentru analiza fezabilității și planificarea taskurilor. Modulul responsabil de determinarea acestor parametrii este o parte integrantă a sistemului cadru propus și poate fi constituită de către o unealtă software de analiză și determinare a valorilor timpilor maximi de execuție ai unui cod pe o platformă dată (unelte de analiză WCET).

Dacă perioada taskului si termenul de execuție al acestuia sunt alese de utilizator în limita condițiilor impuse de aplicație, timpul de execuție este un rezultat direct al setului de instrucțiuni și al resurselor accesate. Acesta trebuie bine determinat.

În practică, timpul de execuție pentru un anumit task, variază de la execuția unei instanțe la alta. De aceea s-a ales ca reprezentare un parametru C_i, ce corespunde timpului maxim de execuție pentru o instanță a taskul *i*, adică timpului de execuție pentru cazul cel mai defavorabil al rulării taskului *i.* Determinarea corectă a acestui timp este extrem de importantă.

Dacă se consideră un timp mai mic decât timpul maxim de execuție, se poate ajunge la depășirea termenelor limită, ceea ce, pentru un sistem critic, poate avea efecte catastrofale. Dacă în schimb, se consideră un timp considerabil mai mare decât termenul limită, aceasta poate duce la scăderea fezabilității setului de taskuri. Adică un set poate fi planificabil cu un anumit algoritm pentru un set de valori pentru $C_i = x_i$, dar poate fi neplanificabil pentru un set de valori $C_i = x_i + \zeta_i$, cu $\zeta_i > 0$.

Pentru o determinare cât mai exactă a timpului de execuție maxim, avem nevoie în primul rând, ca arhitectura sistemului pe care rulează aplicația să fie predictibilă, după cum a fost specificat în descrierea platformei țintă (v. subcapitolul 4.3.1).

În practică, există două principale metode de determinare a timpului maxim de execuție. Prima metodă este bazată pe măsurarea timpilor de execuție pentru diferite cazuri de execuție ale unui task (ex. prin marcarea momentelor de lansare în execuție și terminare a execuției taskului respectiv și calculul diferenței între acestea, sau cu ajutorul uneltelor hardware și software de tip *trace* care permit urmărirea execuției codului în timp real și măsurarea timpului necesar execuției acestuia, prin folosirea unor temporizatoare interne dispozitivului de *trace*). Exemple de astfel de unelte sunt Tanto și Tantino produse de firma Hitex.

Prin aceste metode se pot măsura valorile timpilor de execuție pentru un număr finit de cazuri. Astfel, aceste metode, numite dinamice, determină un timp maxim de

execuție observabil, care de cele mai multe ori este mai mic decât timpul maxim de execuție real, ceea ce face metoda nepotrivită pentru sistemele de timp real critice.

A doua metodă, cea statică, presupune analiza execuției taskului și limitarea superioară a acesteia (i.e. găsirea unei funcții ce limitează superior funcția ce reprezintă execuția taskului). Timpul de execuție al unui task poate fi reprezentat ca o funcție de parametrii de intrare, de numărul de iterații din interiorul fiecărei bucle de cod, de timpii de acces ai diferitelor resurse și de starea sistemului.

$$\varsigma_i(\Delta, I, \Theta, S) -> \mathbf{R} \tag{4-30}$$

unde ς reprezintă timpul de execuție pentru taskul *i*, Δ reprezintă setul datelor de intrare a taskului, reprezentate atât prin valoare cât și prin numărul lor (dacă acestea influențează timpul de execuție al codului), I reprezintă setul de instrucțiuni incluzând limitele pentru execuția buclelor, θ repezintă timpii de acces la diferite resurse, iar *S* reprezintă starea sistemului. Valorile funcției sunt numere naturale reprezentând valoarea timpului de execuție exprimată în cicluri de tact ale procesorului.

Atunci C_i va fi maximul (pentru condițiile de funcționare date ale sistemului) unei funcții ς' ce mărginește superior funcția ς. În Figura 11 se poate vedea un exemplu pentru care funcția ς reprezintă o funcție liniară ce mărginește funcția ς. Știind că numărul maxim al datelor de intrare este 50, rezultă ca funcția ς' este definită pe intervalul [0,50], atunci, se poate considera ca timp maxim de execuție maximul funcției $\varsigma(50)=250$ cicluri de clock (CC).

Astfel, în sistemele timp-real critice, metoda de determinare a parametrului C_i potrivită este cea bazată pe mărginirea superioară și maximizarea funcției timpului de execuție și nu metode bazate pe estimarea acestei valori. Un studiu de referință asupra metodelor de determinare a timpului maxim de execuție îl reprezintă lucrarea [106].

În metodologia generală de determinare a timpului maxim de execuție întâlnim următoarele etape:

- Împărțirea codului taskului pe blocuri elementare și determinarea grafului de parcurgere a acestora;
- Determinarea unei limite superioare pentru execuția fiecărui bloc (un număr maxim de câte ori se poate executa un anumit bloc într-o parcurgere a căii ce-l conține);
- Determinarea timpului maxim de execuție pentru fiecare bloc;
- Pentru procesoare mai complexe (care conțin pipeline, cache sau alte elemente nesecvențiale) determinarea dependenței de context a taskului și a instrucțiunilor în sine. Procedeu pe care nu-l vom referi în această lucrare, unde avem în vedere un sistem țintă strict predictibil.
- Determinarea timpilor de acces la resurse (ex. timpul de acces la o memorie externă pentru citirea unui cuvânt de date);
- Determinarea căii celei mai costisitoare, din punct de vedere al timpului de execuție, pentru taskul respectiv și calcularea timpului maxim de execuție.

Anumite unelte de analiză permit (chiar cer în unele cazuri) introducerea de informații suplimentare de către utilizator fie prin adnotări la cod, fie prin fișiere separate de configurare. Informațiile furnizate în acest fel sunt:

- Harta memoriei și informații despre aceasta (timpi de acces, moduri de lucru, etc.)
- Informații despre execuția codului (limite pentru numărul de execuție al unor bucle, informații despre întreruperi externe, etc.)
- Limite ale domeniilor de valori pentru diferite variabile

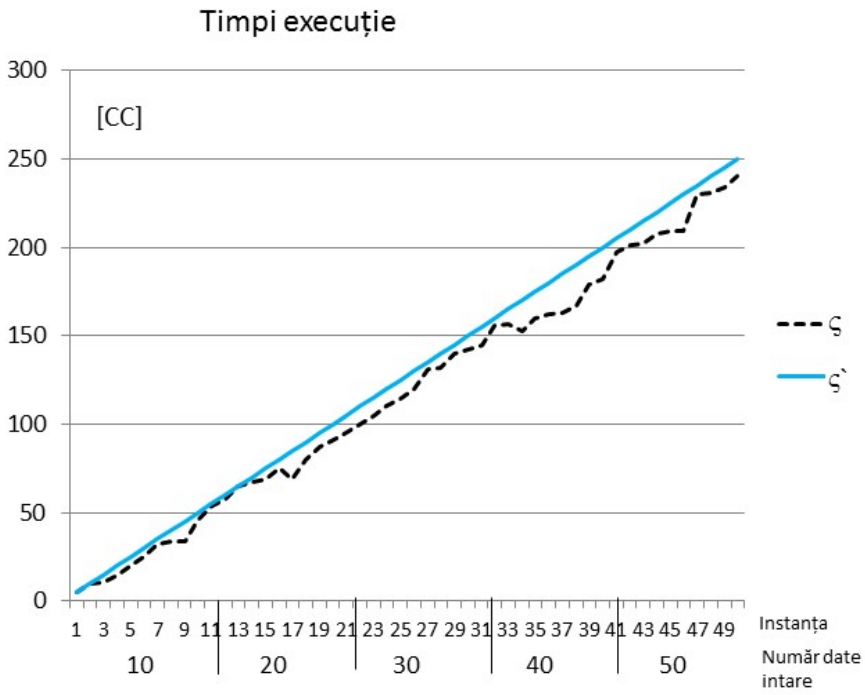


Figura 11.Timpii de execuție și aproximarea lor

Exemple de astfel de unelte bazate pe metode statice pentru determinarea timpului maxim de execuție sunt: Bound-T produs de firma Tidorum Ltd și aiT produs de firma AbsInt.

Pașii principali care trebuie urmați în determinarea timpilor de execuție sunt de obicei realizați de componente diferite ale uneltelor și sunt următorii:

1. Analiza căilor de execuție posibile – în urma acestei etape se realizează un graf al programului, unde nodurile sunt reprezentate de secvențe de cod compacte, fără salturi.
2. Analiza valorilor datelor – este utilă pentru a determina adresele de memorie accesate (de aici și timpii de acces care pot fi diferiți în funcție de ce modul de memorie e accesat), limitele superioare pentru unele bucle, etc.
3. Analiza și predicția comportării pipeline-ului – se determină timpii de execuție pentru fiecare bloc (nod din graf), pentru toate stările posibile în care se poate afla procesorul la intrarea în bloc și se calculează maximul
4. Determinarea căii cele mai defavorabile și a timpului maxim de execuție – se calculează contribuția fiecărui bloc la timpul de execuție total (produsul între costul de timp și numărul maxim de execuții) și se alege calea cea mai costisitoare din punct de vedere al timpului.

Aceste etape principale sunt reprezentate schematic în Figura 12 ce urmează, unde prin dreptunghiuri sunt reprezentați principalii pași, iar prin elipse principalele rezultate obținute în fiecare etapă.

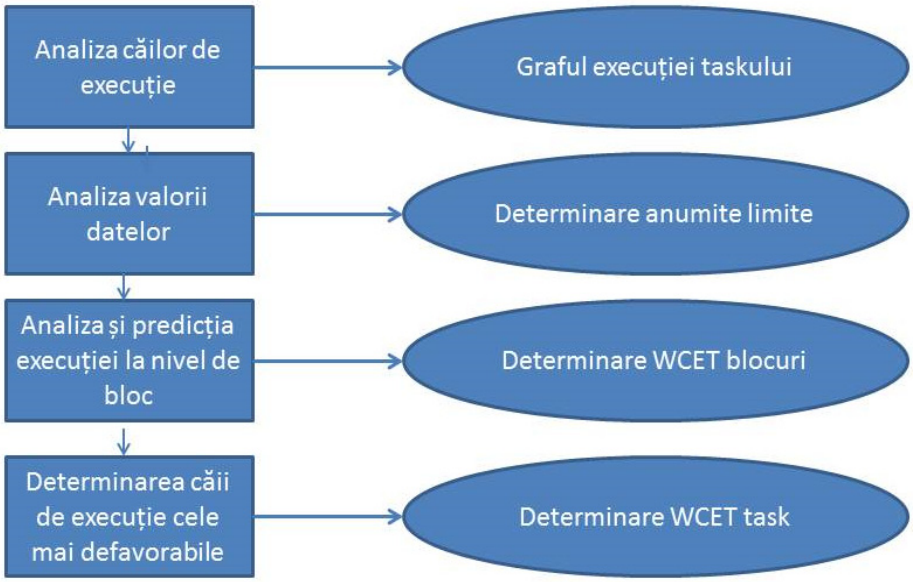


Figura 12. Determinarea timpilor de execuție ai unui task

4.6.2 Studiu de caz: AiT - AbsInt

Exemple de determinare a timpului maxim de execuție pentru diferite secvențe de cod au fost realizate cu un software al firmei AbsInt numit AiT și pot fi vizualizate în figurile următoare (Figura 13 și Figura 14), unde sunt indicate acele blocuri care fac parte din calea cea mai lungă de execuție, iar în dreptunghiul superior este calculat timpul de execuție maxim pentru secvența de cod dată.

Uneltele existente calculează timpii de execuție ținând cont doar de procesor. Modelul acestuia poate lua în considerare, în anumite cazuri și periferia, accesul la memorii externe, magistrale, dar se limitează la structura procesorului și nu și la alte module hardware ale sistemului timp-real. Astfel timpii de acces la diferite resurse externe nu sunt luați în considerare, de aici rezultă că această sarcină revine utilizatorului.

Pe de altă parte adăugarea costului datorat întreruperilor unui task trebuie făcută tot de către utilizator. Astfel determinarea timpului maxim de execuție pentru un task se va face în doi pași:

1. Determinarea timpului maxim de execuție pentru task cu ajutorul uneltelor descrise anterior (C_i^{ex}).

2. Determinarea costului de timp pentru lansarea în execuție a taskului (schimbarea de context, dacă e cazul, lansarea propriu zisă și revenirea la contextul inițial) (C_i^o).

Timpul maxim de execuție al unui task va fi:

$$C_i = C_i^{ex} + C_i^0 \tag{4-31}$$

Tot cu ajutorul acestor unelte se vor calcula și parametrii de timp ce reprezintă tranzițiile dintr-o stare energetică în alta: o_{j_max} din relația (4-24).

Pentru determinarea parametrilor, pentru fiecare stare de lucru considerată, se va determina timpul maxim de acces în acea stare din orice altă stare considerată.

Acești parametri se pot determina prin scrierea unor funcții separate ce trec procesorul sau diferitele dispozitive dintr-o stare energetică în alta și de a trece aceste funcții prin uneltele de analiză în vederea determinării timpului maxim de execuție. Un astfel de exemplu poate fi vizualizat în figura de mai jos, unde este analizată o funcție de configurare a regulatorului de tensiune pentru nucleul procesorului, în mod dinamic, pentru un procesor bazat pe arhitectura ARM Cortex-M3.

Uneori există anumiți timpi ce nu pot fi determinați direct de uneltele de analiză și trebuie specificați de către utilizator. Un astfel de exemplu este timpul de stabilizare a tactului procesor când îi este modificată frecvența. Acest timp poate fi determinat din valorile de catalog și specificat cu ajutorul sistemelor de adnotare ale uneltelor de analiză, care îl vor integra în calculul timpului total al funcției de tranziție date.

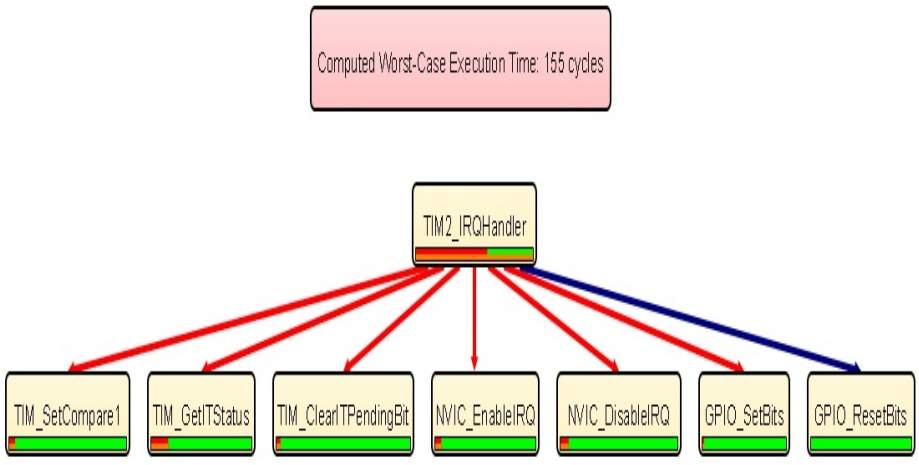


Figura 13. Exemplu de calcul al timpului de execuție

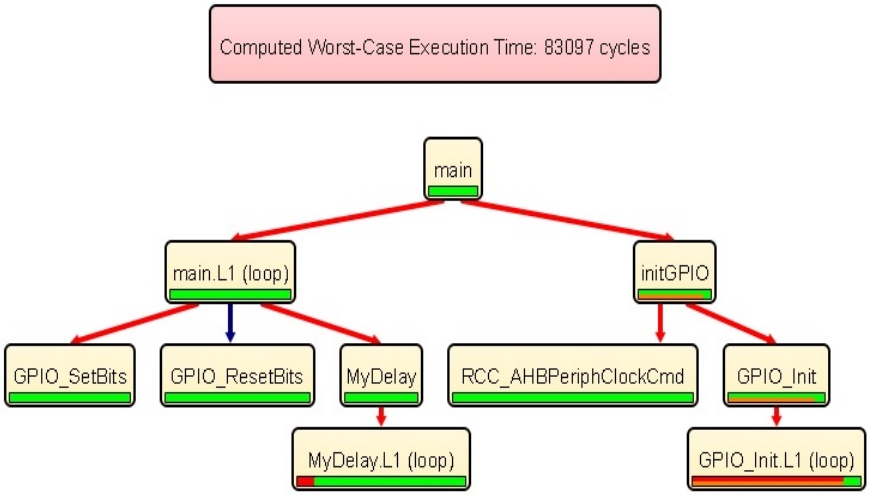


Figura 14. Exemplu de calcul al timpului de execuție

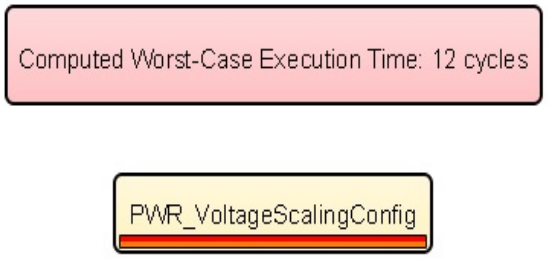


Figura 15. Timpul de execuție al unei funcții de tranziție

Exemple de măsurători pentru diferite componente de timp ale parametrului o_{j_max} discutați anterior se găsesc în tabelele următoare.

În primul tabel sunt definite o serie de stări active A*.* și o serie de stări inactive S/LP/ST/SB*.*. Pentru stările active o_{j_max} este definit ca fiind timpul de intrare în starea respectivă. Timpul de intrare poate avea mai multe componente după cum apare și în tabel. Astfel pentru intrarea într-o stare Ax.x avem timpul de setare al domeniului de tensiuni de alimentare pentru nucleul procesorului (Setare V_{core}) plus timpul de setare al frecvenței oscilatorului plus timpul de stabilizare al acestui oscilator. Observație: considerăm ca sursă de tact pentru nucleul procesorului doar oscilatorul MSI. Dacă intrarea în starea Ax.x se face dintr-o altă stare Ax.y atunci nu mai trebuie setat regulatorul de tensiune, domeniul tensiunilor de alimentare pentru nucleul procesorului fiind același, dar cum o_{j_max} reține maximul timpilor de intrare într-o anumită stare, vor fi reținute toate componentele.

Exemplu de calcul al o_{j_max} pentru starea A1.1:

$$o_{j\_\max} = SetareVcore + SetareFrecventaMSI + tstab = (12+11)CC + 2us \quad (4\text{-}32)$$

Această valoare va fi exprimată în funcție de frecvența de lucru a procesorului *f*, unde $1CC = 1/f$, relație care poate fi reprezentată și în funcție de frecvența nominală înlocuind *f* cu $\varepsilon \cdot f_{nom}$ (v. relația (4-29)).

În ceea ce privește stările inactive, vom considera tranzițiile doar din starea activă în starea *sleep* corespunzătoare din punct de vedere al frecvenței și tensiunii de alimentare, adică doar din Ax.y în starea Sx.y (de exemplu din starea A3.1 în starea S3.1). În acest caz timpul de intrare în stare S*.* este dat doar de timpul de setare al stării inactive (*Setare Sleep Mode).* Iar timpul de ieșire este dat de timpul de ieșire din starea inactivă (*twusleep).*

$$o_{j\_\max} = SetareSleepMode + twusleep \quad (4\text{-}33)$$

Tabelul 4. Stări și timpi tranziție STM32L152

Cod stare	Descriere Stare	Intrare stare	Ieșire stare
A1	Activ-Vcore Range1	Setare Vcore + Setare frecvența oscilator MSI	
A1.1	MSI 4.194 MHz (range 6)	tstab(msi)range6-v range 1	
A1.2	MSI 2.097 MHz (range 5)	tstab(msi)range5	
A1.3	MSI 1.048 MHz (range 4)	tstab(msi)range4	
A1.4	MSI 524.288 KHz (range 3)	tstab(msi)range3	
A1.5	MSI 262.144 KHz (range 2)	tstab(msi)range2	
A1.6	MSI 131.072 KHz (range 1)	tstab(msi)range1	
A2	Activ-Vcore Range2	Setare Vcore + Setare frecvența oscilator MSI	
A2.1	MSI 4.194 MHz	tstab(msi)range6-v range 2	
A2.2	MSI 2.097 MHz	tstab(msi)range5	
A2.3	MSI 1.048 MHz	tstab(msi)range4	
A2.4	MSI 524.288 KHz	tstab(msi)range3	
A2.5	MSI 262.144 KHz	tstab(msi)range2	
A2.6	MSI 131.072 KHz	tstab(msi)range1	
A3	Activ-Vcore Range3	Setare Vcore + Setare frecvența oscilator MSI	
A3.1	MSI 4.194 MHz	tstab(msi)range6-v range 3	
A3.2	MSI 2.097 MHz	tstab(msi)range5	
A3.3	MSI 1.048 MHz	tstab(msi)range4	
A3.4	MSI 524.288 KHz	tstab(msi)range3	
A3.5	MSI 262.144 KHz	tstab(msi)range2	
A3.6	MSI 131.072 KHz	tstab(msi)range1	
S1	Sleep Vcore Range 1	Setare Sleep Mode	twusleep
S1.1	MSI 4.194 MHz		twusleep

Cod stare	Descriere Stare	Intrare stare	Ieșire stare
S1.2	MSI 2.097 MHz		twusleep
S1.3	MSI 1.048 MHz		twusleep
S1.4	MSI 524.288 KHz		twusleep
S1.5	MSI 262.144 KHz		twusleep
S1.6	MSI 131.072 KHz		twusleep
S2	Sleep Vcore Range 2		twusleep
S2.1	MSI 4.194 MHz		twusleep
S2.2	MSI 2.097 MHz		twusleep
S2.3	MSI 1.048 MHz		twusleep
S2.4	MSI 524.288 KHz		twusleep
S2.5	MSI 262.144 KHz		twusleep
S2.6	MSI 131.072 KHz		twusleep
LPS1	LowPowerSleep	Setare low power sleep	twulpsleep
LPS1.1	Vcore Range 3 MSI 32KHz	Setare low power sleep + tstab(msi)range 0	twulpsleep
ST1	STOP	Setare low power sleep + tstab(msi)range 0+ Setare stop mode	twustop
SB1	STANDBY	Setare low power sleep + tstab(msi)range 0+ Setare standby mode	Tstby1

Tabelul 5. Măsurători parametri temporali

Mărime	Valoare [CC]	Mărime	Valoare [us]
Funcție setare domeniu Vcore	12	tstab(msi)range0	40
Funcție setare frecvența oscilator MSI	11	tstab(msi)range1	20
Funcție setare MSI ca sursă de tact	11	tstab(msi)range2	10
Funcție configurare regulator LPR	19	tstab(msi)range3	4
Funcție setare sleep mode	18	tstab(msi)range4	2,5
Funcție setare low power sleep	8	tstab(msi)range5	2
Funcție setare stop mode	22	tstab(msi)range6-v range 1,2	2
Funcție setare standby mode	21	tstab(msi)range6-v range 3	3
		twusleep	0,36
		twulpsleep	32
		twustop	237
		Tstby1	50

4.7 Modulul de analiză a execuției aplicațiilor timp-real

4.7.1 Descrierea generală a modulului

În cele ce urmează, va fi propus și prezentat un modul responsabil cu analiza execuției aplicațiilor timp-real ce vor rula pe platforma țintă. Modulul este reprezentat de către un mediu software ce rulează pe un sistem de calcul de uz general.

Principalele funcționalități vizate de către acest modul sunt:

- *Selectarea acelor stări de funcționare, care satisfac cerințele aplicației, pentru fiecare dispozitiv elementar din sistemul țintă.*
- *Determinarea configurațiilor active minime necesare și inactive pentru fiecare dispozitiv.*
- *Dezvoltarea modelului de taskuri pentru o aplicație dată, prin completarea valorilor parametrilor modelului propus în subcapitolul 4.4*
- *Analiza aplicației din punct de vedere al comportamentului energetic și temporal prin simularea execuției acesteia cu ajutorul diferitelor medii de execuție și planificare propuse în subcapitolul 4.8*
- *Selecția unui mediu de execuție, dintr-o serie de medii propuse, pentru aplicația dată în funcție de cerințele aplicației și rezultatele simulărilor.*

Acest modul implementează două aspecte principale ale cadrului de analiză și planificare a aplicațiilor timp-real, propus în această lucrare. Pe de o parte, implementează o componentă statică (offline) de analiză și una de preprocesare a taskurilor aplicației și a modelului acestora pentru un algoritm de planificare cu funcție de eficientizare a consumului de energie propus și implementat de către modulul de planificare a taskurilor. Astfel, cele două module (de analiză și de planificare a execuției aplicațiilor timp-real), funcționează în strânsă legătură unul cu celălalt, putând fi văzute și ca două componente ale unui singur sistem: o componentă de analiză *offline* a aplicației din punct de vedere al planificabilității și al consumului energetic și o componentă *online* reprezentând mediul de planificare și execuție al taskurilor aplicației timp real.

Cele două componente sunt reprezentate la nivel de bloc în Figura 16:

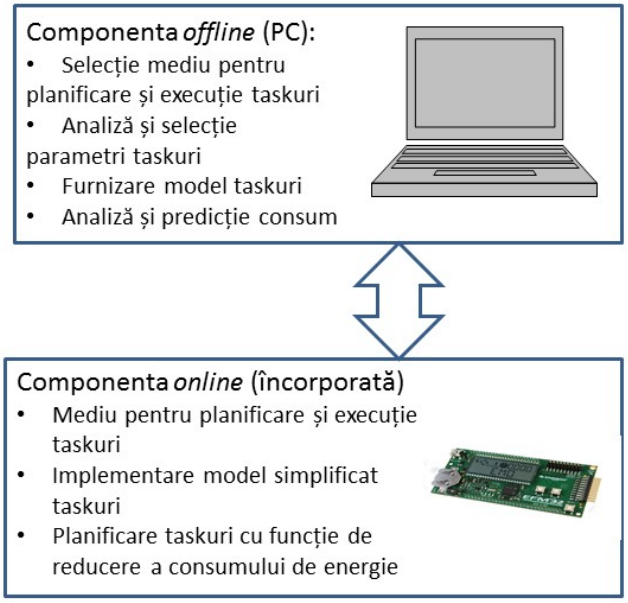


Figura 16. Componentele sistemului de analiză și planificare

Pe de altă parte, componenta *online* este reprezentată de către acele componente software care rulează direct pe platforma țintă și anume: mediul de execuție și planificare a taskurilor cu funcție de eficientizare a consumului de energie, ce cuprinde un model al taskurilor simplificat față de cel folosit pentru componenta online, așa cum va apărea în subcapitolul 5.1.2 .

4.7.2 Integrarea modulului de analiză în sistemul cadru

Modulul de analiză este o parte componentă esențială a sistemului cadru. Acesta se află într-o strânsă legătură cu principalul modul al sistemului cadru, reprezentat de către sistemul de planificare a aplicației timp real pe platforma țintă. Legătura cu acest modul, dar și cu celelalte componente din sistemul cadru, cu care comunică acest modul, poate fi vizualizată în Figura 17.

În această figură se poate observa că modulul primește ca parametru de intrare modelul taskurilor așa cum a fost prezentat în subcapitolul 4.4.2 , pe care îl prelucrează și îl trimite modulului de planificare.

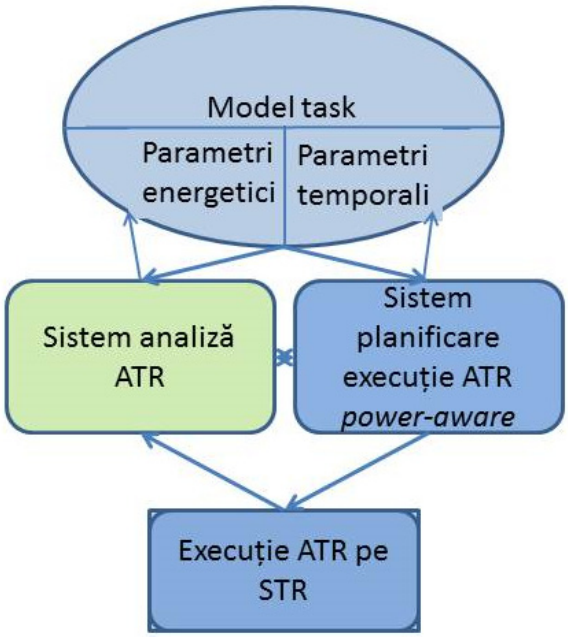


Figura 17. Integrarea modulului de analiză în sistemul cadru

Unul din rolurile acestui modul este acela de a particulariza și simplifica modelul, astfel încât să obținem un model ce poate fi reprezentat pe platforma țintă, având în vedere că aceasta este reprezentată de către o platformă încorporată cu resurse limitate.

Pe de altă parte, sistemul de analiză va propune o serie de medii de execuție și planificare a taskurilor aplicației, ce vor rula pe platforma țintă. Utilizatorul va alege mediul dorit și în mod implicit și algoritmul de planificare aferent acestui mediu, în funcție de specificul aplicației și de platforma pe care aceasta urmează să ruleze.

Componenta statică, apoi, va realiza o analiză de planificabilitate a setului de taskuri conform algoritmului de planificare ales, și o analiză a parametrilor dispozitivelor conform ecuațiilor (4-16),(4-20) și (4-21), pentru a determina valorile tuturor parametrilor cuprinși în aceste ecuații. Se are în vedere faptul că resursele de memorie, computaționale și temporale ale sistemului țintă sunt limitate. De aici rezultă că atât algoritmul ce va rula pe această platformă cât și parametrii conținuți în sistemul țintă trebuie să fie simplificați cât mai mult cu putință. Astfel, datele trebuie preprocesate printr-un algoritm static, ce va rula pe o platformă auxiliară mult mai complexă, reprezentată de un sistem compus dintr-un PC și un mediu software de analiză.

În continuare va fi descrisă metodologia de lucru. Aceasta fiind reprezentă grafic în Figura 18.

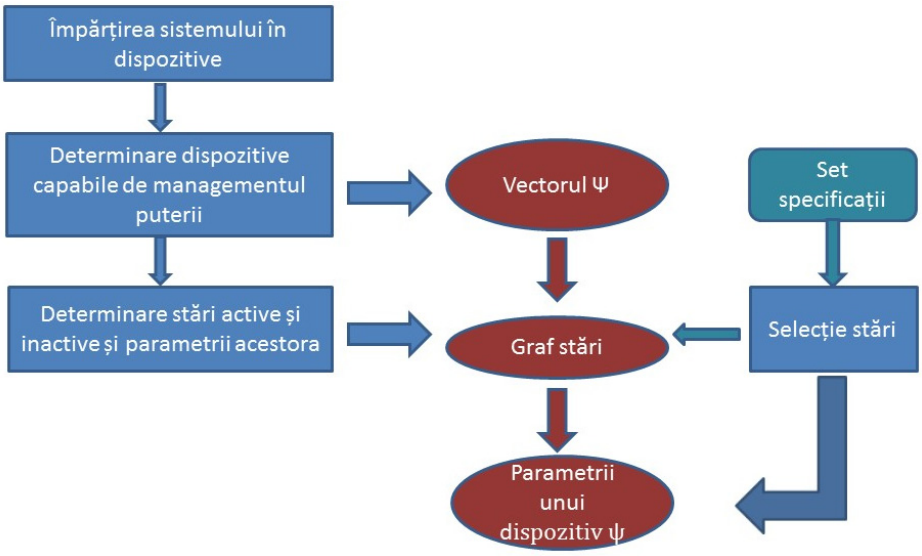


Figura 18. Metodologia de determinare a parametrilor modelului taskului

Modulul de analiză primește vectorul de *dispozitive elementare*, o serie de stări definite prin parametrii necesari pentru calculul energiei consumate în această stare, dați de formula (4-25) și prin valorile timpilor de tranzițiile între aceste stări și a surplusului de energie datorat acestor tranziții.

Dintre aceste stări se vor elimina acelea care nu fac față cerințelor aplicației, astfel:

1. Pentru fiecare dispozitiv se vor sorta stările în ordine crescătoare a performanței lor (în practică s-a observat că această sortare corespunde și cu sortarea crescătoare a puterii lor specifice).
2. Se va determina o frecvență de lucru minimă pentru procesor pe baza specificațiilor de timp ale aplicației și a algoritmului de planificare ales. Această frecvență minimă va determina starea S^A pentru procesor. Aceasta va fi starea de funcționare a procesorului (sau a nucleului acestuia) cu frecvența de lucru mai mare sau egală cu frecvența determinată teoretic în acest pas.
3. Se va parcurge fiecare task în parte și se va specifica starea activă minimă pentru fiecare dispozitiv elementar utilizat de taskul respectiv. Dintre aceste stări se va alege starea maximă, care va reprezenta starea S^A pentru acel dispozitiv, corespunzătoare *configurației active specifice* procesorului. Pe lângă aceste taskuri se va lua în considerare și unul fictiv reprezentat de sistemul de operare, astfel încât și cerințele de funcționare ale acestuia să fie luate în vedere.
4. La parcurgerea fiecărui task se vor elimina stările inactive pentru care timpul de activare depășește o valoarea limită Ω, specifică algoritmului de planificare, astfel selectându-se *configurațiile inactive* corespunzătoare fiecărui *dispozitiv elementar*.

Pe baza informațiilor primite anterior se vor alcătui *m* grafuri de tranziții, câte unul pentru fiecare dispozitiv, în care fiecare nod va reprezenta o stare de funcționare conform formulei (4-25) și fiecare arc va reprezenta o potențială tranziție între două stări, tranziție modelată de:

$$A_j^{k,k+i} = \{\eta_{k,k+i}, L_{k,k+i}, t_{k,k+i}\} \tag{4-34}$$

unde $A_j^{k,k+i}$ reprezintă tranziția din starea S_j^k în starea S_j^{k+i}, $\eta_{k,k+i}$ reprezintă energia consumată datorită tranziției, $L_{k,k+i}$ reprezintă limita de rentabilitate a tranziției (intervalul minim de timp cât dispozitivul trebuie să stea în starea *k+i,* pentru ca tranziția să ofere o reducere de energie), iar $t_{k,k+i}$ reprezintă timpul necesar tranziției.

De asemenea între grafuri vor exista arce, care arată că un dispozitiv aflat în starea *k* va putea funcționa cu alt dispozitiv aflat în starea *k+i.*

Aceste arce sunt de forma:

$$A_{j,j+l}^{k,k+i} \tag{4-35}$$

unde $A^{k,k+i}{}_{j,j+l}$ reprezintă conexiunea dispozitivului *j* din starea S_j^k cu dispozitivul *j+l* din starea $S^{k+i}{}_{j+l}$.

În Figura 19 se poate vedea un exemplu de astfel de graf.

În această figură, există trei dispozitive elementare: ψ_0, ψ_1 și ψ_2. Astfel, vectorul de dispozitive elementare devine $\Psi \equiv \{\psi_0, \psi_1, \psi_2\}$.

Primul dispozitiv are mai multe stări de funcționare active; astfel se va determina o frecvență de funcționare (ex. 32MHx) și pe baza acesteia se va alege starea activă: S_0^1*={19mW,32MHz}.* Dispozitivul se poate afla și într-o serie de alte stări (S_0^2- S_0^3), cu un consum diferit energie.

Al doilea dispozitiv are patru stări de funcționare S_1^1- S_1^4. Tranzițiile posibile dintre stările unui dispozitiv sunt reprezentate de săgeți neîntrerupte, iar arcele date de ecuația (4-35), reprezentând arce între stările compatibile ale unor dispozitive diferite, sunt reprezentate de săgeți cu linii întrerupte.

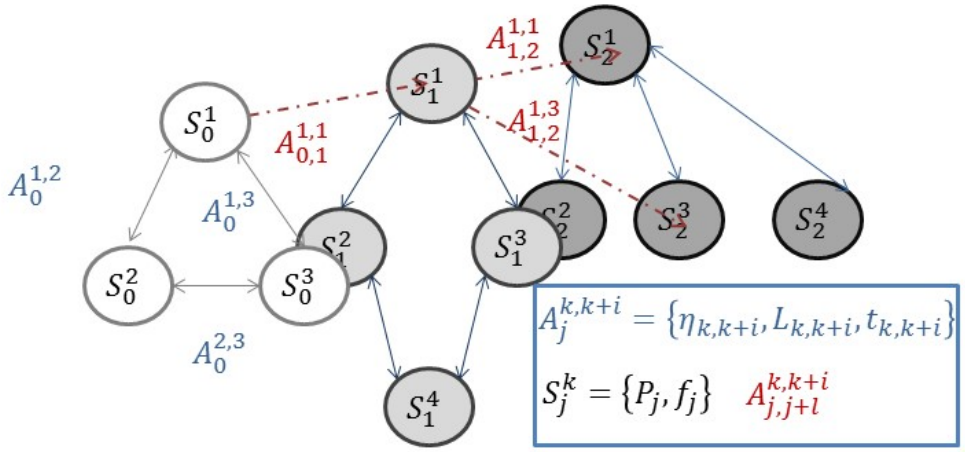


Figura 19. Exemplu de graf de stări pentru trei dispozitive

Printr-un algoritm de căutare, se va determina pentru fiecare task o stare de funcționare activă și o stare de repaus minimă corespunzând fiecărui dispozitiv.

S^A_j va fi starea maximă dintre stările active pentru dispozitivul *j* selectate pe baza cerințelor fiecărui task. Această stare va corespunde *configurației active specifice* definită în (4-23). Analog, starea de repaos S^R_j va fi starea maximă dintre stările inactive. În urma acestui pas, pentru fiecare dispozitiv se vor considera doar aceste două stări. Astfel, parametrii reprezentând surplusul de timp și energie datorat tranzițiilor, vor fi reprezentați doar de tranzițiile din starea S^A_j și S^R_j și invers (4-24).

Acești parametri vor fi transmiși modulului de planificare a aplicației timp-real, care este implementată direct pe platforma țintă.

4.8 Modulul de planificare a execuției aplicațiilor timp-real

4.8.1 Descrierea generală a modulului

Modulul de planificare a execuției aplicațiilor timp-real este reprezentat de către un mediu suport pentru planificarea și execuția acestor aplicații și de către un algoritm de planificare a taskurilor aplicației, cu funcție de eficientizare a consumului de energie electrică al platformei țintă, datorat rulării aplicației vizate.

Acest modul primește ca date de intrare o opțiune de selecție a unui mediu de planificare și execuție, din mai multe variante posibile, un model al taskurilor simplificat corespunzător aplicației și mediului ales, oferite de către modulul de analiză a aplicației.

Modulul este implementat direct pe platforma țintă și cunoaște mai multe variante, în funcție de platforma țintă și mediul suport ales. Peste acest mediu de execuție și planificare clasică a aplicațiilor timp-real critice va rula un mecanism de planificare cu funcție de eficientizare a consumului, prezentat pe larg în capitolul 5 .

În continuare vor fi propuse trei variante de medii de execuție și planificare a taskurilor timp real, ce au fost implementate pe procesoare bazate pe arhitecturi ARM 7 și ARM CortexM3.

4.8.2 Mediul suport pentru execuția si planificarea task-urilor timp-real

În continuare sunt propuse diferite variante pentru mediul suport al execuției si planificării taskurilor unei aplicații timp real critice.

Un astfel de mediu este oferit de către un nucleu de sistem de operare numit HARETICK, prezentat mai pe larg în [107, 108]. Avantajele oferite de către acest mediu sunt:

- *Predictibilitate*: În primul rând HARETICK, ca urmare a faptului că este un sistem dedicat aplicațiilor critice, oferă o serie de condiții de planificabilitate pentru taskurile unei aplicații, care atâta timp cât sunt îndeplinite garantează că toate condițiile de timp vor fi îndeplinite. În plus, mediul de planificare și execuție oferă un grad sporit de predictibilitate, în sensul că toate deciziile de planificare și momentele de lansare în execuție ale unui task pot fi determinate în avans și sunt cunoscute în momentul

rulării. Mai mult, sistemul oferă suport pentru planificarea și execuția perfect periodică a instanțelor fiecărui task.

- *Determinism*: Prin faptul că au fost eliminate orice alte surse de întrerupere în afară de cea de ceas, practic s-au redus principalele surse de nedeterminism din sistem. În plus, metodele de implementare ale sistemului, bazate în special pe alocare statică, pe execuție de cod perfect predictibilă și pe execuția taskurilor într-o manieră non-preemptivă contribuie la menținerea determinismului sistemului. Astfel, dacă și suportul hardware este unul determinist, putem spune că întreg sistemul suport pentru execuția aplicațiilor timp-real este unul determinist.

Cu toate acestea, HARETICK prezintă și unele dezavantaje: o rată de planificabilitate scăzută, o utilizare procesor scăzută și inflexibilitate.

Astfel, în continuare, vor fi prezentate trei variante de medii de execuție pentru aplicații timp real, bazate pe HARETICK:

1. HARETICK configurat cu un algoritm de planificare static non-preemptiv – folosit ca mediu unic și independent
2. HARETICK extins cu un algoritm de planificare hibrid – extins față de prima variantă prin adăugarea unui mediu de planificare și execuție mai flexibil pentru taskurile timp-real critice care nu necesită o execuție perfect periodică sau pentru alte tipuri de taskuri care nu sunt critice
3. HARETICK folosit ca o extensie peste un sistem de operare timp-real existent.

Ultimele două soluții sunt originale. Acestea presupun combinarea mediului HARETICK cu alte două medii dintre care un sistem de operare timp-real consacrat, obținând medii hibride de planificare și execuție pentru taskuri, care pe de o parte se folosesc de avantajele oferite de un mediu de execuție neîntreruptibil, determinist, ca HARETICK, pe de altă parte, se folosesc de avantajele altor medii mai dinamice, flexibile, cu un grad de încărcare procesor mai mare.

4.8.2.1 HARETICK-FENP

În acest paragraf este prezentat pe scurt un mediu de planificare și execuție, reprezentând o parte esențială a unui sistemului de operare HARETICK [107, 108].

HARETICK este un nucleu de sistem de operare timp-real, dezvoltat în cadrul laboratorului DSPLabs, al Universității Politehnica Timișoara.

Acest sistem are la bază o sursă de reprezentare și măsurare a timpului sistem numită ceas timp-real RTC (*Real Time Clock*), care este în implementarea inițială a sistemului singura sursă de întrerupere acceptată. Detecția oricărui alt eveniment a fost menită, a se face doar prin mecanisme de tip *polling*. Astfel, elementul central al acestui sistem îl reprezintă RTC, atât planificarea cât și lansarea în execuție a fiecărui task fiind bazată pe acesta.

Sistemul HARETICK este unul modular. Principalele lui componente fiind:

- BOOT - un modul responsabil cu încărcarea HARETICK-ului pe sistemul țintă și pornirea acestuia.
- SYSINIT – un modul responsabil cu inițializarea sistemului, configurarea modulelor fizice folosite de către sistemul de operare (ex. configurare surse de ceas și frecvențe de lucru pentru procesor și periferice, etc.), configurarea parametrilor sistemului de operare.
- LOADER - un modul responsabil cu încărcarea taskurilor aplicației ce va rula pe sistem prin setarea parametrilor și referirea secvențelor de cod ale taskurilor și încărcarea acestora în structurile sistemului de operare.

- HSCD/PREHSCD – un modul responsabil cu planificarea taskurilor aplicației. Rolul său principal este cel de a încărca date într-o o tabelă (numită tabela HDIS) folosită de către modulul HDIS pentru a lansa în execuție câte un task. Principalele date conținute în tabela HDIS sunt identificatorii taskurilor și valorile de timp ale momentelor lor de lansare în execuție. PREHSCD este modulul responsabil cu încărcarea primului set de date. HSCD va determina și va încărca în mod periodic aceste date în tabelă. Modulul este văzut ca un task critic, periodic (*ModX*).
- HDIS – un modul responsabil cu lansarea în execuție a câte unui task. Este activat de către întreruperea de ceas timp-real și este implementat ca o rutină de tratare a acestei întreruperi. Modulul este format din alte două sub module: HDIS_PRE ce rulează înainte de lansarea fiecărui task și HDIS_SUF ce rulează după execuția fiecărui task, durata de execuție a celor două componente fiind adăugată la durata de execuție a codului unui task și fiind astfel inclusă în valoare WCET-ului taskului respectiv.

Funcționarea sistemului poate fi descrisă pe scurt în felul următor: pornirea sistemului începe prin execuția modului BOOT care încarcă efectiv nucleul HARETICK și îl lansează în execuție. Această execuție începe efectiv cu modulul SYSINIT, care configurează platforma fizică pentru condițiile de operare ale HARETICK-ului, execuție continuată prin încărcarea referinței codului și a parametrilor taskurilor într-o tabelă a descriptorilor numită PDT (*Process Description Table*). Execuția continuă cu lansarea modulului PREHSCD care determină și încarcă în tabela HDIS indicatorii taskurilor și momentele de lansare în execuție ale acestora referite de ceasul timp-real. Datele în tabelă aflându-se în permanență ordonate după valorile momentelor de lansare. Această secvență de execuție este reprezentată în Figura 20.

După ce modulul PREHSCD a terminat de completat tabela HDIS, acestea setează primul moment de întrerupere pentru lansarea primului task. Aceste taskuri periodice care sunt planificate și rulate cu ajutorul nucleului HARETICK se numesc ModX-uri (Module executabile). Reactualizarea tabelei HDIS se face în mod periodic de către modulul HSCD, lansat ca un ModX. Lansarea unui ModX în execuție se face în felul următor: când se ajunge cu timpul de execuție la valoarea următorului eveniment de lansare al unui ModX se produce o cerere de întrerupere de către RTC, aceasta este deservită imediat de către rutina de deservire corespunzătoare, rutină care implementează modulul HDIS. La începutul rutinei este lansat modulul HDIS_PRE, responsabil cu selectarea taskului care trebuie lansat în execuție și cu lansarea acestuia, după ce execuția taskului s-a finalizat rulează modulul HDIS_SUF, responsabil cu citirea următorului moment de întrerupere și setarea următoarei întreruperi, sau cu chemarea prefixului direct, dacă intervalul între cele două lansări este prea scurt. Un exemplu de succesiune a modulelor HDIS, HSCD și a două taskuri μ_1 și μ_2 este prezentat în Figura 21. În această figură avem reprezentată succesiunea execuției a două ModX-uri normale și a ModX-ului HSCD, fiecare execuție fiind precedată de către HDIS_PRE și încheiată cu HDIS_SUF.

Algoritmul ales ca suport pentru planificarea taskurilor în sistemul propus în acest capitol este numit FENP (*Fixed Execution Non-Preemptive*).

Algoritmul de planificare FENP este unul ciclic, *non-preemptiv*, care planifică taskurile de timp real critice astfel încât între două instanțe consecutive ale aceluiași task să fie în permanență o distanță temporală contantă egală cu perioada taskului (v. Figura 22).

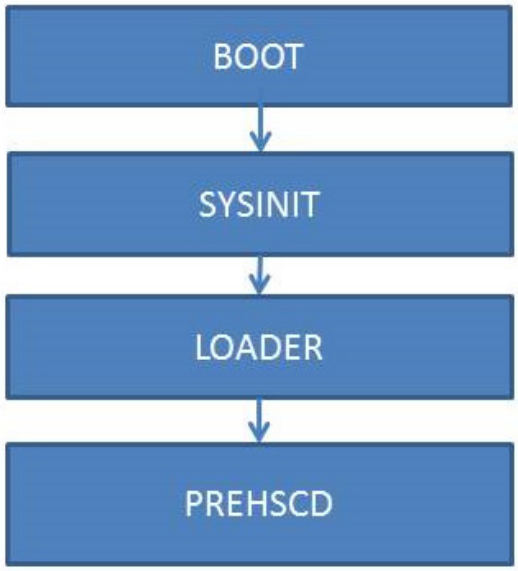


Figura 20. Modulele HARETICK

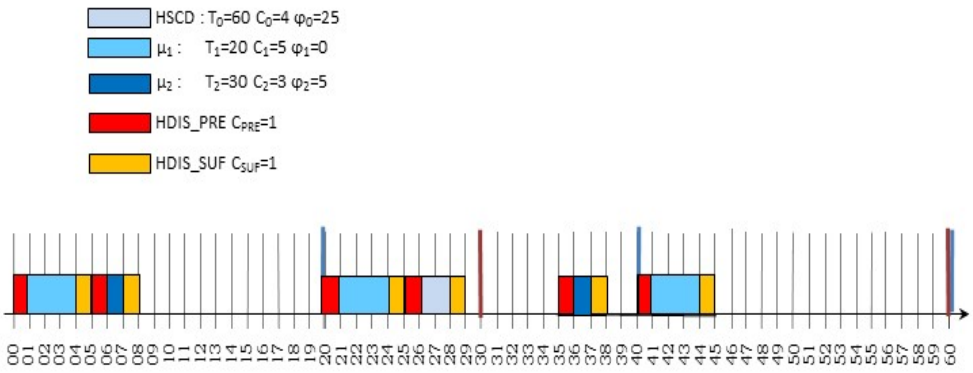


Figura 21. Exemplu de rulare de ModX-uri

$$\begin{cases} \boldsymbol{S}^{FENP} \equiv \{\mu_1, \ldots, \mu_i, \mu_{i+1}, \ldots, \mu_m\} \\ \mu_i \equiv (T_i, C_i, \varphi_i) \ . \ i = 1..m \\ T_i \le T_{i+1} \\ s_{i,k+1} = s_{i,k} + T_i = \varphi_i + kT_i \ , \forall \mu_i \in \boldsymbol{S}^{FENP}, k \in \mathbf{N} \end{cases} \quad (4\text{-}36)$$

Formula reprezentând mecanismul de planificare al taskurilor FENP, S^{FENP} numite ModX-uri și notate cu μ_i. Acestea sunt reprezentate de parametrii temporali, perioadă T_i, timp de execuție C_i și timp de start în cadrul perioadei sau fază φ_i. ModX-urile sunt ordonate în ordine descrescătoare a perioadei lor.

Momentul de start al unei instanțe *k+1* a taskului *i* poate fi calculat prin adunarea unei perioade a acestui ModX la momentul de start anterior *k.*

Avantajele oferite de către acest algoritm și a mediului de planificare și execuție care îl implementează sunt:

- *Analiza fezabilității*: algoritmul oferă posibilitatea unei analize statice, care să determine dacă un set de taskuri este planificabil, printr-un set de condiții de necesitate și un set de condiții de suficiență.
- *Predictibilitatea*: taskurile planificate și executate în acest mediu oferă cel mai mare grad de predictibilitate posibil, pe lângă faptul garanției respectării termenelor de timp pentru terminarea execuției taskurilor, acest mecanism oferă garanția respectării momentelor de timp pentru lansarea în execuție a fiecărei instanțe a unui task, calculate în prealabil.
- Execuția taskurilor într-o manieră *non-preemptivă* oferă un plus de determinism și evită o serie de probleme legate de partajarea resurselor.

Cu toate acestea, algoritmul prezintă și o serie de dezavantaje, dintre care menționăm o rată foarte scăzută de succes în ceea ce privește planificabilitatea unui set de taskuri. Această rată de succes este sub 10% pentru seturi de taskuri cu utilizare procesor peste 50% [7]

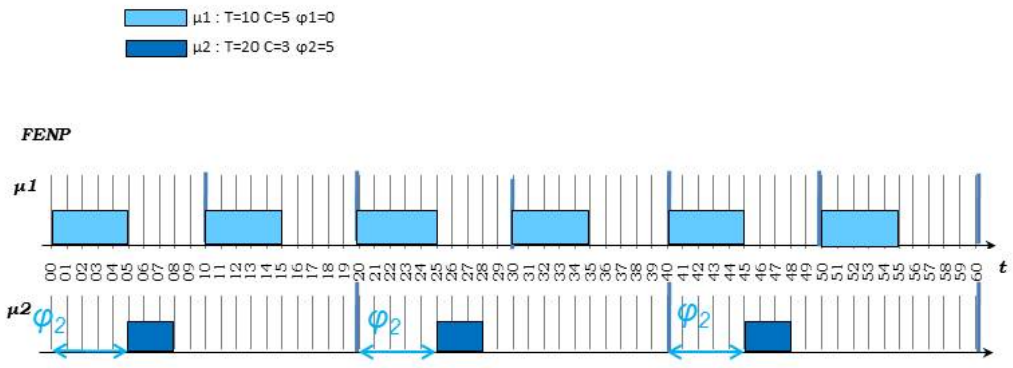


Figura 22. FENP

4.8.2.2 H²RTS

În acest paragraf este prezentată a doua propunere de mediu de planificare și execuție a taskurilor timp-real critice și anume un sistem hibrid, care combină cu succes tehnicile de planificare statice cu cele dinamice. Algoritmul a fost implementat pe o platformă bazată pe arhitectura ARM-7 și publicat cu numele de H²RTS (*Hybrid Hard Real-Time Scheduler)* [109].

După cum s-a menționat anterior, tehnicile de planificare statice, neîntreruptibile, ca FENP oferă o serie de avantaje pentru taskurile timp-real critice, care necesită o interacțiune perfect sincronă cu mediul înconjurător. Aceasta este oferită în primul rând de capabilitatea acestei tehnici de a planifica taskuri fără variațiuni între momentele de lansare în execuție a instanțelor unui task calculate de la începutul perioadei sale (fără *jitter* de execuție). Dar după cum am menționat deja, acest tip de mecanisme de planificare au și importante dezavantaje dintre care menționăm: o rată relativ mică de fezabilitate, o utilizare procesor maximă mult mai redusă decât a mecanismelor de planificare dinamice, *preemptive*, o flexibilitate mică (în sensul că dacă un parametru de timp al unui task se modifică trebuie realizată o reanaliză și o replanificare a întregii aplicații).

În continuare propunem un mecanism hibrid care să îmbunătățească rata de planificabilitate a aplicațiilor timp-real și să ofere un plus de flexibilitate fără a pierde din predictibilitate. După cum s-a arătat prin rezultatele experimentale publicate în [109], acest algoritm aduce o îmbunătățire substanțială a ratei de planificabilitate (între 5 și 55%), fără a altera gradul de predictibilitate al sistemului.

Ideea de bază a algoritmului implică împărțirea unui set de taskuri S în două subseturi executate în contexte de lucru diferite:

- Primul este numit contextul FENP și este destinat execuției taskurilor critice care necesită o execuție perfect periodică în cadrul perioadei lor. Aceste taskuri vor fi executate într-o manieră *non-preemptivă* având prioritate maximă.
- Al doilea context este numit MEDF (fiind un context bazat pe EDF modificat). Acesta planifică și execută celelalte task-uri timp real critice, după cum vom vedea în continuare.

În sistem există și un al treilea context de execuție, numit BGND (de la Background în limba engleză), de data aceasta pentru taskuri care nu sunt timp-real.

Un exemplu de funcționare al celor trei medii de execuție poate fi văzut în Figura 23. Aici două ModX-uri, executate în contextul FENP, *μ1* și *μ2* întrerup două taskuri executate în context MEDF, la rândul său taskul *ε1* întrerupe contextul BGND.

După cum a fost deja menționat în subcapitolul 4.8.2.1. Algoritmul de planificare FENP este unul ciclic, *non-preemptiv*, care planifică taskurile de timp real critice astfel încât între două instanțe consecutive ale aceluiași task să fie în permanență o distanță temporală contantă egală cu perioada taskului.

Algoritmul MEDF reprezintă o versiune modificată a algoritmului clasic de planificare EDF. Acesta va alege în fiecare moment al planificării acel task care are cel mai apropiat *deadline* și care nu a fost deja executat în perioada sa curentă. Execuția unui task rulat în contextul MEDF poate fi întreruptă doar de unul sau mai multe taskuri din mediul FENP, niciun task din contextul MEDF neavând capabilitatea de a întrerupe alte taskuri din MEDF sau FENP. Această abordare evită problemele legate de schimbările complexe între contextele de execuție.

Modelul matematic al taskurilor și al algoritmilor de planificare pentru mediul de planificare și execuție H²RTS este următorul:

$$\begin{cases} S \equiv \{S^{FENP}, S^{MEDF}\} \\ un\ sistem\ de\ N = n + m\ taskuri \end{cases} \quad (4\text{-}37)$$

Formula anterioară reprezintă sistemul de taskuri ce conține atât taskurile FENP, S^{FENP}, cât și taskurile MEDF S^{MEDF}

Formula matematică de planificare a taskurilor FENP a fost deja prezentată în subcapitolul 4.8.2.1.

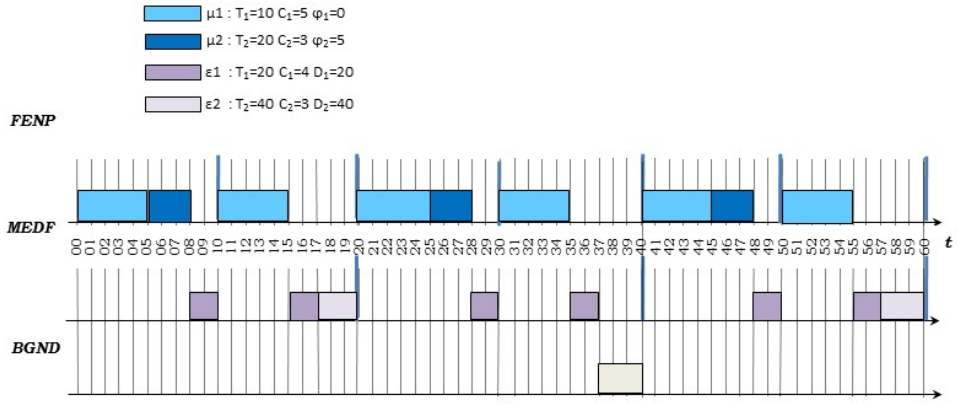


Figura 23. Cele trei contexte de execuție propuse

Formula matematică reprezentând mecanismul de planificare al taskurilor MEDF este următoarea:

$$\begin{cases} \boldsymbol{S}^{MEDF} \equiv \{\varepsilon_1, \ldots, \varepsilon_j, \varepsilon_{j+1}, \ldots, \varepsilon_n\} \\ \varepsilon_j \equiv (T_j, C_j, D_j)\ .\ D_j \le T_j,\ j = 1..n \\ T_j \le T_{j+1} \\ s_{j,k+1} = \max(t, kT_j), k \in \mathbf{N} \quad \text{and} \\ \qquad j \,|\, \forall i \ne j, (kT_j + D_j - t) \le (kT_i + D_i - t) \end{cases} \quad (4\text{-}38)$$

Taskurile MEDF, S^{MEDF} sunt notate cu ε_j. Acestea sunt reprezentate de parametrii temporali, perioadă T_j, timp de execuție C_j și deadline D_j. Timpul de start al unui task este acel moment de planificare t pentru care taskul nu a fost planificat în perioada sa curentă și pentru care taskul j are *deadline*-ul cel mai apropiat. În lipsa prezenței vreunui task FENP în sistem, algoritmul MEDF se comportă identic cu algoritmul EDFNP (*Earliest Deadline First Non-Preemptive*) (v. Figura 24).

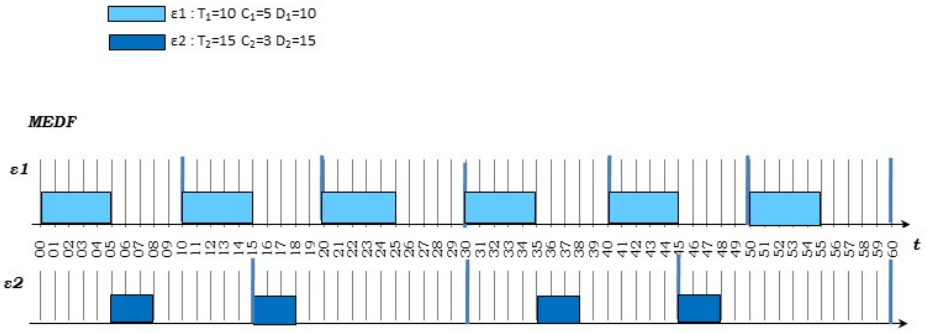


Figura 24. MEDF

4.8.2.3 Hybrid RTOS

O a treia variantă de mediu de execuție și planificare este reprezentată de către un sistem de operare timp-real în care HARETICK se integrează ca o extensie a mediului de execuție formând un nou sistem hibrid.

Ca mediu suport pentru integrarea HARETICK a fost ales un sistem de operare timp-real consacrat numit FreeRTOS[110]. Acesta este produs de către Real Time Engineers Ltd.. Este un sistem mic, bine documentat, a cărui surse (scrise în cea mai mare parte în C) sunt disponibile în mod gratuit, implementat pe peste 30 de arhitecturi hardware diferite. Toate acestea, fac ca acest sistem să fie unul popular, fiind totodată ușor de utilizat.

Acest sistem de operare a fost ales ca suport pentru cea de a treia variantă de mediu de execuție și planificare, în primul rând pentru că îndeplinește o serie de condiții, impuse pe de o parte particularităților hardware ale sistemului țintă, pe de alta cerințelor de operare ale aplicațiilor timp-real vizate, și anume:

- *Dimensiunea mică* - nucleul sistemului de operare trebuie să aibă o dimensiune redusă, datorată faptului că în cazul dispozitivelor încorporate în general și în particular, a nodurilor de senzori, se folosesc memorii de dimensiuni reduse. FreeRTOS îndeplinește această cerință, consumând între 5 și 10 kB de memorie ROM, iar memoria RAM necesară începe de la câțiva kB și se extinde, în funcție de cerințele aplicației care rulează pe acest sistem.
- *Sistem predictibil* – sistemul trebuie să fie unul de timp-real, astfel să permită specificarea parametrilor temporali esențiali pentru un task de timp-real și anume specificarea cel puțin a unuia din parametrii perioadă și *deadline*, pe care să-i respecte în procesul de execuție al taskurilor. Sistemul vizat permite specificarea perioadei sau a timpului minim între două instanțe ale aceluiași task.
- *Timpul sistem* - trebuie să existe un mod de reprezentare a timpului sistem, la care să aibă acces programatorul/dezvoltatorul sistemului de operare. În sistemele de timp real, timpul fiind o coordonată esențială pe toată durata funcționării sistemului.
- *Metoda de planificare* – sistemul trebuie să ofere o metodă de planificare pentru taskurile timp-real, care să respecte condițiile de timp ale aplicației. Se preferă o metodă complementară celei folosite de către HARETICK, și anume

ca metoda să fie *preemptivă*, cu un grad mai mare de flexibilitate și grad mai mare de utilizare procesor, față de FENP. FreeRTOS folosește un mecanism de planificare *preemptiv*, bazat pe priorități, implementat pe baza întreruperii de ceas ale sistemului.

- *Managementul consumului electric* – sistemul trebuie să fie capabil să implementeze sau să susțină implementarea unui mecanism de management al consumului.

Pe lângă aceste cerințe s-a avut în vedere în primul rând arhitecturile pe care a fost implementat sistemul: faptul că a fost implementat pe arhitecturi deterministe, capabile de management al consumului electric având o însemnătate deosebită în alegerea acestui sistem. În plus faptul că este ușor portabil și ușor de utilizat, fac din FreeRTOS să fie candidatul principal pentru baza dezvoltării mediului propus.

Sistemul propus este un hibrid (numit Hybrid RTOS), bazat pe extensia unui sistem de operare deja existent, în cazul nostru FreeRTOS, prin adăugarea unui nou context de planificare și execuție a taskurilor bazat pe HARETICK, independent și izolat (la nivel de resurse) de sistemul de operare inițial.

Sistemul este prezentat la nivel de bloc în Figura 25.

Fiecare mediu de execuție are planificatorul său și modulul său pentru lansarea în execuție a taskurilor. Fiecare mediu are sursa sa de timp, așa cum se poate vedea în figură. Cele două medii de execuție sunt practic izolate.

În Hybrid RTOS, extensia HARETICK reprezintă un mediu de execuție non-*preemptiv*, prioritar față de sistemul de operare inițial.

Acest sistem a fost implementat pe un procesor EFM32G890 și EFM32GG990 bazate pe o arhitectură ARM-Cortex M3

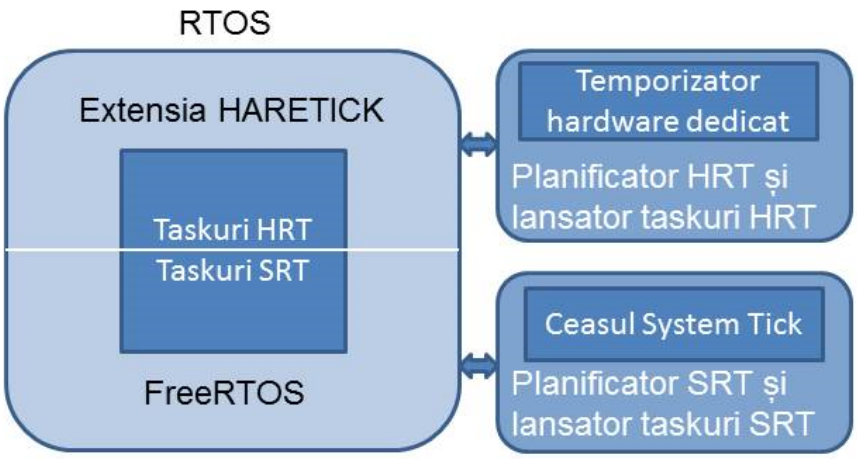


Figura 25. Integrare HARETICK în FreeRTOS

4.9 Concluzii

Acest capitol propune un sistem cadru pentru analiza comportamentului energetic și temporal și pentru planificarea și execuția aplicațiilor timp-real (critice), cu funcție de eficientizare a consumului de energie.

Acest sistem este numit TEEARTS și este conceput ca un sistem modular, unde fiecare componentă îndeplinește una sau mai multe funcționalități ale sistemului, așa cum au fost acestea precizate în subcapitolul 4.1 :

- *Sistemul timp real țintă*, constituie baza unui *model de consum de energie la nivel de sistem pentru o platformă țintă dată, ce face parte dintr-o serie de clase de dispozitive timp-real încorporate, definite în prealabil* și este descris pe larg în subcapitolul 4.3 . Tot aici este realizată descrierea generală a sistemelor țintă vizate și sunt oferite diferite variante de arhitectură și configurație ale acestora.
- Pe baza modelului energetic al sistemului țintă și pe baza specificațiilor aplicației timp-real este propus *un model al comportării energetice și temporale a taskurilor aplicației*, model prezentat pe larg în subcapitolul 4.4
- *Modulul pentru măsurarea și determinarea parametrilor energetici*, descris în subcapitolul 4.5 , împreună cu *modulul pentru analiza și determinarea parametrilor temporali*, descris în subcapitolul 4.6 , îndeplinesc funcționalitatea de *măsurarea și determinarea parametrilor modelului consumului de energie de la punctul precedent, pentru o anumită aplicație dată*.
- *Planificarea aplicației cu ajutorul unui mecanism de planificare, pe mai multe niveluri, pe platforma țintă, cu funcție de eficientizare a consumului de energie electrică* și *execuția aplicației pe sistemul țintă* sunt realizate de către modulul de planificare descris în subcapitolul 4.8 .
- *Realizarea unei analize a comportării aplicației timp-real direct pe sistem* este realizată de către modulul de analiză propus și prezentat în 4.7 .

Pentru fiecare modul este dedicate un subcapitol separat, pentru fiecare este furnizată o scurtă descriere a modulului și a integrării acestuia în sistemul cadru, sunt propuse una sau mai multe variante de implementare și este prezentat un scenariu de funcționare, cu exemple de date furnizate de către acest modul, acolo unde este cazul.

Sistemul propus TEEARTS este un sistem unitar, format din elemente existente preluate și integrate în sistem ca: modulul pentru măsurarea și determinarea parametrilor energetici și modulul pentru analiza și determinarea parametrilor temporali și din elemente originale, propuse și dezvoltate în cadrul acestei cercetări ca: modelul de comportament energetic și temporal al sistemului țintă și cel al taskurilor aplicațiilor timp-real și algoritmul de planificare a acestora, cu funcție de eficientizare a consumului de energie electrică, din cadrul modului de planificare a aplicațiilor.

Prin cele prezentate în acest capitol au fost atinse primele obiective ale cercetării doctorale și anume: *1.Conceperea și dezvoltarea unui model de consum de energie la nivel de sistem pentru o serie de clase de dispozitive timp-real încorporate, definite în prealabil și 2. Conceperea și dezvoltarea unui cadru, reprezentând un mediu hardware și software și a unei metodologii aferente, pentru măsurarea și determinarea parametrilor modelului consumului de energie de la punctul precedent.*

5 Planificarea execuției aplicațiilor timp-real

Motto: *"Un algoritm trebuie să fie văzut pentru a fi crezut."* ***Donald Knuth***

În acest capitol va fi propus și prezentat, la nivel de detaliu, un algoritm pentru planificarea taskurilor timp-real critice, cu funcție de eficientizare a consumului de energie electrică.

În continuare va fi prezentată o particularizare a modelului taskurilor din sistemul cadru TEEARTS, prezentat în capitolul anterior, pentru a servi ca model de comportare energetică și temporală a taskurilor planificate cu algoritmul propus.

Urmând ca partea esențială a acestui capitol să fie constituită din propunerea și din descrierea funcționalității și a principiilor de implementare ale unui algoritm de reducere a consumului de energie datorat aplicațiilor timp-real ce rulează pe o platformă țintă.

5.1 Particularizarea modelului taskurilor

Modelul taskurilor este modelul prezentat în subcapitolul 4.4 , particularizat în funcție de algoritmul de planificare ales și în funcție de rezultatele oferite de către modulul de analiză al aplicației timp-real prezentat în subcapitolul 4.8 .

În cele ce urmează va fi reluat foarte pe scurt acest model și apoi prezentată varianta simplificată a acestuia, cea care este implementată direct pe platforma țintă și constituie baza planificării cu funcție de eficientizare a consumului de energie electrică propuse în continuare.

5.1.1 Modelul complet

După cum a fost menționat, sistemul de analiză conține un model energetic și temporal pentru taskurile aplicațiilor timp-real. Acesta modelează comportamentul temporal al aplicației, permițând totodată și determinarea un consum maxim la nivel de task și de aplicație, pentru o platformă dată, prin integrarea unor parametri de consum energetic în modelul taskului. Astfel modelul propus pentru un task, conține pe lângă parametrii temporali și o serie de parametri energetici, după cum s-a precizat în (4-16):

$$\theta_i \equiv \{T_i, C_i, D_i, \varphi_i, \Psi_i\}$$

Parametrii temporali *T (perioada), C (timpul de execuție), D (deadline-ul)* și *φ (faza)* vor fi specificați la frecvența nominală a procesorului.

Parametrii energetici ai unui task sunt dați în primul rând de setul de dispozitive din sistem, capabile să-și ajusteze consumul de energie în mod dinamic prin metode software, numite *dispozitive elementare* (4-20*)*:

$$\Psi \equiv \{\psi_0, \ldots, \psi_j, \psi_{j+1}, \ldots, \psi_m\}$$

unde, ψ_j este dispozitivul elementar *j.*

Fiecare dispozitiv elementar este caracterizat printr-un graf de stări active și inactive. Un nod S_j^k, al grafului, reprezintă starea *k* de funcționare, pentru dispozitivul *j,* descrisă prin parametrii (4-25):

$$S_j^k = \{P_j^k, f_j^k\}$$

unde, P_j^k reprezintă puterea corespunzătoare stării S_j^k, și f_j^k reprezintă frecvența de lucru a dispozitivului ψ_j în starea S_j^k.

Arcele din graf, reprezintă tranzițiile între stări și sunt descrise prin parametrii (4-34):

$$A_j^{k,k+i} = \{\eta_{k,k+i}, L_{k,k+i}, t_{k,k+i}\}$$

unde $A_j^{k,k+i}$ reprezintă tranziția din starea S_j^k în starea S_j^{k+i}, $\eta_{k,k+i}$ reprezintă energia consumată datorită tranziției, $L_{k,k+i}$ reprezintă limita de rentabilitate a tranziției (intervalul minim de timp cât dispozitivul trebuie să stea în starea *k+i,* pentru ca tranziția să ofere o reducere de energie), iar $t_{k,k+i}$ reprezintă timpul necesar tranziției.

Grafurile fiecărui dispozitiv vor fi interconectate, printr-o serie de alte arce între noduri din grafuri diferite care vor modela proprietatea că un dispozitiv *j* aflat în starea S_j^k poate funcționa cu un dispozitiv *j+l* aflat în starea S_{j+l}^k (4-35) :

$$A_{j,j+l}^{k,k+i}$$

unde $A_{j,j+l}^{k,k+i}$ reprezintă conexiunea dispozitivului *j* din starea S_j^k cu dispozitivul *j+l* din starea S_j^{k+i}.

5.1.2 Modelul simplificat

Pornind de la modelul anterior al taskurilor și al dispozitivelor se va realiza o variantă simplificată a acestuia, variantă folosită în implementările direct pe placa țintă. Această variantă este caracterizată printr-un număr mai redus de parametri și printr-un număr mai redus de stări. Astfel, dintre stările definite pentru un dispozitiv, pe baza cerințelor de funcționalitate și temporale ale taskurilor, se vor alege doar două: una activă $S^A{}_j$ și una inactivă $S^R{}_j$, astfel, dispozitivul *j* va fi caracterizat doar de următorii parametri:

$$\psi \equiv \{S_j^A, S_j^R, L_j, t_{j\_R\_A}\} \quad (5\text{-}1)$$

unde $S^A{}_j$ este *starea activă minimă* (starea corespunzătoare configurației *active specifice),* determinată conform cerințelor setului de taskuri pentru dispozitivul ψ_j Starea poate avea o valoare convențională nulă dacă dispozitivul nu e folosit. $S^R{}_j$ este *starea inactivă* (de repaus, definită ca fiind stare de funcționare cu consum redus, în care *device*-ul nu poate deservi nicio cerere) cu consumul minim în care poate intra dispozitivul elementar ψ_j conform cerințelor aplicației, L_j este limita de rentabilitate *(Break-Even Time)* pentru a trece din starea $S^A{}_j$ în starea $S^R{}_j$, $t_{j\_R\_A}$ este timpul de trecere din $S^R{}_j$ în starea $S^A{}_j$. Atât L_j și $t_{j\_R\_A}$ sunt exprimați în cicluri de tact procesor pentru frecvența nominală.

$S^A{}_j$ și $S^R{}_j$ sunt stări energetice definite de:

$$S_j^{A/R} = \{P_j, \eta_j, f_j\} \quad (5\text{-}2)$$

unde, P_j reprezintă puterea corespunzătoare stării $S^{A/R}{}_j$, η_j reprezintă energia maximă consumată în timpul tranziției din starea $S^A{}_j$ în $S^R{}_j$ și invers, f_j reprezintă frecvența de lucru a dispozitivului ψ_j în starea $S^A{}_j$. Pentru starea $S^R{}_j$, valoarea lui f_j poate fi ignorată.

Modelarea taskului, conform formulei (4-16), rămâne neschimbată în modelul simplificat.

Conform formulei energiei, energia consumată de dispozitivul j pe durata C_j a execuției unei instanțe ale unui task θ_j, este:

$$E_{i,j}(0, C_i) = \int_0^{C_i} P_i(t)dt \tag{5-3}$$

Dacă funcția de putere este constantă, atunci ecuația devine:

$$E_{i,j} = P_j \cdot C_i \tag{5-4}$$

unde P_j, este puterea specifică stării în care se află dispozitivul j pe durata execuției taskului *i.*

La această energie consumată, se adaugă surplusul de energie η_j(descris în relația (5-2). Astfel energia totală pentru un dispozitiv j pe durata execuției unei instanțe a taskului i este estimată ca fiind (în cel mai defavorabil caz):

$$E_{i,j\_tot} = E_{i,j} + \eta_j \tag{5-5}$$

Facem următoarea observație: un dispozitiv ψ_j își păstrează starea energetică pe durata unui task. Astfel, el poate fi activat, dezactivat, trecut dintr-o stare în alta doar la începutul și/sau la sfârșitul unui task. Nu se vor face tranziții dintr-o stare în alta în timpul rulării taskului.

5.2 Algoritmul TEEPARTS

În continuare va fi propus un algoritm de reducere a consumului de energie electrică, numit TEEPARTS *(Time and Energy Efficient Power Aware Real-Time Scheduling)*, dezvoltat ca o extensie pentru un tip de algoritmi de planificare pentru taskurile aplicațiilor timp-real critice. Tipul de algoritm de planificare vizat trebuie să îndeplinească următoarele condiții:

- Să fie unul determinist.
- Să fie destinat rulării taskurilor într-o manieră *non-preemptivă* (care nu acceptă întreruperi pe parcursul execuției lor) sau cu un număr limitat de niveluri de întrerupere.
- Să permită determinarea în avans a următorului task care va fi lansat și al momentului de lansare al acestuia.

Algoritmul de reducere a consumului energetic prezentat în continuare face parte din categoria algoritmilor de reducere a energiei *inter-task,* orice modificare de stare a sistemului sau a dispozitivelor sale, făcându-se la începutul și/sau sfârșitul execuției unui task și nu pe durata acesteia. Această abordare contracarează eventualele probleme cauzate de faptul că în practică, există situații în care nu se cunoaște momentul exact de utilizare al unui dispozitiv de către un task în cadrul execuției sale, păstrând astfel nivelul de predictibilitate al sistemului nealterat. Mai mult, algoritmii *inter-task* sunt mai ușor de implementat.

Mecanismul de reducere a energiei propus, cuprinde două componente: o componentă statică (offline), implementată de către modulul de analiză a aplicației și o componentă dinamică (online), implementată de către modulul de planificare.
Componenta statică a fost prezentată pe larg în subcapitolul 4.7 . Aceasta are rolul de a analiza planificabilitatea setului cu un anumit algoritm de planificare și de a furniza modelul simplificat al taskurilor pentru componenta online.

Componenta dinamică este reprezentată de un algoritm de eficientizare a consumului energetic, implementat direct pe placă, fiind integrat în mediul de planificare și execuție al taskurilor.

Ipoteza de lucru: Se va considera un mediu de execuție al taskurilor *non-preemptiv*, static, pentru care momentele de timp pentru lansarea fiecărui task sunt cunoscute sau se pot determina în orice moment.

În continuare va fi descris algoritmul de reducere al consumului energetic:

Se aleg două puncte de decizie semnificative în rularea algoritmului (înainte de lansarea unui task și după terminarea execuției acestuia). În fiecare din aceste puncte va rula o secvență din algoritm. Astfel, modulul software care implementează secvența de algoritm din momentul lansării în execuție al unui task se va numi TEE_PRE, iar cel care va implementa secvența de după terminarea execuției taskului se va numi TEE_SUF. Timpul necesar execuției acestor dispozitive se va însuma cu durata maximă de execuție a unui task, C_i , fiind luat astfel în calcul în algoritmii de planificare. Un exemplu de execuție a două taskuri și a momentelor de execuție a modulelor TEE_PRE și TEE_SUF este prezentat în Figura 26. În acest exemplu avem un set de două taskuri cu următorii parametri temporali: perioada T_1=10 unități de timp, timpul de execuție C_1=5 și faza φ_1=0 și respectiv T_2=20 unități de timp, timpul de execuție C_2=3 și faza φ_2=5. Din timpul total de execuție o unitate este reprezentată de către timpul de execuție al prefixului TEE_PRE, care se execută la începutul fiecărui task și o unitate este reprezentată de către timpul de execuție al sufixului TEE_SUF, care se execută la sfârșitul fiecărui task.

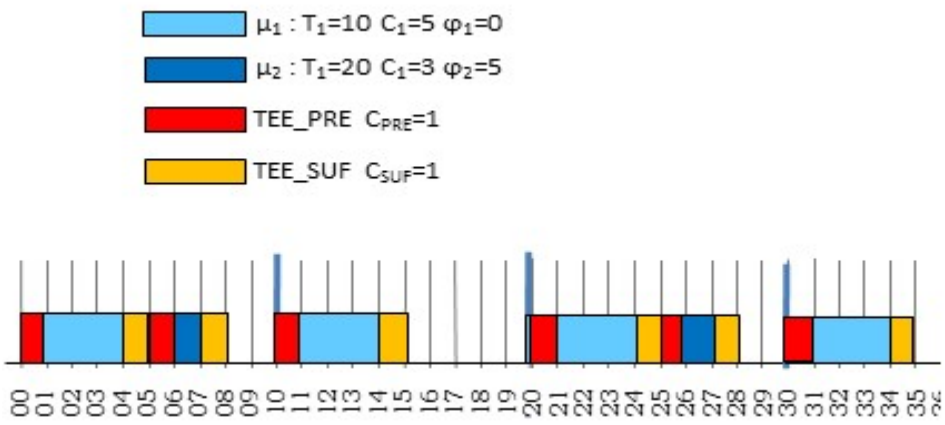


Figura 26. Exemplu de execuție a modulelor TEE_PRE și TEE_SUF

Se va considera o valoare limită pentru timpul de activare al oricărui dispozitiv elementar ca fiind:

$$\Omega \equiv \max(\Omega_a, \Omega_p) \tag{5-6}$$

unde Ω_a reprezintă timpul necesar activării unui dispozitiv elementar din TEE_PRE și Ω_p reprezintă timpul necesar (pre)activării unui dispozitiv din TEE_SUF.

În mod particular limita Ω poate fi considerată a fi mai mică decât timpul de execuție cel mai defavorabil al celor două module TEE_PRE și TEE_SUF însumate.

$$\Omega \leq C_{PRE} + C_{SUF} \tag{5-7}$$

unde, C_{PRE} este timpul de execuție cel mai defavorabil pentru TEE_PRE și C_{SUF} este timpul cel mai defavorabil pentru TEE_SUF.

Considerând această limită rezultă de asemenea că timpul de activare este mai mic strict decât execuția unui task.

Modulul TEE_PRE are rolul de a asigura disponibilitatea dispozitivelor necesare pentru taskul ce tocmai urmează să fie lansat, numit și taskul curent. Aceasta se realizează prin trecerea dispozitivului în starea S_i^A, dacă acesta se află în starea S_i^R. Organigrama algoritmului implementat de acest modul, este reprezentată în Figura 27.

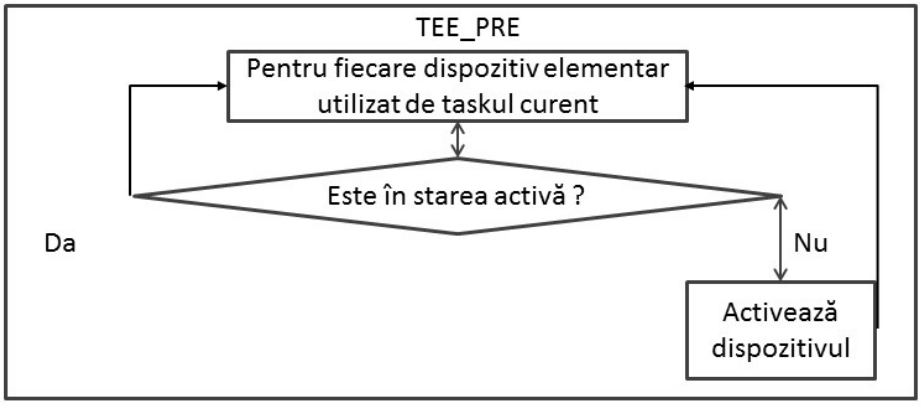


Figura 27. Organigrama modulului TEE_PRE

Modulul TEE_SUF are rolul de a determina dacă urmează un interval de timp cât procesorul nu este folosit (timp *idle*) și durata acestuia. Dacă există un astfel de interval se vor inactiva toate acele dispozitive pentru care timpul *idle* procesor depășește limita de rentabilitate a dispozitivului și toate dispozitivele care nu sunt folosite de următorul task. De asemenea toate dispozitivele folosite de următorul task care au perioada de activare mai mare decât Ω_a trebuie activate în prealabil pentru acest task. Organigrama algoritmului implementat de acest modul este reprezentată în Figura 28.

Aceste aspecte legate de funcționalitatea celor două module sunt prezentate și sub formă de pseudocod în Figura 29.

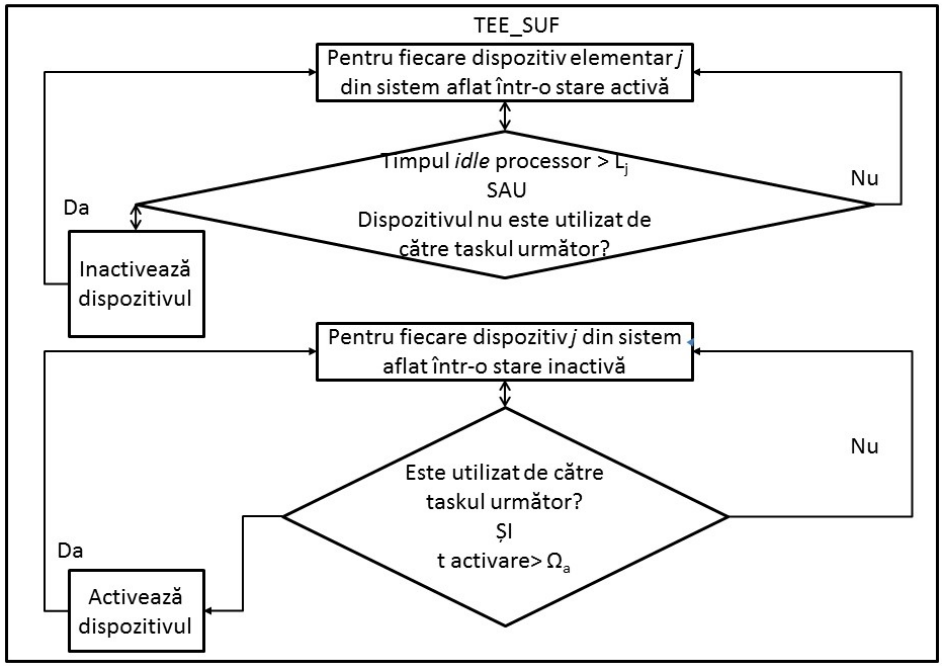


Figura 28. Organigrama modulului TEE_SUF

```
                        TEEPARTS:
TEE_PRE:
  Extragere id task curent din tabela HDIS
  PENTRU fiecare dispozitiv elementar utilizat de taskul curent
    DACA dispozitivul nu este in stare activa
        activeaza dispozitivul
        marcheaza activarea dispozitivului
  Lansare executie taskul curent
TEE_SUF:
  Extragere id task urmator din tabela HDIS
  Extragere moment lansare in executie task urmator din tabela HDIS
  Determinare timp idle procesor
  PENTRU fiecare dispozitiv elementar din sistem
     DACA este in stare activa
        DACA timpul idle este mai mare decat limita de rentabilitate (L) a dispozitivului
         SAU dispozitivul nu este utilizat de catre taskul urmator
             inactiveaza dispozitivul
             marcheaza inactivarea dispozitivului
     ALTFEL
        DACA este utilizat de catre taskul urmator
         SI timpul de activare este mai mare decat limita de timp pentru activare din TEE_PRE (omega_a)
             activeaza dispozitivul
             marcheaza activarea dispozitivului
```

Figura 29. Algoritmul TEEPARTS în pseudocod

5.2.1 Integrare TEEPARTS în FENP. Particularități

Integrarea algoritmului de reducere a energiei electrice propus în subcapitolul 5.2 în sistemul de operare HARETICK se face prin integrarea algoritmului TEEARTS peste mecanismul de planificare și lansare în execuție a taskurilor aplicației.

Algoritmul TEEPARTS a fost special dezvoltat pentru algoritmi de planificare *non-preemptivi* așa cum este FENP. Astfel, integrarea celor doi algoritmi este una facilă.

În primul rând structurile celor doi algoritmi sunt asemănătoare, ambii algoritmi folosind aceleași două puncte de decizie: la începutul și la sfârșitul execuției unui task. Astfel, modulul TEE_PRE al algoritmului TEEPARTS poate fi integrat cu ușurința în modulul HDIS_PRE al sistemului HARETICK. În mod analog, modulul TEE_SUF poate fi integrat în modulul HDIS_SUFIX.

Ca o consecință a acestui fapt, C_{PRE} și C_{SUF} din formula (5-7) se vor referi la timpii cei mai defavorabili de execuție ai modulelor integrate TEE_PRE+HDIS_PRE și respectiv TEE_SUF+HDIS_SUF.

Integrarea noului model de taskuri propus în subcapitolul 5.1.2 în structurile de date ale sistemului HARETICK se face în mod relativ simplu:

- Structura de date care implementează modelul unui task are parametrii de timp aceeași și pentru algoritmul FENP și pentru TEEPARTS, astfel pentru a obține noul model de taskuri se va extinde vechiul model prin adăugarea vectorului de dispozitive ψ_i din formula (4-16).
- Pentru a modela acest vector de dispozitive se declară o nouă structură de tip tablou. Fiecare înregistrare din acest tablou reprezentând un dispozitiv elementar, modelat prin parametrii formulei (5-1). Pe lângă acești parametri se va reține pentru fiecare dintre stările active sau inactive ale dispozitivului o funcție de intrare în acea stare pentru dispozitivul respectiv.
- Timpul *idle* procesor se va calcula ca fiind:

$$T_{idle} = \tau_{st\_urm} - \tau_{fin} \tag{5-8}$$

unde τ_{st_urm} reprezintî cel mai apropiat moment de start al unui ModX față de momentul curent, iar τ_{fin} este momentul de finalizare a execuției taskului curent incluzând modulul executiv HDIS_SUF.

Un exemplu de implementare pe o platforma țintă dată va fi prezentat în Capitolul 6 .

5.2.2 Integrare TEEPARTS în H²RTS. Particularități

Integrarea algoritmului TEEPARTS în sistemul de operare HARETICK extins prin implementarea algoritmului H²RTS se face ca în cazul precedent tot prin integrarea algoritmului TEEPARTS peste mecanismul de planificare și lansare în execuție a taskurilor aplicației.

Și în acest caz structurile celor doi algoritmi sunt asemănătoare, ambii algoritmi folosind aceleași două puncte de decizie: la începutul și la sfârșitul execuției unui task, doar că în acest caz avem de a face cu două module diferite de tip prefix pentru executiv, unul pentru taskurile planificate cu algoritmul FENP altul pentru cele planificate cu MEDFNP. Astfel, modulul de prefix al algoritmului TEEPARTS trebuie integrat în ambele module de prefix (pentru fiecare din cele două medii de execuție).

Integrarea noului model de taskuri propus în subcapitolul 5.1.2 în structurile de date ale sistemului HARETICK se va face prin integrarea noului model al taskurilor în ambele modele de taskuri precedente:

- structura de date care implementează modelul unui task are parametrii de timp aceeași pentru algoritmul FENP, pentru MEDF și pentru TEEPARTS, astfel pentru a obține noul model de taskuri se va extinde vechiul model prin adăugarea vectorului de dispozitive Ψ din formula
- (4-16) la ambele modele inițiale de taskuri.
- datorită existenței unui mediu *preemptiv*, pe lângă condiția din TEE_SUF, pentru inactivarea unui dispozitiv, în cazul în care timpul *idle* procesor este mai mare decât limita de rentabilitate a dispozitivului, condiție ca acesta să nu fie utilizat de către taskul FENP următor, se va verifica și condiția ca acesta să nu fie utilizat nici de către taskul MEDF tocmai întrerupt. Astfel, organigrama algoritmului ce rulează în modulul de TEE_PRE rămâne neschimbată, iar organigrama pentru TEE_SUF se modifică conform reprezentării din Figura 30.
- modul de calcul al timpului *idle* procesor va fi unul diferit:

$$T_{idle} = \tau`_{st\_urm} - \tau`_{fin} \tag{5-9}$$

unde, $\tau`_{st\_urm}$ este cel mai apropiat moment de start al unui task al setului S din relația (4-37), față de momentul curent, iar $\tau`_{fin}$ este momentul de finalizare a execuției taskului curent (incluzând modulul executiv, dacă e cazul).

- vectorul de dispozitive elementare este același ca pentru algoritmul anterior, acesta nedepinzând de algoritmul de planificare pe baza căruia se face extensia TEEPARTS.

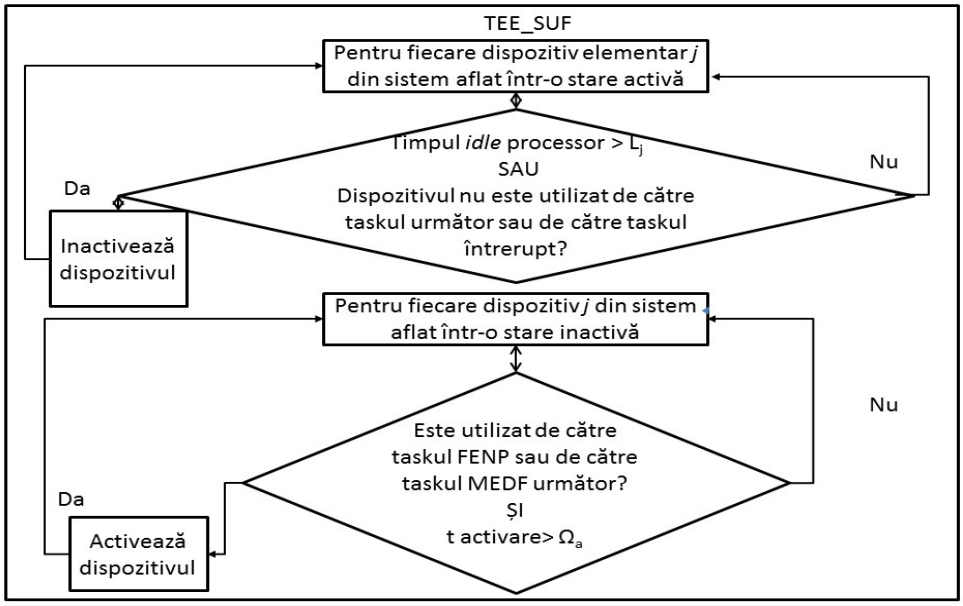


Figura 30. Organigrama modulului TEE_SUF pentru H^{2}RTS

5.2.1 Integrare TEEPARTS în Hybrid RTOS. Particularități

Integrarea algoritmului de reducere a energiei propus, în sistemul de operare Hybrid RTOS se poate face, deocamdată doar în mediul de execuție al taskurilor critice (peste extensia HARETICK), prin integrarea algoritmului TEEARTS peste mecanismul de planificare și lansare în execuție a taskurilor critice, integrarea acestuia făcându-se peste algoritmul FENP, integrare care a fost deja prezentată în subcapitolul 5.2.1 . Există și o serie de excepții legate de particularitatea sistemului:

- Procesorul va fi inactivat de către mediul FreeRTOS, când nu are de rulat niciun task propriu.
- În varianta curentă algoritmul TEEPARTS nu poate inactiva dispozitive utilizate de către taskurile rulate în mediul FreeRTOS.

5.3 Concluzii

În acest capitol este prezentat o soluție de reducere a consumului de energie electrică datorat rulării unei aplicații timp-real (critice), pe o platformă țintă. Soluția constă într-un algoritm de planificare a taskurilor timp real critice, cu funcție de eficientizare a consumului de energie electrică, dezvoltat ca o extensie a unor algoritmi pentru planificarea taskurilor timp-real, îndeosebi pentru taskuri critice, care trebuie să îndeplinească o serie de condiții dintre care: să fie un algoritm de planificare determinist, destinat rulării taskurilor într-o manieră non-preemptivă sau cu un număr limitat de niveluri de întrerupere și să permită determinarea în avans a următorului task care va fi lansat și al momentului de lansare al acestuia.

Algoritmul se bazează pe o variantă simplificară a modelului taskurilor în sistemul TEEARTS, propus în capitolul anterior și cuprinde două etape: una statică de determinare a parametrilor modelului și a două stări de funcționare pentru fiecare dispozitiv folosit (una activă și una inactivă) și una dinamică implementată direct pe platforma țintă, reprezentată de planificarea efectivă a taskurilor și de managementul dispozitivelor folosite.

Reducerea consumului de energie prin planificarea aplicației cu ajutorul algoritmului TEEPARTS și execuția acesteia pe platforma țintă, reprezintă principala funcționalitate a sistemului cadru propus, astfel realizându-se obiectivul principal al acestei cercetări doctorale, cel de a: *dezvolta un mecanism de planificare, pe mai multe niveluri, pentru sisteme timp-real încorporate, cu funcție de eficientizare a consumului de energie electrică.*

6 Validarea sistemului cadru TEEARTS și evaluări de performanță

Motto: *"Realitatea este o platformă de care ne lovim de multe ori".* ***Nicolae Iorga***

În acest capitol va fi prezentată validarea sistemului cadru TEEARTS. Se va pune accent în mod special pe validarea algoritmului de planificare TEEPARTS, prin implementarea acestuia pe două platforme țintă, și prin măsurarea consumului unei aplicații planificate cu și fără algoritmul TEEPARTS propus.

În continuare vor fi descrise platformele țintă și se vor prezenta detalii de implementare specifice acestora și mediului de execuție și planificare ales. Varianta de implementare propusă este analizată din punct de vedere al impactului asupra sistemului de operare, având în vedere următorii parametri: surplusul de memorie utilizat, influența asupra latenței întreruperilor, asupra jitter-ului de execuție a taskurilor și încărcarea suplimentară procesor.

Tot în acest capitol sunt prezentate rezultate ale analizei algoritmului propus, obținute atât prin măsurare directă pe platformele țintă vizate, cât și cu ajutorul modulului de analiză prin intermediul unui simulator numit TEEPATSsim.

6.1 Implementarea TEEPARTS folosind platformele EFM32-G8XX-STK și EFM32GG-STK3700

6.1.1 Descrierea platformelor

Două exemple de platforme țintă le constituie EFM32-G89XX-STK și EFM32GG-STK3700 (v.Figura 31). Acestea sunt plăcuțe de dezvoltare produsă de către compania Energy Micro (Silicon Labs). Prima placă cuprinde un procesor EFM32G890 din familia Gecko, iar a doua un procesor EFM32GG990 din familia Giant Geko, ambele bazate pe o arhitectură ARM-Cortex M3 Arhitectura aceasta a fost aleasă pentru că îndeplinește condițiile sistemului țintă așa cum au fost definite în subcapitolul 4.3 și anume:

- *Arhitectură deterministă* – În primul rând procesorul are la bază o arhitectură deterministă, oferind predictibilitate la nivel de instrucțiune.
- *Interacțiunea cu mediul*: - Capacitatea sistemului de a interacționa cu mediul este furnizată prin capacitatea procesorului de a utiliza surse de întrerupere și prin faptul că aceste plăcuțe de dezvoltare au o serie de senzori integrați și suportă și alte extensii ce pot conține componente de interacțiune cu mediul.
- *Coordonata de timp*: - Suportul pentru implementarea coordonatei de timp este dat în primul rând de temporizatoare (*timers*), de diferite surse de tact (ce pot fi sincronizate). În acest sistem, coordonatele de timp și întreruperile sunt strâns legate.

Pe de altă parte, sistemul vizat are capabilități de reducere a consumului de energie electrică prin control software. Dintre mecanismele de reducere a consumului de energie electrică descrise în primul capitol, vizăm cele mai semnificative două tipuri: cele bazate pe DVS si cele bazate pe DPM. Procesorul poate fi alimentat la tensiuni diferite, poate să-și schimbe frecvența de lucru in mod dinamic, are predefinite patru

stări de funcționare cu consum energetic diferit (EM0-3). Toate aceste treceri într-o anumită stare de funcționare se întâmplă într-un mod predictibil, într-un timp mai mic sau egal cu o limită maximă determinată apriori, fie din specificațiile de catalog fie pe baza măsurătorilor realizate de modulul de măsurare al sistemului cadru propus.

O altă caracteristică ce recomandă acest sistem este faptul că acesta suportă alimentare de la o sursă de alimentare fixă sau de la baterie astfel fiind potrivit atât pentru dezvoltare și testare cât și pentru funcționarea ca nod de senzori în aplicații reale.

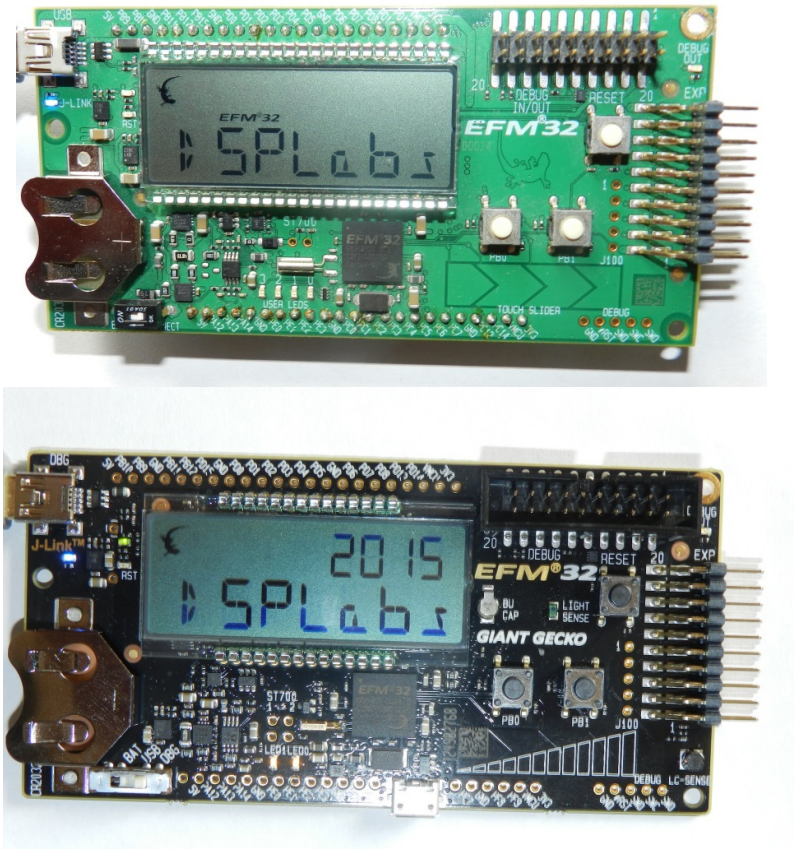


Figura 31. Sisteme țintă

6.1.2 Implementare

Implementarea algoritmului TEEPARTS s-a făcut ca integrare a acestuia în nucleul HARETICK, folosind ca algoritm de planificare FENP.

Procesorul pe care s-a efectuat implementarea este un procesor cu arhitectura ARM-Cortex M3 pe 32 de biți.

Ca suport pentru reprezentarea ceasului sistem a fost ales un sistem de trei *timer-e* TIMER0:TIMER1:TIMER2, fiecare pe câte 16 biți, conectate în cascadă astfel încât să funcționeze ca un *timer* unitar pe 48 de biți. Acesta are în primul rând scopul de a păstra timpul sistem, pe baza căruia se vor lua toate deciziile legate de planificarea și execuția taskurilor, iar în al doilea rând *timer-ul* funcționează ca o sursă unică de întrerupere în sistem. Astfel, rutina de tratare a acestei întreruperi implementează de fapt executivul sistemului: modulul HDIS. Pentru că *timer-ul* este singura sursă de întrerupere din sistem, orice detecție de evenimente se face cu ajutorul mecanismelor de tip *polling.*

Nucleul HARETICK folosește o serie de structuri de date, dintre care două sunt esențiale în modelarea taskurilor și planificarea lor: o tabelă de structuri ce conține descriptorii unui task numită tabela PDT și o tabelă de structuri ce conține datele necesare executivului, numită tabela HDIS.

Structurile de date ale nucleului s-au modificat după cum urmează:

Structurii de date care modelează un task și care reprezintă o înregistrare în tabela descriptorilor numită tabela PDT, i s-a adăugat un câmp numit DEVICES, acesta modelează vectorul de dispozitive din formula (4-16), în sensul marcării că un anumit dispozitiv din sistem este folosit de către taskul respectiv. Astfel structura descriptorilor unui task devine (v. Tabelul 6):

Tabelul 6. Structura înregistrărilor din tabela PDT modificată

Offset	**Denumire**	**Semnificație**
0	PID (16 biți)	Un identificator pentru fiecare task FENP (ModX).
	ATR (16 biți)	Atribute suplimentare, rezervat pentru alte implementări
1:2	PER_Hi:PER_Lo	Perioada ModX-ului în cicluri de clock ale timpului sistem
3:4	DLN_Hi:DLN_Lo	Intervalul limită (*deadline*)
5:6	DLY_Hi:DLY_Lo	Faza taskului
7	WCE	Durata maximă de execuție
8:9	ECT_Hi:ECT_Lo	Contorul de execuții, rezervat pentru alte implementări
10:11	SCT_Hi:SCT_Lo	Timpul de start următor efectiv utilizat de FENP
12	CODE	Adresa de start a zonei de cod a ModX-ului
13	***DEVICES***	***Vectorul de dispozitive utilizate de către ModX***

Tabela HDIS rămâne neschimbată (v.Tabelul 7):

Tabelul 7. Structura înregistrărilor din tabela HDIS

Offset	**Denumire**	**Semnificație**
0	PID	Identificatorul fiecărui task FENP
1:2	STime_Hi:STime_Lo	Timpul de start

Pe lângă aceste structuri, este declarată și folosită o structură nouă, care implementează modelarea unui dispozitiv elementar conform formulei (5-1) (v.Tabelul 8).

Tabelul 8. Structura unui dispozitiv elementar

Offset	Denumire	Semnificație
0	***DEV_ID***	***Identificatorul fiecărui dispoziv***
1	***ID_S_A***	***Identificatorul stării minime active***
2	***ID_S_R***	***Identificatorul stării minime inactive***
3:4	***L_Hi:L_Lo***	***Limita de rentabilitate***
5:6	***T_R_A_Hi:T_R_A_Lo***	***Timpul de activare al dispozitivului***

Pentru vectorul de dispozitive elementare este definit un tablou, cu *Nr_Dispozitive* intrări, de astfel de structuri, unde *Nr_Dispozitive* reprezintă numărul de dispozitive elementare din sistem. În plus, se va mai reține și o variabilă care va marca la nivel de bit dacă un dispozitiv elementar este activ (valoarea 1 logic) sau inactiv (valoarea 0) în fiecare moment al rulării sistemului și a taskurilor aplicației.

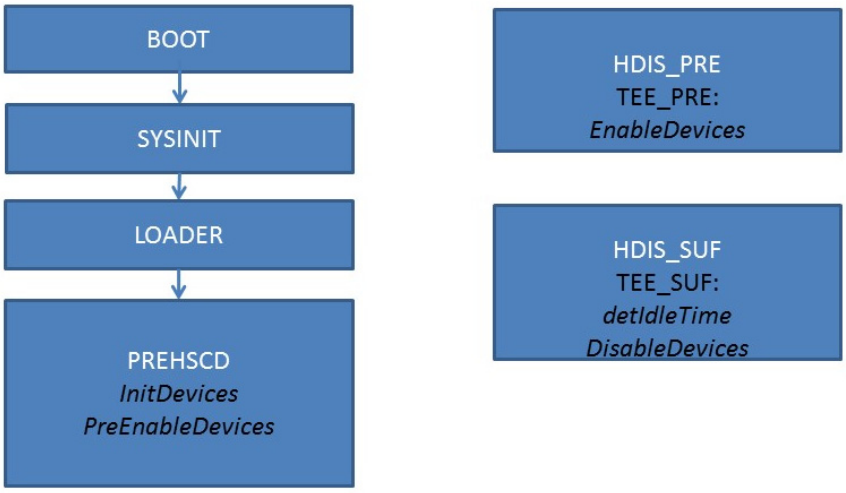
Pentru fiecare dispozitiv, va exista definită și implementată o funcție de activare a acestuia (de trecere din starea inactivă S^R în starea activă S^A) și o funcție de inactivare a acestuia (de trecere din starea inactivă în cea activă).

În cele ce urmează va fi prezentată implementarea algoritmului TEEPARTS. În primul rând au fost modificate trei module esențiale ale nucleului HARETICK: PREHSCD și cele două componente ale executivului HDIS_PRE și HDIS_SUF, prin extensie. Acestor module li s-au adăugat apeluri de funcții (în cazul modulelor implementate în C) sau secțiuni de cod (pentru secțiunile implementate în limbaj de asamblare). Aceste modificări pot fi vizualizate în Figura 32, unde cu negru sunt marcate extensiile de cod impuse de algoritmul TEEPARTS și cu alb sunt marcate modulele originale ale nucleului HARETICK.

După cum se poate vedea, prima modificare este efectuată în modulul PREHSCD. Astfel modulul, înainte de a lansa în execuție aplicația efectivă, populează tabloul de dispozitive cu datele din Tabelul 8 prin funcția *InitDevices*(). Apoi tot din PREHSCD vor fi pornite toate dispozitivele necesare primului task ce urmează a fi lansat în execuție prin funcția *PreEnableDev*().

Pe de altă parte pornirea și oprirea dispozitivelor pentru fiecare task în parte se efectuează în cadrul modului HDIS, prin cele două componente ale sale HDIS_PRE și HDIS_SUF. Modulul HDIS_PRE a fost extins prin apelul funcției *EnableDevices*(), care implementează efectiv funcționalitatea modulului TEE_PRE, funcționalitate deja descrisă în subcapitolul 5.2 . În mod analog, modulul HDIS_SUF a fost extins printr-o secțiune de cod care determină timpul *idle* procesor (*detIdleTime*) și printr-un apel de funcție *DisableDevices* care implementează funcționalitatea modulului TEE_SUF a cărui funcționalitate a fost deja descrisă tot în subcapitolul 5.2 .

Figura 32. Modulele HARETICK modificate



În plus, mediul de execuție a taskurilor care nu sunt timp real, mediu numit Background (BGND), va trece procesorul în starea inactivă S^R corespunzătoare procesorului, când nu are nimic de procesat. Ieșirea din starea inactivă se face prin activarea întreruperii sursei de timp care gestionează mediul de planificare și execuție FENP, în implementarea curentă, astfel, procesorul este tratat separat de către algoritm, deși este modelat la fel cu celelalte dispozitive elementare.

6.2 Analiza aplicațiilor timp-real pe platformele țintă vizate

Pentru a analiza mediul de execuție cu noul mecanism de planificare TEEPARTS și a-l compara cu varianta cu FENP simplu s-au ales o serie de parametri reprezentativi pentru analiza performanțelor unui algoritm de planificare și a mediului de execuție aferent.

Pentru măsurarea parametrilor s-a ales un cadru de măsurare reprezentat de sistemul țintă, un analizor logic LA1032, un PC pe care rulează *sofware-ul* pentru analizorul logic numit zlgLogic și un mediu de dezvoltare integrat pentru analiza codului ce rulează pe placa țintă, numit Keil uVision. Sistemul este prezentat în Figura 33.

A. Utilizarea memoriei

Extensia TEEPARTS peste nucleul de operare HARETICK aduce un surplus mic de memorie utilizată. Astfel, prin extensia TEEPARTS se alocă static în plus față de structurile folosite de HARETICK în total 8 câmpuri de date, a câte un cuvânt fiecare. Surplusul de memorie adus de codul rutinelor ce implementează TEEPARTS este de aproximativ 0.5%, față de codul inițial (v.Figura 34).

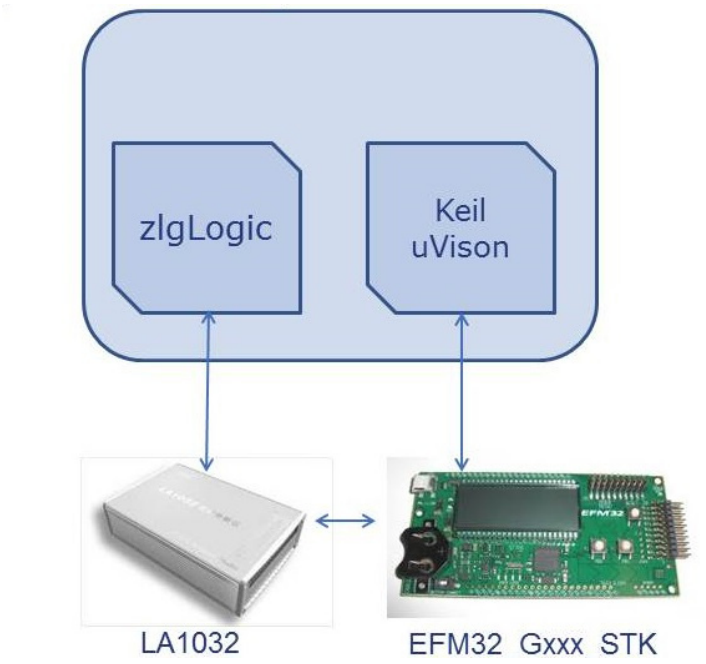


Figura 33. Cadrul de măsurare

```
        Cu extensia TEEPARTS:
    Total RO  Size (Code + RO Data)                23408 (  22.86kB)
    Total RW  Size (RW Data + ZI Data)              6816 (   6.66kB)
    Total ROM Size (Code + RO Data + RW Data)      23608 (  23.05kB)

Fără extensia TEEPARTS:
    Total RO  Size (Code + RO Data)                23296 (  22.75kB)
    Total RW  Size (RW Data + ZI Data)              6816 (   6.66kB)
    Total ROM Size (Code + RO Data + RW Data)      23496 (  22.95kB)
```

Figura 34. Memoria utilizată - Captură log Keil uVision

B. Latența întreruperilor și jitter

Consider latența întreruperilor definită ca *"suma timpilor de blocare cât timp nucleul se află în așteptarea răspunsului la întrerupere, în procesul de salvare contextului, de determinare a sursei de întrerupere și de invocare a handler-ului*". [111].

Deoarece, în HARETICK singura sursă de întrerupere este întreruperea de ceas RTC (dată de un timer), orice alt eveniment fiind detectat prin mecanisme de polling, latența întreruperilor este deterministă. Aceasta este dată pe de o parte de latența hardware a întreruperii (12CC în implementarea curentă), pe de alta de către rutina de

tratare a întreruperii care are rolul de a lansa modulul executiv HDIS_PRE. Astfel, latența totală a întreruperilor este de:

$$L_{hw\_\text{int}} + L_{t\_\text{int}\_handler} = 12CC + 56CC \quad (6\text{-}1)$$

unde, L_{hw_int} reprezintă latența hardware și $L_{t\_int\_handler}$ reprezintă latența datorată rutinei de tratare a întreruperii (din momentul de start al rutinei până la lansarea HDIS_PRE).

Pentru că TEEPARTS nu intervine decât în modulul HDIS_PRE, acesta nu influențează latența întreruperilor, așa cum a fost definită aceasta anterior și nici nu introduce jitter. Astfel, nici în HARETICK simplu nici în extensia cu TEEPARTS nu există jitter în tratarea întreruperilor, aspect ce poate fi văzut și în studiul de caz următor.

C. Încărcare suplimentară procesor

Încărcarea suplimentară procesor este definită, pentru un sistem de operare, ca fiind "*surplusul de timp consumat de către sistemul de operare, față de utilizarea procesor efectivă*" (datorată aplicațiilor) [112].

În HARETICK aceasta este egală cu timpul de execuție al modulelor de planificare (HSCD) și executiv (HDIS). Conform măsurătorilor efectuate această încărcare este egală cu 372 CC pentru HDIS și 14 240 CC pentru HSCD.

Cu extensia TEEPARTS se mărește încărcarea datorată modulului HDIS, cea datorată modulului HSCD rămânând neschimbată. Astfel noua încărcare datorată modulului HDIS diferă în funcție de numărul de dispozitive elementare definite și în funcție de tipul acestora și deci implicit de funcțiile de activare și dezactivare corespunzătoare acestor dispozitive.

Tabelul următor concluzionează cele prezentate anterior, furnizând o comparație minimală între cele două medii de planificare și execuție discutate.

Tabelul 9. Tabel comparativ

Parametru	***HARETICK+FENP***	***HARETICK+TEEPARTS***
Utilizare memorie	Mică (~23KB)	Mică (~23KB)
Latența întreruperilor	12CC Predictibilă Mică, dată de latența hardware a întreruperilor	68CC Predictibilă Mică, dată de latența hardware a întreruperilor
Jitter întrerupere	Inexistent	Inexistent
Încărcare suplimentară procesor	Deterministă	Deterministă

6.3 Studii de caz și evaluarea performanțelor

6.3.1 Studiu de caz

În mediul HARETICK, cu extensia TEEPARTS în configurația și implementarea descrisă în capitolul precedent, a fost ales un exemplu de set de taskuri format din două ModX-uri perfect periodice, care aprind și sting de un număr de ori un LED și afișează un text pe ecranul LCD al plăcii. Astfel, au fost selectate trei dispozitive elementare, nucleul

procesorului, portul C al perifericului GPIO și controlerul pentru ecranul LCD. Se consideră următorii parametri:

Tabelul 10. Parametrii dispozitivelor elementare

ID dispozitiv	Denumire	Starea activă	Starea inactivă	Limita de rentabilitate	Timpul de activare
0	Nucleu CPU	EM0-32MHz	EM1	112CC	48CC
1	GPIO	Cu tact	Fără tact	240CC	112CC
2	LCD controler	Activ, cu tact	Inactiv, fără tact	600CC	283CC

Taskurile au fost planificate cu algoritmul FENP și au următorii parametri. Modulul HSCD este de asemenea văzut și tratat ca un ModX:

Tabelul 11. Tabela PDT

HSCD		
Denumire	**Valoare**	**Semnificație**
PID (16 biți)	0	ID HSCD
ATR (16 biți)	0	Rezervat
PER_Hi:PER_Lo	8000000	4s
DLN_Hi:DLN_Lo	8000000	4s
DLY_Hi:DLY_Lo	506000	253 ms
WCE	600	0,3 ms
ECT_Hi:ECT_Lo	0x00000000FFFFFFFF	Număr nelimitat de execuții
SCT_Hi:SCT_Lo	0	Se va seta ulterior de către HSCD
CODE	HSCD	Eticheta reprezentând adresa codului ModX-ului HSCD
DEVICES	1	Este folosit doar nucleul procesorului
Blink 1		
Denumire	**Valoare**	**Semnificație**
PID (16 biți)	1	ID ModX1
ATR (16 biți)	0	Rezervat
PER_Hi:PER_Lo	1000000	0,25s
DLN_Hi:DLN_Lo	1000000	0,25s
DLY_Hi:DLY_Lo	0	0
WCE	6000	3 ms
ECT_Hi:ECT_Lo	0x00000000FFFFFFFF	Număr nelimitat de execuții
SCT_Hi:SCT_Lo	0	Se va seta ulterior de către HSCD

CODE	BLINK1	Eticheta reprezentând adresa codului ModX-ului BLINK1
DEVICES	7	Nucleul procesorului, PORTUL C GPIO și controlerul LCD
Blink 2		
Denumire	**Valoare**	**Semnificație**
PID (16 biți)	2	ID ModX2
ATR (16 biți)	0	Rezervat
PER_Hi:PER_Lo	500000	0,5s
DLN_Hi:DLN_Lo	500000	0,5s
DLY_Hi:DLY_Lo	6000	3 ms
WCE	6000	3 ms
ECT_Hi:ECT_Lo	0x00000000FFFFFFFF	Număr nelimitat de execuții
SCT_Hi:SCT_Lo	0	Se va seta ulterior de către HSCD
CODE	BLINK2	Eticheta reprezentând adresa codului ModX-ului BLINK2
DEVICES	7	Nucleul procesorului, PORTUL C GPIO și controlerul LCD

6.3.2 Evaluarea performanțelor

În urma rulării algoritmului TEEPARTS pe sistemul țintă, pentru cazul prezentat, a fost observată o reducere a energiei consumate cu peste 60%. Apoi, a fost considerat un alt studiu de caz, cu același set de taskuri, dar eliminând controlerul LCD. Astfel au fost considerate doar două dispozitive elementare: nucleul procesor și portul C al perifericului GPIO, cu aceeași parametri ca în cazul precedent. De data aceasta s-a obținut o reducere de energie de 65%, după cum se poate vedea și din compararea celor două capturi (cu și fără TEEPARTS), efectuate cu software-ul de analiză al consumului (Simplicity Studio), furnizat de către sistemul țintă printr-un cadru de măsurare furnizat de către firma producătoare a sistemului (Energy Micro/Silicon Labs) (v. Figura 35 și Figura 36).

Cele două figuri reprezintă consumul de energie pentru o hiperpedioadă a setului de taskuri. Execuția întregului set de taskuri se repetă periodic la un interval dat de hiperperioada setului. Aceasta este reprezentată de cel mai mic multiplu comun al perioadelor taskurilor din set.

Pentru platforma EFM32GG-STK3700, pentru aceeași aplicație, a fost obținută o performanță asemănătoare (64,09%).

Pe lângă reducerea consumului de energie, cel mai important aspect care a fost validat prin implementarea pe sistemul țintă, este faptul că TEEPARTS nu reduce în niciun fel determinismul și predictibilitatea sistemului. Acest aspect se reflectă mai ales prin faptul că taskurile cu și fără TEEPARTS rămân perfect periodice (v. Figura 37 și Figura 38).

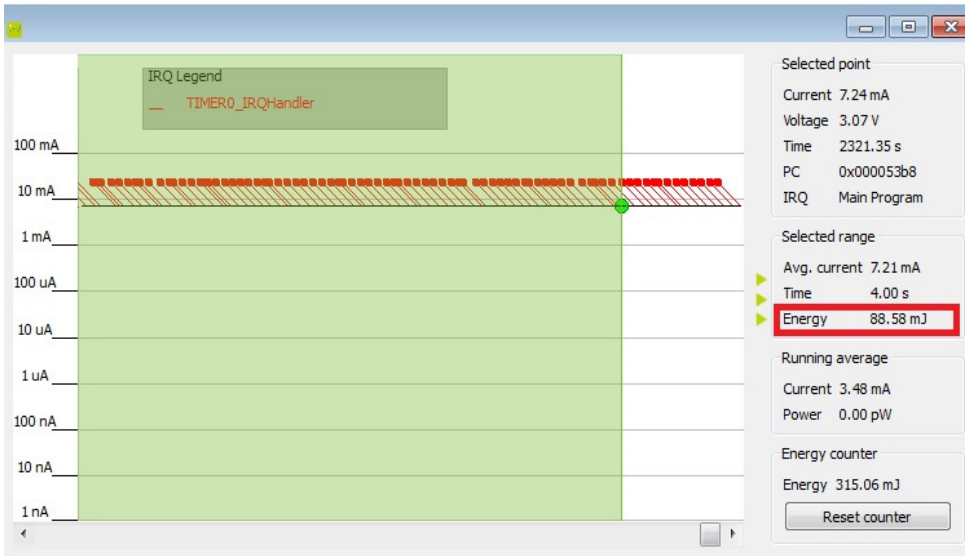


Figura 35. Captură consum energie pentru o hiperperioadă fără TEEPARTS

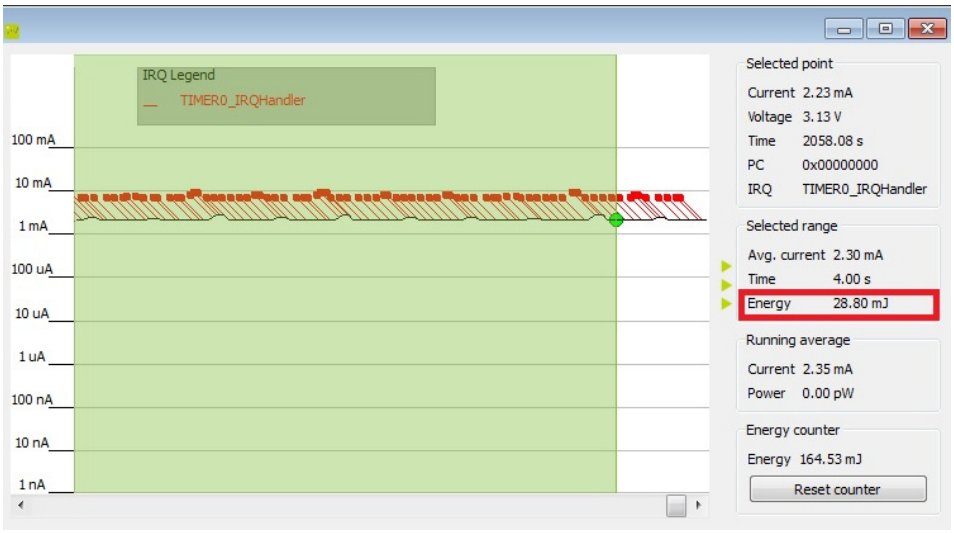


Figura 36. Captură consum energie pentru o hiperperioadă cu TEEPARTS

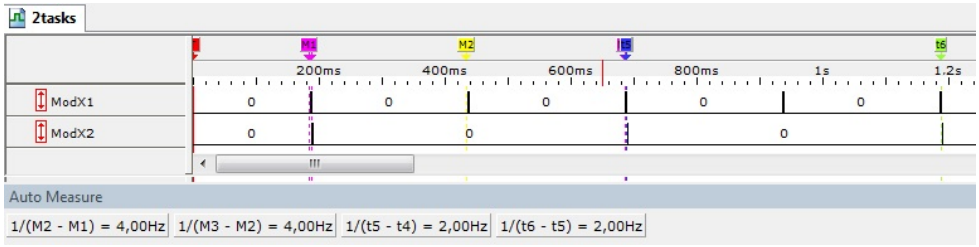


Figura 37. Execuția fără jitter a două ModX-uri cu FENP simplu

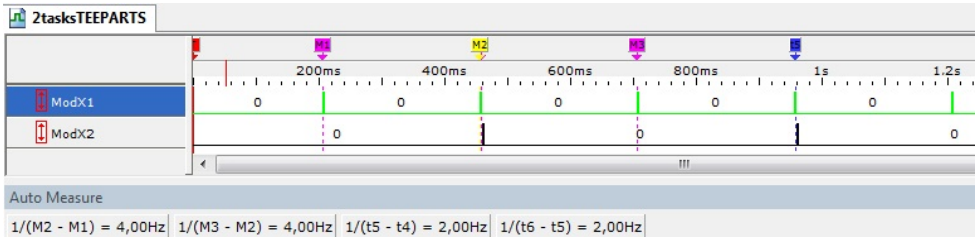


Figura 38. Execuția fără jitter a două ModX-uri cu FENP + TEEPARTS

Aplicația descrisă anterior și prezentată ca studiu de caz a fost de asemenea rulată pe platforma EFM32-G8xx-STK folosind sistemul de operare FreeRTOS. Acest sistem a fost configurat să treacă procesorul, atunci când nu este folosit, în starea energetică EM2, caracterizată printr-un consum de 2,7 μW. FreeRTOS folosește un mecanism de reducere a consumului de energie bazat pe DPM, care vizează doar procesorul. Performanțele comparative ale celor două mecanisme de reducere a consumului de energie pot fi vizualizate în Figura 39.

În această figură se observă că cel mai mare consum de energie este dat de rularea aplicației cu sistemul de operare FreeRTOS fără sisteme de management al puterii (FreeRTOS-noPM), HARETICK-ul nesolicitând același consum, nici în varianta simplă fără TEEPARTS. În ceea ce privește reducerea consumului de energie pentru cazurile considerate, aceasta este reprezentată grafic în Figura 40, atât ca valoare absolută exprimată în mJ cât și ca procentaj. Din figură observăm că deși folosind algoritmul implementat cu FreeRTOS obținem o valoare ușor superioară în mJ, datorată consumului mai mare al aplicației implementate cu FreeRTOS simplu, ca procentaj TEEPARTS este mai eficient pentru cazurile considerate.

Deși putem vorbi despre reduceri de energie asemănătoare între cele două tehnici de eficientizare a consumului, doar TEEPARTS bazat pe mecanismul de planificare FENP oferă suport pentru execuția taskurilor perfect periodică (i.e. fără jitter de lansare în execuție a taskurilor).

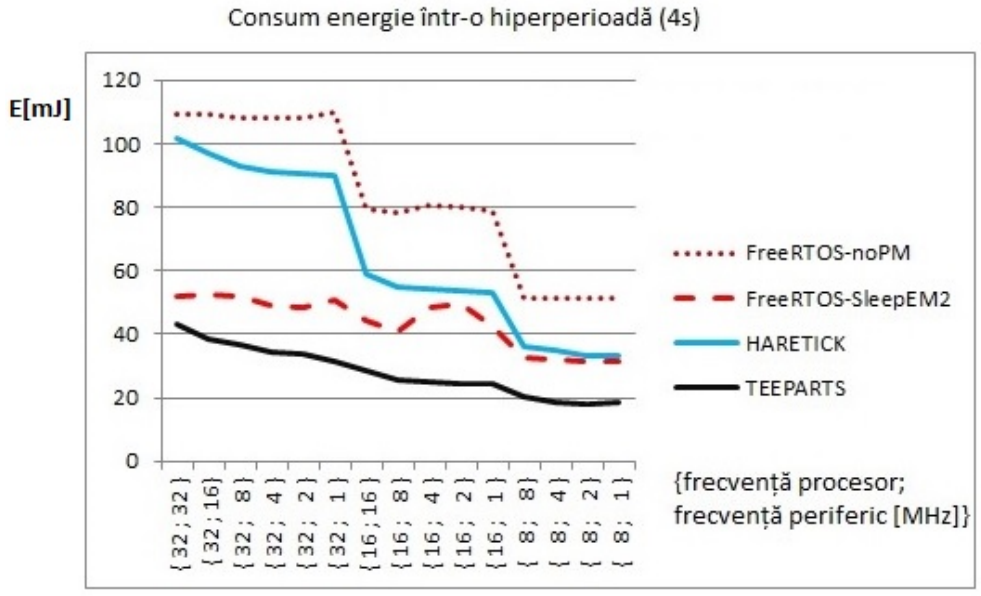


Figura 39. Consum energie într-o hiperperioadă

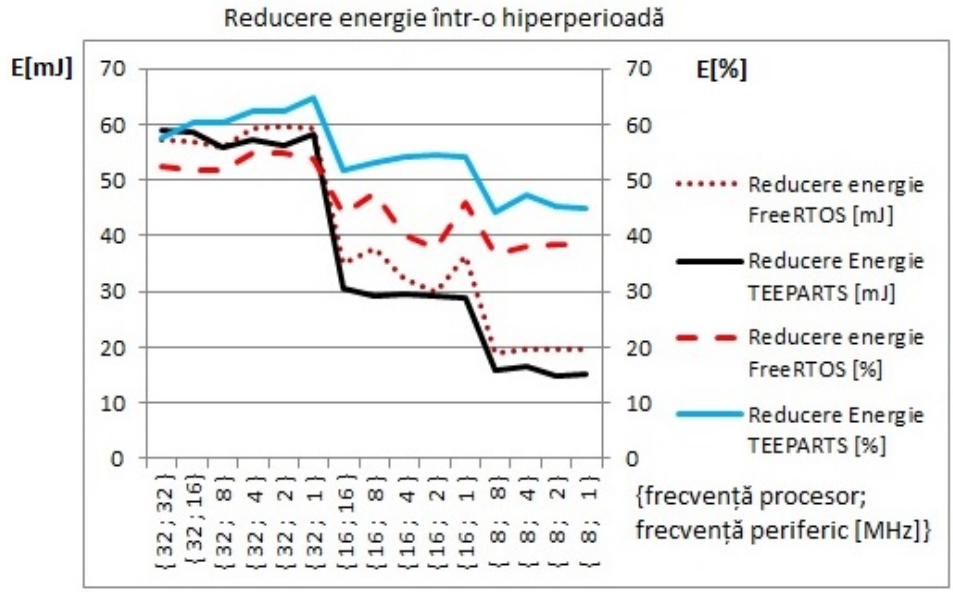


Figura 40. Reducere comparativă de energie între FreeRTOS și TEEPARTS

În cele ce urmează vor fi prezentate câteva rezultate ale analizei de performanță a algoritmului TEEPARTS, realizate cu ajutorul simulatorului TEEPARTSsim. În Figura 41 este reprezentă evoluția performanțelor de reducere a consumului de energie electrică în funcție de factorul de utilizare procesor (U).

În acest caz U este exprimat în procente și este definit ca fiind:

$$U = \sum_{i=0}^{n} \frac{C_i}{T_i} \cdot 100 \qquad (6\text{-}2)$$

unde C_i reprezintă timpul de execuție al taskului *i,* iar T_i reprezintă perioada acestuia.

După cum era de așteptat performanța, dată de procentajul cu care este redusă energia folosind algoritmul TEEPARTS, scade o dată cu creșterea factorului de utilizare procesor, totuși această scădere nu este una extremă, ceea ce face ca acest algoritm să fie eficient chiar și la valori ale utilizării procesor de peste 40%.

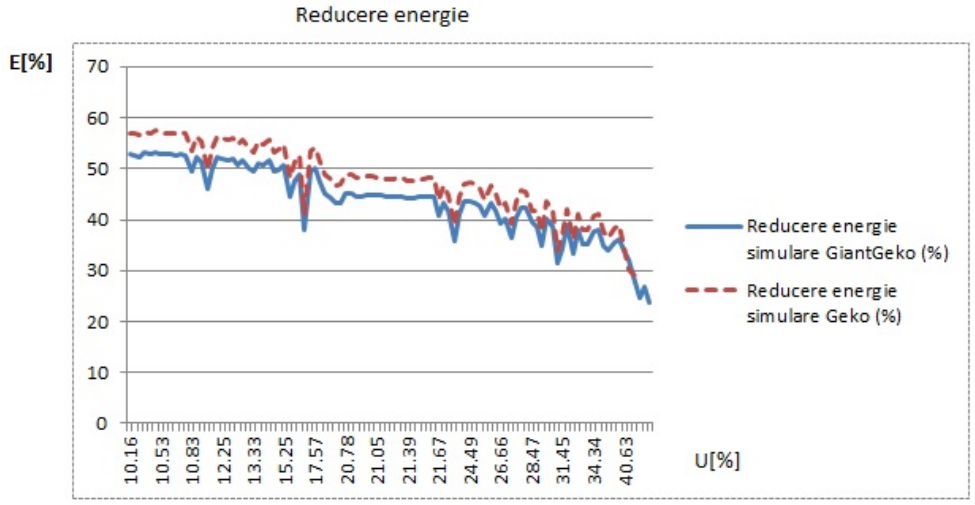


Figura 41. Reducerea consumului de energie în funcție de utilizarea procesor

6.4 Concluzii

În acest capitol este prezentată validarea algoritmului TEEPARTS prin implementarea acestuia pe două platforme țintă reprezentate de către EFM32-G8xx-STK și EFM32GG-STK3700.

Tot în acest capitol este prezentat un sistem folosit pentru a analiza performanțele algoritmului de planificare din punct de vedere al memoriei utilizate și al impactului asupra comportării temporale a sistemului de operare peste care a fost implementat. Rezultatele obținute în urma acestei analize sunt prezentate și sintetizate într-un mic tabel.

Acest algoritm a fost analizat și din punct de vedere al performanțelor asupra reducerii de energie, pe de o parte cu ajutorul modulului de determinare a parametrilor

energetici, pe baza unor studii de caz; iar pe de altă parte prin realizarea unor simulări cu ajutorul unui mediu de simulare TEEPARTSsim, dezvoltat în cadrul program de cercetare.

Pentru cazurile considerate s-au obținut reduceri de energie între 50 și 66%, determinate în mod direct prin măsurători. Iar, cu ajutorul simulatorului s-a obținut o reducere de energie între 25 și 60%, pentru o variație a factorului de utilizare procesor între 10 și 41%.

Principalul rol al acestui capitol este de a prezenta validarea algoritmului pe platformele considerate și de a prezenta comportamentul aplicațiilor planificate cu acest algoritm ce rulează pe platforme reale. Simulatorul are doar rolul de a face estimări (pesimiste) asupra unor eventuale scenarii de execuție pentru care rularea pe platforma propriu zisă nu este la îndemână sau este anevoioasă.

Prin cele prezentate a fost îndeplinit și obiectivul final al cercetării doctorale și anume: *validarea mecanismului de planificare, prin execuția aplicației pe sistemul țintă și realizarea unei analize a comportării aplicației timp-real direct pe sistem*.

7 Concluzii și perspective

Motto: *"Scopul gândirii este să ajute la obținerea unei concluzii, să proiecteze încheierea posibilă pe baza datelor existente."* ***John Dewey***

7.1 Concluzii

Lucrarea de față sintetizează activitatea de cercetare și dezvoltare efectuată în cadrul programului de cercetare doctorală cu titlul "Tehnici de planificare pentru sisteme timp real cu funcție de eficientizare a consumului de energie electrică", efectuat de către mine sub conducerea științifică a domnului profesor dr. ing. Vladimir I. Crețu.

Teza de față abordează o serie de domenii conexe, dintre care principalele două sunt reprezentate de către: *cel al sistemelor timp-real* și cel al *sistemelor încorporate.*

Tema cercetării cuprinse în această teză este cea *a cadrelor suport și a metodologiilor de reducere a energiei electrice în sistemele timp-real încorporate*.

Obiectivele finale ale sistemului cadru propus în urma acestei cercetări și ale tezei de doctorat, sunt, așa cum apar în subcapitolul 1.3 :

1. *Conceperea și dezvoltarea unui model de consum de energie la nivel de sistem pentru o serie de clase de dispozitive timp-real încorporate, definite în prealabil.*
2. *Conceperea și dezvoltarea unui cadru, reprezentând un mediu hardware și software și a unei metodologii aferente, pentru măsurarea și determinarea parametrilor modelului consumului de energie de la punctul precedent.*
3. *Dezvoltarea unui mecanism de planificare, pe mai multe niveluri, pentru sisteme timp-real încorporate, cu funcție de eficientizare a consumului de energie electrică.*
4. *Validarea mecanismului de planificare, prin execuția aplicației pe sistemul țintă și realizarea unei analize a comportării aplicației timp-real direct pe sistem.*

Primele două obiective au fost atinse prin conceperea și dezvoltarea unui sistem cadru, cu mai multe componente, și a unui model de consum pentru o serie de platforme țintă vizate, pentru care s-au prezentat o serie de specificații. Această activitate a fost prezentată în capitolul 4

Pentru prezentarea atingerii obiectivului al treilea a fost dedicat un capitol separat (capitolul 5), acest obiectiv reprezentând principalul scop al acestei cercetări. În acest capitol este astfel propus și detaliat un algoritm de planificare pentru taskuri timp-real critice (și nu numai), cu funcție de eficientizare a consumului de energie electrică, algoritm dezvoltat pe două niveluri, primul constituit dintr-o componentă statică implementată pe un PC, iar al doilea constituit dintr-o componentă dinamică implementată direct pe platforma țintă.

Capitolul 6 este dedicat ultimului obiectiv, cel al validări mecanismului de planificare, prin execuția acestuia pe o platformă țintă reală, dar și prin analiza performanțelor mecanismului de planificare. Pentru cazurile considerate s-au obținut reduceri de energie între 50 și 66%, determinate în mod direct prin măsurători. Iar, cu

ajutorul simulatorului s-a obținut o reducere de energie între 25 și 60%, pentru o variație a factorului de utilizare procesor între 10 și 41%.

7.2 Sinteza contribuțiilor

În cele ce urmează va fi prezentat un scurt rezumat al contribuțiilor cuprinse în acest referat, structurate la nivel de capitole:

- În Capitolul 3 :
  - este realizată o analiză comparativă a principalelor metode existente de eficientizare a consumului de energie electrică prin mecanisme de control software pentru sisteme timp-real încorporate, evidențiind totodată impactul pe care acestea îl pot avea asupra sistemelor timp-real critice, activitate ce a fost parțial publicată și în [A2].
- În Capitolul 4 :
  - este definit și specificat sistemul țintă pentru care este propus un model de comportament temporal și energetic al sistemului la nivel de task.
  - este propus un sistem de analiză energetică și temporală a comportamentului unui sistem țintă pe parcursul execuției fiecărui task al unei aplicații, format dintr-un un mediu de analiză a consumului de energie pe baza unui model la nivel de task, ce este publicat, într-o formă preliminară, în [A8].
  - sunt propuse două variante de extensie ale unui sistem de operare numit HARETICK ce constituie două medii de execuție și planificare hibride a căror propunere și implementare pe platforme fizice au fost publicate în [A1] și [A3].
  - este prezentată integrarea unei serii de module de măsurare, de planificare și analiză a comportamentului energetic și temporal al aplicațiilor timp-real ce rulează pe platforma țintă, într-un sistem cadru mai general numit TEEARTS. Acest sistem fiind propus ca un sistem unitar într-o formă originală.
- În Capitolul 5 :
  - este propus și dezvoltat un algoritm de planificare numit TEEPARTS pentru taskuri timp-real critice (și nu numai), cu funcție de eficientizare a consumului de energie electrică, algoritm dezvoltat pe două niveluri, primul constituit dintr-o componentă statică implementată pe un PC, iar al doilea constituit dintr-o componentă dinamică implementată direct pe platforma țintă.
- În Capitolul 6 :
  - este prezentată implementarea algoritmului TEEPARTS pe un exemplu de platformă țintă în scopul validării modelului energetic și temporal pentru taskuri și al algoritmului propus
  - este prezentat un studiu de caz și o analiză a performanțelor acestui algoritm pentru acest caz și o serie de alte cazuri derivate din primul.

7.3 Perspective de dezvoltare

În determinarea valorilor parametrilor modelului s-au folosit o serie de unelte hardware și software, neexistând un mediu integrat care să le cuprindă pe toate. De aici rezultă și o posibilă direcție de dezvoltare ulterioară: dezvoltarea unui mediu integrat care să determine atât parametrii de timp cât și cei de consum de energie pentru un cod dat la nivel de instrucție.

Pe de altă parte mediile de analiză și măsurare a parametrilor nu iau în considerare mai multe dispozitive ale unui sistem țintă în același timp. Sistemul de măsură a timpilor de execuție ia in considerare doar procesorul și eventual și accesele la memorie, dacă sunt specificate de utilizator, nu se iau în considerare perifericele, magistralele și timpii de acces la alte dispozitive care pot fi conectate la procesor. Pe de altă parte sistemele de măsurare a energiei fie iau în considerare procesorul, fie întreaga placă, dar nu fiecare dispozitiv în parte. De aici rezultă o altă posibilă direcție de dezvoltare.

Algoritmul de reducere a energiei electrice vizează doar o serie de algoritmi de planificare, modelul taskurilor poate fi extins și poate fi folosit și la dezvoltarea unor alți algoritmi de reducere a energiei electrice, pe baza altor algoritmi de planificare, decât cei statici, *non-preemptivi*.

Referințe bibliografice

[1] V. Weber, "Smart Sensor Networks: Technologies and Applications for Green Growth," *OECD Digital Economy Papers* vol. 167, no. December 2009.

[2] M. Marcu, C. Stangaciu, A. Topirceanu, D. Volcinschi, and V. Stangaciu, "Wireless Sensors Solution for Energy Monitoring, Analyzing, Controlling and Predicting," in *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. vol. 57, G. Par and P. Morrow, Eds., ed: Springer Berlin Heidelberg, 2011, ISBN: 978-3-642-23583-2, pp. 1-19.

[3] B. S. Popescu, A. Stancovici, V. Stangaciu, C. Certejan, and M. Marcu, "Intelligent Wireless Distributed Network for Power Consumption Monitoring and Analysis," in *Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference* 2009, pp. 730-735.

[4] J. Daintith. (2004, May). *"real-time system." A Dictionary of Computing*. Available: http://www.encyclopedia.com/doc/1O11-realtimesystem.html

[5] M. V. Micea, V. Cretu, and V. Groza, "Predictable Signal Generation with the Hard Real-Time Operating Kernel HARETICK," in *Instrumentation and Measurement Technology Conference, IMTC* Proceedings of the IEEE 2005, pp. 2097-2102.

[6] G. Buttazzo, "Research trends in real-time computing for embedded systems," *SIGBED Rev.,* vol. 3, no. 3, pp. 1-10, 2006.

[7] M. V. Micea, "Proiectarea sistemelor timp-real pentru aplicatii critice de achizitie si prelucrare numerica de semnal," PhD Thesis, Departament of Computer Science and Engineering, "Politehnica" University Timisoara, 2004.

[8] J. Liu, *Real-Time Systems*. New Jersey - United States of America: Prentice Hall, 2000, ISBN:

[9] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo, *Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms*. Boston / Dordrecht / London: Kluwer Academic Publishers, 1998, ISBN:

[10] Renesas Electronics Corporation. (2010, June). *R8C Family General RTOS Concepts*. Available: http://documentation.renesas.com/doc/products/tool/apn/res05b0008_r8cap.pdf

[11] M. V. Micea, *Proiectarea sistemelor timp-real pentru aplicatii critice*. Timisoara: Editura Orizonturi Universitare, 2005, ISBN:

[12] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *Real-Time Systems Symposium*, Proceedings., Twelfth 1991, pp. 129-139.

[13] M. Bertogna, G. Buttazzo, M. Marinoni, G. Yao, F. Esposito, and M. Caccamo, "Preemption points placement for sporadic task sets," in *Real-Time Systems (ECRTS), 2010 22nd Euromicro Conference on*, 2010, pp. 251-260.

[14] E. Bini and G. C. Buttazzo, "Measuring the performance of schedulability tests," *Real-time systems,* vol. 30, no. 1-2, pp. 129-154, 2005.

[15] L. George, Nicolas Rivierre, and M. Spuri., "Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling - Raport de Recherche," Institut National de Recherche en Informatique et en Automatique 1996.

[16] E. Bini, T. H. C. Nguyen, P. Richard, and S. K. Baruah, "A Response-Time Bound in Fixed-Priority," *IEEE Transactions on Computers,* vol. 58, no. February 2009.

[17] G. Buttazzo and P. Gai, "Efficient EDF Implementation for Small Embedded Systems," in *OSPERT 2006 - Workshop on Operating Systems Platforms for Embedded Real-Time applications*, Dresden - Germany, 2006.

[18] A. Sravan, S. Kundu, and A. Pal, "Low Power Sensor Node for a Wireless Sensor Network," in *VLSI Design, 2007. Held jointly with 6th International Conference on Embedded Systems., 20th International Conference on*, 2007, pp. 445-450.

[19] F. J. Atero, J. J. Vinagre, E. Morgado, and M. R. Wilby, "A Low Energy and Adaptive Architecture for Efficient Routing and Robust Mobility Management in Wireless Sensor Networks," in *Distributed Computing Systems Workshops, 31st International Conference on, ICDCSW*, 2011, pp. 172-181.

[20] J. V. Capella, A. Bonastre, R. Ors, and J. J. Serrano, "A Pollution Monitoring System Based on an Energy-Efficient Wireless Sensor Networks Architecture," in *17th Telecommunications Forum, TELFOR* Serbia, Belgrade, 2009.

[21] J. V. Capella, A. Bonastre, J. J. Serrano, and R. Ors, "A New Robust, Energy-efficient and Scalable Wireless Sensor Networks Architecture Applied to a Wireless Fire Detection System," in *Wireless Networks and Information Systems,International Conference on, WNIS*, 2009, pp. 395-398.

[22] V. Stangaciu, M. V. Micea, and V. I. Cretu, "Low-level communication time analysis in real-time wireless sensor networks," in *Robotic and Sensors Environments (ROSE), 2014 IEEE International Symposium on*, 2014, pp. 83-87.

[23] M. Stratulat, *Circuite integrate digitale*. Timisoara: Editura Politehnica, 2004, ISBN: 973-625-161-6.

[24] C. M. Krishna and Y.-H. Lee, "Voltage-Clock-Scaling Adaptive Scheduling Techniques for Low Power in Hard Real-Time Systems," *Computers, IEEE Transactions on,* vol. 52, no. 12, pp. 1586- 1593, December 2003.

[25] O. S. Unsal and I. Koren, "System-Level Power-Aware design Techniques in Real-Time Systems," Proc. IEEE, Special Issue on Real-Time Systems, 2003, pp. 1055–1069.

[26] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Determining Optimal Processor Speeds for Periodic Real-Time Tasks with Different Power Characteristics," in *Real-Time Systems, 13th Euromicro Conference on*, 2001, pp. 225- 232.

[27] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined Dynamic Voltage Scaling and Adaptive Body Biasing for Lower Power Microprocessors under Dynamic Workloads," in *Computer Aided Design, 2002. ICCAD 2002. IEEE/ACM International Conference on* 2002, pp. 721- 725.

[28] J. Wang, B. Ravindran, and T. Martin, "A Power-Aware, Best-Effort Real-Time Task Scheduling Algorithm," in *Software Technologies for Future Embedded Systems. IEEE Workshop on*, 2003, pp. 21- 28.

[29] H. Aydin, R. Melhem, and D. Mosse, "Periodic Reward-Based Scheduling and Its Application to Power-Aware Real-Time Systems," in *Handbook of Scheduling: Algorithms, Models and Performance Analysis*, J. Leung, Ed., ed: CRC Press, 2004, ISBN: pp. 36.1 - 36.17.

[30] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Power-Aware Scheduling for Periodic Real-Time Tasks," *Computers, IEEE Transactions on,* vol. 53, no. 5, pp. 584- 600, May 2004.

[31] F. Bellosa, "The Benefits of Event-Driven Accounting in Power-Sensitive Systems," Proceedings of the SIGOPS European Workshop, 2000.

[32] R. Xu, D. Mosse, and R. Melhem, "Minimizing Expected Energy Consumption in Real-Time Systems Through Dynamic Voltage Scaling," *ACM Trans. Comput. Syst.,* vol. 25, no. 4, Article:9, 2007.

[33] C. Scordino and G. Lipari, "Using Resource Reservation Techniques for Power-Aware Scheduling," Proceedings of the 4th ACM international conference on Embedded software (EMSOFT '04), New York, USA, 2004, pp. 16-25.

[34] (2005, February). *ADSP-BF537 Blackfin® Processor Hardware Reference, Revision 2.0*. Available: http://www.analog.com/en/processors-dsp/blackfin/processors/manuals/resources/index.html

[35] A. Andrei, P. Eles, Z. Peng, M. T. Schmitz, and B. M. A. Hashimi, "Energy Optimization of Multiprocessor Systems on Chip by Voltage Selection," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 15, no. 3, pp. 262-275, March 2007.

[36] R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "Mitigating Parameter Variation with Dynamic Fine-Grain Body Biasing," in *Microarchitecture, 2007. MICRO 2007. 40th Annual IEEE/ACM International Symposium on*, 2007, pp. 27-42.

[37] W. Kim, D. Shin, H.-S. Yun, J. Kim, and S. L. Min, "Performance Comparison of Dynamic Voltage Scaling Slgorithms for Hard Real-Time Systems," in *Real-Time and Embedded Technology and Applications Symposium*, Proceedings. Eighth IEEE 2002, pp. 219- 228.

[38] Z. Lu, Y. Zhang, M. Stan, K. Lach, and K. Skadron, "Procrastinating Voltage Scheduling with Discrete Frequency Sets," in *Design, Automation and Test in Europe. DATE*, Proceedings, 2006, pp. 6-10.

[39] S. Mohan, F. Mueller, M. Root, W. Hawkins, C. Healy, D. Whalley, and E. Vivancos, "Parametric Timing Analysis and its Application to Dynamic Voltage Scaling," *ACM Trans. Embed. Comput. Syst. ,* vol. 10, no. 2, Article:25, January 2011.

[40] R. Xu, R. Melhem, and D. Mossé, "A Unified Practical Approach to Stochastic DVS Scheduling," in *Proceedings of the 7th ACM & IEEE international conference on Embedded software (EMSOFT '07)*, New York, USA, 2007, pp. 37-46.

[41] Y. Shin, K. Choi, and T. Sakurai, "Power Optimization of Real-Time Embedded Systems on Variable Speed Processors," in *Computer Aided Design, 2000. ICCAD-2000. IEEE/ACM International Conference on*, 2000, pp. 365-368.

[42] W.-C. Kwon and T. Kim, "Optimal Voltage Allocation Techniques for Dynamically Variable Voltage Processors," *ACM Trans. Embed. Comput. Syst. ,* vol. 4, no. 1, pp. 211-230, February 2005.

[43] F. Yao, A. Demers, and S. Shenker, "A Scheduling Model for Reduced CPU Energy," in *Foundations of Computer Science. Proceedings., 36th Annual Symposium on*, 1995, pp. 374-382.

[44] T. Ishihara and H. Yasuura, "Voltage Scheduling Problem for Dynamically Variable Voltage Processors," in *Proceedings of the 1998 International Symposium on Low Power Electronics and Design (ISLPED '98)*, Monterey, CA, 1998.

[45] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez, "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems," in *Real-Time Systems Symposium, RTSS* Proceedings. 22nd IEEE 2001, pp. 95- 105.

[46] Y. Zhu and F. Mueller, "Exploiting synchronous and asynchronous DVS for feedback EDF scheduling on an embedded platform," *ACM Trans. Embed. Comput. Syst.,* vol. 7, no. 1, Article:3, December 2007.

[47] D.-R. Chen;, C.-C. Hsu;, and M.-F. Lai;, "Time-Efficient Power-Aware Scheduling for Periodic Real-Time Tasks," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1-8.

[48] T. Zitterell and C. Scholl, "A Probabilistic and Energy-Efficient Scheduling Approach for Online Application in Real-Time Systems," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, 2010.

[49] P. Mejia-Alvarez, E. Levner, and D. Mosse, "Adaptive Scheduling Server for Power-Aware Real-Time Tasks," *ACM Trans. Embed. Comput. Syst. ,* vol. 3, no. 2, pp. 284-306, May 2004.

[50] W. Kim, J. Kim, and S. Lyul-Min, "Dynamic Voltage Scaling Algorithm for Fixed-Priority Real-Time Systems Using Work-Demand Analysis," in *Low Power Electronics and Design. ISLPED. Proceedings of the 2003 International Symposium on*, 2003, pp. 396- 401.

[51] J.-J. Chen, "Expected Energy Consumption Minimization in DVS Systems with Discrete Frequencies," Proceedings of the 2008 ACM

symposium on Applied computing (SAC '08), New York, USA, 2008, pp. 1720-1725.

[52] A. Soria-Lopez, P. Mejia-Alvarez, and J. Cornejo, "Feedback Scheduling of Power-Aware Soft Real-Time Tasks," in *Computer Science, 2005. ENC 2005. Sixth Mexican International Conference on*, 2005, pp. 266-273.

[53] G. Sudha Anil Kumar, G. Manimaran, and Z. Wang, "Energy-Aware Scheduling of Real-Time Tasks in Wireless Networked Embedded Systems," in *Real-Time Systems Symposium. RTSS. 28th IEEE International* 2007, pp. 15-24.

[54] B. Mochocki, X. S. Hu, and G. Quan, "A Realistic Variable Voltage Scheduling Model for Real-Time Applications," in *Computer Aided Design. ICCAD. IEEE/ACM International Conference on*, 2002, pp. 726-731.

[55] S. Saewong and R. Rajkumar, "Practical Voltage-Scaling for Fixed-Priority RT-Systems," in *Real-Time and Embedded Technology and Applications Symposium*, Proceedings. The 9th IEEE 2003, pp. 106-114.

[56] J. Pouwelse, K. Langendoen, and H. Sips, "Energy Piority Scheduling for Variable Voltage Processors," in *Low Power Electronics and Design, International Symposium on*, 2001, pp. 28-33.

[57] R. Jejurikar and R. Gupta, "Energy Aware Task Scheduling with Task Synchronization for Embedded Real Time Systems," Proceedings of International Conference on Compilers, Architecture and Synthesis for Embedded Systems, Grenoble, France, 2002.

[58] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Adaptive Dynamic Power Management for Hard Real-Time Systems," in *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, 2009, pp. 23-32.

[59] L. Yuan and G. Qu, "ALT-DVS: Dynamic Voltage Scaling with Awareness of Leakage and Temperature for Real-Time Systems," in *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, 2007, pp. 660-670.

[60] P. Chowdhury and C. Chakrabarti, "Static task-scheduling algorithms for battery-powered DVS systems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 13, no. 2, pp. 226-237, February 2005.

[61] S. Lee, I. Shin, W. Kim, I. Lee, and S. L. Min, "A Design Framework for Real-Time Embedded Systems with Code Size and Energy Constraints," *ACM Trans. Embedd. Comput. Syst. ,* vol. 7, no. 2, Article:18, February 2008.

[62] J. Lemieux. (2003, June). *Introduction to ARM Thumb*. Available: http://www.eetimes.com/discussion/other/4024632/Introduction-to-ARM-thumb

[63] T. Lirong and T. Arslan, "An evolutionary power management algorithm for SoC based EHW systems," in *Evolvable Hardware, 2003. Proceedings. NASA/DoD Conference on*, 2003, pp. 117-124.

[64] B. Zhao and H. Aydin, "Minimizing Expected Energy Consumption Through Optimal Integration of DVS and DPM," in *Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009. IEEE/ACM International Conference on*, 2009, pp. 449-456.

[65] M. K. Bhatti, C. Belleudy, and M. Auguin, "Hybrid Power Management in Real Time Embedded Systems: an Interplay of DVFS and DPM Techniques," *Real-Time Syst.,* vol. 42, no. 2, March 2011.

[66] P. Rong and M. Pedram, "Power-Aware Scheduling and Dynamic Voltage Setting for Tasks Running on a Hard Real-Time System," in *Design Automation, 2006. Asia and South Pacific Conference on*, Proceedings ASPDAC, 2006, pp. 473–478.

[67] R. Jejurikar and R. Gupta, "Procrastination Scheduling in Fixed Priority Real-Time Systems," Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems (LCTES '04), New York, USA, 2004, pp. 57-66.

[68] L. Yuan, S. Leventhal, and G. Qu;, "Temperature-Aware Leakage Minimization Technique for Real-Time Systems," in *Computer-Aided Design, 2006. ICCAD '06. IEEE/ACM International Conference on*, 2006, pp. 761-764.

[69] K. Choi, W. Lee, R. Soma, and M. Pedram, "Dynamic Voltage and Frequency Scaling Under a Precise Energy Model Considering Variable and Fixed Components of the System Power Dissipation," in *Computer Aided Design, 2004. ICCAD-2004. IEEE/ACM International Conference on*, 2004, pp. 29- 34.

[70] B. Zhang, R. Simon, and H. Aydin, "Harvesting-aware energy management for time-critical wireless sensor networks with joint voltage and modulation scaling," *Industrial Informatics, IEEE Transactions on,* vol. 9, no. 1, pp. 514-526, 2013.

[71] K. Huang, L. Santinelli, J.-J. Chen, L. Thiele, and G. C. Buttazzo, "Applying Real-Time Interface and Calculus for Dynamic Power Management in Hard Real-Time Systems," *Real-Time Syst. ,* vol. 42, no. 2, March 2011.

[72] H. Cheng and S. Goddard, "EEDS_NR: An Online Energy-Efficient I/O Device Scheduling Algorithm for Hard Real-Time Systems with Non-preemptible Resources," in *Real-Time Systems, 2006. 18th Euromicro Conference on*, 2006, pp. 260-270.

[73] H. Cheng and S. Goddard, "Online energy-aware I/O device scheduling for hard real-time systems," presented at the Proceedings of the conference on Design, automation and test in Europe: Proceedings, Munich, Germany, 2006.

[74] V. Swaminathan and K. Chakrabarty, "Pruning-based energy-optimal device scheduling for hard real-time systems," presented at the Proceedings of the tenth international symposium on Hardware/software codesign, Estes Park, Colorado, 2002.

[75] V. Swaminathan and K. Chakrabarty, "Energy-conscious, deterministic I/O device scheduling in hard real-time systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 22, no. 7, pp. 847-858, 2003.

[76] V. Swaminathan and K. Chakrabarty, "Pruning-based, energy-optimal, deterministic I/O device scheduling for hard real-time systems," *ACM Trans. Embed. Comput. Syst.,* vol. 4, no. 1, pp. 141-167, 2005.

[77] V. Devadas and H. Aydin, "DFR-EDF: A unified energy management framework for real-time systems," in *Real-Time and Embedded Technology and Applications Symposium (RTAS), 2010 16th IEEE*, 2010, pp. 121-130.

[78] T. L. Martin and D. P. Siewiorek, "Nonideal battery and main memory effects on CPU speed-setting for low power," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on,* vol. 9, no. 1, pp. 29-34, 2001.

[79] M. Bambagini, M. Bertogna, M. Marinoni, and G. Buttazzo, "On the Impact of Runtime Overhead on Energy-Aware Scheduling," presented at the Workshop on Power, Energy, and Temperature Aware Real-time Systems (PETARS 2013), Philadelphia, USA, 2013.

[80] L. Yuan and G. Qu, "Analysis of energy reduction on dynamic voltage scaling-enabled systems," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on,* vol. 24, no. 12, pp. 1827-1837, December 2005.

[81] D. C. Snowdon, S. M. Petters, and G. Heiser, "Power Measurement as the Basis for Power Management," in *WS Operating System Platforms for Embedded Real-Time Applications*, 2005.

[82] H. Kweon, Y. Do, J. Lee, and B. Ahn, "An Efficient Power-Aware Scheduling Algorithm in Real Time System," in *Communications, Computers and Signal Processing, 2007. PacRim 2007. IEEE Pacific Rim Conference on*, 2007, pp. 350-353.

[83] V. Konstantakos, A. Chatzigeorgiou, S. Nikolaidis, and T. Laopoulos, "Energy Consumption Estimation in Embedded Systems," *Instrumentation and Measurement, IEEE Transactions on,* vol. 57, no. 4, pp. 797-804, April 2008.

[84] Z. Gao, G. Dai, P. Liu, and P. Zhang;, "Energy-Efficient Architecture for Embedded Software with Hard Real-Time Requirements in Partial Reconfigurable Systems," in *Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conference on*, 2009, pp. 387-392.

[85] G. Callou, P. Maciel, E. Tavares, E. Andrade, B. Nogueira, C. Araujo, and P. Cunha, "Energy Consumption and Execution Time Estimation of Embedded System Applications," *Microprocess. Microsyst,* vol. 35, no. 4, pp. 426-440, 2011.

[86] G. Zeng, H. Tomiyama, and H. Takada, "A Generalized Framework for Energy Savings in Hard Real-Time Embedded Systems," *IPSJ Trans. on System LSI Design Methodology,* vol. 2, no. pp. 167-179 2009.

[87] P. Ranganathan, S. Rivoire, and J. D. Moore, "Models and Metrics for Energy-Efficient Computing," in *Advances in Computers*. vol. 75, M. V. Zelkowitz, Ed., ed Burlington: Academic Press, 2009, ISBN: 978-0-12-374810-2, pp. 159-233.

[88] Embedded Microprocessor Benchmark Consortium (EEMBC). (February). *Energy-Bench TM version 1.0 power/energy benchmarks*. Available: http://www.eembc.org/benchmark/power_sl.php

[89] M. Marcu, B.Popescu, A. Stancovici, V. Stangaciu, and C.Certejan, "Power Characterization of Electric, Electronic and Computing Devices," *Scientific Bulletin of the Electrical Engineering Faculty,* vol. 1/2009, no. , 2009.

[90] C. S. Stangaciu, M. V. Micea, and V. I. Cretu, "Energy efficiency in real-time systems: A brief overview," in *Applied Computational Intelligence and Informatics (SACI), 2013 IEEE 8th International Symposium on*, 2013, pp. 275-280.

[91] J. A. Stankovic and K. Ramamritham, "What is predictability for real-time systems?," *Real-time systems,* vol. 2, no. 4, pp. 247-254, 1990.

[92] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks,* vol. 7, no. 3, pp. 537-568, 2009.

[93] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava, "Energy-Aware Wireless Microsensor Networks " *Signal Processing Magazine, IEEE,* vol. 19, no. 2, pp. 40-50, 2002.

[94] M. Bhardwaj, "Power-Aware Systems," Master of Science, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, 2001.

[95] M. A. M. Vieira, C. N. Coelho, Jr. , D. C. da Silva, Jr. , and J. M. da Mata, "Survey on wireless sensor network devices " in *Emerging Technologies and Factory Automation, 2003.*, Proceedings. ETFA '03. IEEE Conference, 2003, pp. 537-544.

[96] I. Iordan, *Dicționarul explicativ al limbii române, ediția a II-a*: Editura Univers Enciclopedic, 1998, ISBN:

[97] A. Devices, "Small and Thin ±5 g Accelerometer ADXL320," A. Devices, Ed., ed, 2004.

[98] ST, "LSM303DLH Sensor module: accelerometer and 3-axis magnetometer," ST, Ed., ed, 2009.

[99] W. Qin, M. Hempstead, and W. Yang, "A Realistic Power Consumption Model for Wireless Sensor Network Devices," in *Sensor and Ad Hoc Communications and Networks, 2006. SECON '06. 2006 3rd Annual IEEE Communications Society on*, 2006, pp. 286-295.

[100] T. Instruments, "CC1101 Low-Power Sub-1 GHz RF Transceiver," T. Instruments, Ed., ed.

[101] D. Lymberopoulos and A. Savvides, "XYZ: a motion-enabled, power aware sensor node platform for distributed sensor network applications," in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, 2005, pp. 449-454.

[102] E. Bini, G. Buttazzo, and G. Lipari, "Minimizing CPU energy in real-time systems with discrete speed management," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 8, no. 4, p. 31, 2009.

[103] YEInternational. *MS8050 High Accuracy 53000 Counts Bench Model Multimeter*. Available: http://www.yeint.fi/files/products/MASTECH_MS8050.pdf

[104] Hameg Instruments. *Quadruple High-Performance. Power Supply HM7044* Available: http://www.teknetelectronics.com/DataSheet/HAMEG/WEBHAMEGHM7044.pdf

[105] Silicon Labs. (December). *Simplicity Studio*. Available: http://www.silabs.com/products/mcu/Pages/simplicity-studio.aspx/simplicity-studio

[106] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, and T. Mitra, "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Transactions on Embedded Computing Systems (TECS),* vol. 7, no. 3, p. 36, 2008.

[107] M. V. Micea, "HARETICK: A real-time compact kernel for critical applications on embedded platforms," in *Proc. 7th Intl. Conf. Development and Applic. Syst., DAS*, 2004, pp. 16-23.

[108] M. V. Micea and V. I. Cretu, "Highly predictable execution support for critical applications with HARETICK kernel," *AEU - International Journal of Electronics and Communications,* vol. 59, no. 5, pp. 278-287, 2005.

[109] M. V. Micea, C. S. Stangaciu, V. Stangaciu, and V. I. Cretu, "Improving the efficiency of highly predictable wireless sensor platforms with hybrid scheduling," in *Robotic and Sensors Environments (ROSE), 2012 IEEE International Symposium on*, 2012, pp. 73-78.

[110] R. Barry, "FreeRTOS Reference Manual," *Real Time Engineers Ltd,* no. 2011.

[111] S. Baskiyar and N. Meghanathan, "A survey of contemporary real-time operating systems," *Informatica (Slovenia),* vol. 29, no. 2, pp. 233-240, 2005.

[112] S. Deshmukh and N. Mhala, "Comparison of Open Source RTOSs Using Various Performance Parameters," *International Journal of Electronics Communication and Computer Engineering,* vol. 4, no. 2, pp. 86-91, 2013.

Publicații

Publicații indexate ISI:

[A1].**C. S. Stangaciu**, M. V. Micea, and V. I. Cretu, "Hard Real-Time Execution Environment Extension for FreeRTOS," *in Robotic and Sensors Environments (ROSE), 2014 IEEE International Symposium on*, 2014.
[A2].**C. S. Stangaciu**, M. V. Micea, and V. I. Cretu, "Energy efficiency in real-time systems: A brief overview," in *Applied Computational Intelligence and Informatics (SACI), 2013 IEEE 8th International Symposium on*, 2013, pp. 275-280.
[A3]. M. V. Micea, **C. S. Stangaciu**, V. Stangaciu, and V. I. Cretu, "Improving the efficiency of highly predictable wireless sensor platforms with hybrid scheduling," in Robotic and Sensors Environments (ROSE), 2012 IEEE International Symposium on, 2012, pp. 73-78.
[A4]. M. V. Micea, V. Stangaciu, **C. Stangaciu**, and C. Filote, "Sensor-Level Real-Time Support for XBee-Based Wireless Communication," *in Advances in Intelligent and Soft Computing*. vol. 145, F. L. Gaol and Q. V. Nguyen, Eds., ed Berlin, Germany: Springer Berlin Heidelberg, 2012, ISBN: 978-3-642-28307-9, pp. 147 - 154.
[A5].B.S. Popescu, A. Stancovici, V. Stangaciu, **C. Certejan**, M. Marcu, "Intelligent wireless distributed network for power consumption monitoring and analysis"*, Science and Technology for Humanity (TIC-STH), 2009 IEEE Toronto International Conference* , vol., no., pp.730-735, 26-27 Sept. 2009
[A6].M.V. Micea, **C. Certejan**, V. Stangaciu, R. Cioarga, V. Cretu, E. Petriu,, "Inter-task communication and synchronization in the hard real-time compact kernel HARETICK," *Robotic and Sensors Environments, 2008. ROSE 2008. International Workshop on* , vol., no., pp.19-24, 17-18 Oct. 2008

Patent indexat ISI:

[A7]. M. G. Marcu, A. Stancovici, V. Stangaciu, **C. Stangaciu**, A. Topirceanu, B. Popescu, D. Volcinschi, and S. O. Fuicu, "SYSTEM FOR MEASURING AND ANALYZING THE ENERGY CONSUMPTION IN ELECTRIC DEVICES BY USING CONSUMPTION SIGNATURES," RO127698-A0.

Publicații indexate BDI:

[A8].**C. S. Stangaciu**, M. V. Micea, and V. I. Cretu, "Analysis and Improvements in Energy Consumption Models for RTS," acceptat pentru publicare in *IEEE 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics (SACI 2015)*, 2015.
[A9].M. Marcu, **C. Stangaciu**, A. Topirceanu, D. Volcinschi, and V. Stangaciu, "Wireless Sensors Solution for Energy Monitoring, Analyzing, Controlling and Predicting," *in Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. vol. 57, G. Par and P. Morrow, Eds., ed: Springer Berlin Heidelberg, 2011, ISBN: 978-3-642-23583-2, pp. 1-19.
[A10]. M. Marcu, B. Popescu, A. Stancovici, V. Stangaciu, **C. Certejan**, "Power Characterization of Electric, Electronic and Computing Devices*", Scientific Bulletin of the Electrical Engineering Faculty*, 1/2009