

POLITEHNICA University of Timisoara
Department of Computer Science & Engineering
Digital Signal Processing Laboratories – DSP Labs
2, V. Parvan Bv., 300223 – Timisoara, ROMANIA
Tel: +40 256 403271, Fax: + 40 256 403214
Web: <http://dsplabs.utt.ro>



Mihai V. MICEA

**Proiectarea și implementarea
sistemelor timp-real
pentru aplicații critice de achiziție
și prelucrare numerică de semnal**

Teză de doctorat

**Conducător științific
Prof. dr. ing. Vladimir CREȚU**

2004

*Celor două familii ale mele:
Cea de acasă și cea de la școală.*

Cuvânt înainte

Mulțumesc domnului prof. dr. ing. Vladimir Crețu, conducătorul științific de doctorat, pentru ajutorul și îndrumarea continuă și plină de răbdare, acordate nu numai pe parcursul urcușului lung și abrupt al activității mele de doctorat, ci în toți anii de excelentă colaborare. Mai mult decât profesor și coleg, domnul Crețu reprezintă pentru mine un adevărat prieten.

Activitatea mea de cercetare la doctorat a beneficiat de sprijinul prețios al multor colegi din cadrul Universității "Politehnica" din Timișoara și din cadrul Departamentului de Calculatoare. Recunoștința mea se îndreaptă către prof. dr. ing. Mircea Stratulat, pentru întregul sprijin moral și material (echipamente de laborator și de măsură, documentație, etc.). Discuțiile purtate în domeniul sistemelor timp-real cu prof. dr. ing. Nicolae Robu s-au dovedit extrem de utile, fiind germeni ai unor importante idei din lucrare. Menționez de asemenea colaborarea prețioasă și sprijinul colegilor ing. Lucian Pățcaș, ing. Zsolt Husz și ing. Alexandru Joni. Lista poate continua pe multe pagini; mulțumesc tuturor colegilor, datorită cărora simt în fiecare zi că aparțin unei mari și calde familii.

Programul de colaborare dintre Departamentul Calculatoare și compania Motorola, Inc. (respectiv, Freescale, Inc.), cu Centrul DSP (MDCR) București, a reprezentat un pilon important pe care s-a sprijinit și construit activitatea mea de doctorat. Mulțumesc pe această cale domnilor dr. ing. Andreas Wild, ing. Paul Marino, ing. Daniel Broșteanu și ing. Costel Ilaș.

Deși nu e o specialistă de renume în domeniu, contribuția cea mai importantă la această teză o are soția mea Claudia, care m-a înconjurat în toți acești ani cu imensa ei dragoste, răbdare și devotament. Îi mulțumesc din suflet că a dus la capăt această muncă, umăr la umăr cu mine.

Adânca mea recunoștință se îndreaptă către părinții mei, Maria și Ioan, mai ales pentru că prin sacrificiile și dragostea lor am devenit ceea ce sunt astăzi.

În fine, dar nu în cele de pe urmă, mulțumesc bunului Dumnezeu pentru că mi-a dat norocul de a mă putea bucura în fiecare clipă de toate lucrurile minunate cu care El mă înconjoară.

Autorul,
Septembrie 2004

Abstract

Embedded systems and digital signal processing (DSP) systems are widely used in today's digital control applications, requiring in most cases real-time behavior of the hardware-software components. Many applications have a critical impact on the environment and/or on the human factor which they interact with. Examples of such applications include: modern flight control systems, fly-by-wire, auto-pilot, automotive control, industrial mechatronics, nuclear plant surveillance, and so on.

There are two essential characteristics a hardware-software platform has to meet in order to provide correct operation results for critical applications: (a) the entire process of system/application development should include the time coordinate, and (b) the system must provide maximum of predictability for the hard real-time tasks.

Although a very large number of projects have been developed in the field of real-time and embedded systems, both in the industry and the academic communities, there still are many important issues to be addressed. Three problems we consider of key importance are: (i) the large number and variety of real-time system development and analysis methodologies, correlated with the lack of their compatibility and standardization, many of them lacking also in providing proper integration of the time coordination; (ii) most current hardware-software architectures are designed and optimized to provide a good *average case* behavior, while the requirements of real-time applications specify a correct behavior of the system even in the *worst case operating conditions*; (iii) the unrestricted use of interrupts and of the associated asynchronous mechanisms and tasks in real-time systems generates severe predictability problems.

This work addresses the "Design and Implementation of Real-Time Systems for Critical Applications of Digital Signal Acquisition and Processing" topic, providing a set of homogenous solutions to the main problems stated above and to some other related issues.

A uniform set of models for real-time signals and tasks has been developed and proved, based on a proper representation of the time. The *ModX* is the resulting model for hard real-time tasks, defining modular, periodic tasks, which are scheduled and executed in a non-preemptive environment. The asynchronous events occurring in the system are processed using polling techniques and proper task scheduling.

The OPEN-HARTS system (Operating Environment for Hard Real-Time Systems) is a new methodology introduced for the development and implementation of hard real-time systems and applications, based on the previously defined set of models for time, signals and tasks. The methodology specifies the interconnection of two sub-systems: INVERTA (Integrated Visual Environment for Real-Time Application Analysis and Development) and HARETICK (Hard Real-Time Compact Kernel).

In order to provide timing guarantee and to maximize the operation predictability of critical and hard real-time applications, special task scheduling techniques must be considered with care. The thesis focuses on studying and

developing non-preemptive scheduling algorithms for hard real-time tasks, and the associated mechanisms to solve the problem of asynchronous event processing within hard real-time systems.

As a case study for the principles and mechanisms introduced and discussed with the OPEN-HARTS methodology, a test version of the HARETICK kernel has been developed. The kernel is implemented on the Motorola DSP56307EVM platform and provides a fully operational environment for applications containing both hard real-time as well as soft (non-) real-time tasks. The applications used to test the HARETICK kernel proved its correct behavior, according to their functional and timing specifications. For example, a particular application demonstrates that the kernel provides the necessary support for generating perfectly periodic output signals which satisfy all their specified parameters (period, impulse length, etc.), thus proving that the entire system can reach the maximum level of predictability – one of the main objectives of our research.

In nature, as well as in the practice of digital control applications, there are many phenomena and sequences of actions that perform without admitting interrupts. On the other hand, many interruptible operations can also be sliced in non-preemptive sub-sequences. This work proves the theoretical and practical validity of the non-preemptive techniques, used in conjunction with a proper and homogenous set of models for time, events and actions. The simulation and experimental results provided by real-time systems based on non-preemptive principles are similar with those provided by classical systems, with respect to the correctness of the operations performed. Nevertheless, when predictability and timing guarantees are needed, the non-preemptive based systems are the only ones able to operate correctly even in the *worst case conditions*.

Cuprins

SECȚIUNEA I. INTRODUCERE	1
1 Introducere	3
2 Sisteme timp-real de achiziție și prelucrare numerică de semnal	7
2.1 Sisteme de achiziție și prelucrare numerică de semnal	7
2.1.1 Sisteme de achiziții de date	8
2.1.2 Sisteme de prelucrare numerică a semnalelor	12
2.2 Sisteme timp-real	15
2.3 Timpul ca și coordonată esențială a specificării și operării STR.....	18
2.4 Tehnici de planificare	21
2.5 Exemple de sisteme timp-real.....	25
2.5.1 Sistemul Giotto.....	25
2.5.2 Sistemul Spring	28
3 Problemele dezvoltării STR stricte pentru aplicații critice de achiziție și prelucrare numerică de semnal	31
3.1 Dezvoltarea STR – concepte și arhitecturi	31
3.2 Predictibilitatea execuției task-urilor TR cu termene stricte	32
3.3 Specificarea, programarea și analiza temporală a aplicațiilor TR critice	34
3.4 Modelul unui STR strict pentru aplicații critice	35
SECȚIUNEA II. CONTRIBUȚII LA PROIECTAREA ȘI DEZVOLTAREA STR STRICTE PENTRU APLICAȚII CRITICE	37
4 Informația de timp, evenimente și acțiuni. Modelul task-ului TR strict	39
4.1 Timpul sistem și un model temporal pentru dezvoltarea STR stricte.....	39
4.2 Evenimente în sisteme TR stricte	42
4.3 Tratarea evenimentelor: acțiuni TR stricte	46
4.4 Modelul task-ului TR strict: ModX-ul.....	57
5 Planificarea aplicațiilor TR stricte	65
5.1 Introducere.....	65
5.2 Planificarea non-preemptivă a ModX-urilor independente	68
5.2.1 Modelul setului de task-uri independente	68
5.2.2 Algoritmul de planificare MLFNP	72
5.2.3 Algoritmul de planificare EDFNP	77
5.2.4 Condițiile de planificabilitate	80
5.3 Planificarea online non-preemptivă a ModX-urilor independente	85
5.3.1 Planificarea online în cicluri cu număr constant de execuții.....	86
5.3.2 Planificarea online în cicluri periodice	92

5.4 Planificarea non-preemptivă a ModX-urilor independente cu execuție fixă	98
5.4.1 Introducere	98
5.4.2 Modelul matematic al ModX-urilor cu execuție fixă	100
5.4.3 Planificarea unui set de două FModX-uri	102
5.4.4 Planificarea unui set cu un număr oarecare de FModX-uri	113
5.5 Concluzii	115
6 Evaluarea performanțelor algoritmilor de planificare non-preemptivă	117
6.1 Introducere: metodele utilizate pentru analiza performanțelor	117
6.2 Evaluarea comparativă a algoritmilor EDFNP și MLFNP	123
6.3 Evaluarea comparativă a algoritmilor de planificare online non-preemptivă	127
6.4 Evaluarea algoritmului FENP	131
6.5 Concluzii	134
7 Modelul STR strict pentru aplicații critice APNS și de control încorporat	137
7.1 Descrierea generală și caracteristicile sistemului OPEN-HARTS	137
7.2 Principiile de operare ale sistemului propus	140
SECȚIUNEA III. CONTRIBUȚII LA IMPLEMENTAREA ȘI EVALUAREA STR STRICTE	143
8 Nucleul de operare HARETICK: caracteristici generale, componente și relația dintre acestea	145
8.1 Caracteristici generale	145
8.2 Componentele HARETICK	147
8.2.1 Încărcarea și lansarea sistemului: BOOT	149
8.2.2 Inițializarea sistemului: SYSINIT	149
8.2.3 Comunicația de date: DATALINK	149
8.2.4 Interfațarea cu operatorul: MONITOR	150
8.2.5 Încărcarea unei aplicații: LOADER	150
8.2.6 Planificatorul task-urilor TR lejere: SSCD	151
8.2.7 Planificatorul ModX-urilor: HSCD	151
8.2.8 Comutarea execuției ModX-urilor: HDIS	152
8.2.9 Raportarea stării curente a sistemului: STATREPO	153
8.2.10 Subrutina de raportare a evenimentelor sistem: TiLT	154
8.3 Relația dintre componentele nucleului HARETICK	154
9 Reprezentarea și gestionarea timpului și a ModX-urilor în cadrul sistemului	157
9.1 Timpul ca parametru esențial de operare al HARETICK	157
9.2 Structurile destinate reprezentării și gestionării timpului în HARETICK	158
9.2.1 Timpul absolut sistem	158
9.2.2 Ceasul timp-real	159
9.2.3 Timpul de planificare	161

9.3 Mecanismele de sincronizare a structurilor de gestionare a timpului în HARETICK.....	163
9.4 Reprezentarea ModX-urilor în cadrul HARETICK	164
9.4.1 Reprezentarea ModX-urilor în memorie	165
9.4.2 Tabela de simboluri.....	167
9.4.3 Tabela descriptorilor de procese	167
9.5 Operarea de principiu a ModX-urilor în cadrul HARETICK.....	169
9.6 Concluzii.....	173
10 Funcționarea de principiu a sistemului	175
10.1 Contextele de execuție sistem și comutarea acestora	175
10.2 Stările ModX-urilor	181
10.3 Executivul TR al HARETICK: HDIS	185
10.3.1 Caracteristici generale.....	185
10.3.2 Elemente de implementare și analiză a codului executivului	188
10.4 Planificatorul HRT al HARETICK: HSCD.....	193
10.4.1 Principii de proiectare, implementare și operare ale HSCD	193
10.4.2 Analiza codului HSCD.....	196
10.5 Concluzii.....	200
11 Testarea și evaluarea performanțelor sistemului HARETICK	203
11.1 Testarea și analiza operării executivului HDIS	203
11.2 Aplicații pentru evaluarea performanțelor sistemului	210
11.2.1 Generarea unui semnal rectangular perfect periodic.....	211
11.2.2 Evaluarea operării ModX-urilor fantomă.....	217
11.3 Concluzii.....	219
SECȚIUNEA IV. CONCLUZII FINALE ȘI PERSPECTIVE	221
12 Concluzii finale și perspective	223
12.1 Concluzii.....	223
12.2 Studiu comparativ cu alte sisteme timp-real.....	228
12.3 Rezumat al contribuțiilor	231
12.4 Concluzii finale.....	234
12.5 Perspective de cercetare și dezvoltare	234
Bibliografie	237
Lista lucrărilor publicate	249

SECȚIUNEA I. INTRODUCERE

Secțiunea de față conține capitolele de introducere ale lucrării, în care se prezintă tema și principalele domenii abordate în cadrul tezei.

Capitolul 2 prezintă o scurtă trecere în revistă a principalelor concepte actuale în domeniul sistemelor timp-real de achiziție și prelucrare numerică de semnal (APNS) și de control digital încorporat (embedded systems).

În Capitolul 3 sunt prezentate o serie de probleme curente în domeniu, fiind evidențiată de asemenea și importanța soluționării acestora pentru ca proiectarea și implementarea sistemelor și aplicațiilor timp-real stricte să se poată efectua în concordanță cât mai strânsă cu specificațiile impuse. Preocupările din cadrul cercetărilor noastre de doctorat, prezentate în materialul de față, se axează pe introducerea și studierea unui set de soluții la aceste probleme.

1 Introducere

Teza de față prezintă contribuțiile proprii, rezultate din activitatea noastră de cercetare-dezvoltare cuprinsă în programul de doctorat cu tema "Contribuții la achiziția și prelucrarea numerică în timp-real a semnalelor utilizând sisteme de calcul distribuite", sub coordonarea științifică a prof. dr. ing. Vladimir CREȚU.

Sistemele de control digital încorporat (embedded systems) și achiziția și prelucrarea numerică a semnalelor (DSP-based systems) sunt două domenii înrudite, de interes major în preocupările actuale ale comunității academice și industriale, fapt dovedit și de numărul imens de aplicații ce integrează astfel de sisteme în aproape toate domeniile activității umane. O cerință esențială impusă acestor sisteme și aplicații este operarea în timp-real, cu garantarea respectării termenelor de timp impuse de specificațiile de proiectare și de mediu. Un număr foarte mare de aplicații au un impact critic asupra mediului înconjurător și/sau asupra factorului uman cu care interacționează. Exemple de aplicații critice de achiziție, prelucrare numerică a semnalelor și control digital încorporat sunt sistemele de control al zborului din avioanele moderne (*avionics, fly-by-wire, auto-pilot*), sisteme de navigație, sistemele de control ale rachetelor, navetelor și stațiilor spațiale, echipamentul de supraveghere și control al automobilelor (*automotive*), linii de fabricație automatizată, sistemele de supraveghere și control al centralelor nucleare, și multe altele. Eșecul unor astfel de sisteme în îndeplinirea *la termenele specificate* a sarcinilor programate poate avea consecințe catastrofale asupra mediului înconjurător, ducând chiar la pierderi de vieți omenești.

În prezent există o cantitate covârșitoare de informație ce tratează domeniile de mai sus. Majoritatea documentației necesare activității noastre de cercetare-dezvoltare a fost colectată începând cu anul 2000, având acces la o serie de resurse importante, recunoscute la nivel mondial în comunitatea academică și industrială: societățile științifice IEEE și ACM (SUA), site-uri de Web ale unor instituții și laboratoare de renume în domeniu, baze de informație ale unor firme majore (Motorola, Texas Instruments, QNX), etc. Informația colectată a fost procesată în cea mai mare parte a sa, până în anul 2002. Din materialul studiat, a rezultat faptul că *problematika sistemelor timp-real (TR) stricte destinate aplicațiilor critice de achiziție și prelucrare numerică a semnalelor (APNS) și de control digital încorporat* nu este rezolvată în mod satisfăcător, existând încă o serie de probleme importante:

- Introducerea timpului ca o coordonată esențială în toate etapele dezvoltării, analizei și implementării sistemelor și aplicațiilor TR;
- Unificarea și integrarea tuturor modelelor și fazelor de dezvoltare ale sistemelor TR (STR) într-o metodologie omogenă;
- Utilizarea cu precauție a structurilor și mecanismelor clasice, concepute pentru creșterea eficienței de operare a sistemelor pentru cazurile de operare tipice (de medie), exemplul cel mai important fiind utilizarea fără restricții a întreruperilor și a mecanismelor asincrone;

- Existența unei mari varietăți de metode de specificare și verificare formală a STR. Până în prezent însă, nici una nu este încă acceptată în întregime de mediul academic și în industrie. Principala problemă constă din decalajul important între rezultatele cercetărilor din domeniul metodelor formale timp-real (modelele concepute) și implementările practice reușite.

Activitatea noastră de cercetare, sintetizată în lucrarea de față, își propune să aducă o serie de contribuții și soluții pentru rezolvarea problemelor de mai sus. Abordarea din lucrare se axează pe următoarele idei principale:

- Utilizarea întreruperilor în sistemele TR stricte reprezintă o problemă din punctul de vedere al predictibilității, afectând capacitatea acestora de a garanta în orice condiții de operare respectarea termenelor impuse task-urilor TR stricte.
- Pentru a oferi predictibilitate maximă sistemelor TR stricte destinate aplicațiilor critice de achiziție și prelucrare numerică a semnalelor și de control digital încorporat, este necesară abordarea modelelor non-preemptive în ceea ce privește definirea semnalelor, a task-urilor și conceperea algoritmilor de planificare.
- Dezvoltarea viabilă a sistemelor și aplicațiilor TR stricte trebuie să aibă la bază, pentru toate fazele sale, metode și modele omogene, compatibile, în care timpul este o coordonată cheie.
- În procesul de dezvoltare a aplicațiilor TR stricte, faza de analiză offline, apriori execuției sale pe sistemul țintă, reprezintă o cerință obligatorie.

Principalele obiective propuse în cadrul acestui program de cercetare-dezvoltare sunt strâns legate de ideile cheie enumerate anterior:

- (1) Conceperea, dezvoltarea și demonstrarea unui set omogen de modele pentru semnale și task-uri TR, bazate pe un sistem corespunzător de reprezentare a timpului real.
- (2) Conceperea unei metodologii unitare de dezvoltare și implementare a sistemelor TR pentru aplicații critice de achiziție și prelucrare numerică de semnal. Elementul central al tuturor fazelor implicate (proiectare, specificare, verificare, analiză, implementare și testare) este sistemul de modele realizat în cadrul primului obiectiv.
- (3) Studiul detaliat al tehnicilor și metodelor ce permit operarea unui sistem TR strict cu garantarea respectării termenelor de timp impuse de aplicațiile critice și maximizarea predictibilității. Interesul a fost focalizat pe studiul algoritmilor de planificare non-preemptivă a seturilor de task-uri timp-real și a mecanismelor ce trebuie dezvoltate pentru rezolvarea elementelor asincrone din operarea sistemelor TR stricte.

Rezultatele obținute în cadrul activității de doctorat sunt sintetizate în teza de față. Datorită volumului relativ mare de informație prezentată, teza este structurată în secțiuni.

Prima secțiune conține capitolele de introducere ale lucrării. Astfel, capitolul următor prezintă o scurtă trecere în revistă a conceptelor actuale, ce vor fi abordate

ca subiecte de interes ale lucrării, din domeniul sistemelor TR și de control digital încorporat.

Cu toate că domeniul sistemelor TR a beneficiat de o atenție deosebită din partea comunității academice și industriale în ultimele decenii, există încă o serie de probleme nerezolvate, câteva fiind amintite în paragrafele anterioare. În Capitolul 3 sunt evidențiate și discutate un număr de subiecte care necesită încă soluționare și care vor fi abordate în capitolele următoare ale lucrării.

Secțiunea a doua prezintă contribuțiile aduse la proiectarea, modelarea și dezvoltarea sistemelor TR stricte pentru aplicații critice. În capitolele secțiunii sunt introduse și studiate o serie de modele care pot fi utilizate în tratarea omogenă a fazelor de concepție, proiectare, specificare, verificare și analiză a STR stricte.

Un model temporal este propus și analizat (Capitolul 4), urmând a se constitui într-un element de bază pentru toate fazele dezvoltării STR stricte. În continuare sunt clasificate și studiate semnalele cu care interacționează sistemele TR și, utilizând modelul temporal sistem, este derivat un set minimal și complet de parametri temporali ce pot modela semnalele din punct de vedere al specificării formale. Comportarea în timp a task-urilor TR este de asemenea abordată pe baza aceluiași model temporal, rezultând un set de parametri și relații, cu ajutorul cărora este propus și discutat modelul task-ului TR strict, ModX-ul.

Problema planificării seturilor de task-uri TR este abordată în Capitolul 5, din perspectiva modelelor pentru timp, semnale și task-uri introduse în Capitolul 4, rezultând un studiu detaliat al algoritmilor de planificare non-preemptivă pentru seturi de ModX-uri (task-uri TR stricte, periodice), atât din perspectiva operării algoritmice, cât și a operării în cadrul STR (operare online). Tot aici sunt discutate în detaliu aspectele legate de modelarea și planificarea seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora.

Evaluarea performanțelor și studiul comparativ al algoritmilor de planificare non-preemptivă este prezentată în Capitolul 6. Ca rezultat, se discută selecția și modul de combinare a acelor algoritmi care prezintă o comportare optimă pentru clasa de aplicații TR critice asupra căreia se axează preocupările din lucrare.

Modelul unui sistem complet, OPEN-HARTS, care unifică toate conceptele introduse și discutate în această secțiune, este prezentat în Capitolul 7. Sistemul OPEN-HARTS este conceput ca un set de soluții pentru problemele ridicate de dezvoltarea unitară a unui sistem sau aplicații TR stricte, utilizând modelele de timp sistem și semnal, și ModX-urile cu planificare și execuție non-preemptivă.

Secțiunea a III-a conține contribuțiile aduse la implementarea și evaluarea sistemelor timp-real stricte pentru aplicații critice. În capitolele secțiunii sunt prezentate detaliile de implementare și testare ale unui sistem de operare (nucleu) timp-real cu caracteristici care îi permit garantarea respectării termenelor de timp pentru task-urile critice ale aplicațiilor de achiziție și prelucrare numerică de semnal, sau de control încorporat. Astfel, nucleul de operare HARETICK reprezintă un studiu de caz, complet și consistent, privind implementarea, testarea și evaluarea principiilor, modelelor teoretice și algoritmilor discutați în cadrul secțiunii anterioare.

Capitolul 8 trece în revistă caracteristicile generale ale nucleului HARETICK, caracteristici ce derivă din faptul că au fost vizate sistemele DSP și de control digital încorporat (embedded systems). HARETICK a fost conceput ca un set omogen de componente software sistem de bază care asigură operarea de ansamblu a sistemului

conform principiilor TR vizate. Descrierea acestor componente (tipul, rolul, principiul de operare, detalii semnificative de implementare, etc.) este realizată în același capitol.

Timpul, ca parametru esențial de operare al sistemului, este reprezentat și gestionat cu ajutorul unor structuri și mecanisme dedicate. Capitolul 9 prezintă elementele de implementare și de sincronizare ale celor trei structuri de gestionare a timpului din HARETICK: timpul sistem absolut, ceasul de timp-real și timpul de planificare. În continuare sunt detaliate structurile de date și mecanismele angajate de nucleu pentru reprezentarea și gestionarea task-urilor TR stricte (ModX-urilor).

Funcționarea de principiu a sistemului se bazează pe cele două contexte de execuție complementare – contextul strict și cel lejer. Descrierea celor două contexte, a modului de operare al task-urilor din cadrul acestora și a elementelor ce caracterizează stările task-urilor TR stricte pe parcursul existenței lor în cadrul nucleului, este realizată în Capitolul 10. Componenta principală a HARETICK – executivul timp-real – este de asemenea prezentată în detaliu, subliniindu-se rolul său de bază: asigurarea execuției temporizate a ModX-urilor și rezolvarea comutării corecte a celor două contexte de execuție. Finalul capitolului este dedicat prezentării planificatorului contextului strict al nucleului.

Capitolul 11 tratează aplicațiile dezvoltate și implementate pentru testarea și evaluarea practică a operării nucleului HARETICK. Rezultatele obținute cu ajutorul măsurătorilor practice cu echipamente specializate (osciloscop digital, generator de funcții programabile, etc.) sunt prezentate și discutate în detaliu.

Secțiunea a IV-a, cu concluziile desprinse pe parcursul tratării subiectelor propuse în lucrare, un studiu comparativ cu alte sisteme din domeniu, rezumatul contribuțiilor, precum și perspectivele de continuare a cercetării, încheie teza.

Întreaga activitate de cercetare-dezvoltare din cadrul programului de doctorat a fost realizată în Laboratoarele de Prelucrare Numerică a Semnalelor (DSPLabs) ale Departamentului de Știința și Ingineria Calculatoarelor, Universitatea "Politehnica" din Timișoara, laboratoare coordonate de autor și de către prof. dr. ing. Vladimir CREȚU. Diverse elemente și rezultate au fost obținute în colaborare cu Laboratorul de Circuite Integrate și Sisteme Numerice de Achiziție (coordonator prof. dr. ing. Mircea STRATULAT), Laboratorul Multidisciplinar de Testare a Mașinilor Electrice ("D109", coordonatori: prof. dr. ing. Marius BIRIESCU și prof. dr. ing. Vladimir CREȚU) și Laboratorul pentru Studiul Dinamicii Fluidelor (coordonatori: prof. dr. ing. Liviu ANTON și prof. dr. ing. Alexandru BAYA).

Începând cu anul 2001, programul de doctorat a beneficiat și de suportul companiei Motorola, Incorporated, constând, printre altele, din oferirea sistemelor de evaluare pentru procesoare numerice de semnal Motorola DSP56303 EVM, DSP56824 EVM, și DSP56307 EVM care a fost utilizată ca platformă pentru implementarea și testarea nucleului HARETICK.

2 Sisteme timp-real de achiziție și prelucrare numerică de semnal

Pe măsură ce sistemele digitale de calcul și control sunt integrate în tot mai multe medii și aplicații din toate domeniile de activitate umane, a crescut importanța și interesul acordat sistemelor de achiziție și prelucrare numerică a semnalelor, a sistemelor de control digital încorporat (*embedded systems*) și, implicit, a sistemelor software ce operează pe astfel de platforme.

Sistemele de achiziție și prelucrare numerică de semnal (APNS) și cele de control digital încorporat sunt în majoritatea cazurilor utilizate în aplicații cu cerințe de operare în timp real și, în multe cazuri, implementează funcții critice de procesare.

Capitolul de față reprezintă o scurtă trecere în revistă a unor concepte actuale legate de sistemele și aplicațiile APNS timp-real, ce vor fi abordate apoi în restul lucrării, cu accent pe cele cu impact critic asupra mediului și/sau a operatorilor umani.

2.1 Sisteme de achiziție și prelucrare numerică de semnal

Datorită numărului imens de aplicații ce implică sisteme numerice de procesare, în ultimii ani au apărut noi tipuri de probleme și domenii de interes în ingineria sistemelor digitale [Baya 99, Micea 00b, Micea 00e, Micea 00f, Micea 01a, Micea 01b, Micea 01c, Cretu 02, Cretu 03]:

- Interfațarea sistemelor numerice la mediul înconjurător al aplicațiilor
- Dezvoltarea de algoritmi eficienți pentru procesarea datelor și a semnalelor
- Încapsularea echipamentelor digitale de control și prelucrare a datelor în produse industriale și de larg consum
- Rezolvarea cerințelor de operare timp-real
- Specificarea, proiectarea și implementarea de arhitecturi și protocoale eficiente pentru comunicații digitale.

Domeniul sistemelor de achiziție și prelucrare numerică a semnalelor (APNS) se ocupă în principal cu studierea soluțiilor ce tratează primele două puncte de mai sus (vezi Figura 2-1) [Toma 96, Pentek 96, NatInst 99, Micea 00c].

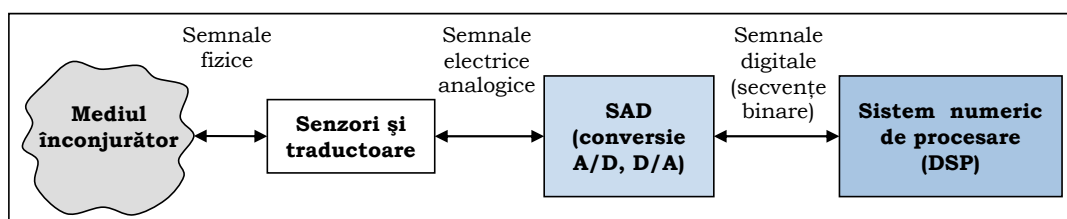


Figura 2-1. Interfațarea sistemelor digitale la mediul înconjurător

2.1.1 Sisteme de achiziții de date

Sistemele de achiziții de date (SAD) sunt dispozitive electronice hibride (analogice și digitale), cu rolul principal de interfață între sistemele digitale de procesare și mediul înconjurător. Funcțiile de bază ale SAD cuprind următoarele: (a) condiționarea semnalelor, (b) conversia analog-digitală, (c) conversia digital-analogică, (d) interfață digitală cu mediul, și (e) comunicația de date cu sistemul gazdă (cu DSP).

(a) Condiționarea semnalelor

Semnalele fizice provenite (prelevate) din mediul înconjurător (de exemplu: sunet, lumină, mișcare, presiune, umiditate, debit al fluidelor, semnale biologice, unde seismice, etc.) sunt convertite în semnale electrice (curent sau, de regulă, tensiune) cu ajutorul senzorilor și traductoarelor. Operația reciprocă, prin care sistemele digitale (DSP) acționează asupra mediului, este efectuată cu ajutorul actuatorilor.

Condiționarea (adaptarea) semnalelor implică operații de adaptare a domeniului de scală utilizat de senzori/traductoare și actuatori la cel utilizat de circuistica de conversie analog-digitală și digital-analogică din cadrul SAD. Tot la acest nivel se pot aplica filtrări preliminare ale semnalelor, eliminându-se astfel benzile de frecvență care nu sunt necesare.

(b) Conversia analog-digitală

Conversia analog-digitală (ADC, Analog-to-Digital Conversion) a semnalelor reprezintă funcția cheie a SAD și constă din setul de operații care stabilesc o corespondență directă între o valoare electrică analogică de intrare (curent sau tensiune electrică) și un cod binar de ieșire, de lungime (precizie) finită.

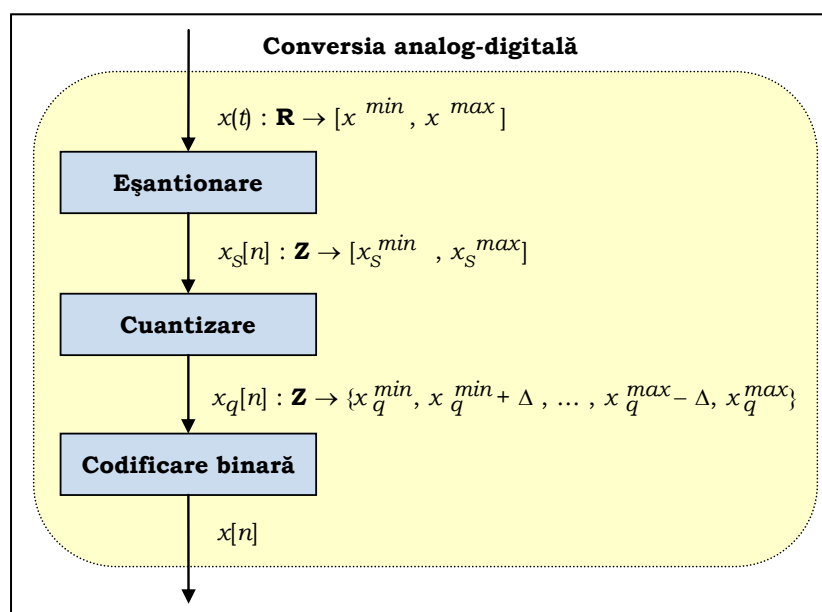


Figura 2-2. Procedura de conversie analog-digitală

ADC este un proces ce se desfășoară în trei faze (vezi Figura 2-2), executate secvențial de către un dispozitiv electronic specializat, convertorul analog-numeric [Burr-Brown 95a, Burr-Brown 95b]: eșantionarea, cuantizarea și codificarea binară.

Eșantionarea reprezintă operația de prelevare de eșantioane de valori din semnalul analogic de intrare ($x(t)$ în figura precedentă). De regulă eșantionarea se efectuează la intervale de timp constante (*eșantionare periodică*, cu perioada T_s), rezultând semnalul discret în raport cu timpul $x_s[n]$. Relația stabilită de operația de eșantionare între cele două variabile de timp, cea continuă t și cea discretă n , este: $t = n \cdot T_s$, cu $n \in \mathbf{Z}$. Domeniul de valori al semnalului rezultat după eșantionare este tot unul continuu, delimitat de extremele minimă și maximă ale semnalului de intrare obținut de la senzori, valorile eșantioanelor fiind astfel exprimate cu precizie infinită (Figura 2-2).

Cuantizarea tratează problema reprezentării cu precizie finită a valorilor eșantioanelor de semnal – problemă comună tuturor sistemelor digitale. Valorile obținute după cuantizare aparțin unei mulțimi finite, discrete de valori, în care fiecare două elemente consecutive diferă cu valoarea *treptei de cuantizare* Δ , conform relației (2-1):

$$\Delta = \frac{x_q^{\max} - x_q^{\min}}{2^b - 1} = \frac{FSR}{L - 1} \quad (2-1)$$

unde: b este numărul de biți ce compun codul binar rezultat (rezoluția conversiei analog-digitale, sau precizia binară a conversiei), FSR este domeniul de scală al operației (Full Scale Range), și $L = 2^b$ reprezintă numărul total de nivele de cuantizare (numărul total de combinații binare pe b biți, sau cardinalul domeniului de valori ale cuantizării).

Codificarea binară poate utiliza mai multe scheme posibile [Albanus 91]: USB (Unipolar Straight Binary), BOB (Bipolar Offset Binary), BTS (Bipolar Two's Complement), etc.

(c) *Conversia digital-analogică*

Conversia digital-analogică (DAC, Digital-to-Analog Conversion) este reciprocă procedurii de conversie analog-digitale, punând în corespondență fiecare cod binar pe b biți furnizat la intrare, cu o mărime electrică analogică de ieșire (curent sau tensiune) proporțională [TexasInst 00a, TexasInst 00b].

(d) *Interfața digitală cu mediul (Digital I/O)*

În prezent există multe traductoare care transformă mărimile fizice direct în semnale digitale (de exemplu, traductoarele de rotație la mouse, traductoare de poziție, senzori video CCD, etc.). Majoritatea SAD dispun de o interfață digitală de intrare-ieșire, paralelă, programabilă și cu logică tampon.

(e) *Comunicația de date cu sistemul gazdă*

Comunicația de date dintre SAD și sistemul gazdă (sistemul de prelucrare a semnalelor digitizate) reprezintă o problemă importantă, ce vizează aspecte ca:

- Viteza sau rata de transfer a datelor: aplicațiile moderne, ca sisteme multimedia, echipamente de telecomunicații digitale, interfețe evoluate, etc., necesită interfețe de comunicații de înaltă performanță;
- Arhitectura interfețelor de comunicații: în funcție de cerințele aplicațiilor implementate și de arhitectura APNS (dacă SAD se găsește pe o aceeași placă sau la distanță față de DSP), interfața de comunicații va fi de tip local sau la distanță;
- Complexitatea protocoalelor de comunicație;
- Standardizarea: pentru obținerea unor APNS cât mai modulare, flexibile și portabile, interfața de comunicații trebuie să respecte standarde bine specificate. Exemple uzuale: interfața serială RS232, USB, FireWire, interfața paralelă (SPP, EPP, ECP), CAN, IrDA, PCI, etc.

Arhitectura generală a SAD (vezi Figura 2-3) corespunde principalelor funcții descrise anterior, fiind compusă din următoarele module [Cretu 99, Micea 00c]: ADCM – modulul de conversie analog-digitală, DACM – modulul de conversie digital-analogică, IOM – interfața digitală intrare-ieșire, DAQCDM – modulul de comandă a sistemului de achiziții, ICOMM – modulul de comunicații cu sistemul gazdă, și IBUS – magistrala internă a SAD (date, adrese și comenzi).

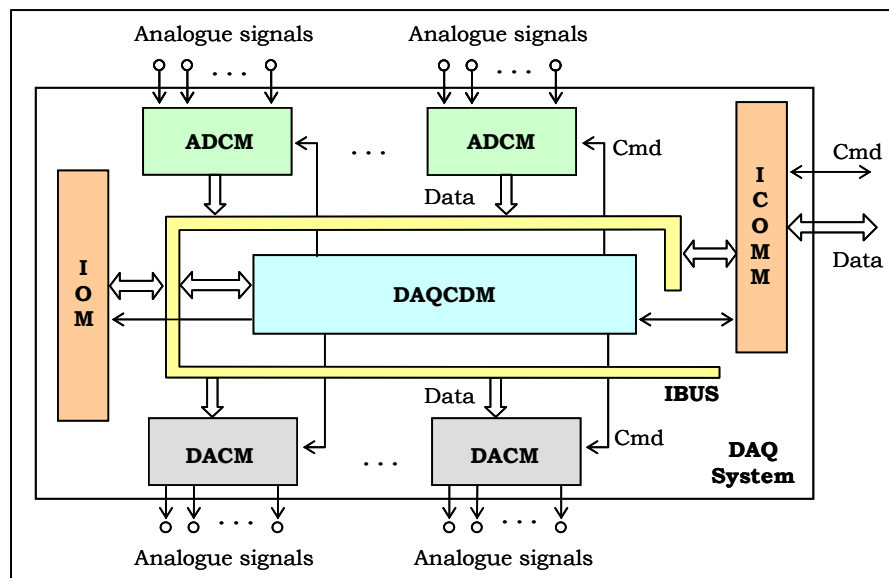


Figura 2-3. Arhitectura generală a SAD

Modulele de conversie analog-digitală (ADCM) și digital-analogică (DACM) au o arhitectură simetrică (vezi Figura 2-4), incluzând circuite specializate de condiționare a semnalelor, multiplexare/demultiplexare analogică [TexasInst 00c], eșantionare și memorare, și conversie [Burr-Brown 95a]. Semnalele de control

uzuale pentru ADCM și DACM includ: linii de control ale circuitelor de condiționare de semnal (de exemplu, controlul coeficientului de amplificare/atenuare a tensiunii de intrare – "Gain Ctrl"), liniile de selecție a multiplexării, comanda eșantionării și memorării (" S/\bar{H} " – "Sample/Hold"), comanda inițierii unui nou ciclu de conversie ("SCV" – "Start Conversion"), semnalul ce confirmă terminarea ciclului curent de conversie ("EOC" – "End of Conversion"), etc.

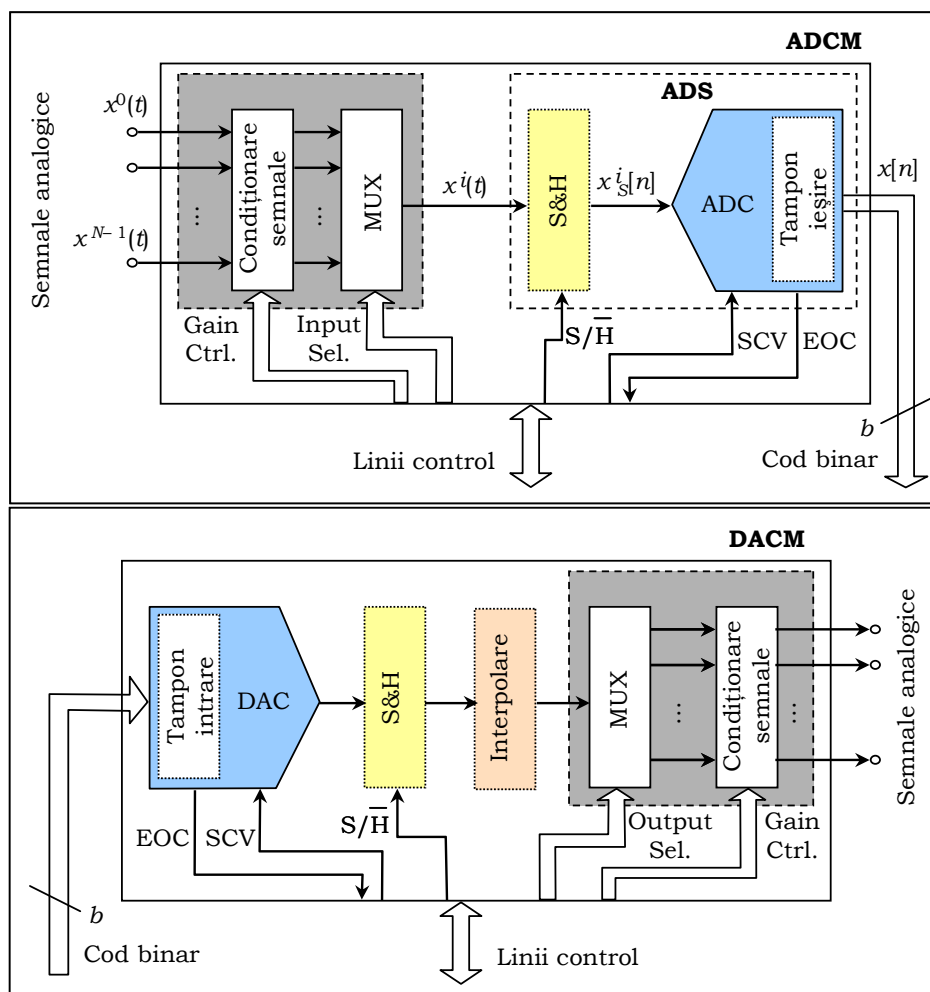


Figura 2-4. Modulul de conversie analog-digitală, ADCM (sus), și modulul de conversie digital-analogică, DACM (jos)

Controlul operării tuturor modulelor și blocurilor funcționale ale SAD este realizat de către modulul de comandă al sistemului (DAQCDM). Un modul tipic de comandă al SAD trebuie să includă următoarele funcții:

- Decodarea adreselor pentru selecția diferitelor module ale SAD;
- Implementarea de regiștri programabili speciali de date, comenzi și de stare, puși la dispoziția programelor de aplicații de pe sistemul gazdă la care e conectat SAD, pentru controlul cât mai flexibil al achiziției;
- Implementarea de mecanisme corespunzătoare de sincronizare a operării componentelor sistemului (de exemplu, pe bază de timere programabile);

- Controlul operațiilor diferitelor blocuri funcționale ale SAD: ajustarea amplitudinii semnalelor de intrare sau de ieșire ("Gain Ctrl"), selecția la multiplexare a canalului analogic curent, extragerea unui eșantion, lansarea unui ciclu de conversie, etc.

Observație

Pornind de la tipul de eșantionare (eșantionarea periodică) și de la modul de operare a dispozitivelor de conversie utilizate în marea majoritate a SAD, semnalele de control a acestor procese sunt strict periodice, având perioade și termene bine stabilite, care trebuie respectate întocmai pentru ca achiziția de semnal să se desfășoare corect.

Aceste cerințe legate de comanda operațiilor de achiziție ale SAD impun o comportare strictă de timp-real pentru sistemele și aplicațiile aferente.

2.1.2 Sisteme de prelucrare numerică a semnalelor

Sistemele de prelucrare numerică a semnalelor (DSP – Digital Signal Processing systems) sunt componentele cheie ale oricărui APNS. Un sistem DSP este un dispozitiv electronic, un program sau un algoritm autonom, cu două funcții principale: (i) prelucrarea semnalelor digitale conform unui algoritm bine definit, și (ii) controlul și comanda tranzacției de semnale cu mediul înconjurător prin intermediul sistemelor SAD.

Din punct de vedere al implementării sistemelor DSP, există patru abordări curente [Micea 00c]:

(1) Automate cu stare finită

Algoritmii de prelucrare numerică necesari unei aplicații particulare sunt implementați ca un automat cu stare finită, utilizând diverse componente digitale discrete integrate. De exemplu, FPGA (Field Programmable Gate Array) pot fi utilizate pentru implementarea unui astfel de automat.

Avantajele acestei abordări includ viteza mare de procesare și fiabilitatea înaltă a sistemelor, recomandându-le astfel pentru aplicații timp-real critice. Pe de altă parte, există o serie de dezavantaje majore: lipsa acută de flexibilitate și de scalabilitate, restricționarea numărului și a complexității algoritmilor de prelucrare numerică ce pot fi implementați, limitarea complexității generale a sistemului impusă de ingineria și de tehnologiile curente, etc.

(2) Sisteme bazate pe microcontrollere

Microcontrollerele reprezintă în continuare cea mai frecventă soluție pentru implementarea logicii digitale de procesare și control în sistemele încorporate, datorită vitezei lor de procesare și a fiabilității relativ mari, combinate cu flexibilitatea oferită de implementarea algoritmilor de procesare ca rutine software.

Arhitectura internă a microcontrollerelor oferă majoritatea resurselor necesare implementării sistemelor de procesare a datelor și control digital: CPU, memorie

internă RAM și ROM, porturi intrare-ieșire programabile, timere programabile, mecanisme de întreruperi și DMA, etc. Cu toate acestea, din cauză că arhitectura microcontrollerelor este destinată procesărilor generice de date, multe aplicații DSP ce implică manipulări de date și calcule complexe la viteze mari nu pot fi implementate corespunzător pe microcontrollere.

(3) *Sisteme bazate pe procesoare specializate (procesoare numerice de semnal)*

Relativ recent a fost dezvoltată o nouă clasă de procesoare specializate, care oferă soluții eficiente pentru aplicațiile de prelucrare numerică de semnal. Procesoarele DSP (Digital Signal Processor) derivă din arhitecturile microcontrollerelor, la care au fost aduse îmbunătățiri în direcția facilitării execuției eficiente a operațiilor de manipulare a datelor și a celor aritmetice complexe, implicate frecvent în algoritmi de prelucrare numerică a semnalelor [Motorola 95, TexasInst 97].

Un exemplu de operație complexă de prelucrare numerică, implementată cu eficiență maximă de procesoarele DSP, este *convoluția* a două semnale digitale [Lynn 92, Ifeachor 93, Motorola 95, Proakis 96, Oppenheim 96]:

$$z[n] = x[n] * y[n] = \sum_k x[k] \cdot y[n-k], \quad k \in \mathbf{Z} \quad (2-2)$$

unde: $x[n]$ și $y[n]$ sunt cele două semnale digitale procesate, iar $z[n]$ este semnalul rezultat în urma convoluției.

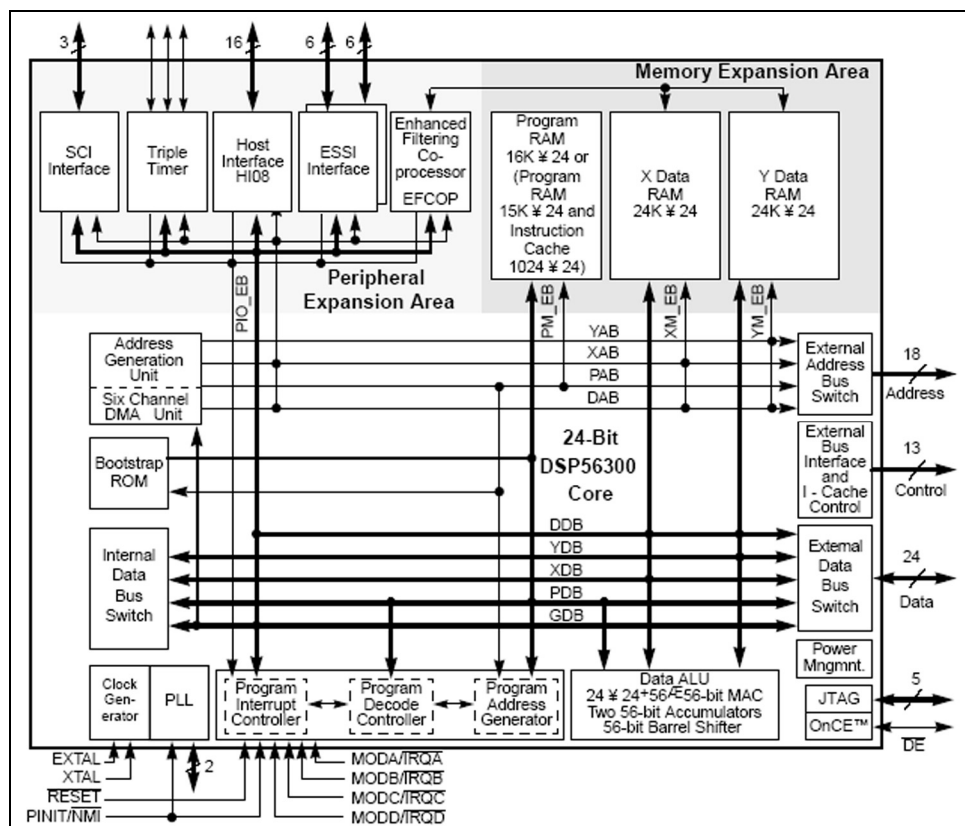


Figura 2-5. Schema bloc a procesorului Motorola DSP56307 (sursa: [Motorola 98])

Arhitectura internă a procesoarelor DSP (exemplificată în Figura 2-5 pentru procesorul Motorola DSP56307 [Motorola 98, Motorola 00]) dispune de blocuri funcționale care operează autonom în paralel, unități aritmetice și logice care implementează de exemplu operația de multiplicare și acumulare (MAC), necesară în (2-2), într-un singur ciclu mașină, unități de generare a adreselor, etc. Arhitectura familiei de procesoare Motorola DSP56k dispune, de exemplu, de următoarele facilități suplimentare ce permit implementarea extrem de eficientă a algoritmilor și programelor de prelucrare numerică de semnal:

- O configurație și interconectare internă a blocurilor funcționale și a magistralelor, bazată pe arhitectura Harvard modificată [Motorola 95, Motorola 00], ce permite operarea cu un grad mare de paralelism. De asemenea, interfețele periferice pot opera, la rândul lor, autonom și în paralel cu nucleul procesorului;
- Existența unui suport puternic pentru diferite tipuri de structuri de date, oferit la nivelul regiștrilor și a modurilor de adresare: structuri tampon liniare și circulare, adresare cu deplasament specificat a structurilor tip tabelă, adresare cu inversare de biți ("bit-reversed addressing") pentru accesarea factorilor implicați în calculul Transformărilor Fourier Rapide (FFT), etc.;
- Suport hardware pentru programarea și execuția rapidă a buclelor și a secvențelor repetitive de instrucțiuni;
- Regiștri ortogonali, ce pot fi utilizați de majoritatea instrucțiunilor procesorului ca regiștri de uz general, sau ca regiștri cu funcții specializate;
- Existența a diverse moduri de operare, orientate pe controlul consumului de energie.

Avantajele oferite de procesoarele DSP (performanță ridicată, costuri relativ scăzute, flexibilitate de programare, consum redus de putere, unelte soft de dezvoltare a aplicațiilor la costuri moderate, etc.) le recomandă ca una dintre cele mai eficiente soluții în domeniul prelucrării numerice timp-real a semnalelor și al controlului digital încorporat [TexasInst 92, Gai 02].

(4) *Software specializat, executat pe calculatoare numerice*

Algoritmii DSP pot fi implementați ca programe executate pe calculatoare personale sau specializate. Soluția are avantajul maximei flexibilități și portabilități, prin utilizarea unor limbaje de programare de nivel înalt, ca C sau Java, pentru care există o largă varietate de unelte software de dezvoltare și analiză.

Pe de altă parte, viteza relativ moderată de execuție a algoritmilor pe calculatoare ce rulează sub platforme clasice de operare (MS Windows, Linux, etc.), precum și lipsa de predictibilitate indusă de aceste sisteme de operare, nu recomandă abordarea de față ca soluție pentru sisteme de control autonome, industriale, cu cerințe stricte de timp-real impuse de multe aplicații în domeniu.

2.2 Sisteme timp-real

Oxford Dictionary of Computing dă următoarea definiție pentru un sistem timp-real [Ostroff 92]:

"Un sistem timp-real (STR) este 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, aceluiași variații. Decalajul dintre timpii de intrare și cei de ieșire trebuie să fie suficient de mic, relativ la tipul aplicațiilor implementate."

[Young 82] definește un STR ca fiind: "Orice activitate sau sistem de procesare a informației, care trebuie să răspundă după o perioadă finită și bine specificată, unor stimuli de intrare generați extern."

Așadar, operarea corectă a STR nu depinde doar de rezultatul logic al procesării, ci și de timpul în care sunt produse rezultatele [Burns 90, Stankovic 92a, Stankovic 92b, Panzieri 93, Ghosh 94, Parashkevov 94].

Sistemele de calcul pot fi împărțite în trei mari categorii [Berry 00]:

- *Sisteme transformaționale*: sunt sistemele care procesează un set de date de intrare pentru a obține un set de date de ieșire, după care se opresc. Majoritatea programelor de procesare numerică se încadrează în această categorie: compilatoare, editoare/procesoare de texte, etc.
- *Sisteme interactive*: interacționează cu mediul (și cu operatorii/utilizatorii) într-o manieră de tip "cerere-răspuns" (modelul *master-slave*). Sistemul oferă anumite servicii de procesare, așteaptă cereri de tip client pe care le deservește pe rând. Exemple: sistemele de operare, sistemele de gestionare a bazelor de date, motoare de căutare pe Web, etc.
- *Sisteme reactive*: denumite și *sisteme reflexe*, reacționează în mod continuu la semnale de intrare (stimuli) provenind din mediul fizic înconjurător, generând semnale de ieșire (actuatori). Sistemele reactive au o comportare strict legată de cea a intrărilor, trebuind să reacționeze cu succes în ritmul dictat de mediul înconjurător. Exemple: controloarele de proces, sistemele de control digital încorporat, sistemele APNS, etc.

Sistemele de calcul și aplicațiile reale nu aparțin strict doar uneia din categoriile de mai sus. Astfel, sistemele TR sunt, în mod uzual, sisteme reactive, dar care permit un anumit nivel de interacțiune, cu operatorul de exemplu. Figura 2-6 prezintă modelul de principiu al STR, în care sunt evidențiate sistemul de control (STR), sistemul controlat (mediul înconjurător) și partea de interacțiune cu operatorul uman. Mediul în care operează un anumit STR are un rol extrem de important în conceperea și proiectarea acestuia [Stankovic 92b]. O pondere majoră a specificațiilor STR derivă din restricțiile pentru semnalele de intrare/ieșire impuse sistemului de către mediul înconjurător. De exemplu, perioada de apariție a unui anumit semnal de intrare are implicații directe asupra intervalului maxim de timp pe care STR îl are la dispoziție pentru a trata acest semnal și a genera ieșirile corespunzătoare. Acest punct de vedere constituie un element central al preocupărilor noastre legate de proiectarea și implementarea STR stricte, descrise în lucrarea de față.

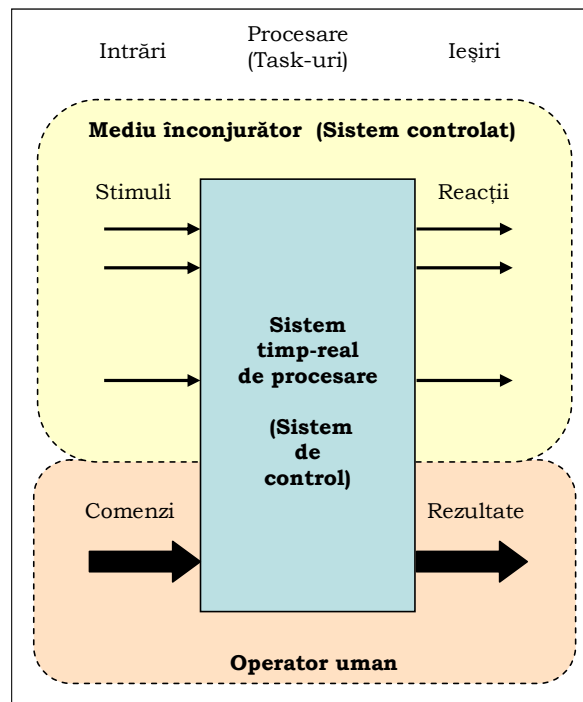


Figura 2-6. Schema de principiu a STR

Multe sisteme controlate (medii înconjurătoare) au caracteristici foarte bine definite (de exemplu, o linie de asamblare, motorul unui automobil, uși cu operare automată, etc.), fiind clasificate ca *deterministe*. STR care operează în astfel de medii vor fi, de asemenea, deterministe. Un sistem este considerat determinist dacă o aceeași secvență de intrări va produce de fiecare dată o aceeași secvență de ieșiri [Berry 00]. Determinismul este o caracteristică de bază a sistemelor reactive și unul din criteriile care le deosebește față de sistemele interactive. Identificarea (sau modelarea) unui mediu determinist în care va opera un STR, stă la baza specificării și proiectării sistemului.

Cerințele de comportare corectă în timp a STR sunt impuse de impactul fizic pe care îl au sistemele de control asupra mediului înconjurător. Prin urmare timpul devine în cazul sistemelor și aplicațiilor TR o coordonată esențială, cu impact în toate fazele dezvoltării și exploatării acestora, de la specificare a întregului sistem și până la execuția la nivel de task. Cerințele de timp pentru task-urile STR pot fi clasificate după diverse criterii, cel mai uzual criteriu fiind periodicitatea. De exemplu, majoritatea task-urilor care procesează informație obținută de la senzori sau cele de acționări asupra mediului sunt de tip periodic și cu cerințe de timp severe.

Problema respectării termenelor de timp impuse de aplicație (mediu) sistemelor TR le împarte în următoarele categorii:

- *STR critice*: includ funcții (task-uri) pentru care respectarea termenelor de execuție este o problemă critică. Exemple de astfel de task-uri sunt cele de monitorizare a temperaturii reactorului într-o centrală nucleară, sistemele de navigație ale unei rachete de croazieră sau sistemul tip "fly-by-wire" de control al zborului avioanelor moderne. Nerespectarea termenelor specificate

pentru aceste sisteme are consecințe catastrofale pentru mediu, pierderi de vieți omenești, etc.

- *STR stricte*: includ task-uri pentru care depășirea termenelor de execuție implică o comportare eronată a sistemului. Cu alte cuvinte, valoarea de execuție a task-urilor TR stricte este anulată în cazul în care execuția acestora depășește termenele specificate. Nerespectarea termenelor în STR stricte are efecte destul de grave asupra aplicațiilor ce le utilizează (avarierea sau distrugerea de echipamente și bunuri materiale, apariția de erori în ciclul de producție, anularea de diverse tranzacții, etc.).
- *STR lejere*: sunt sisteme ce conțin numai funcții (task-uri) a căror valoare de execuție nu se anulează, ci se diminuează doar, odată cu depășirea termenelor specificate. Exemple de astfel de sisteme se găsesc în domeniul multimedia și al comunicațiilor uzuale. În cazul unei video-conferințe, pierderea unui cadru de imagine la recepție nu este o problemă catastrofală, comunicația desfășurându-se în continuare. Bineînțeles însă că valoarea operării întregului sistem este cu atât mai mare cu cât se pierd mai puține cadre video.

Tehnicile și metodele implicate în cazul task-urilor TR stricte (critice) diferă foarte mult față de cele utilizate în cazul task-urilor TR lejere. Astfel, pentru STR stricte, este necesară încă din faza de specificare, introducerea unor metode consistente care să permită definirea și verificarea cât mai corectă a comportamentului în timp. Mai departe, în multe cazuri, task-urile TR stricte sunt pre-alocate (incluzând rezervarea resurselor necesare, pentru intervalele de timp corespunzătoare execuției lor) și pre-planificate, având ca efect garantarea respectării termenelor de execuție ale acestora. În cazul task-urilor TR lejere, se utilizează frecvent algoritmi de planificare uzuali (care nu sunt de tip TR) sau algoritmi care vizează explicit constrângerile de timp ale task-urilor, dar urmăresc doar performanța în situațiile de operare medie (încărcare medie a sistemului, situații uzuale de operare, etc.). Așa cum este evidențiat și în [Stankovic 92b], *task-urile TR stricte sunt mult mai dificil de tratat decât cele TR lejere, iar sistemele care operează cu ambele tipuri de task-uri simultan, sunt cu atât mai complexe.*

Fiabilitatea STR reprezintă un subiect extrem de important și de complex. Problema fiabilității trebuie luată în considerare ca un element esențial încă din faza de proiectare a sistemului TR, implicând atât partea de hardware cât și partea de software și trebuie integrată împreună cu constrângerile temporale ale sistemului. Pe lângă tehnicile implicate în garantarea respectării termenelor task-urilor critice și stricte, în cele mai multe cazuri printr-o analiză "offline" a fezabilității și prin rezervarea apriori a resurselor necesare (chiar dacă acestea vor fi neutilizate o bună parte a timpului de operare), trebuie de asemenea luate în calcul apariția erorilor și comportarea sistemului la diferite grade de încărcare.

În concluzie, un sistem TR nu este doar un sistem rapid, ci un sistem *suficient de rapid pentru a putea garanta respectarea termenelor de execuție a task-urilor, pentru situația cea mai defavorabilă de operare.*

2.3 Timpul ca și coordonată esențială a specificării și operării STR

Cercetările din ultimii peste 30 de ani în domeniul sistemelor TR subliniază necesitatea introducerii coordonatei temporale în toate fazele dezvoltării STR: specificare, verificare/validare, programare, analiză, planificare și execuție.

Domaniul temporal a fost studiat în numeroase lucrări de specialitate și au fost propuse diferite modele și logici temporale [Allen 83, Barbacci 86, Melliar-Smith 87, Jahanian 88, Halpern 91, Ostroff 92, Bucci 95, Mattolini 96, Chen 98, Bellini 00, Berry 00, Mattolini 01]. Aceste modele sunt în general concepute pentru utilizarea în fazele de analiză a cerințelor și de specificare a STR, și se concentrează mai degrabă asupra modelării comportării sistemelor decât asupra aspectelor structurale și funcționale ale acestora.

Comportarea unui sistem se referă la modul în care acesta reacționează la stimulii externi și la evenimentele interne – aspect critic al sistemelor reactive (și deci, al STR). Aspectele *structurale* se referă la modul de descompunere a sistemului în subsisteme, iar aspectele *funcționale* se referă la transformările informației în cadrul sistemului (procesarea informației).

Logicile temporale sunt definite ca structuri de forma:

$$\langle \mathfrak{T}, R, V \rangle \quad (2-3)$$

unde \mathfrak{T} reprezintă domeniul temporal, $R \subseteq \mathfrak{T} \times \mathfrak{T}$ reprezintă relația definită de logică asupra elementelor domeniului \mathfrak{T} și V este funcția de evaluare a formulelor (propozițiilor) logicii:

$$V: F \times \mathfrak{T} \rightarrow \{\top, \perp\} \quad (2-4)$$

În (2-4), F este setul de formule ale logicii, iar $\{\top, \perp\}$ – mulțimea valorilor de adevăr ("adevărat" și "fals"). Funcția V asignează câte o valoare de adevăr fiecărei formule (propoziții) ale logicii temporale.

Pentru logicile temporale, relația R este denumită *relația de precedență* și e notată cu " $<$ ". De regulă, " $<$ " este o relație tranzitivă și nereflexivă, realizând o ordonare parțială a domeniului temporal, \mathfrak{T} [Bellini 00]. Proprietățile pe care relația " $<$ " le conferă domeniului temporal sunt următoarele:

Proprietatea 2-1. Tranzitivitatea

$$(t_i < t_j) \wedge (t_j < t_k) \Rightarrow (t_i < t_k), \text{ pentru } \forall t_i, t_j, t_k \in \mathfrak{T}.$$

Proprietatea 2-2. Nereflexivitatea

$$- t < t, \text{ pentru } \forall t \in \mathfrak{T}^+.$$

Proprietatea 2-3. Liniaritatea

$$(t_i < t_j) \vee (t_i = t_j) \vee (t_j < t_i), \text{ pentru } \forall t_i, t_j \in \mathfrak{T}.$$

Proprietatea 2-4. Predecesorul

$$\text{Pentru } \forall t_i \in \mathfrak{T}, \exists t_j \in \mathfrak{T}, \text{ astfel încât } t_j < t_i.$$

Proprietatea 2-5. Succesorul

Pentru $\forall t_i \in \mathfrak{T}, \exists t_j \in \mathfrak{T}$, astfel încât $t_i < t_j$.

Proprietatea 2-6. Limita stângă (începutul)

$\exists t_i \in \mathfrak{T}$, pentru care $\neg \exists t_j \in \mathfrak{T}$, astfel încât $t_j < t_i$.

Proprietatea 2-7. Limita dreaptă (sfârșitul)

$\exists t_i \in \mathfrak{T}$, pentru care $\neg \exists t_j \in \mathfrak{T}$, astfel încât $t_i < t_j$.

Proprietatea 2-8. Domeniul continuu

Pentru $\forall t_i, t_j \in \mathfrak{T}$ cu $t_i < t_j \Rightarrow \exists t_k \in \mathfrak{T}$, astfel încât $t_i < t_k < t_j$.

Proprietatea 2-9. Domeniul discret

Pentru $\forall t_i \in \mathfrak{T}, \exists t_j \in \mathfrak{T}$, cu $t_i < t_j$, pentru care $\neg \exists t_k \in \mathfrak{T}$, astfel încât $t_i < t_k < t_j$.

Proprietatea 2-6 exprimă faptul că domeniul temporal este limitat în trecut, iar Proprietatea 2-7, că e limitat în viitor. Proprietatea 2-8 semnifică faptul că între oricare două instanțe de timp există întotdeauna o a treia. Se observă că există proprietăți care se exclud reciproc, cum sunt de exemplu, Proprietatea 2-8 cu Proprietatea 2-9. Fiecare din aceste proprietăți definesc câte o structură particulară pentru domeniul temporal utilizat. Astfel, de exemplu, sistemele digitale de calcul utilizează un domeniu temporal liniar, discret și limitat în trecut (la momentul $t_0 = 0$). În acest fel, se poate asocia câte o stare a sistemului pentru fiecare instanță de timp. Pe de altă parte, un sistem fizic oarecare poate utiliza un domeniu temporal liniar, continuu și nelimitat.

Pe o structură particulară a domeniului temporal, timpul poate fi modelat după două metode principale: logici punctuale și logici bazate pe intervale.

Logicile temporale punctuale se bazează pe exprimarea relațiilor dintre evenimente în termeni de instanțe de timp (puncte din domeniul temporal). Aceste logici definesc intervalele ca seturi conectate de puncte succesive. Exprimarea relațiilor dintre intervalele în care se verifică anumite evenimente este mai dificilă utilizând modelarea punctuală a timpului.

Logicile temporale bazate pe intervale sunt mult mai expresive, fiind capabile a descrie evenimente în cadrul unui interval de timp. O instanță de timp (un punct din domeniul temporal) este reprezentată ca un interval de dimensiune (durată) unitară. Relațiile dintre intervale sunt exprimate mai ușor și mai intuitiv în aceste modele, utilizând operatori specifici comparativi (exprimă relațiile dintre intervale) și combinatorici (exprimă modul de combinare al intervalelor), sau operatori care precizează limitele intervalelor pe baza valorilor de adevăr ale predicatelor. Figura 2-7 ilustrează relațiile posibile între două intervale, împreună cu operatorii comparativi aferenți, conform modelului temporal bazat pe intervale propus în [Allen 83] și [Allen 94].

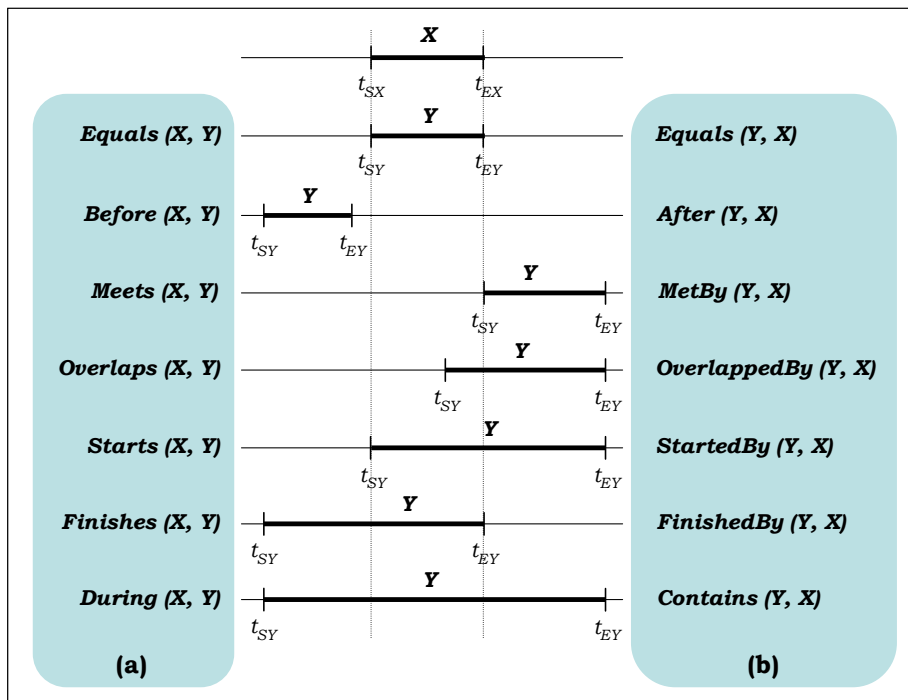


Figura 2-7. Relațiile posibile, directe (a) și reciproce (b), între două intervale temporale

Operatorii utilizați în modelul temporal ilustrat în Figura 2-7 pot fi transformați în operatori ce precizează limitele intervalelor, prin adăugarea unei *metrici* pentru timp și utilizând operatorul elementar \square ("totdeauna, în viitor", ce exprimă conceptul de necesitate în viitor). Astfel,

$$\square_{[2,9]} A \tag{2-5}$$

semnifică faptul că formula (propoziția) A este adevărată pentru toate instanțele de timp, de la cea de-a doua și până la cea de-a noua, de la momentul curent de timp în care se face evaluarea. În aceste condiții, operatorul $Before(X, Y)$ din Figura 2-7 va fi echivalent cu:

$$\square_{[t_{sx}, t_{ex}]} X \Rightarrow (\square_{[t_{sy}, t_{ey}]} Y) \wedge (t_{ey} < t_{sx})$$

Logicile temporale cu metrică de timp permit definirea de relații temporale cantitative, cum ar fi distanța dintre două evenimente sau durata unor evenimente, exprimate în unități de timp. *Posibilitatea exprimării de constrângeri temporale cantitative este esențială pentru specificarea STR.*

Timpul poate fi modelat în manieră *implicită* sau *explicită*, și *relativă* sau *absolută*. Un model temporal este implicit atunci când semnificația formulilor depinde de evaluarea momentului curent al timpului, acesta fiind considerat implicit în formule. De exemplu, formula (2-5) este echivalentă cu:

$$\forall t \in [t_0 + 2, t_0 + 9]. A(t)$$

unde t_0 este evaluarea timpului (instanța curentă de timp). Pe de altă parte, modelul temporal explicit exprimă timpul curent prin intermediul unei variabile, prezentă în toate formulele modelului. Specificarea explicită a timpului permite exprimarea

oricărei proprietăți utile a timpului-real. De asemenea, modelele temporale explicite permit specificarea de expresii care nu au sens în domeniul temporal, cum ar fi, de exemplu, activarea unui predicat pentru instanțele pare de timp [Bellini 00].

Referirea timpului este considerată absolută când valoarea timpului curent este determinată în funcție de un ceas sistem general, idealizat (se presupune că nu există nici un fel de devieri ale ceasului de la timpul absolut). În aceste modele, duratele intervalelor și momentele instanțelor de timp sunt exprimate direct în secunde sau milisecunde, etc. (adică au valori absolute, corespunzătoare ceasului referit). Modelele temporale relative exprimă duratele și instanțele de timp în unități temporale, relativ la o instanță a timpului absolut, exprimată în secunde, milisecunde, etc. Determinarea valorii absolute a elementelor temporale pentru modelul relativ, este amânată pentru faza implementării sistemului.

2.4 Tehnici de planificare

Rolul algoritmilor de planificare în cadrul STR este găsirea de soluții fiabile pentru asignarea în timp a resursei principale a sistemului – procesorul (sau procesoarele), către task-urile sistemului, astfel încât să nu existe suprapuneri ale execuțiilor acestora pe parcursul operării.

În literatura curentă există studii teoretice, simulări și implementări efectuate asupra unui număr extrem de mare de algoritmi de planificare pentru STR [Liu 73, Kim 80, Jeffay 91, Mercer 92, Panzieri 93a, Panzieri 93b, Ghosh 94, Ramamritham 94, Howell 95, George 96, Kang 98, Letia 00, Radulescu 02]. La proiectarea unui STR, alegerea unui anumit algoritm de planificare (sau a unei politici de planificare) poate depinde de o varietate de considerente, ca: numărul de procesoare ale sistemului, omogenitatea sau eterogenitatea acestora, modul lor de cuplare, tipul aplicațiilor ce vor fi executate în sistem, etc. Așa cum se subliniază și în [Ramamritham 94], date fiind numărul și diversitatea extrem de mari a abordărilor legate de algoritmi de planificare, o analiză exhaustivă a domeniului ar fi imposibilă. Chiar și o clasificare unitară și sintetică a tehnicilor de planificare este dificil de realizat. Câteva criterii de clasificare a algoritmilor de planificare pentru STR sunt trecute în revistă în continuare:

- 1) După numărul procesoarelor sistemului:
 - algoritmi mono-procesor;
 - algoritmi multiprocesor.
- 2) După modul de cuplare a procesoarelor sistemului:
 - algoritmi pentru sisteme multiprocesor (cu memorie partajată);
 - algoritmi pentru sisteme distribuite.

3) După modul (momentul) de generare a planificării:

- algoritmi statici (planificare generată offline);
- algoritmi dinamici (planificare generată online, în timpul operării sistemului).

4) În funcție de acceptarea sau nu a întreruperilor:

- algoritmi preemptivi (se admite întreruperea task-ului planificat și executat curent în sistem, de către un altul cu prioritate mai mare, de exemplu);
- algoritmi non-preemptivi (pe durata execuției unui task nu sunt admise întreruperi).

Un factor deosebit de important ce influențează politicile de planificare pentru STR îl reprezintă necesarul de resurse sistem (cuantificate mai ales în timp de procesare) pe care îl implică execuția algoritmilor de planificare. Importanța acestui criteriu derivă și din considerentul că, practic, task-ul de planificare nu aparține aplicației propriu-zise; utilizarea procesor corespunzătoare task-ului de planificare reducând astfel, resursele puse de sistem la dispoziția aplicației. Există două caracteristici care afectează puternic necesarul de resurse sistem, impus de către algoritmi de planificare: complexitatea algoritmilor și posibilitatea efectuării unei analize offline a planificabilității aplicației.

Algoritmii de planificare statică, mono-procesor, au complexitatea cea mai mică, fiind astfel studiați și implementați în majoritatea sistemelor TR ce permit această abordare. La capătul opus al scalei complexității, se situează algoritmi de planificare dinamică, destinați sistemelor multiprocesor și distribuite.

Algoritmii de planificare statică presupun existența unei analize offline a planificabilității aplicației ce urmează a fi executată pe STR. Rezultatul analizei constă într-o tabelă care specifică planificarea și execuția fiecărui task în parte, împreună cu timpii lor de execuție, sau este stabilită (prin asignarea de priorități, de exemplu) ordinea în care executivul STR va lansa în execuție task-urile aplicației. Cei mai frecvenți algoritmi de planificare statică utilizați în cadrul sistemelor TR sunt algoritmi de *planificare ciclică* [Ramamritham 94, Ghosh 94] și algoritmul *RMS (Rate Monotonic Scheduler)* [Liu 73, Kim 80, Audsley 91]. De exemplu, RMS asignează priorități distincte pentru fiecare task al aplicației, în manieră monoton descrescătoare a frecvenței de execuție a acestora, astfel încât prioritatea mai mare este atribuită task-ului cu frecvența mai mare și așa mai departe.

Avantajul tehnicilor statice de planificare constă în obținerea unei predictibilități mari a operării sistemului, cât și în complexitatea redusă a algoritmilor, facilitând astfel implementarea acestora pe arhitecturi simple, destinate unor aplicații bine precizate, de control digital al mediilor complet deterministe. Există însă o serie de dezavantaje majore: lipsa acută de flexibilitate (modificarea unui parametru al unui task oarecare, sau modificare unei condiții de operare sau de mediu, implică reluarea analizei offline și recalcularea întregii planificări), utilizarea procesor relativ mică a aplicațiilor pe care algoritmi sunt capabili a le planifica și fenomenul de inversare a priorităților (*priority inversion* – un task cu prioritate mai mare, ce dorește să acceseze o anumită resursă partajată a sistemului, este blocat de către un task cu

prioritate mai mică, ce a fost întrerupt de către primul în timpul accesării aceleiași resurse).

Dezavantajele algoritmilor statici de planificare enumerate mai sus au constituit unul dintre motivele principale pentru care au fost concepuți și studiați algoritmi de planificare dinamică. Acești algoritmi asignează în mod dinamic prioritățile task-urilor aplicației, în timpul operării sistemului, aplicând diverse criterii pentru selectarea task-ului ce urmează a fi programat și executat la momentul curent. *EDF* (*Earliest Deadline First*), *MLF* (*Minimum Laxity First*) și *MUF* (*Maximum Urgency First*) [Liu 73, Ghosh 94, Ramamritham 94, Panzieri 93a, Panzieri 93b] sunt trei dintre cei mai eficienți algoritmi de planificare dinamică, deși s-a demonstrat că, în contextele preemptive de operare, nu se poate garanta respectarea termenelor de timp ale task-urilor, în situații tranzitorii de supraîncărcare a sistemului.

EDF asignează în mod dinamic priorități task-urilor aplicației după criteriul celui mai apropiat termen de execuție (*deadline*): task-ul cu cel mai apropiat termen primește prioritatea cea mai mare. *MLF*, reprezintă o versiune modificată a lui *EDF*, în care se ține cont și de durata de execuție a task-urilor, nu numai de termenele de execuție. *Intervalul disponibil planificării* (*laxity interval*) este un parametru ce caracterizează comportarea temporală a task-urilor STR, fiind definit ca intervalul de timp rămas, din momentul curent, până la expirarea termenului de execuție, considerând durata maximă de execuție a task-ului. Intuitiv, laxitatea măsoară flexibilitatea (lejeritatea) disponibilă la un moment dat pentru planificarea unui task. *MLF* asignează prioritatea cea mai mare task-ului cu laxitatea cea mai mică. O altă variantă a lui *EDF*, care încearcă să îmbunătățească predictibilitatea mai ales în situațiile tranzitorii de supraîncărcare a sistemului, este *MUF*. Algoritmul prevede pentru fiecare task, asignarea (statică) a unui parametru ce specifică în mod explicit gradul de urgență, și determinarea dinamică a unei priorități care este invers proporțională cu laxitatea task-ului (ca în cazul *MLF*). Gradul de urgență al task-urilor este o combinație a două priorități fixe: (a) nivelul critic, cu precedență mai mare decât prioritatea dinamică a task-ului, și (b) prioritatea utilizator, cu precedență mai mică decât prioritatea dinamică. Prin această combinație, se încearcă ajutarea algoritmilor de planificare în diferențierea dintre task-uri mai importante și mai puțin importante, prin noțiunea de "prioritate" specificată de utilizator.

Puternica răspândire a sistemelor de calcul distribuite din ultimul deceniu a impus necesitatea abordării sistemelor TR distribuite, în cadrul căruia, planificarea reprezintă un subiect central de interes. Numărul foarte mare de tehnici de planificare propuse și studiate dovedește atenția de care se bucură domeniul STR distribuite în prezent. Planificarea task-urilor în STR distribuite se află la capătul superior al scalei, atât din punct de vedere al complexității algoritmilor (de ordin nepolinomial, NP [Rădulescu 02]), cât și al resurselor de calcul (timp) necesare. În consecință, planificarea task-urilor într-un sistem distribuit nu mai poate aplica seturi de criterii bine definite, în operații de căutare exhaustivă pentru selecția task-urilor ce vor fi planificate, ci sunt necesare abordări euristice, probabilistice și de aproximare.

Au fost propuse diverse metode pentru măsurarea și compararea performanțelor algoritmilor de planificare. *Raportul de utilizare procesor* (sau simplu, *utilizarea procesor – utilization factor*), discutat în detaliu în [Liu 73], reprezintă un parametru de bază, utilizat atât ca și criteriu de apreciere a performanței algoritmilor de

planificare, cât și pentru derivarea condițiilor de fezabilitate a planificării (vezi și Capitolul 5):

$$U^M = \sum_{i=1}^n \frac{C_i}{T_i} \quad (2-6)$$

unde: C_i reprezintă durata de execuție a task-ului i din setul M de task-uri supus planificării, T_i este perioada task-ului i , și $n = |M|$ este numărul de task-uri din M .

Pe baza factorului de utilizare procesor, se definește parametrul denumit *limita superioară a planificabilității* (*least upper bound to the processor utilization factor* [Liu 73], *SB – schedulable bound* [Ghosh 94], *BU – breakdown utilization* [Panzieri 93a, Panzieri 93b]), reprezentând utilizarea procesor maximă pentru care se poate garanta respectarea termenelor impuse task-urilor dintr-un set supus algoritmului de planificare evaluat:

$$SB_A (= BU_A) = \max_{\forall M} \left\{ U^M \mid M \text{ e fezabil cu } A \right\} \quad (2-7)$$

unde A este algoritmul de planificare evaluat.

Evaluarea algoritmilor de planificare cu ajutorul limitei superioare a planificabilității exprimată de (2-7), dă următoarele rezultate, spre exemplu pentru algoritmul *RMS* [Ghosh 94]: $SB_{RMS} = 0.693$ pentru seturi generice de task-uri (de dimensiuni apropiate de infinit), $SB_{RMS} = 0.88$ pentru cazurile în care perioadele task-urilor au o distribuție uniformă, și $SB_{RMS} = 1.00$ doar în cazul în care perioadele task-urilor sunt armonice ale celei mai mici perioade din set. Pe de altă parte, algoritmul *EDF* se caracterizează printr-o limită superioară a planificabilității unitară (100%) pentru toate seturile generice de task-uri ($SB_{EDF} = 1.00$), dar, așa cum a fost menționat anterior, algoritmul nu poate garanta respectarea termenelor tuturor task-urilor pentru cazurile de supraîncărcare tranzitorie a sistemului.

Alte exemple de parametri destinați evaluării performanțelor algoritmilor de planificare sunt [Panzieri 93a, Panzieri 93b]: *timpul mediu, normalizat, de răspuns NMRT* (*Normalized Mean Response Time*) și *factorul de garantare GR* (*Guaranteed Ratio*). *NMRT* este definit ca raportul dintre intervalul de timp scurs din momentul în care un task este gata de execuție și până se termină execuția lui, și timpul procesor consumat pentru execuția efectivă a task-ului. Cu cât *NMRT* este mai mare, cu atât este mai mare timpul "de inactivitate" al task-urilor, eficiența algoritmului fiind deci mai scăzută.

Factorul de garantare *GU*, reprezintă raportul dintre numărul de task-uri pentru care algoritmul evaluat poate garanta execuția cu respectarea termenelor, față de totalul task-urilor care sunt gata de execuție.

Din punctul de vedere al sistemelor TR stricte, destinate aplicațiilor critice, predictibilitatea operării este o cerință esențială. Deși s-a demonstrat faptul că algoritmi de planificare non-preemptivă necesită – în cazul general – resurse (timp) de calcul cu ordin de complexitate strict nepolinomial (NP-hard) [Cheng 88, Jeffay 91, Panzieri 93a], ei prezintă potențialul garantării execuției task-urilor cu respectarea termenelor specificate, chiar și în condițiile cele mai dezavantajoase de operare ale sistemului (*worst case operation*), spre deosebire de abordările preemptive.

Având în vedere acest considerent (vezi și Capitolul 3), în lucrarea de față vom aborda problema planificării non-preemptive a seturilor de task-uri modelate special pentru sisteme TR stricte.

2.5 Exemple de sisteme timp-real

În prezent există, în diverse faze de realizare (de la specificare și modelare, până la variante și prototipuri funcționale), o varietate extrem de mare de sisteme timp-real și metodologii de dezvoltare a aplicațiilor TR. Nici unul dintre acestea însă nu a ajuns să se impună în mediul academic și industrial în așa fel încât să se poată vorbi de o standardizare în domeniu. Câteva motive pentru această situație vor fi discutate în Capitolul 3.

În paragrafele următoare se vor trece pe scurt în revistă câteva sisteme TR și metodologii de dezvoltare a acestora, pe care le-am considerat semnificative din perspectiva stadiului de realizare a acestora (a funcționalității), precum și a tipurilor de probleme timp-real pe care le tratează.

2.5.1 Sistemul Giotto

Sistemul "Giotto" reprezintă o metodologie completă de programare pentru sisteme de control digital încorporat, mono-procesor sau distribuite, cu capabilități timp-real stricte [Henzinger 03b], dezvoltată la "University of California at Berkeley", Statele Unite.

Sistemul "Giotto" se bazează pe două componente principale: (1) limbajul de programare și compilatorul Giotto și (2) mediul (micro-nucleul) de operare ("runtime system", "programmable microkernel") pentru sisteme TR.

(1) *Limbajul de programare și compilatorul Giotto*

Limbajul de programare Giotto [Henzinger 01, Henzinger 03a], definește arhitecturi software în cadrul cărora se poate specifica și programa comportamentul funcțional și temporal al aplicațiilor timp-real. Limbajul Giotto se bazează pe următoarele paradigme legate de STR: evenimentele și acțiunile sunt periodice și sincronizate, iar operarea STR este temporizată ("time-triggered systems").

Din punct de vedere formal, limbajul Giotto prezintă următoarele componente de bază:

(a) *Task-urile*: reprezintă elementele funcționale ale unei aplicații. Task-urile Giotto sunt periodice și modulare.

(b) *Modurile*: reprezintă o grupare a unei anumite configurații de invocări (activări) repetate ale unui subset fixat de task-uri ale programului.

Operarea unui program Giotto se găsește într-un anumit *mod* la un moment dat. În cadrul unui *mod*, fiecare task este invocat cu o anumită frecvență (de unde rezultă perioada fiecărui task).

Durata de execuție a unui anumit *mod*, denumită *rundă*, rezultă ca cel mai mic multiplu comun al perioadelor task-urilor componente. Operarea unui program Giotto este văzută ca o succesiune de *runde* ale unuia sau mai multor *moduri*. Tranziția de la un *mod* la altul este efectuată cu ajutorul entității *comutator de moduri (mode switch)*.

(c) *Porturile*: sunt echivalentele logice ale unor locații de memorie, destinate comunicației de date și caracterizate de tipuri bine definite.

Informația stocată în fiecare *port* este "persistentă", în sensul că își păstrează valoarea până la o nouă actualizare.

Există mai multe categorii disjuncte de porturi:

- *sensor ports*: intrări, actualizate de către mediul înconjurător (senzori);
- *actuator ports*: ieșiri, actualizate de programul Giotto;
- *task ports*: entități echivalente cu parametrii de intrare/ieșire ai task-urilor;
- *mode ports*: entități echivalente cu parametrii de comunicație dintre *modurile* programului Giotto.

Un task poate să-și mențină o anumită stare de la o execuție la alta, prin intermediul unui set de *porturi private*, inaccesibile în exteriorul task-ului.

(d) *Drivererele*: reprezintă funcții de conversie a valorilor unor porturi pentru alte porturi. Cu alte cuvinte, *drivererele* Giotto sunt secvențe de cod sistem pentru preprocesarea și transmiterea datelor între porturile locale ale driverelor.

Drivererele pot fi condiționate cu ajutorul a câte unui predicat care se evaluează pentru un anumit *port* de intrare al *driverului*. De exemplu, un task oarecare, invocat de program la un moment dat, nu poate fi executat decât dacă evaluarea condiției aplicate *portului de senzor* al task-ului dă rezultate pozitive.

În plan logic, formal, execuția driverelor se efectuează *instantaneu*, spre deosebire de cea a task-urilor, care are loc pe durata perioadei corespunzătoare. Prin urmare, *drivererele* implementează *operații atomice*, executate în regim non-preemptiv (cu sistemul de întreruperi dezactivat).

Compilerul Giotto este o componentă esențială a metodologiei, având sarcina principală de a genera cod executabil pentru o platformă dată, dintr-un program sursă Giotto care reprezintă o specificare funcțională și temporală a aplicației. Codul executabil generat conține trei elemente principale: (i) *codul de funcționalitate* a task-urilor aplicației ce pot fi scrise de exemplu în C; (ii) *E-codul*, adică instrucțiunile ce definesc comportarea reactivă a task-urilor aplicației, de exemplu timpii de activare și termenele limită ale task-urilor ca reacție la evenimentele externe ale sistemului; (iii) *S-codul*, ce conține o planificare fezabilă a aplicației pe arhitectura țintă dată.

Pentru a reuși etapa de compilare a unei aplicații Giotto (cu generarea unei planificări fezabile), este necesară calcularea apriori compilării a timpilor maximi de execuție (WCET) pentru secvențele de cod funcțional ale task-urilor. Metodologia Giotto nu furnizează în acest sens nici un mecanism care să ajute la obținerea

timpilor WCET. Pentru ușurarea fazei de compilare a aplicațiilor Giotto, se pot utiliza biblioteci de tip "run-time" ce furnizează de exemplu nivelul de funcții și primitive de planificare și comunicație inter-task pentru o platformă dată. De asemenea, se pot introduce în program adnotări speciale: "Giotto-H" pentru specificarea unor caracteristici și performanțe hardware ale platformei țintă, "Giotto-M" pentru specificarea mapării task-urilor pe unitățile de procesare și a elementelor de comunicație pe arhitectura rețelei în cazul sistemelor distribuite, și "Giotto-S" pentru planificarea task-urilor și a elementelor de comunicație.

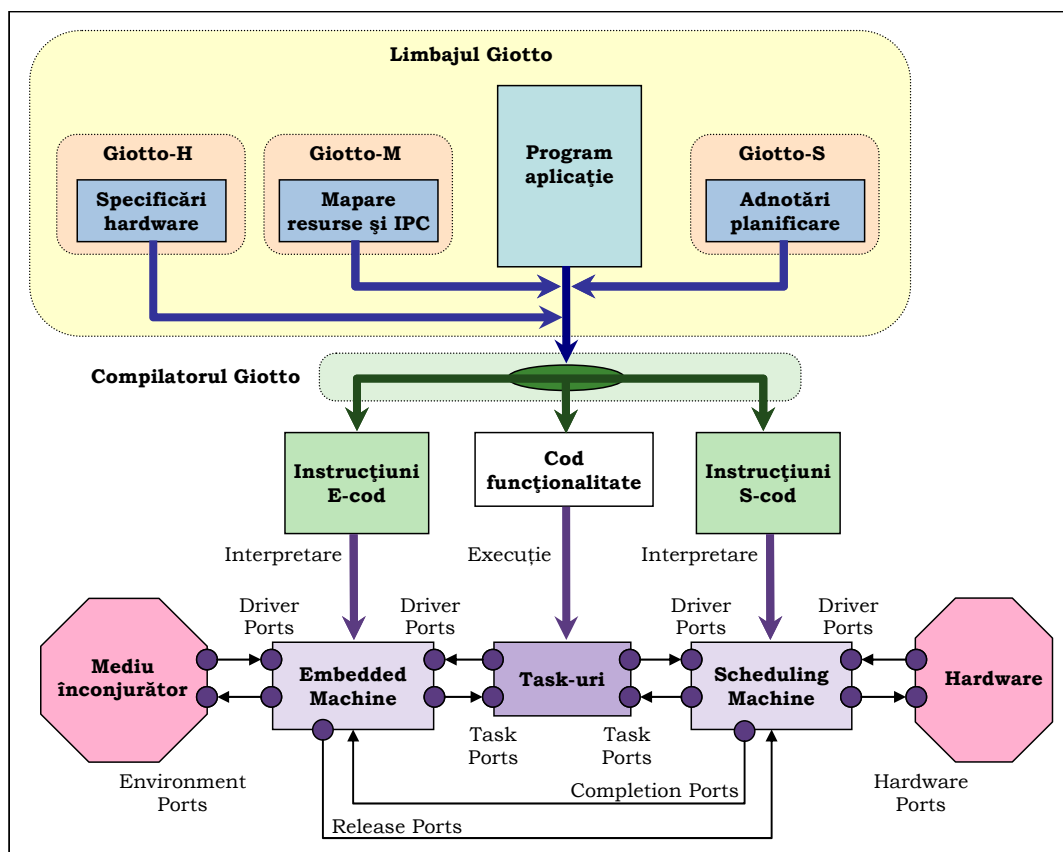


Figura 2-8. Principalele elemente ale metodologiei Giotto

(2) Mediul (micro-nucleul) de operare

Metodologia Giotto specifică [Henzinger 02, Kirsch 03, Henzinger 03c] o arhitectură logică particulară pentru execuția aplicațiilor timp-real pe diverse platforme țintă. Mediul de operare este partajat în două mașini virtuale: (a) "E-machine" (Embedded machine), care interpretează și execută "E-codul", supervizând execuția "logică" a task-urilor aplicației relativ la evenimentele externe ale sistemului; (b) "S-machine" (Scheduling machine), care interpretează "S-codul" produs de compilator, supervizând execuția "fizică" a task-urilor aplicației pe baza resurselor sistemului.

Mașina "E" reprezintă componenta reactivă a sistemului de control digital încorporat, independent de resursele hardware sau de constrângerile impuse de planificare. Pe de altă parte, mașina "S" reprezintă componenta "pro-activă" a

sistemului, gestionând execuția pe procesoarele disponibile a task-urilor active la un moment dat în sistem. Mașina "S" este corespondenta planificatoarelor din sistemele timp-real. Cele două mașini virtuale compun micro-nucleul programabil propus de metodologia Giotto pentru execuția aplicațiilor timp-real. Figura 2-8 prezintă principalele elemente ale metodologiei Giotto și relația dintre ele.

2.5.2 Sistemul Spring

Sistemul "Spring" [Stankovic 91a, Stankovic 91b, Stankovic 91c, Stankovic 99] este dezvoltat la Universitatea Massachusetts, Amherst, Statele Unite, pentru a servi ca suport pentru aplicații timp real în medii de calcul distribuite.

Arhitectura "SpringNet" se bazează pe o rețea de noduri multiprocesor (vezi Figura 2-9), fiecare nod operând sub nucleul "Spring Kernel" [Stankovic 90]. Nodurile sistemului conțin unul (sau mai multe) procesoare de aplicații, unul (sau mai multe) procesoare sistem, un subsistem I/O și unul pentru comunicații. Fiecare procesor dintr-un nod dispune de o memorie locală, ce poate fi accesată și de către procesoarele sistem.

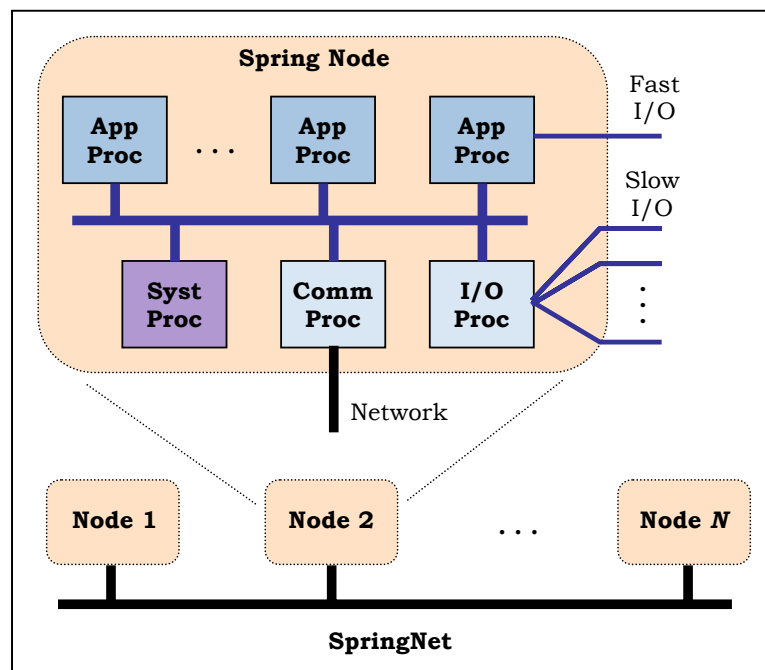


Figura 2-9. Sistemul "SpringNet"

Arhitectura sistemului "Spring" se bazează pe implementarea unui set de idei ce vizează operarea sistemelor timp-real distribuite în condiții cât mai bune de predictibilitate, flexibilitate și fiabilitate:

- partiționare funcțională,
- segmentare/partiționare a resurselor,
- prealocare selectivă a resurselor,
- garantarea apriori pentru task-urile critice și garantarea dinamică (on-line) pentru task-urile esențiale,

- integrarea planificării task-urilor pe procesoare împreună cu alocarea resurselor sistem,
- diferențierea între constrângerile de timp și cele de importanță pentru task-uri,
- utilizarea de informații suplimentare în timpul operării sistemului (de exemplu, termene, importanța task-urilor, cerințe de fiabilitate, etc.) și capacitatea de a modifica dinamic aceste informații. Aceste proprietăți conferă *reflectivitate* sistemului.

Partajarea funcțională semnifică faptul că fiecare nod al lui "SpringNet" este un sistem multiprocesor, compus din procesoare aplicație care execută task-uri aplicație garantate și planificate apriori, un procesor (sau mai multe procesoare) sistem, care execută algoritmi de planificare și primitivele sistem, un procesor pentru comunicații și unul sau mai multe procesoare I/O. Procesorul sistem prelevează procesoarele aplicație de încărcarea suplimentară impusă de planificare și de rutinele de operare ale nucleului "Spring", având ca rezultat creșterea semnificativă a vitezei sistemului și evitarea problemelor de predictibilitate ce pot apărea în execuția task-urilor garantate, din cauza întreruperilor externe sau a operării sistemului. În cazul defectării procesorului sistem, funcțiile acestuia vor putea fi preluate de unul din procesoarele aplicație.

Segmentarea/partajarea resurselor sistemului este realizată în entități bine definite ce includ: task-uri și grupuri de task-uri, segmente de memorie pentru cod, stivă, variabile locale, porturi, discuri virtuale, etc. Atât task-urile și grupurile de task-uri, cât și primitivele nucleu sunt limitate ca timp și ca resurse. Cu alte cuvinte, pentru fiecare task se cunosc: timpul de execuție (WCET – Worst Case Execution Time) și cerințele de resurse (WCRR – Worst Case Resource Requirements) pentru cazurile cele mai defavorabile.

Sistemul reține și utilizează o cantitate relativ mare de informație cu privire la caracteristicile și parametrii task-urilor și ai grupurilor de task-uri. Astfel, pe lângă WCET și WCRR, se mai pot utiliza parametri ca: nivelul de importanță, relații de precedență, cerințe de fiabilitate, etc. Nucleul utilizează în mod dinamic aceste informații pentru a planifica task-urile pentru execuție și pentru a rezerva resursele sistem necesare.

Algoritmul de planificare ce stă la baza operării nucleului "Spring" împarte setul de task-uri timp-real sistem și ale aplicației în task-uri critice (nerespectarea condițiilor de timp impuse produce consecințe majore) și task-uri esențiale (operarea sistemului și a aplicației nu este afectată în mod critic în cazul depășirii termenelor specificate). În cazul task-urilor critice, execuția acestora este planificată pe procesoarele sistemului iar resursele sunt rezervate apriori, pe toată durata operării. În cazul task-urilor esențiale, planificarea se face dinamic. Algoritmul încearcă planificarea și garantarea task-urilor esențiale adăugând pe rând câte un astfel de task la setul celor garantate până în momentul curent și verificând dacă noul set de task-uri este fezabil. Principalii parametri considerați la planificarea fiecărui task sunt: momentul de apelare, termenul limită sau perioada, WCET, setul task-urilor predecesoare și necesarul de resurse sistem (utilizate fie în mod exclusiv, fie în mod partajat). Fiind conceput pentru sisteme timp-real distribuite, algoritmul de planificare aplică o euristică ce constă din căutarea iterativă într-un arbore cu planificări posibile ale unui nou task ce se adaugă setului de task-uri garantate.

Această tehnică încearcă practic, la fiecare pas, extinderea planificării curente. Dacă o anumită ramură a arborelui rezultă într-o planificare nefezabilă, se reia căutarea prin tehnici de "backtracking" pe alte ramuri posibile.

În [Niehaus 93, Burleson 93] este propus conceptul interesant de "co-procesor de planificare" (SSCoP – "Spring Scheduling Co-Processor"), care practic implementează hardware algoritmul de planificare al sistemului Spring. Generarea timpilor de planificare pentru task-urile analizate, calcularea funcției de selectare euristică a task-ului următor, revenirea și controlul tehnicii de "backtracking", etc., sunt operații efectuate de blocuri funcționale ale SSSCoP, toate acestea fiind controlate de o mașină cu stare finită. Pentru sistemul Spring, SSSCoP este în multe privințe similar cu o memorie inteligentă, rapidă și de dimensiuni mici. Toți regiștrii SSSCoP sunt mapați în spațiul de adrese al sistemului, co-procesorul fiind încărcat cu parametrii aferenți task-urilor, iar la terminarea unui ciclu de execuție, se poate citi planificarea obținută (în cazul în care există o planificare fezabilă), constând din lista ordonată a task-urilor împreună cu timpii de execuție asociați.

Rezultatele simulărilor SSSCoP demonstrează eficiența acestei soluții [Niehaus 93]: o creștere de peste 30 de ori a vitezei în comparație cu planificatorul software, iar viteza de execuție a secvenței principale din algoritmul de planificare este îmbunătățită cu trei ordine de mărime.

3 Problemele dezvoltării STR stricte pentru aplicații critice de achiziție și prelucrare numerică de semnal

3.1 Dezvoltarea STR – concepte și arhitecturi

Marea majoritate a lucrărilor din domeniul sistemelor TR evidențiază faptul că dezvoltarea sistemelor și aplicațiilor TR trebuie să parcurgă o serie de etape (obligatorii) [Mok 83, Stankovic 88, Stankovic 92a, Stankovic 92b, Ostroff 92, Panzieri 93b, Parashkevov 94, Ghosh 94, Ramamritham 94, Chapman 94, Bellini 00, Letia 00]:

- 1) Concepție și proiectare
- 2) Specificarea funcțională (programare) și de comportare în timp
- 3) Verificare/validare formală
- 4) Analiza temporală de program și de fezabilitate a execuției
- 5) Generarea codului, a planificării și implementarea pe o arhitectură țintă
- 6) Testarea implementării

Toate etapele dezvoltării STR trebuie să aibă în vedere introducerea timpului ca și coordonată esențială a sistemului, și nu doar ca o caracteristică în funcție de care să se adapteze sistemul gata dezvoltat [Stankovic 92b]. Tot în această lucrare este evidențiată dificultatea cerințelor unei proiectări corespunzătoare a STR, ce derivă din următorul paradox: majoritatea tehnicilor de specificare, proiectare și verificare se bazează pe abstractizări (care ignoră detaliile de implementare), pe când, pe de altă parte, constrângerile de timp derivă din detalii concrete de implementare și ale mediului.

În prezent, un număr încă foarte mare de sisteme TR (în special cele stricte) reprezintă practic implementări ad-hoc, puternic orientate pe aplicațiile vizate și particularizate pentru arhitecturile implementării. Multe sisteme TR se bazează pe adaptarea și optimizarea (în sensul creșterii vitezei de reacție) a unor versiuni de sisteme de operare time-sharing tradiționale. Un exemplu în acest sens îl reprezintă sistemul *Real-Time Linux*, care derivă din varianta de *UNIX* pentru calculatoare personale – *Linux*, prin instalarea unor pachete soft (*patches*) care modifică unele structuri de date și rutine ale nucleului, pentru a permite execuția proceselor ținând cont de unele specificații de timp.

Sistemele de operare tradiționale se bazează pe noțiunea că task-urile aplicației emit cereri pentru resurse sistem, iar sistemul de operare este proiectat pentru a deservi aceste cereri (în manieră asincronă). Rezultă astfel o comportare bună în situațiile comune de operare ("average case behavior").

Transformarea unui sistem de operare time-sharing clasic pentru a servi ca platformă pentru aplicații TR, se face de regulă adaptând nucleul în scopul creșterii vitezei de reacție a sistemului [Stankovic 92b]:

- sunt vizate schimbările mai rapide de context;
- nucleul va avea dimensiuni mai mici (cu funcționalitatea minimală asociată, ca rezultat);
- sistemul va răspunde mai rapid la întreruperi externe;
- se minimizează intervalele în care întreruperile sunt dezactivate;
- sunt introduse și gestionate fișiere secvențiale speciale, capabile a acumula date într-un ritm rapid.

Pentru a face față cerințelor temporale impuse de operarea TR, nucleul unui sistem tradițional va fi adaptat astfel:

- gestionează un ceas de timp-real (*RTC – Real-Time Clock*);
- se oferă un mecanism de planificare bazat pe priorități;
- sunt puse la dispoziția task-urilor aplicațiilor mecanisme speciale pentru alarme și contoare de timp cu expirare ("timeouts");
- task-urile pot invoca primitive sistem pentru a stabili întârzieri și pentru a-și întrerupe/relua execuția.

Din punct de vedere al arhitecturilor sistem, în prezent există o multitudine de mecanisme și concepte special prevăzute pentru a crește eficiența și viteza de procesare a sistemelor. Acestea prezintă însă probleme în ceea ce privește predictibilitatea STR, de exemplu îngreunând sau chiar împiedicând calculul timpilor maximi de execuție (*WCET – Worst Case Execution Time*) ai task-urilor programului [Stankovic 92a, Stankovic 92b, Colnarić 93, Chapman 94]:

- paralelismul operării componentelor interne ale procesoarelor moderne;
- mecanismele de pipeline din procesoare;
- memoriile cache;
- mecanismele de întrerupere;
- utilizarea resurselor partajate;
- memoria virtuală;
- accesarea directă a memoriei (DMA), prin mecanisme tip "cycle stealing";
- magistralele sistem ce permit transferuri asincrone de date (în special pentru sisteme cu resurse partajate).

Tehnicile și conceptele de tipul celor menționate mai sus induc o comportare puternic asincronă și impredictibilă din punct de vedere al timpilor de execuție pentru sistemele TR, și trebuie în consecință, evitate în faza de proiectare a arhitecturii hardware a STR.

3.2 Predictibilitatea execuției task-urilor TR cu termene stricte

Predictibilitatea execuției task-urilor în sistemele TR stricte reprezintă o caracteristică și în același timp o problemă extrem de importantă. Una dintre consecințele cele mai importante ale asigurării predictibilității unui task oarecare,

reprezintă posibilitatea garantării respectării termenelor și a comportamentului în timp specificate pentru task-ul respectiv, pe tot parcursul operării sistemului, incluzând situațiile cele mai defavorabile (de exemplu, în cazuri de încărcare maximă sau de supra-solicitare a resurselor sistemului).

Predictibilitatea are implicații directe în conceperea mecanismelor de alocare a resurselor sistem (procesor, memorie, accese I/O, comunicații, etc.), impunând necesitatea integrării restricțiilor de timp [Stankovic 92b]. În particular, proiectarea algoritmilor de planificare pentru sistemele TR, trebuie să aibă în vedere asigurarea predictibilității de operare, cel puțin pentru task-urile TR cu specificații stricte de comportare temporală.

În [Ramamritham 94] se remarcă faptul că pentru a verifica predictibilitatea sistemelor TR (adică, pentru a prezice dacă task-urile își vor respecta constrângerile temporale impuse), este necesară efectuarea unei analize a planificabilității (sau, o verificare a fezabilității) sistemului. Pentru cazul sistemelor TR stricte, unde respectarea specificațiilor temporale este o cerință esențială, ce trebuie garantată în orice condiții de operare, faza de analiză a planificabilității trebuie efectuată înainte lansării în execuție a sistemului (aplicației). Prin urmare, o etapă esențială în dezvoltarea aplicațiilor TR critice este *analiza offline* a fezabilității acestora.

O problemă majoră din punctul de vedere al predictibilității este utilizarea neîngrădită a întreruperilor [Stewart 01]:

- Întreruperile sunt cea mai frecventă cauză a fenomenului de inversare a priorităților (*priority inversion*) dintr-un sistem cu planificare pe bază de prioritate: execuția unui task cu prioritate mai mare ce va accesa o anumită resursă partajată, este blocată de un task cu prioritate mai mică ce a fost întrerupt de către primul, în timpul accesării aceleiași resurse.
- Rutinele de tratare a întreruperilor (*interrupt handlers*) pot întrerupe orice task din sistem, indiferent de prioritatea acestuia. Pe de altă parte, aceste rutine nu sunt (nu pot fi) planificate, corespunzând de regulă apariției unor evenimente asincrone în sistem.
- Rutinele de tratare a întreruperilor sunt executate în contexte diferite de cele ale task-urilor aplicației, forțând astfel utilizarea variabilelor globale pentru a putea schimba informație cu restul aplicației.
- Din punct de vedere al testării aplicațiilor, întreruperile sunt dificil de simulat și de testat, pe de o parte din cauză că uneltele soft de testare permit foarte rar setarea de puncte de control a programului în interiorul handlerelor, iar pe de altă parte, din cauza asincronismului apariției întreruperilor.

Se impune în consecință, mai ales în cazul sistemelor TR stricte, minimizarea pe cât posibil a utilizării întreruperilor. În extremis, [Stewart 01] subliniază faptul că un sistem TR ideal nu prezintă nici o întrerupere.

Capitolul 5 al lucrării conține o discuție și mai detaliată a avantajelor și dezavantajelor celor două abordări – preemptivă și non-preemptivă. Tot aici va fi prezentată o clasă de aplicații TR stricte, ce conțin task-uri cu execuție fixă în cadrul perioadei și a căror planificare nu poate fi rezolvată cu algoritmi preemptivi.

3.3 Specificarea, programarea și analiza temporală a aplicațiilor TR critice

În prezent, există o varietate mare de metode de specificare și verificare formală a sistemelor și aplicațiilor TR, din care însă, nu a reușit nici una să fie acceptată în întregime de mediul academic și industrie. Standardizarea tehnicilor și modelelor utilizate în dezvoltarea sistemelor TR reprezintă una din problemele importante din domeniu.

Pe de altă parte, majoritatea metodelor de specificare formală utilizate pentru sistemele TR derivă din abordări clasice, care nu pun un accent deosebit pe coordonata timp [Chapman 94, Parashkevov 94]: Time Petri Nets, Timed Petri Nets (bazate pe rețelele Petri), Timed CSP, CSP+T, CCSR, Timed CCS (bazate pe algebre de proces), etc. De aici rezultă o serie de inconveniente:

- Multe metode se bazează pe conceptul de timp global (și de ceasuri sincronizate global, în cazul STR distribuite). În practică, această premisă este foarte dificil, dacă nu imposibil, de implementat.
- Fiind derivate din tehnici ne-temporale, metodele de specificare formală au posibilități de expresie limitate de predecesorii lor.
- Există un decalaj important între rezultatele cercetărilor din domeniul metodelor formale de timp-real și implementările practice din industrie.

Din cele de mai sus se desprinde necesitatea conceperii unor unelte puternice și intuitive, destinate specificării incrementale și verificării automatizate a proiectelor sistemelor TR.

Faza ce urmează specificării și verificării formale a STR – programarea aplicațiilor – necesită, la rândul ei, dezvoltarea unor limbaje specializate, cu următoarele caracteristici principale [Parashkevov 94]:

- dependența cât mai redusă față de hardware și sistemul de operare a platformelor țintă;
- trebuie să posede facilitățile limbajelor moderne de programare (set puternic de tipuri de date, programare structurată, modularitate, interfețe bine definite, posibilitatea compilării separate a modulelor, etc.);
- trebuie să ofere primitivele necesare accesării și controlului elementelor hardware ca registri, adrese I/O, întreruperi, etc.;
- furnizarea de suport natural pentru exprimarea explicită a concurenței, a comunicării și sincronizării proceselor aplicației;
- trebuie să ofere programatorului mecanismele necesare descrierii explicite a proprietăților și constrângerilor temporale ale codului programelor: execuție periodică, contoare de timp, termene pentru task-uri și grupuri de task-uri, întâzieri, etc.;
- restricționarea sau chiar interzicerea construcțiilor de program și a structurilor de date ce nu sunt limitate în timp și spațiu: recursivitatea, structuri de memorie dinamică, bucle cu un număr nelimitat sau necunoscut de iterații la momentul compilării, etc.;
- trebuie să ofere informații suplimentare utilizate în analiza planificabilității codului compilat.

3.4 Modelul unui STR strict pentru aplicații critice

Sintetizând cerințele și problemele enumerate în paragrafele anterioare, se poate defini un model generic, ce poate fi urmărit în conceperea tehnicilor și metodelor de dezvoltare a sistemelor TR stricte, orientate pe aplicații critice de achiziție și prelucrare numerică a semnalelor (DSP-based systems) și de control digital încorporat (embedded systems). Modelul STR strict prezintă următoarele caracteristici de bază:

- Definirea clară a mai multor faze necesare în dezvoltarea STR: specificarea, proiectarea, verificarea/validarea, analiza de program și analiza fezabilității.
- Timpul, ca și coordonată esențială a operării sistemului, va trebui introdus ca parametru cheie în toate fazele dezvoltării STR.
- Metodele cu care se abordează fazele dezvoltării STR trebuie să fie unitare și omogene.
- Asigurarea predictibilității execuției task-urilor TR stricte ale sistemului reprezintă o cerință esențială, ce afectează toate etapele dezvoltării și implementării.
- Algoritmii de planificare trebuie atent concepuți astfel încât să asigure execuția predictibilă, cu respectarea termenelor impuse task-urilor TR stricte și conferind sistemului, pe de altă parte, eficiență și flexibilitate cât mai mari.
- Programarea task-urilor aplicațiilor TR stricte trebuie realizată cu ajutorul unor limbaje care să permită exprimarea de constrângeri temporale și totodată să permită analiza temporală a programelor (determinarea timpilor WCET, BCET, etc.).
- Pentru asigurarea predictibilității, următoarele mecanisme și concepte sistem trebuie evitate:
 - memorii cache
 - memorii virtuale
 - transferurile DMA (mai ales cele de tipul "cycle stealing")
 - magistralele sistem ce permit transferuri asincrone de date
- O atenție specială trebuie acordată utilizării (limitate) a următoarelor mecanisme:
 - întreruperile (conferă sistemului un comportament puternic asincron și, deci, impredictibil)
 - pipelining
- Sistemul gestionează un ceas de timp-real propriu, care definește domeniul temporal sistem.
- Comunicarea inter-task-uri trebuie să utilizeze mecanisme speciale, predictibile, care să evite operarea asincronă.
- Accesul la resurse partajate va trebui controlat și rezolvat prin mecanisme de sincronizare predictibile, sau, dacă se poate, chiar evitat.

SECȚIUNEA II. CONTRIBUȚII LA PROIECTAREA ȘI DEZVOLTAREA STR STRICTE PENTRU APLICAȚII CRITICE

Capitolele secțiunii de față conțin contribuțiile aduse în domeniul proiectării și dezvoltării sistemelor timp-real stricte. Secțiunea reprezintă astfel fundamentul teoretic al tezei, ce se bazează pe ideea introducerii mecanismelor non-preemptive în proiectarea, analiza și dezvoltarea STR stricte pentru aplicații critice de APNS și control digital încorporat.

Capitolul 4 introduce și studiază setul omogen de modele pentru timpul real, semnale și task-uri, ce constituie baza proiectării și dezvoltării STR stricte pentru aplicații critice. Modelul task-urilor TR stricte, ModX-ul, este prezentat și analizat în detaliu.

Planificarea aplicațiilor TR stricte este abordată în Capitolul 5, accentul fiind pus pe tehnicile de planificare non-preemptivă, capabile să garanteze operarea cu maximă predictibilitate a STR stricte, în orice condiții. Sunt studiați doi dintre algoritmi cei mai utilizați în prezent pentru planificarea non-preemptivă și sunt introduse o serie de adaptări și de noi algoritmi, în funcție de diversele cerințe impuse de tipul aplicațiilor timp-real ce se doresc implementate.

Capitolul 6 prezintă analiza performanțelor algoritmilor de planificare discutați în această secțiune. Evaluarea performanțelor a fost realizată pe baza unor programe de test special dezvoltate și executate pe cele 12 stații de calcul ale laboratorului DSPLabs, rezultând un total de peste 150000 de teste.

Modelul unui sistem complet (al unei metodologii), OPEN-HARTS, care unifică toate conceptele introduse și discutate în această secțiune, este prezentat în Capitolul 7. Sistemul OPEN-HARTS este conceput ca un set de soluții pentru problemele ridicate de proiectarea și dezvoltarea unitară a sistemelor și aplicațiilor TR stricte.

4 Informația de timp, evenimente și acțiuni. Modelul task-ului TR strict

Timpul, ca și coordonată esențială a STR stricte, trebuie să fie implicat în toate fazele de dezvoltare: specificare și verificare formală, programare, analiză, planificare și execuție.

Capitolul de față analizează informația necesară pentru a descrie comportarea în domeniul timp a elementelor unui STR strict și utilizarea acesteia în toate fazele dezvoltării sistemului. Analiza are ca rezultat extragerea unui set minim necesar de parametri temporali (predicate), suficient de intuitiv pentru a fi utilizat cu ușurință în specificarea și programarea sistemului și cu ajutorul cărora se construiește în final un model al task-ului TR strict.

4.1 Timpul sistem și un model temporal pentru dezvoltarea STR stricte

Funcționarea sistemelor de calcul în general, și a STR în particular, se bazează pe o frecvență de tact generată de un oscilator intern sau extern. Frecvența de tact definește așa-numitul "ciclu procesor" sau "ciclu instrucțiune" al unității de procesare a sistemului, reprezentând durata execuției unei instrucțiuni procesor (exprimată în secunde, Hz sau perioade de tact). În același timp, frecvența de tact alimentează baza de timp a STR, anume ceasul de timp-real (*RTC*, *real-time clock*). Prin urmare, din punctul de vedere al STR, granularitatea de timp minimă cu care sistemul poate lucra este egală cu durata unui ciclu procesor.

Exemplul 4-1.

Pentru exemplificare, să considerăm cazul procesorului numeric de semnale Motorola DSP56307 [Motorola 98, Motorola 99, Motorola 00], care, pentru o aplicație oarecare, operează la o frecvență de tact,

$$CoreCLK = 32 \text{ MHz.}$$

Ciclu procesor pentru DSP56307 este egal cu o perioadă de tact, rezultând astfel granularitatea de timp minimă ca fiind egală cu durata ciclului procesor, adică

$$\Delta t_{CLK} = 31.25 \text{ ns.}$$

Mai departe, în cazul în care se utilizează un RTC cu un factor de divizare (prescalare) de 4, unitatea de timp măsurată de către RTC este:

$$\Delta t_{RTC} = 4 \cdot \Delta t_{CLK} = 125 \text{ ns.}$$

□

Concluzia celor de mai sus este că sistemele digitale de calcul, incluzând și STR, operează cu valori discrete de timp. STR utilizează ca bază de timp pentru tratarea evenimentelor și a task-urilor, *ceasul de timp-real*, *RTC*.

Utilizarea RTC a cărei arhitectură uzuală include un contor de timp, definește domeniul temporal de operare al sistemelor TR. Astfel,

$$t_0 = 0$$

poate fi identificat cu momentul startării sistemului (cuplarea la tensiune și inițializarea contorului RTC), fiind denumit "momentul inițial".

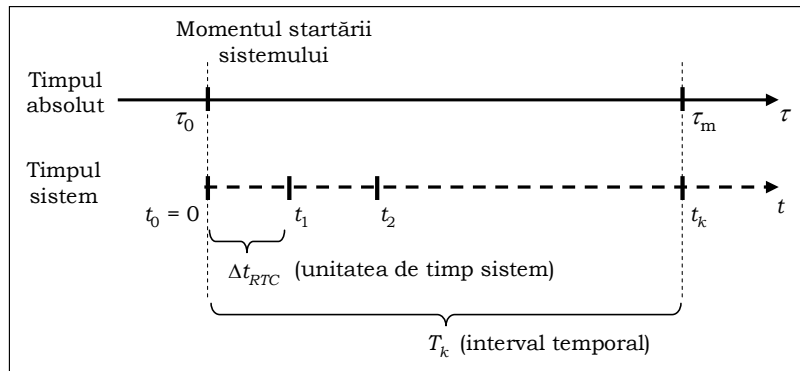


Figura 4-1. Timpul sistem și timpul absolut pentru STR

În Figura 4-1 sunt reprezentate cele două domenii temporale: timpul sistem și cel absolut, la care se raportează. Variabilele de timp au fost notate distinct, τ pentru timpul absolut și t pentru timpul sistem. La momentul τ_0 are loc startarea STR (deci, $t_0 = 0$).

Oricare două instanțe succesive ale timpului sistem, t_i și t_{i+1} sunt distanțate cu Δt_{RTC} în domeniul temporal absolut, și cu 1 în domeniul temporal al sistemului. Două evenimente care apar în intervalul Δt_{RTC} sunt văzute ca fiind simultane de către sistem. De aici rezultă valoarea unității temporale sistem și faptul că aceasta permite existența metricilor temporale.

Lungimea unui interval temporal oarecare T_k poate fi determinată astfel:

$$\begin{cases} T_k = \tau_m - \tau_0 = k \cdot \Delta t_{RTC} & \text{pentru timpul absolut} \\ T_k = t_k - t_0 = k & \text{pentru timpul sistem} \end{cases} \quad (4-1)$$

Legătura dintre cele două domenii temporale este sintetizată de relația de corespondență a unei instanțe de timp (un punct din cele două domenii):

$$\tau_i = \tau_0 + (t_i - t_0) \cdot \Delta t_{RTC} \quad (4-2)$$

Importanța relației (4-2) și a utilizării ambelor modele temporale derivă din faptul că în faza de specificare a comportamentului temporal al STR, utilizatorul/programatorul de aplicații judecă evenimentele din perspectiva timpului absolut, pe când sistemul TR operează în domeniul temporal discret, propriu. În faza imediat următoare din cadrul procesului de dezvoltare a sistemului/aplicației TR, anume în faza de verificare și validare, este necesară convertirea în mod corespunzător a specificațiilor, din modelul temporal absolut în cel sistem.

În concluzie, modelul temporal propus pentru dezvoltarea STR stricte are următoarele caracteristici principale (vezi Capitolul 2):

- Domeniul temporal are o structură liniară, discretă;
- Existența momentului inițial: $t_0 = 0$ (adică domeniul temporal este limitat la stânga);
- Domeniul temporal sistem poate fi echivalat cu mulțimea numerelor naturale:

$$t \in \mathbf{N} \quad (4-3)$$

- Momentul inițial corespunde startării STR, adică momentului τ_0 din domeniul temporal absolut;
- Unitatea de timp corespunde unui interval de lungime Δt_{RTC} din domeniul temporal absolut;
- Timpul sistem permite exprimarea de relații cantitative între instanțe individuale (puncte) sau între intervale temporale. Astfel, modelul temporal prevede existența unei metrice pentru timp;
- Cu ajutorul metricei pentru timp se poate determina lungimea unui interval temporal, cu (4-1);
- Corespondența unei instanțe de timp din domeniul temporal sistem în cel absolut se poate determina cu (4-2).

Deoarece modelul temporal propus permite operarea atât cu instanțe punctuale de timp cât și cu intervale temporale, pentru clarificare, introducem următoarele notații care vor fi utilizate în continuare în lucrare:

Tabelul 4-1. Notații ale entităților temporale utilizate în modelul propus

Entitate	Notăție	Semnificație
Instanță de timp	t	Variabila de timp sau parametru. Utilizat ca parametru, identifică un anumit moment de timp (punct în domeniul temporal). Distincția dintre domeniul temporal absolut și domeniul temporal sistem se va face explicit, acolo unde e cazul: <ul style="list-style-type: none"> • $t \in \mathbf{N}$, pentru modelul temporal sistem, • $t \in \mathbf{R}$, pentru modelul temporal absolut.
Interval temporal	$T = [t_{sT}, t_{eT}]$	Este caracterizat printr-o instanță de început (t_{sT}) și una sfârșit (t_{eT}), care pot fi precizate (exprimare explicită, absolută) sau nu (exprimare implicită, relativă). Utilizat pentru a identifica un anumit interval temporal cât și pentru a exprima lungimea (durata) intervalului.

4.2 Evenimente în sisteme TR stricte

Evenimentele sunt definite ca o modalitate prin care diverși agenți (în cazul nostru, STR, utilizatorul/programatorul de aplicații TR, etc.) clasifică anumite tipare semnificative de schimbare [Allen 94]. Există destul de puține restricții în stabilirea semanticii evenimentelor, cu excepția că acestea trebuie să implice cel puțin un obiect într-un interval de timp sau cel puțin o schimbare de stare. Diferența dintre "evenimente" și "stări" poate fi exprimată intuitiv și prin faptul că primele descriu schimbări pe când celelalte descriu aspecte care nu se schimbă. Cu alte cuvinte, se poate spune că evenimentele "apar" și stările "se mențin".

Într-o accepțiune mai restrânsă și din punctul de vedere al specificării și operării STR prezentat în lucrarea de față, evenimentele sunt descrise prin *semnale* ce determină schimbări de stare ale sistemului.

Semnalele pot fi clasificate după mai multe criterii, trei dintre acestea, care ni se par de interes în problema dezvoltării STR stricte, fiind:

- a) După modul (rata) apariției semnalelor:
 - a.1) semnale periodice
 - a.2) semnale sporadice (aperiodice)
- b) După numărul de apariții:
 - b.1) semnale cu apariție singulară
 - b.2) semnale cu apariție temporară
 - b.3) semnale cu apariție permanentă
- c) După sursa semnalelor, raportată la STR:
 - c.1) semnale de intrare
 - c.2) semnale de ieșire
 - c.3) semnale interne

În cele ce urmează vom analiza toate aceste tipuri de semnale, extrăgând principalele caracteristici cu privire la comportarea în timp. Rezultatul analizei va sta la baza construirii unui model minimal, intuitiv al semnalelor care să poată fi utilizat în fazele de specificare, verificare/validare și analiză a sistemelor TR în general, și a STR stricte în particular.

a.1) *Semnale periodice*

Semnalele periodice sunt caracterizate din punct de vedere al comportamentului în timp prin faptul că au o perioadă cunoscută. Notăm perioada unui semnal oarecare S_i cu $T_{pr}^{S_i}$. Dacă cea de-a k apariție a lui S_i are loc la momentul t_k , atunci cea de-a $(k+1)$ apariție va avea loc la momentul

$$t_{k+1} = t_k + T_{pr}^{S_i} \quad (4-4)$$

Exemple de semnale periodice sunt liniile de tact din comunicațiile sincrone, liniile de date și control din sistemele de achiziție a datelor, etc.

a.2) *Semnale aperiodice (sporadice)*

Semnalele aperiodice se caracterizează ca și comportare în timp prin faptul că intervalele de timp dintre mai multe apariții consecutive nu sunt egale. Așadar, comportamentul în timp al semnalelor aperiodice este mult mai puțin restricționat în comparație cu semnalele periodice. În cazul sistemelor și al mediilor deterministe (cazul considerat în lucrarea de față), se poate însă stabili sau determina un interval de timp minim, notat cu $T_{pr}^{S_i}$, care separă oricare două apariții consecutive ale unui semnal aperiodic S_i . Dacă cea de-a k apariție a lui S_i are loc la momentul t_k , atunci cea de-a $(k+1)$ apariție va avea loc la momentul

$$t_{k+1} \geq t_k + T_{pr}^{S_i} \quad (4-5)$$

Se observă că relațiile (4-4) și (4-5), ce specifică momentul unei noi apariții a semnalului periodic, respectiv aperiodic, S_i , față de instanța ultimei apariții, sunt asemănătoare. Această observație permite abordarea (tratarea) celor două tipuri de semnale într-o manieră unitară, așa cum se va vedea în subcapitolul următor.

Exemple de semnale aperiodice sunt comunicațiile asincrone, interfețele cu operatorul, subsistemele de comandă cu bucle de reacție și reglare, interfețe intrare-ieșire ce operează pe bază de întreruperi, etc.

b.1) *Semnale cu apariție singulară*

Semnalele cu apariție singulară sunt în general semnale care startează sau opresc diverse procese. În cazul acestora, se pune problema comportării sistemului *până* în momentul apariției semnalului și *după* apariție. Apariția în sine semnalului este un eveniment asincron. Tratarea acestui tip de semnale va fi detaliată în subcapitolul următor.

Exemple de semnale cu apariție singulară sunt comenzile utilizator sau sistem, de tip "start/stop", comenzile de schimbare a modului de operare, etc.

b.2) *Semnale cu apariție temporară*

Semnalele cu apariție temporară caracterizează anumite procese ce se desfășoară pe un interval de timp limitat de-a lungul operării sistemelor și a interacțiunii acestora cu mediul. Faptul că un anumit semnal S_i are apariție temporară poate fi caracterizat prin durata intervalului temporal în care au loc evenimentele apariției semnalului (durata intervalului în care semnalul este "activ", $T_{act}^{S_i}$). În cazul în care semnalul este și periodic, durata intervalului de apariții poate fi echivalată cu numărul de apariții:

$$N^{S_i} = \frac{T_{act}^{S_i}}{T_{pr}^{S_i}}$$

unde $T_{pr}^{S_i}$ este perioada semnalului S_i .

Se poate observa că semnalele cu apariție singulară (b.1) pot fi tratate ca un caz particular de semnale cu apariție temporară, în condițiile în care numărul de apariții este egal cu 1.

Exemple de semnale cu apariție temporară sunt interfețele cu operatorul care permit comandarea mai multor moduri disjuncte de operare ale sistemului.

b.3) Semnale cu apariție permanentă

Semnalele cu apariție permanentă caracterizează procesele care se desfășoară pe întreaga durată a operării sistemelor. Acest tip de semnale pot fi considerate la rândul lor un caz particular de semnale cu apariție temporară, în condițiile în care numărul de apariții și durata intervalului de apariții sunt infinite.

Exemple de semnale cu apariție permanentă se întâlnesc în cazul sistemelor de achiziție numerică de semnal care operează în mod continuu (de exemplu, sistemele de achiziționare a parametrilor meteorologici din stațiile meteo, termometre digitale, sisteme de control digital al operării automate a ușilor, etc.).

c) Clasificarea după sursa semnalelor, raportată la STR

Figura 4-2 exemplifică cele trei tipuri de semnale, clasificate după sursa acestora: semnale de intrare (S_5), de ieșire (S_6) și semnale interne (S_1, S_2, S_3 și S_4).

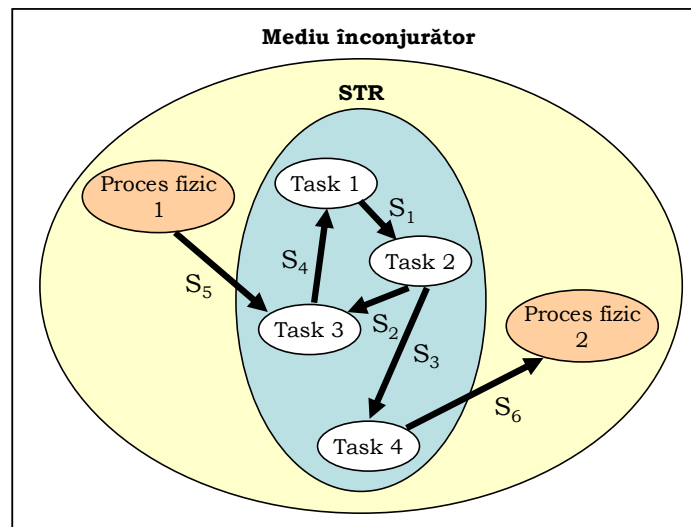


Figura 4-2. Ilustrare a celor trei tipuri de semnale: de intrare, de ieșire și interne

c.1) Semnale de intrare

Semnalele de intrare au ca sursă mediul înconjurător în cadrul căruia operează STR. Față de aceste semnale, STR se comportă ca un sistem reactiv: apariția semnalului de intrare este urmată de tratarea acestuia în cadrul sistemului (cu ajutorul unui task sau al unui set de task-uri specifice). De regulă, rezultatele tratării semnalelor de intrare se concretizează în alte semnale – interne sau de ieșire, reprezentând reacția STR.

Tratarea semnalelor de intrare este o problemă deosebit de importantă pentru dezvoltarea și operarea STR stricte, și modul în care este rezolvată diferențiază diferitele arhitecturi și concepte de sisteme TR. Problema va fi abordată în detaliu în subcapitolul următor.

c.2) Semnale de ieșire

Semnalele de ieșire sunt semnalele prin intermediul cărora STR acționează asupra mediului înconjurător. Relativ la aceste semnale, STR se comportă ca un sistem interactiv, de tip "master". Toate caracteristicile semnalelor sunt stabilite de către sistemul care le generează.

c.3) Semnale interne

Semnalele interne sunt rezultatul interacțiunii dintre task-urile sistemului și constau de regulă din: semnale de sincronizare și de notificare, și semnale ce conțin informație schimbată între task-uri (comunicația de date și parametri, *IPC – Inter-Process Communication*). Caracteristicile temporale ale semnalelor interne sunt dictate de caracteristicile de comportare în timp a task-urilor implicate.

Tabelul 4-2 oferă o reprezentare sintetică a modelului semnalelor prin descrierea principalilor parametri de comportare temporală ai semnalelor, discutați până acum. Parametrii din tabel vor fi utilizați pentru modelarea semnalelor în faza de specificare a STR, urmând a avea implicații în definirea comportării temporale ale task-urilor sistem.

Tabelul 4-2. Modelul semnalelor pentru specificarea STR

Parametru	Descriere	Categorie de semnale
$T_{pr}^{S_i}$	Perioada semnalului S_i .	Semnale periodice
	Durata minimă de timp ce separă oricare două apariții consecutive ale semnalului S_i .	Semnale aperiodice (sporadice)
$T_{act}^{S_i}$	Durata intervalului în care au loc apariții ale semnalului S_i .	Semnale temporare
N^{S_i}	Numărul de apariții ale semnalului S_i .	
	$N^{S_i} = 1$	Semnale singulare
	$N^{S_i} = n$, finit	Semnale temporare
	$N^{S_i} = \infty$	Semnale permanente

4.3 Tratarea evenimentelor: acțiuni TR stricte

Într-o accepțiune generică, acțiunile sunt definite ca activități pe care un sistem (obiect, persoană, etc.) le desfășoară în anumite condiții [Allen 94]. Din perspectiva tematicii abordate în lucrarea de față, acțiunile sunt văzute în strânsă relație de cauzalitate cu evenimentele. Cu alte cuvinte, apariția unui anumit eveniment sau a unei secvențe bine definite de evenimente, provoacă o acțiune, și reciproc, o acțiune anume, provoacă apariția unui eveniment sau a unui set bine precizat de evenimente. Relația de cauzalitate eveniment \leftrightarrow acțiune este în strânsă concordanță cu accepțiunea noastră de STR determinist (care operează într-un mediu determinist) și este de asemenea în legătură directă cu proprietatea de sistem reactiv (sau reflex) – proprietate importantă în cazul STR stricte.

Din perspectiva STR, o acțiune este reprezentată printr-un task sau o secvență bine definită de task-uri specifice (subgraf de task-uri). Relația de cauzalitate dintre evenimente (semnale) și acțiuni (task-uri) implică faptul că unele proprietăți ale semnalelor determină proprietăți similare pentru task-uri, și reciproc, mai ales în ceea ce privește comportamentul temporal în cazul STR.

Înainte de a aborda problema tratării semnalelor de către task-urile unui STR se impune introducerea unui model temporal preliminar al task-ului TR.

Definiția 4-1.

Un *task* (notat M_i) reprezintă o secvență de instrucțiuni pe care sistemul o execută în scopul desfășurării unor acțiuni. Execuția task-urilor în cadrul sistemelor TR se face într-o manieră *planificată*, ținând cont de coordonata timp și de caracteristicile temporale ale task-urilor. □

Fiecare task al unui STR este *invocat* de către sistem pentru a fi executat, fie ca răspuns al sistemului la un eveniment sau la un set de evenimente, fie pentru a se obține apariția unui eveniment sau a unui set de evenimente. Notăm *momentul de invocare* (*invocation time, request time*) al task-ului M_i cu $t_{rq}^{M_i}$.

Task-urile unui STR au, de regulă un număr oarecare (mai mare sau egal cu 1) de execuții. Notăm cu N^{M_i} numărul total de execuții ale task-ului M_i ($N^{M_i} \geq 1$). Fiecare ciclu k de execuție începe din momentul invocării $t_{rq,k}^{M_i}$, și se termină la momentul invocării pentru ciclul următor, $t_{rq,k+1}^{M_i}$.

Definiția 4-2.

Definim *durata de execuție* a task-ului M_i (*execution interval*) pentru ciclul k de execuție, notată cu $T_{ex,k}^{M_i}$, intervalul de timp (*nenu*) necesar execuției lui M_i în cadrul sistemului (intervalul de timp în care procesorul sistemului este alocat execuției lui M_i). □

Așa cum rezultă din Capitolele 2 și 3 ale lucrării, durata de execuție a unui task variază de la o execuție la alta. Așadar, $T_{ex,k}^{M_i}$ nu este constantă, depinzând de o serie de factori, cum ar fi valoarea momentană a informației preluată de la task-urile executate anterior și de calea pe care o urmează execuția în cadrul secvențelor de instrucțiuni ale task-ului (de exemplu, în cazul instrucțiunilor de ramificare de tip "for" sau "case"), etc. În cazul sistemelor TR și în particular, al STR stricte, faptul că durata de execuție a task-urilor nu e constantă de la o execuție la alta constituie o problemă dificilă, în special pentru proiectarea și implementarea algoritmilor de planificare (vezi Capitolul 3). Vom reveni la acest subiect important în capitolele următoare ale lucrării.

Definiția 4-3.

Numim *periodă* a task-ului M_i (*period*), notată cu $T_{pr}^{M_i}$, durata minimă a unui ciclu de execuție al lui M_i , adică intervalul minim de timp definit de oricare două momente de invocare consecutive ale task-ului M_i :

$$T_{pr}^{M_i} = \min_{1 \leq k \leq N^{M_i}} \left\{ \left(t_{rq,k+1}^{M_i} - t_{rq,k}^{M_i} \right) \right\} \quad (4-6)$$

unde $k \in \{1, \dots, N^{M_i}\}$ reprezintă numărul ciclului curent de execuție. □

Se observă că Definiția 4-3 și relația (4-6) permit atât existența task-urilor periodice, cât și a celor aperiodice (sporadice) ale unui STR. Acestea pot fi definite într-o manieră unitară, asemănătoare cu cea a semnalelor periodice și aperiodice (relațiile (4-4), (4-5) și [Jeffay 91]).

Definiția 4-4.

Fie $t_{rq,k}^{M_i}$ cel de-al k -lea moment de invocare al task-ului M_i (adică, momentul ce definește cel de-al k -lea ciclu de execuție al lui M_i). M_i se numește *task periodic*, de perioadă $T_{pr}^{M_i}$, dacă prezintă o comportare în timp ce respectă următoarele reguli:

- i) Cea de-a $(k + 1)$ invocare a lui M_i apare exact la momentul:

$$t_{rq,k+1}^{M_i} = t_{rq,k}^{M_i} + T_{pr}^{M_i} \quad (4-7)$$

- ii) Cea de-a k -a execuție a lui M_i trebuie să înceapă nu mai devreme de $t_{rq,k}^{M_i}$ și să se termine nu mai târziu de $t_{rq,k}^{M_i} + T_{pr}^{M_i}$. □

Reformulând Definiția 4-4, spunem că task-ul periodic M_i solicită $T_{ex,k}^{M_i}$ unități din timpul procesor în fiecare interval $\left[t_{rq,k}^{M_i}, t_{rq,k}^{M_i} + T_{pr}^{M_i} \right]$.

Definiția 4-5.

Fie $t_{rq,k}^{M_i}$ cel de-al k -lea moment de invocare al task-ului M_i . M_i se numește *task aperiodic (sporadic)*, dacă prezintă o comportare în timp ce respectă următoarele reguli:

- i) Cea de-a $(k + 1)$ invocare a lui M_i apare la momentul:

$$t_{rq,k+1}^{M_i} \geq t_{rq,k}^{M_i} + T_{pr}^{M_i} \quad (4-8)$$

- ii) Cea de-a k -a execuție a lui M_i trebuie să înceapă nu mai devreme de $t_{rq,k}^{M_i}$.

□

Un sistem poate conține un număr oarecare de task-uri, ce pot fi *independente* sau *dependente* unul față de celălalt. Dependentele dintre task-urile unui STR sunt de două tipuri: (i) *dependență de control (control dependence, control flow)* și (ii) *dependență de date (data dependence, data flow)*, care pot fi grupate formal cu ajutorul așa-numitei *dependențe de program (program dependence)*. Dependentele de program ale task-urilor impun restricții (de multe ori, dificil de rezolvat) pentru planificarea în execuție a acestora. Un task M_j nu poate fi planificat și executat la un moment oarecare t decât în condițiile în care sunt îndeplinite *toate* dependențele sale de program, semnificând de exemplu, ca task-ul M_i să fi obținut prin execuție, până la momentul $(t - 1)$, valorile parametrilor săi de ieșire din care o parte sunt transmiși lui M_j ca parametri de intrare.

Definiția 4-6.

Definim *momentul gata de execuție* al task-ului M_i (*ready time*) pentru ciclul k de execuție, notat cu $t_{rd,k}^{M_i}$, instanța de timp la care task-ul M_i îndeplinește condițiile de execuție și poate fi planificat pentru execuție în ciclul k , în cadrul sistemului TR.

□

Sistemele TR stricte impun termene limită pentru execuția task-urilor. În cazul task-urilor TR stricte, depășirea acestor termene implică funcționarea eronată a sistemului, uneori cu consecințe catastrofale pentru operatori și pentru mediul înconjurător.

Definiția 4-7.

Definim *termenul limită* al task-ului M_i (*deadline*) pentru ciclul k de execuție, notat cu $t_{dl,k}^{M_i}$, instanța de timp până la care execuția lui M_i are valoare în cadrul sistemului TR. □

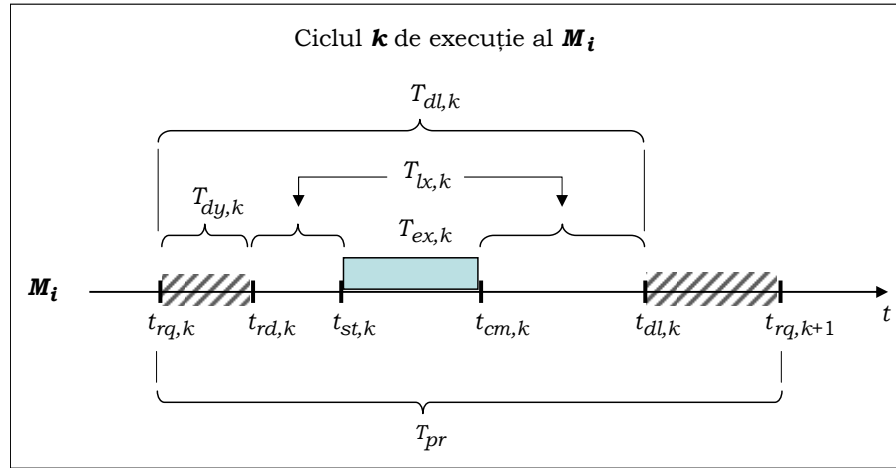


Figura 4-3. Modelul temporal al task-ului TR strict, M_i

Figura 4-3 și Tabelul 4-3 de mai jos sintetizează modelul și proprietățile temporale discutate până în acest punct, ce caracterizează task-ul TR (strict) M_i în cadrul unui ciclu de execuție, k . Intervalele de timp hașurate în figură semnifică faptul că execuția lui M_i nu este posibilă: pe de o parte, în intervalul $[t_{rq,k}^{M_i}, t_{rd,k}^{M_i}]$ task-ul M_i nu este gata pentru execuție (vezi Definiția 4-6), iar pe de altă parte, în intervalul $[t_{dl,k}^{M_i}, t_{rq,k+1}^{M_i}]$ M_i își depășește termenul de execuție, implicând operarea eronată a sistemului.

Între parametrii temporali ai modelului de task TR, prezentați în Tabelul 4-3, există următoarele relații:

$$T_{ex,k}^{M_i} = t_{cm,k}^{M_i} - t_{st,k}^{M_i} \quad (4-9)$$

$$T_{dl,k}^{M_i} = t_{dl,k}^{M_i} - t_{rq,k}^{M_i} \quad (4-10)$$

$$T_{dy,k}^{M_i} = t_{rd,k}^{M_i} - t_{rq,k}^{M_i} \quad (4-11)$$

$$T_{lx,k}^{M_i} = T_{dl,k}^{M_i} - T_{dy,k}^{M_i} - T_{ex,k}^{M_i} \quad (4-12)$$

și relația (4-6):

$$T_{pr}^{M_i} = \min_{1 \leq k \leq N^{M_i}} \left\{ \left(t_{rq,k+1}^{M_i} - t_{rq,k}^{M_i} \right) \right\}.$$

Tabelul 4-3. Parametrii temporali ai task-ului strict M_i

Parametru	Denumire	Semnificație
$t_{rq,k}^{M_i}$	<i>Momentul de invocare (Request time)</i>	Momentul în care M_i este invocat de STR. Definește începutul ciclului k de execuție a lui M_i .
$t_{rd,k}^{M_i}$	<i>Momentul gata de execuție (Ready time)</i>	Momentul de la care M_i poate fi planificat pentru execuție.
$t_{st,k}^{M_i}$	<i>Momentul de start (Start time)</i>	Momentul în care M_i își începe efectiv execuția.
$t_{cm,k}^{M_i}$	<i>Momentul de terminare (Completion time)</i>	Momentul în care M_i își termină execuția și este eliberat procesorul sistemului.
$t_{dl,k}^{M_i}$	<i>Termenul limită (Deadline)</i>	Momentul până la care execuția lui M_i are valoare în cadrul sistemului.
$T_{ex,k}^{M_i}$	<i>Durata de execuție (Execution interval)</i>	Intervalul de timp necesar execuției lui M_i în cadrul sistemului.
$T_{dl,k}^{M_i}$	<i>Intervalul limită (Deadline interval)</i>	Intervalul de timp în care execuția lui M_i poate fi planificată pentru o funcționare corectă a sistemului.
$T_{dy,k}^{M_i}$	<i>Intervalul de întârziere (Delay interval)</i>	Intervalul de timp după care execuția lui M_i poate fi planificată.
$T_{lx,k}^{M_i}$	<i>Intervalul disponibil planificării (Laxity interval)</i>	Intervalul de timp disponibil pentru planificarea lui M_i astfel încât execuția lui să nu depășească termenul limită.
$T_{pr}^{M_i}$	<i>Perioada (Period)</i>	Durata minimă a unui ciclu de execuție a lui M_i .
N^{M_i}	<i>Numărul de execuții (Execution count)</i>	Numărul total de execuții ale lui M_i (numărul de cicluri de execuție).

Pentru ca planificarea task-ului M_i să poată fi fezabilă, adică să existe posibilitatea planificării și execuției lui M_i în cadrul sistemului TR, parametrii săi temporali vor trebui să verifice relațiile:

$$t_{rq,k}^{M_i} \leq t_{rd,k}^{M_i} \leq t_{st,k}^{M_i} < t_{cm,k}^{M_i} \leq t_{dl,k}^{M_i} \leq t_{rq,k+1}^{M_i} \quad (4-13)$$

sau, echivalent:

$$0 < T_{ex,k}^{M_i} \leq T_{dl,k}^{M_i} \leq T_{pr}^{M_i} \quad (4-14)$$

$$0 \leq T_{dy,k}^{M_i} \leq T_{dl,k}^{M_i} - T_{ex,k}^{M_i} < T_{dl,k}^{M_i} \leq T_{pr}^{M_i} \quad (4-15)$$

pentru $\forall k \in \{1, \dots, N^{M_i}\}$.

Relațiile (4-9) până la (4-15) pot fi utilizate ca un prim pas în faza de validare a specificațiilor unui STR din punct de vedere al comportării în domeniul timp.

Urmând clasificarea semnalelor, detaliată în subcapitolul precedent, vom trata în continuare aspectele legate de comportamentul în timp al task-urilor unui sistem TR, vis a vis de cel al semnalelor, vizând în special cazul task-urilor TR stricte.

Semnalele de ieșire

Semnalele de ieșire ale unui STR sunt *generate* de către task-uri ale sistemului. Aici, relația de cauzalitate este de tipul "acțiune → eveniment". Comportamentul în domeniul timp al semnalelor de ieșire este complet determinat de proprietățile temporale ale task-urilor care le generează. Prin urmare, specificațiile de comportare temporală ale unui semnal de ieșire sunt direct determinate de cele ale task-ului care îl generează. De exemplu, un task M_i care este periodic, de perioadă $T_{pr}^{M_i}$, și care are un număr de N^{M_i} execuții, va genera un semnal de ieșire S_j periodic și temporar, de perioadă $T_{pr}^{S_j} = T_{pr}^{M_i}$ și cu un număr $N^{S_j} = N^{M_i}$ de apariții.

În faza de specificare/verificare a STR este necesară punerea în corespondență a parametrilor temporali ai semnalelor de ieșire cu parametrii temporali ai task-urilor care le generează.

Semnalele interne

Semnalele interne al unui STR sunt, la rândul lor, *generate* de către task-uri ale sistemului (fiind deci echivalente cu semnalele de ieșire), iar pentru alte task-uri, reprezintă semnale de intrare. Problema sincronizării și comunicației dintre task-urile STR va fi tratată separat, în paragrafele ce descriu modelul task-ului TR strict ("ModX").

Semnalele de intrare

Despre semnalele de intrare ale unui STR, spunem că sunt *tratate* de către task-uri ale sistemului. Relația de cauzalitate în acest caz este de tipul "eveniment → acțiune", comportamentul în timp al task-urilor ce tratează semnalele de intrare trebuind să corespundă specificațiilor temporale ale semnalelor.

Perspectiva *tratării* semnalelor de intrare de către task-uri ale STR impune adăugarea a încă unui parametru temporal important la modelul de semnal propus în subcapitolul anterior (vezi Tabelul 4-2). Este vorba despre așa-numitul *interval de răspuns* al sistemului (*response time*), $T_{rs}^{S_i}$, la un anumit semnal S_i . Echivalent, timpul de răspuns se referă la task-ul care tratează semnalul respectiv.

Definiția 4-8.

Definim *intervalul de răspuns* pentru semnalul S_i , notat cu $T_{rs}^{S_i}$, intervalul de timp inițiat de momentul apariției lui S_i și în decursul căruia task-ul corespunzător al sistemului trebuie să trateze pe S_i .

□

Tratarea semnalelor de intrare se poate realiza prin două tehnici de bază, diferite: în manieră asincronă, utilizând mecanismul de întreruperi al STR, sau în manieră sincronă, prin baleierea continuă, periodică a stării semnalului de intrare (tehnica denumită *polling*). Mecanismul de întreruperi și tratarea semnalelor de intrare în manieră asincronă presupune planificarea și execuția de task-uri aperiodice, invocate de întreruperi. Existența în cadrul STR a acestor task-uri afectează puternic predictibilitatea și funcționarea deterministă a sistemului (discuția din Capitolul 3).

Așadar, pentru cazul sistemelor TR stricte, este necesară eliminarea task-urilor aperiodice și utilizarea în exclusivitate a celor periodice. Această cerință este realizabilă din punct de vedere practic, din următoarele considerente:

- Chiar dacă o anumite aplicație TR utilizează task-uri aperiodice, există mecanisme demonstrate în literatura de specialitate, care permit transformarea lor în task-uri periodice [Mok 83, Mok 84].
- Task-urile periodice permit analiza operării sistemului TR într-o manieră predictibilă, deterministă și conceperea unor algoritmi de planificare fezabili, care să respecte specificațiile temporale de execuție ale task-urilor TR stricte.
- Utilizarea mecanismului de polling împreună cu mecanisme de stocare tampon (registre tampon, buffere, zone de memorie tampon) pentru tratarea semnalelor de intrare cu ajutorul task-urilor periodice este o soluție fezabilă.
- Două din dezavantajele majore ale acestui tip de abordare constau în scăderea drastică a eficienței sistemului și lipsa acută de flexibilitate. Mecanismele de întreruperi au fost concepute tocmai pentru tratarea cât mai eficient posibil a evenimentelor asincrone de către sistem. Problema constă însă în lipsa de predictibilitate a sistemului implicată de utilizarea întreruperilor [Stewart 01].

Teorema 4-1. Tratarea semnalelor de intrare cu task-uri periodice simple

Considerăm un task TR strict, periodic, simplu, M_i , caracterizat prin următorii parametri temporali:

- $T_{pr}^{M_i}$, perioada lui M_i ,
- $T_{ex}^{M_i}$, durata de execuție a lui M_i : $0 < T_{ex}^{M_i} \leq T_{pr}^{M_i}$,
- Intervalul limită al lui M_i este egal cu perioada acestuia: $T_{dl}^{M_i} = T_{pr}^{M_i}$,
- M_i este gata de execuție la începutul fiecărei perioade: $T_{dy}^{M_i} = 0$.

Dacă M_i tratează un semnal de intrare S_j într-un interval de răspuns $T_{rs}^{S_j}$, atunci perioada lui M_i trebuie să respecte relația¹:

$$T_{pr}^{M_i} \leq \left\lceil \frac{T_{rs}^{S_j} + 1}{2} \right\rceil \quad (4-16)$$

¹ Cu $\lfloor x \rfloor$ ("floor") s-a notat cel mai mare întreg mai mic sau egal cu x , iar cu $\lceil x \rceil$ ("ceiling"), cel mai mic întreg mai mare sau egal cu x .

Demonstrație

Vom demonstra teorema luând în considerare cazul cel mai defavorabil care poate să apară relativ la momentul începerii execuției lui M_i față de momentul apariției lui S_j : semnalul apare în instanța imediat următoare începerii execuției din ciclul curent a lui M_i , iar intervalul de timp dintre două execuții consecutive este maxim (vezi Figura 4-4).

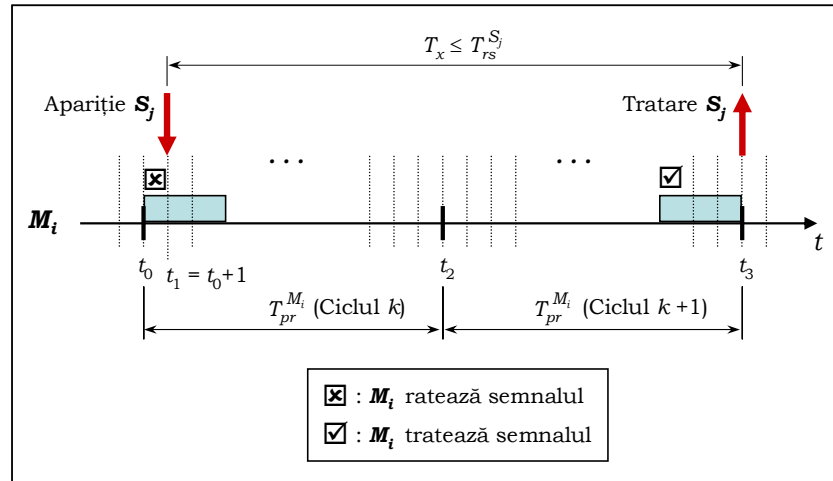


Figura 4-4. Cazul cel mai defavorabil tratării semnalului S_j cu un task M_i

Din figura de mai sus se observă că execuția k a lui M_i ratează apariția lui S_j , starea acestuia (informația furnizată de semnal) fiind reținută într-o structură tampon a sistemului, așa cum am descris mai sus, pentru tratarea în cadrul unei execuții ulterioare a lui M_i . În situația cea mai defavorabilă, execuția $(k + 1)$ a lui M_i are loc la sfârșitul ciclului de execuție următor (la sfârșitul perioadei următoare), pe când execuția curentă are loc la începutul perioadei.

Intervalul de timp scurs de la apariția semnalului S_j și până M_i finalizează tratarea acestuia are valoarea:

$$T_x = 2 \cdot T_{pr}^{M_i} - 1 \quad (4-17)$$

Dar, pentru ca S_j să fie tratat în intervalul de răspuns specificat, trebuie ca $T_x \leq T_{rs}^{S_j}$, de unde, prin înlocuirea lui T_x în (4-17), obținem relația (4-16), cerută a fi demonstrată în enunțul teoremei. □

Teorema 4-1 este utilă în fazele de specificare, verificare și analiză a STR stricte, pentru cazurile în care este necesară deducerea (automată) a comportamentului unor task-uri care tratează semnale de intrare, în condițiile în care se specifică doar parametrii temporali ai semnalelor. Din punctul de vedere al programatorului de aplicații, este mai utilă și mai intuitivă specificarea directă a

comportării temporale ale semnalelor de intrare în sistem decât deducerea manuală a caracteristicilor task-urilor TR stricte care vor trebui să trateze aceste semnale.

În continuare exemplificăm cu câteva cazuri concrete specificarea temporală a unui semnal de intrare și aplicarea Teoremei 4-1 pentru deducerea parametrilor task-ului TR strict care tratează semnalul.

Exemplul 4-2.

Considerăm un STR cu două semnale de intrare, S_1 și S_2 . Pentru tratarea semnalului S_1 cu task-ul M_1 se specifică un interval de răspuns $T_{rs}^{S_1} = 11$, iar pentru tratarea lui S_2 cu M_2 , intervalul de răspuns este $T_{rs}^{S_2} = 10$. Durata de execuție a ambelor task-uri o considerăm $T_{ex}^{M_1} = T_{ex}^{M_2} = 3$ unități de timp.

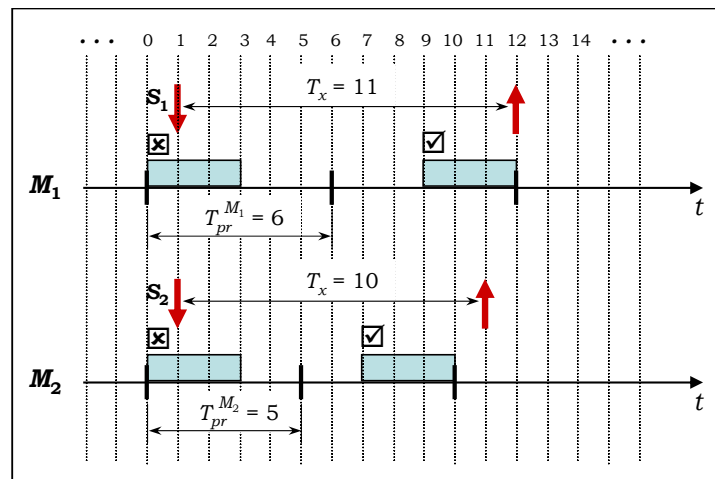


Figura 4-5. Task-urile M_1 și M_2 ce tratează semnalele S_1 , respectiv S_2

Utilizând Teorema 4-1 pentru determinarea parametrilor temporali ai lui M_1 și M_2 (a perioadelor acestora), rezultă (vezi Figura 4-5):

$$T_{pr}^{M_1} \leq \left\lfloor \frac{T_{rs}^{S_1} + 1}{2} \right\rfloor = \left\lfloor \frac{11 + 1}{2} \right\rfloor = 6,$$

pentru M_1 , și:

$$T_{pr}^{M_2} \leq \left\lfloor \frac{T_{rs}^{S_2} + 1}{2} \right\rfloor = \left\lfloor \frac{10 + 1}{2} \right\rfloor = 5.$$

□

Teorema 4-1 permite tratarea cu ajutorul task-urilor TR stricte periodice a semnalelor aperiodice de intrare pentru care se specifică intervalul de răspuns.

În cazurile în care, pentru un anumit semnal aperiodic de intrare S_j nu se specifică intervalul de răspuns $T_{rs}^{S_j}$, ci perioada $T_{pr}^{S_j}$ (intervalul de timp minim dintre oricare două apariții consecutive), se poate considera că timpul de răspuns al task-ului M_i care tratează pe S_j este determinat de această perioadă. Cu alte cuvinte, M_i trebuie să trateze pe S_j până când are loc o nouă apariție a acestuia:

$$T_{pr}^{M_i} \leq \left\lfloor \frac{T_{pr}^{S_j} + 1}{2} \right\rfloor. \quad (4-18)$$

Pentru exemplificare, pot fi considerate cazurile reprezentate în Figura 4-4 și Figura 4-5, cu $T_x = T_{pr}^{S_j}$.

Dacă sunt specificați ambii parametri temporali ai unui semnal aperiodic de intrare S_j , $T_{pr}^{S_j}$ și $T_{rs}^{S_j}$, trebuie întâi verificată în mod obligatoriu relația

$$T_{rs}^{S_j} \leq T_{pr}^{S_j} \quad (4-19)$$

care afirmă că, practic, intervalul alocat tratării lui S_j de către STR nu poate depăși ca durată intervalul celei mai apropiate apariții consecutive a lui S_j . Dacă inegalitatea este confirmată, în formula de determinare a perioadei task-ului M_i rămâne parametrul $T_{rs}^{S_j}$ ca în (4-16).

Formula (4-18) poate fi aplicată și pentru tratarea semnalelor periodice de intrare, fiind o procedură sigură, care evită "scăparea" vreunei apariții a semnalului: în cadrul unei perioade a lui S_j , $T_{pr}^{S_j}$, task-ul TR strict M_i este planificat și executat de două ori. Dacă într-o perioadă a semnalului, acesta apare înaintea execuției lui M_i , S_j este tratat în cel de-al doilea ciclu de execuție, înainte unei noi apariții. Problema acestei abordări însă constă în scăderea drastică a eficienței sistemului – prețul plătit de sistemele TR stricte în schimbul garantării îndeplinirii tuturor specificațiilor temporale ale aplicației, în orice condiții de operare.

Exemplul 4-3.

Considerăm un STR cu un semnal periodic de intrare, S_1 , cu perioada $T_{pr}^{S_1} = 7$, de tratarea căruia este responsabil task-ul TR strict și periodic, M_1 . Durata de execuție a lui M_1 o considerăm constantă pentru toate ciclurile de execuție: $T_{ex}^{M_1} = 2$.

$$\text{Din Teorema 4-1 rezultă o perioadă } T_{pr}^{M_1} \leq \left\lfloor \frac{T_{pr}^{S_1} + 1}{2} \right\rfloor = \left\lfloor \frac{7+1}{2} \right\rfloor = 4$$

pentru M_1 . O configurație posibilă de operare a sistemului este reprezentată grafic în Figura 4-6, iar o scurtă evaluare a eficienței execuției lui M_1 în cadrul sistemului este redată în Tabelul 4-4.

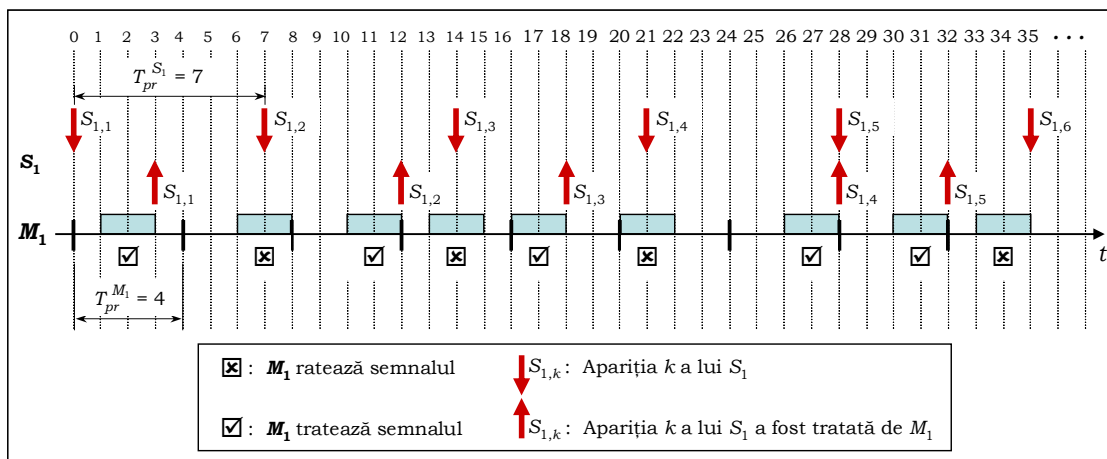


Figura 4-6. Exemplificare a tratării unui semnal de intrare periodic

Tabelul 4-4. Eficiența task-ului TR strict M_1

Total apariții S_1	6
Total execuții M_1	9
Execuții M_1 care tratează S_1	5
Execuții M_1 care ratează S_1	4
Eficiență execuție M_1	55 %

□

Pentru creșterea eficienței tratării semnalelor de intrare periodice, se pot utiliza task-uri TR periodice, cu aceeași perioadă ca a semnalului de intrare, cu condiția sincronizării apariției semnalului cu execuția task-ului. Problema sincronizării dintre un task periodic și un semnal de intrare de aceeași perioadă, este relativ dificil de rezolvat, cu atât mai mult cu cât în foarte multe aplicații practice, sistemul TR și dispozitivul extern care generează semnalul utilizează oscilatoare distincte. Astfel, oricât de bine ar fi specificată perioada pentru semnalul de intrare și pentru task-ul TR strict care îl tratează, după un anumit interval de timp de operare, apare un defazaj sensibil între apariția semnalului și execuția task-ului, rezultând în final la pierderea unei perioade.

O variantă de rezolvare a problemei sincronizării este ca sistemul să se comporte ca un "master" pentru dispozitivul extern care generează semnalul de intrare periodic: fiecare apariție a semnalului să fie declanșată de fapt tot de STR (de un task al sistemului), sub forma activării periodice a unei linii de semnal sau a unei comenzi către dispozitivul extern, iar timpul de răspuns al echipamentului să fie cunoscut.

Observație

Tratarea semnalelor de intrare periodice și aperiodice mai poate fi rezolvată în cadrul STR și prin implementarea unui mecanism de tipul *server periodic de întreruperi* (*sporadic server, aperiodic server, deferrable server*) [Lehoczky 87, Sprunt 89a, Sprunt 89b], care, în principiu, constă dintr-un task periodic ce se

ocupă în cadrul fiecărei execuții de arbitrarea și tratarea, una câte una, a întreruperilor ce apar în sistem.

Această abordare are, la rândul ei, două inconveniente majore:

- durata de execuție a serverului de întreruperi va trebui să fie cel puțin egală cu cea mai mare durată de execuție corespunzătoare tratării oricărui semnal aperiodic din cele gestionate de server,
- perioada serverului va trebui să fie suficient de mică pentru a putea satisface specificațiile temporale ale tuturor semnalelor gestionate, din punct de vedere al intervalului de răspuns și/sau al perioadei acestora.

Cu cât sunt mai multe semnale de intrare (evenimente asincrone) tratate cu mecanismul serverului de întreruperi și cu cât acestea au specificații temporale mai diferite între ele, cu atât mai dificilă este planificarea și execuția periodică a serverului.

4.4 Modelul task-ului TR strict: ModX-ul

Elementele legate de gestionarea timpului, a semnalelor și a task-urilor într-un STR strict discutate până acum în cadrul capitolului de față, cât și problemele curente de proiectare și implementare a STR stricte prezentate în Capitolul 3, au ca rezultat propunerea unui model de task TR strict – *ModX-ul*.

Un *ModX* (*Modul executabil*) reprezintă un task TR periodic, modular, cu specificații complete și stricte de comportare în timp și cu planificare și execuție în context non-preemptiv, destinat a servi ca element de bază în specificarea, proiectarea și dezvoltarea sistemelor și aplicațiilor TR stricte, pentru care determinismul și predictibilitatea operării sunt cerințe esențiale.

Din perspectiva arhitecturii aplicațiilor TR stricte, cât și din punct de vedere funcțional, ModX-ul este un modul software, adică echivalentul unui "bloc de bază" (*basic block*) utilizat în teoria analizei programelor și a compilării [Acho 87, Muchnick 97]. ModX-ul prezintă un set de intrări (parametri de intrare), un corp (bloc) de instrucțiuni care prelucrează intrările, variabilele definite local și variabilele globale, și un set de ieșiri (parametri de ieșire). Din punct de vedere al specificațiilor și al comportamentului în domeniul timp, parametrii de intrare/ieșire ai ModX-ului sunt semnale interne ale sistemului. În plus, un ModX poate interacționa cu semnale de intrare (ModX-ul *tratează* semnale de intrare) și de ieșire (ModX-ul *generează* semnale de ieșire).

Formal, ModX-ul este o componentă a unei aplicații TR reprezentate sub forma unui graf de program, direcționat, aciclic [Ferrante 87, Podgurski 89, Marlowe 90, Niehaus 91, Gupta 96]:

$$\mathbf{G} \equiv \langle \mathbf{M}, \mathbf{V} \rangle \quad (4-20)$$

unde $\mathbf{M} = \{M_i \mid 1 \leq i \leq |\mathbf{M}|\}$ este mulțimea nodurilor grafului \mathbf{G} (ModX-urile aplicației) și $\mathbf{V} = \{(M_i, M_j) \mid M_i, M_j \in \mathbf{M}; 1 \leq i, j \leq |\mathbf{M}|\}$ este mulțimea arcelor lui \mathbf{G} , constând din perechi ordonate de noduri.

Definiția 4-9.

Un ModX M_i este un task TR strict, reprezentat formal prin următorul model:

$$M_i \equiv \langle T, P, S, F \rangle \quad (4-21)$$

unde:

$T = \{T_{pr}^{M_i}, T_{ex}^{M_i}, T_{dl}^{M_i}, T_{dy}^{M_i}, N^{M_i}\}$ este setul specificațiilor de comportare temporală a ModX-ului M_i ,

$P = \{P_{IN}, P_{OUT}, P_{GLB}\}$ este setul parametrilor de intrare, de ieșire, respectiv al variabilelor globale cu care operează M_i ,

$S = \{S_{IN}, S_{OUT}\}$ este setul semnalelor de intrare tratate de M_i și al semnalelor de ieșire generate de M_i , iar

F este setul de instrucțiuni care procesează informația în cadrul M_i , reprezentând specificarea funcțională a ModX-ului. □

În continuare vom aborda pe rând fiecare componentă a structurii modelului de task TR strict propus – ModX-ul.

Task-urile periodice, non-preemptive și predictibilitatea STR stricte

Pentru a se putea obține un sistem TR care să opereze în condiții maxime de predictibilitate și determinism, și ale cărui task-uri să poată fi executate cu respectarea în orice condiții a termenelor impuse, este necesară eliminarea elementelor asincrone, sporadice din sistem. Principalul mecanism care generează o comportare impredictibilă, asincronă pentru sistemele digitale îl reprezintă întreruperile [Stewart 01].

Ca rezultat, ModX-ul a fost conceput ca un task periodic, planificat și executat în cadrul STR în regim non-preemptiv. Aceste două caracteristici principale ale ModX-ului facilitează specificarea și verificarea comportamentului temporal al aplicației, cât și analiza fezabilității planificării task-urilor înaintea încărcării și execuției lor efective în sisteme TR a căror fiabilitate de operare este critică. Un alt avantaj constă în eliminarea problemelor de tip inversare a priorității, tipice sistemelor TR care implementează tehnici de planificare și execuție preemptive, bazate pe priorități.

Abordarea modulară

Pentru concepția modelului de task TR strict a fost aleasă o abordare modulară din mai multe considerente:

- Creșterea flexibilității în procesul de dezvoltare a aplicațiilor TR. Dezvoltarea modulară și vizuală a aplicațiilor, utilizând reprezentări de tip graf, ușurează munca de proiectare, specificare, validare și analiză a acestora.
- Evidențierea rapidă a dependențelor de program (control și date) ale task-urilor aplicației.

- Rezolvarea simplă a comunicației și sincronizării între task-urile aplicației. Există două mecanisme puse la dispoziția programatorului de aplicații: transmiterea de parametri de intrare/ieșire și utilizarea (limitată) a variabilelor globale.
- Ținând cont de faptul că task-urile aplicației sunt periodice, este mai ușoară și mai intuitivă vizualizarea execuției acestora ca module în cadrul aplicației.

Aspectul funcțional și evaluarea duratei de execuție

Din punct de vedere funcțional, ModX-urile reprezintă secvențe de cod ce se execută fără blocare și în regim non-preemptiv. Prin urmare, operațiile implementate de un ModX sunt *operații atomice*, eliminându-se astfel necesitatea mecanismelor de sincronizare și control al acceselor concurente la resurse.

O problemă extrem de importantă pentru studiul STR stricte este determinarea duratei de execuție a task-urilor. Parametrul temporal $T_{ex,k}^{M_i}$ specifică durata de execuție a task-ului M_i , care variază de la un ciclu k de execuție la altul (vezi Definiția 4-2 și Tabelul 4-3). Pentru a se putea dezvolta algoritmi fezabili de planificare, este necesară considerarea pentru fiecare task a unei durate de execuție constante, $T_{ex}^{M_i}$, care să nu varieze de la un ciclu de execuție la altul. Pe de altă parte, planificarea task-urilor unui STR strict trebuie să fie fezabilă pentru cele mai dezavantajoase situații de operare ale sistemului și nu doar pentru cazurile de încărcare medie, de exemplu. Din aceste considerente, durata de execuție a unui task TR strict, pentru fiecare ciclu de execuție, este considerată ca fiind egală cu *WCET* (*Worst Case Execution Time*), adică durata de execuție cea mai defavorabilă [Puschner 89, Puschner 91, Park 91, Burns 91, Chapman 94, Parashkevov 94, Puschner 00]. Durata *WCET* pentru un task oarecare se determină ca fiind suma timpilor de execuție ai instrucțiunilor ce compun calea cea mai lungă de execuție a task-ului și considerând condițiile cele mai defavorabile de operare.

WCET nu poate fi determinat fără o serie de restricții impuse limbajului de programare cu ajutorul căruia se specifică comportamentul funcțional al task-urilor unui STR [Shaw 89, Stoyenko 91, Chung 95, Gerber 97]. Din aceste motive, o serie de limbaje de programare au fost modificate (adăugându-se restricții, etc.) și au fost concepute și dezvoltate noi limbaje de programare, vizând facilitarea analizei programelor TR și a determinării *WCET*: Real-Time EUCLID [Klingerman 86], CRL [Stoyenko 95], etc.

Programarea aplicațiilor TR stricte cu ajutorul ModX-urilor este supusă, la rândul ei, următoarelor restricții principale:

- Utilizarea *recursivității* – este interzisă, din cauza cerințelor de spațiu impredictibile pe care le produce (dimensiunea memoriei necesare).
- *Buclele de program* trebuie să fie limitate ca durată totală și ca spațiu. Cu alte cuvinte, trebuie ca numărul total de iterații ale fiecărei bucle să fie bine determinat. De asemenea, buclele încuibate (*nested loops*) se vor supune acelorași restricții, inclusiv unei limitări a nivelurilor de încuibare, pentru o analiză mai ușoară a programului.

- Utilizarea *instrucțiunilor de ramificare* de tip GOTO sunt permise doar în condițiile în care au ca efect salturi de tip "înainte" ale execuției. În acest fel se păstrează reductibilitatea grafului dependențelor de control ale programului [Acho 86, Muchnick 97] și se permite reprezentarea programului sub formă de graf aciclic. Salturile de tip "înapoi" sunt permise doar în structuri de tip buclă de program (de exemplu: FOR, WHILE, REPEAT).
- Utilizarea *apelurilor de proceduri și funcții* în cadrul ModX-urilor este permisă dar într-o manieră limitată și liniară. Trebuie subliniat că, prin arhitectura modulară de programare propusă prin intermediul ModX-urilor, funcțiile și procedurile pot fi implementate ca ModX-uri individuale. Acest lucru este avantajos și din perspectiva planificării, având în vedere că ModX-urile se execută în context non-preemptiv. Cu cât un ModX particular durează mai mult ca execuție, cu atât va fi mai dificil de planificat în cadrul sistemului TR.
- *Alocarea dinamică a resurselor* (de exemplu, a memoriei) – este interzisă, pentru că mecanismele curente de alocare dinamică prezintă un comportament impredictibil în timp. Pentru a facilita analiza "offline" a sistemului, mecanismele utilizate trebuie să fie statice și liniare.

Limbajele pe care le avem în vedere pentru programarea ModX-urilor trebuie să se apropie cât mai mult de limbajul de asamblare al procesorului țintă și să respecte restricțiile enumerate mai sus. Astfel, într-o primă fază se utilizează direct limbajul de asamblare [Micea 04b], și intenționăm adaptarea unei versiuni de "ANSI C" pentru programarea aplicațiilor TR. Compilarea/asamblarea fiecărui ModX în parte se face cu opțiuni de generare de cod relocabil (bibliotecă de module).

Pentru fiecare ModX M_i din cadrul unei aplicații TR, se va considera o durată de execuție constantă, $T_{ex}^{M_i}$, egală cu WCET al codului ModX-ului M_i (incluzând apelurile interne la proceduri și funcții).

Parametrii de intrare/ieșire ai ModX-ului și comunicația inter-task

Comunicația între ModX-urile aplicației TR se realizează prin intermediul setului de parametri de intrare, P_{IN} , și al celor de ieșire, P_{OUT} , ale fiecărui ModX. Un ModX M_i nu poate începe ciclul curent de execuție până când toți parametrii săi de intrare nu au fost actualizați de ModX-urile care au aceiași parametri ca parametri de ieșire.

P_{GLB} reprezintă setul acelor variabile globale ale aplicației care sunt accesate de către un ModX. Variabilele globale reprezintă un al doilea mecanism de sincronizare și comunicare inter-task pus la dispoziția programatorului de aplicație. Datorită faptului că ModX-urile se execută în context non-preemptiv și, astfel, implementează operații atomice, nu sunt necesare mecanisme suplimentare de control și sincronizare a acceselor concurente la variabilele globale ale aplicației. Se recomandă, totuși, utilizarea cu atenție a variabilelor globale pentru că ele practic ascund dependența normală de date dintre task-urile aplicației, cu implicații în construirea unui graf al dependențelor de program realist. O situație în care se recomandă utilizarea variabilelor globale este aceea în care aplicația folosește structuri de date de dimensiuni mari (cum sunt de exemplu, diverse buffere de semnale sau cadre de

imagini). Prin această tehnică, se evită copierea repetată a structurilor de mari dimensiuni de la o execuție la alta a ModX-urilor, sub formă de parametri de intrare/ieșire.

Observație

Un alt aspect important ce rezultă din faptul că ModX-urile sunt task-uri periodice este posibilitatea transmiterii unei anumite stări curente a unui ModX, de la o execuție a sa, la alta. Această operație este posibilă prin stabilirea unui parametru (sau a mai multora), atât ca parametru de intrare, cât și de ieșire. În acest context însă, este necesară implementarea unui task de inițializare a aplicației (incluzând setarea valorilor inițiale pentru astfel de parametri), care să se execute primul, înaintea celorlaltor ModX-uri. Acesta are un regim special de execuție, nefiind periodic ci executându-se o singură dată.

Tratarea și generarea semnalelor

ModX-urile interacționează cu semnalele de intrare/ieșire ale STR în maniera descrisă în secțiunea precedentă. Din punct de vedere al programatorului de aplicații, setul de semnale $\mathbf{S} = \{\mathbf{S}_{IN}, \mathbf{S}_{OUT}\}$ cu care interacționează un ModX M_i este perceput la nivelul specificării comportamentului temporal al semnalelor. Informația pe care o poartă un semnal oarecare este manipulată de STR prin intermediul variabilelor locale, globale și al parametrilor de intrare/ieșire a ModX-urilor. Astfel, pentru modelul propus, un semnal S_j este reprezentat de parametrii săi temporali:

$$S_j \equiv \{T_{pr}^{S_j}, T_{rs}^{S_j}, T_{act}^{S_j}, N^{S_j}\}, \text{ cu } (S_j \in \mathbf{S}_{IN}) \wedge (S_j \in \mathbf{S}_{OUT}) \quad (4-22)$$

unde: $T_{pr}^{S_j}$ este perioada semnalului S_j (dacă e periodic) sau durata minimă de timp ce separă oricare două apariții consecutive ale acestuia, $T_{act}^{S_j}$ este durata intervalului în care au loc apariții ale semnalului S_j , și N^{S_j} este numărul de apariții ale lui S_j (vezi Tabelul 4-2), iar, dacă S_j este semnal de intrare, $T_{rs}^{S_j}$ reprezintă intervalul de timp inițiat de momentul apariției lui S_j și în decursul căruia M_i trebuie să-l trateze (vezi Definiția 4-8).

Specificarea comportamentului temporal al ModX-urilor

Parametrii temporali ai ModX-ului M_i aparțin mulțimii $\mathbf{T} = \{T_{pr}^{M_i}, T_{ex}^{M_i}, T_{dl}^{M_i}, T_{dy}^{M_i}, N^{M_i}\}$, unde:

- $T_{pr}^{M_i}$ este perioada lui M_i – caracteristică esențială și obligatorie pentru toate ModX-urile, fiind task-uri periodice. Perioada este constantă de-a lungul operării aplicației, deci nu depinde de ciclul curent k de execuție a lui M_i ;

- $T_{ex}^{M_i}$ este durata de execuție a lui M_i . Este de asemenea constantă pe durata operării sistemului TR, fiind egală cu $WCET$ al codului ModX-ului;
- $T_{dl}^{M_i}$ este intervalul limită pentru planificarea și execuția lui M_i în cadrul perioadei acestuia. $T_{dl}^{M_i}$ este un parametru opțional, introdus pentru cazurile în care este necesar ca execuția lui M_i să se finalizeze cu un interval de timp oarecare înainte de terminarea perioadei, pentru toate ciclurile de execuție;
- $T_{dy}^{M_i}$ este intervalul de întârziere a planificării și execuției lui M_i în cadrul perioadei. $T_{dy}^{M_i}$ este, la rândul lui un parametru opțional, ce poate fi utilizat în cazurile în care se dorește ca planificarea și execuția lui M_i să nu poată fi efectuate într-un anumit interval de la începutul fiecărei perioade a lui M_i .
- N^{M_i} este numărul total de execuții ale lui M_i . Este un parametru opțional, care tratează trei cazuri:
 - M_i are un număr nedefinit de execuții (execuție continuă): $N^{M_i} = \infty$,
 - M_i are un număr finit, bine precizat de execuții: $N^{M_i} < \infty$. În acest caz, M_i interacționează cu semnale (generează sau tratează semnale) cu apariție temporară,
 - M_i nu mai este executat de către sistem: $N^{M_i} = 0$. M_i este planificat în continuare, dar nu va mai fi lansat în execuție de către executivul/dispatcher-ul sistemului.

Observație

În cazul în care N^{M_i} este finit, valoarea momentană a acestuia poate fi controlată din două perspective:

i.) În timpul execuției lui M_i , executivul/dispatcher-ul sistemului de operare TR decrementează de fiecare dată pe N^{M_i} cu o unitate, până acesta ajunge la 0. În acest mod, sistemul contorizează execuțiile lui M_i pe parcursul operării. Când N^{M_i} ajunge la 0, executivul nu-l va mai lansa în execuție pe M_i .

ii.) În timpul execuției unui alt ModX, M_j , acesta poate accesa contorul de execuții al lui M_i , reactualizându-l la orice valoare posibilă, conform necesităților aplicației. Prin acest mecanism, se poate relansa în execuție un ModX, chiar dacă acesta are contorul setat în mod curent pe 0. Metoda presupune acordarea unei atenții sporite fluxului de control al programului, care poate fi modificat într-un mod incontrolabil.

Definiția 4-10.

Numim *ModX fantomă*, un ModX M_i , care la un moment dat are contorul de execuții setat pe 0 ($N^{M_i} = 0$). Așa cum s-a văzut mai sus, ModX-urile fantomă sunt planificate în cadrul sistemului TR, dar nu sunt lansate în execuție.



Figura 4-7 exemplifică operarea, pentru două perioade consecutive, a unui ModX oarecare M_i ce are specificați toți parametrii săi temporali (*ModX generic*).

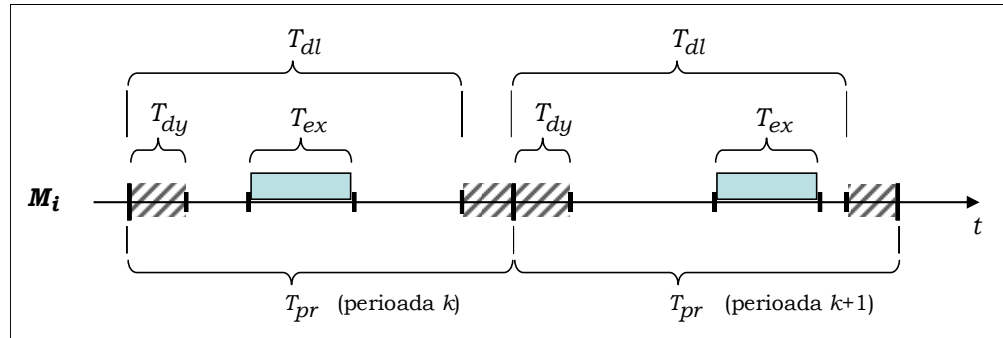


Figura 4-7. ModX-ul generic M_i și parametrii săi temporali

Specificarea temporală minimală a unui ModX M_i , constă din cei doi parametri de bază: perioada $T_{pr}^{M_i}$ și durata de execuție $T_{ex}^{M_i}$. Pe baza acestora, algoritmi de planificare pot analiza fezabilitatea setului de ModX-uri al sistemului TR strict și, în caz că există o variantă de planificare, o pot obține. Setarea celorlalți parametri temporali se va face implicit, în faza de validare a aplicației, după cum urmează:

$$\begin{cases} T_{dl}^{M_i} = T_{pr}^{M_i} \\ T_{dy}^{M_i} = 0 \\ N^{M_i} = \infty \end{cases} \quad (4-23)$$

Relația (4-23) specifică pentru M_i , un interval limită egal cu perioada, un interval de întârziere nul și un număr nedefinit de execuții (operare continuă). Figura 4-8 exemplifică operarea pentru două perioade consecutive a unui ModX M_i ce are specificați doar cei doi parametri temporali de bază (*ModX simplu*).

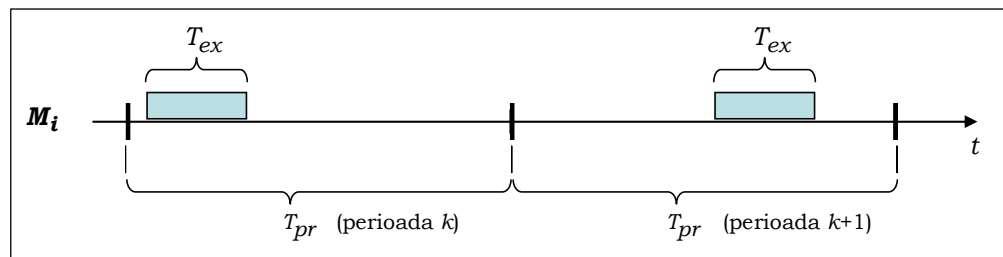


Figura 4-8. ModX-ul simplu M_i și parametrii săi temporali

Verificarea și analiza aplicației TR din punct de vedere al comportamentului temporal se bazează pe următoarele relații stabilite între parametrii temporali ai ModX-urilor și ai semnalelor utilizate în aplicație:

$$0 < T_{ex}^{M_i} \leq T_{dl}^{M_i} \leq T_{pr}^{M_i} \quad (4-24)$$

$$0 \leq T_{dy}^{M_i} \leq T_{dl}^{M_i} - T_{ex}^{M_i} < T_{dl}^{M_i} \leq T_{pr}^{M_i} \quad (4-25)$$

$$T_{pr}^{M_i} \leq \left\lceil \frac{\min(T_{rs}^{S_j}, T_{pr}^{S_j}) + 1}{2} \right\rceil \quad (4-26)$$

pentru $\forall M_i \in \mathbf{M}$ (vezi (4-20)) și $S_j \in \mathbf{S}$ (vezi (4-21)).

Formulele (4-24), (4-25) și (4-26) sunt variante ale relațiilor (4-14), (4-15), respectiv (4-16) și (4-18), adaptate definiției ModX-urilor ca task-uri TR stricte, periodice și cu parametrii temporali (mulțimea \mathbf{T} din (4-21)) constanți pentru toate ciclurile de execuție.

5 Planificarea aplicațiilor TR stricte

5.1 Introducere

Una dintre problemele de interes major în dezvoltarea și analiza sistemelor și aplicațiilor TR o reprezintă planificarea task-urilor. În literatura curentă există studii teoretice și simulări efectuate asupra unei multitudini de algoritmi de planificare pentru STR [Mercer 92, Panzieri 93a, Panzieri 93b, Ghosh 94, Ramamritham 94, George 96, Letia 00, Radulescu 02] (vezi Capitolul 2).

O metodă de clasificare a algoritmilor de planificare este în funcție de admiterea sau nu a întreruperilor în timpul execuției task-urilor sistemului. Rezultă cele două categorii principale de algoritmi de planificare: algoritmi *preemptivi*, care acceptă întreruperea unui task aflat în execuție de către alte task-uri, cu prioritate mai mare de exemplu, și algoritmi *non-preemptivi*, care nu acceptă întreruperi în timpul execuției unui task. Algoritmii de planificare preemptivă au fost modelați și tratați în numeroase lucrări, din care menționăm pe [Liu 73] ca fiind de referință, în timp ce algoritmi non-preemptivi nu s-au bucurat de aceeași atenție, în special din cauză că generează o mulțime de probleme pentru care s-a demonstrat că necesită resurse (timp) de rezolvare de factură strict nepolinomială (*NP-hard*) [Cheng 88, Panzieri 93b]. Cu toate acestea, algoritmi de planificare non-preemptivă sunt importanți dintr-o varietate de motive [Jeffay 91]:

- Pentru multe probleme practice de planificare a task-urilor TR, cum ar fi de exemplu, planificarea task-urilor ce interacționează cu semnale de intrare/ieșire ale sistemului, proprietățile hardware și software ale cuplului STR – dispozitiv periferic nu permit lucrul cu întreruperile sau acesta devine mult prea costisitor;
- Algoritmii de planificare non-preemptivi sunt mai ușor de implementat decât cei preemptivi și, în cele mai multe cazuri, au ca efect o scădere drastică a resurselor suplimentare necesare planificării din timpul operării sistemului (*run-time scheduling*);
- Comportamentul și resursele de timp și de calcul al 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 (*run-time scheduling overhead*) sunt ignorate, rezultând că implementarea unui algoritm non-preemptiv este mai apropiată de modelul formal studiat decât cazul unui algoritm preemptiv;
- Așa cum s-a văzut și în Capitolul 4, task-urile non-preemptive, planificate pe un STR mono-procesor, garantează accesul exclusiv la resursele partajate ale sistemului, eliminând astfel necesitatea și costurile implementării mecanismelor de sincronizare.

În completarea celor de mai sus, așa cum remarcă [Gai 02], planificarea non-preemptivă este o metodă naturală și directă, în special în cazul arhitecturilor de sistem și a aplicațiilor TR de achiziție și prelucrare numerică a datelor în ritm continuu, neîntrerupt (de exemplu, în aplicațiile TR ce au la bază procesoare numerice de semnal, DSP).

Există însă și o serie de dezavantaje ale algoritmilor de planificare non-preemptivă. Un prim neajuns a fost menționat mai sus – generarea de probleme computaționale de ordin strict nepolinomial. Alte dezavantaje includ:

- Necesitatea găsirii unei partajări optime a secvenței de cod a programului în task-uri (problema *granularității* task-urilor), în așa fel încât planificarea non-preemptivă a acestora să poată fi efectuată în condiții optime de eficiență. Dacă se lucrează cu o granularitate prea mică (task-uri de dimensiuni mari), planificarea non-preemptivă devine extrem de dificilă. Dacă, dimpotrivă, se lucrează cu o granularitate prea mare, se complică foarte mult mecanismele de transmitere a parametrilor și de salvare/restaurare a contextului între task-urile aplicației;
- Lipsa de flexibilitate a abordării non-preemptive. 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. Din aceste motive, vor trebui prevăzute și tratate apriori, încă din faza de specificare și proiectare, toate posibilitățile și configurațiile de operare ale sistemului;
- Creșterea ineficienței sistemului, comparativ cu cazurile de planificare preemptivă. Mecanismul întreruperilor a fost conceput tocmai pentru a permite operarea cât mai eficientă a sistemelor digitale, mai ales a celor care interacționează cu semnale (evenimente) asincrone;
- Dificultatea găsirii unor condiții de testare rapidă a planificabilității și de găsire a planificării optime a unui set de task-uri. Pentru multe modele ale aplicațiilor, testarea planificabilității setului aferent de task-uri și găsirea unui algoritm optim de planificare reprezintă probleme computaționale de ordin strict nepolinomial.

Avantajul major al algoritmilor de planificare non-preemptivă constă însă în faptul că permit operarea STR (mai ales a sistemelor stricte) în condiții maxime de predictibilitate și determinism, cu garantarea respectării termenelor de timp impuse task-urilor TR stricte, spre deosebire de algoritmi de planificare ce permit existența întreruperilor, a evenimentelor și task-urilor asincrone, sporadice.

Exemplul 5-1.

Considerăm o stație terestră de emisie-recepție care comunică permanent cu un satelit nestaționar aflat pe o orbită în jurul Pământului. Două dintre funcțiunile principale executate de sistemul digital de control și procesare al stației sunt comunicația de date cu satelitul și comanda dispozitivelor de orientare a antenei parabolice în așa fel încât să fie permanent și perfect aliniată cu satelitul.

Ambele funcții se desfășoară în regim permanent, dar, dacă partea de comunicație poate fi modelată în mod direct ca un task periodic (set de task-uri

periodice), funcția de orientare a antenei poate fi modelată ca un task asincron: necesitatea reorientării antenei fiind semnalată de scăderea sub un anumit prag a intensității semnalului recepționat de la satelit. Evident, sistemul de control și procesare al stației este un sistem TR strict.

Dacă sistemul implementează un algoritm de planificare preemptiv, pot apărea situații în care, prin întreruperea execuției unuia din cele două task-uri de către celălalt, să rezulte pierderea orientării antenei sau a unor pachete de date vehiculate pe legătura de comunicație dintre sistemul terestru și satelit.

Pe de altă parte, abordarea non-preemptivă a problemei presupune o analiză minuțioasă, prealabilă, a planificării și execuției celor două task-uri: cel de orientare și cel de comunicație. Task-ul ce tratează comunicația poate fi modelat ca un task periodic. Task-ul ce tratează orientarea antenei poate fi modelat, de asemenea, ca un task periodic, ținând cont de intervalul de timp minim în care poate apărea necesitatea orientării antenei (parametru ce depinde de viteza satelitului, distanța între satelit și antenă, etc.). Analiza planificabilității non-preemptive a setului compus din cele două task-uri se bazează pe perioadele și pe duratele lor de execuție, verificându-se respectarea termenelor și evitarea suprapunerii execuțiilor în timpul operării.

În cazul în care analiza de fezabilitate a planificării non-preemptive a task-urilor sistemului dă rezultate pozitive, există garanția operării predictibile și cu respectarea strictă a termenelor specificate.

□

În capitolul de față vom aborda problema planificării non-preemptive a task-urilor TR stricte în *sisteme TR monoprosesor*, tratând diverse modele posibile, de la cele mai simple, până la modele mai complexe și mai apropiate de realitate. Vom pleca de la următoarele caracteristici inițiale, comune tuturor cazurilor pe care le vom analiza în continuare:

- i.) Se va studia problema planificării non-preemptive monoprosesor a task-urilor TR stricte, periodice;
- ii.) Modelul task-urilor TR stricte se bazează pe caracteristicile generale ale ModX-urilor (vezi Capitolul 4): task-uri periodice, cu durată de execuție constantă de la un ciclu de execuție la altul;
- iii.) Înainte de execuția efectivă a sistemului de task-uri în cadrul STR, are loc o analiză a fezabilității planificării acestora. Numim această operație *analiză offline*;
- iv.) STR operează într-un mediu determinist și toate specificațiile temporale necesare sunt cunoscute;
- v.) Modelul temporal utilizat este cel descris în Capitolul 4: un domeniu temporal discret, limitat la stânga, nelimitat la dreapta și cu metrică temporală;
- vi.) Toți algoritmi de planificare studiați vor implementa o *planificare fără inserție de timp liber*, adică nu vor introduce în planificare intervale de timp liber în situațiile în care există task-uri invocate de către sistem.

5.2 Planificarea non-preemptivă a ModX-urilor independente

Majoritatea abordărilor din literatură, legate de planificarea non-preemptivă, se concentrează pe modelul cel mai simplu și mai apropiat de ideal, în ceea ce privește setul de task-uri analizat: modelul task-urilor TR stricte, independente [Kim 80, Jeffay 91, Howell 95, George 96, Kang 98].

5.2.1 Modelul setului de task-uri independente

Modelul setului de task-uri cu ajutorul căruia studiem planificarea non-preemptivă a task-urilor independente este definit după cum urmează.

Definiția 5-1.

Definim setul M de ModX-uri simple, independente, ca fiind mulțimea

$$M = \{M_1, M_2, \dots, M_n\}$$

unde, fiecare element M_i al lui M ($1 \leq i \leq n$) este un ModX, cu următoarele caracteristici:

- M_i este un ModX simplu (un task TR strict, periodic, planificat și executat în context non-preemptiv).
- Din punct de vedere al comportamentului în timp, M_i este caracterizat prin următorii parametri temporali:

$$T = \{T_{pr}^{M_i}, T_{ex}^{M_i}, T_{dl}^{M_i}, T_{dy}^{M_i}, N^{M_i}\} \quad (5-1)$$

- $T_{pr}^{M_i}$ este perioada lui M_i , $T_{ex}^{M_i}$ este durata de execuție, $T_{dl}^{M_i}$ este intervalul limită, $T_{dy}^{M_i}$ este intervalul de întârziere și N^{M_i} este numărul total de execuții.
- Între parametrii lui M_i se stabilesc următoarele relații:

$$0 < T_{ex}^{M_i} \leq T_{pr}^{M_i} \quad (5-2)$$

$$T_{dl}^{M_i} = T_{pr}^{M_i} \quad (5-3)$$

$$T_{dy}^{M_i} = 0 \quad (5-4)$$

$$N^{M_i} = \infty \quad (5-5)$$

- Momentul primei invocări a fiecărui ModX M_i este:

$$t_{rq,1}^{M_i} = 0 \quad (5-6)$$

- Execuția lui M_i nu este condiționată de execuțiile altor ModX-uri, adică nu există dependențe de control sau de date între oricare două ModX-uri din M .

□

Setul T al parametrilor temporali din (5-1) specifică un ModX generic, pentru care, în mod obligatoriu, durata de execuție este nenulă și mai mică sau egală cu perioada. Relațiile (5-2) până la (5-4) identifică un ModX simplu (vezi Capitolul 4): intervalul limită (*deadline*) este egal cu perioada, intervalul de întârziere este nul, respectiv, numărul de cicluri de execuție este nelimitat.

Definiția 5-2.

Spunem că setul de ModX-uri M este *fezabil din punct de vedere al planificării*, dacă există un algoritm capabil să planifice toate ModX-urile din M în așa fel încât fiecare să-și respecte specificațiile de comportare temporală pe toată durata operării sistemului.

□

Există două clase de algoritmi de planificare ce pot fi aplicați pentru seturi de ModX-uri independente:

- *Algoritmi de planificare statică*: asignează câte o prioritate fixă de planificare pentru fiecare ModX după un anumit criteriu. La o instanță de timp oarecare, t_x (pentru care procesorul este liber și se poate planifica un ModX), va fi selectat ModX-ul cu prioritatea cea mai mare și care nu a mai fost executat în perioada corespunzătoare lui t_x . De exemplu, un criteriu pentru stabilirea priorităților în cadrul unui set de ModX-uri este asignarea unei priorități invers proporționale cu lungimea perioadei fiecăruia. Cea mai cunoscută tehnică de planificare statică este *RMS (Rate Monotonic Scheduling)* [Liu 73, Kim 80, George 96].
- *Algoritmi de planificare dinamică*: selectarea ModX-ului care va fi planificat la un moment dat t_x , se face în mod dinamic după un criteriu ce include, printre altele, valoarea lui t_x și cerința ca ModX-ul să nu fi fost executat în perioada corespunzătoare lui t_x . De exemplu, un criteriu utilizat în planificarea dinamică este selectarea ModX-ului a cărui perioadă se termină cel mai aproape de momentul t_x (sau, cu alte cuvinte, ModX-ul al cărui *deadline* este cel mai apropiat de t_x). Algoritmul de planificare ce utilizează acest criteriu este *EDF (Earliest Deadline First)* [Liu 73, Kim 80, Jeffay 91, Howell 95, George 96, Kang 98].

În continuare vom analiza doi dintre algoritmii de planificare dinamică, considerați a fi cei mai eficienți pentru tratarea execuțiilor non-preemptive [Kim 80, Jeffay 91, Howell 95, George 96]. Este vorba de algoritmul *MLFNP (Minimum Laxity First Non-Preemptive)*, sau *LLFNP – Least Laxity First Non-Preemptive* și *EDFNP (Earliest Deadline First Non-Preemptive)*.

Lema 5-1.

Considerăm un set M de ModX-uri cu caracteristicile date de Definiția 5-1 și notăm cu T_{LCM} , intervalul de timp egal cu cel mai mic multiplu comun al perioadelor ModX-urilor din M :

$$T_{LCM} = \min \left\{ C \mid T_{pr}^{M_i} / C, \forall M_i \in M \right\} \quad (5-7)$$

unde, cu x/y s-a notat faptul că x divide pe y .

Dacă un algoritm este capabil a planifica ModX-urile din M în intervalul T_{LCM} , atunci setul M este fezabil din punctul de vedere al planificării cu acest algoritm.

Demonstrație

Lema susține că dacă un algoritm este capabil să planifice cu succes ModX-urile din M pe un interval de timp egal cu cel mai mic multiplu comun al perioadelor ModX-urilor, atunci planificarea setului M va reuși pe toată durata operării sistemului.

Setul M conține ModX-uri simple, cu caracteristicile date de Definiția 5-1. Astfel, pentru toate ModX-urile, prima invocare are loc la momentul 0 (5-6), iar comportamentul lor în timp este complet determinat de către perioada și durata fiecăruia de execuție: $T_{pr}^{M_i}$, respectiv $T_{ex}^{M_i}$, pentru $\forall M_i \in M$.

Din (5-6) rezultă că perioadele tuturor ModX-urilor din M sunt aliniate la momentul 0, și, mai mult, ele sunt de asemenea aliniate la fiecare moment de timp care este un multiplu comun al acestor perioade. Cu alte cuvinte, putem spune că, din punct de vedere al perioadelor, setul M are o configurație ciclică, cu perioada de bază T_{LCM} .

Pe de altă parte, algoritmi de planificare vizează ca nici un ModX din M să nu-și depășească termenele specificate, deci ca fiecare ModX să fie executat o dată (și numai odată), pentru fiecare perioadă a sa, de-a lungul operării sistemului.

Rezultă din cele de mai sus că se poate stabili o comportare ciclică și pentru algoritmi de planificare, pe baza intervalului T_{LCM} .

□

Lema 5-1 permite efectuarea analizei offline a fezabilității planificării unui set de ModX-uri independente doar pentru un interval de timp de lungime T_{LCM} . Astfel, dacă un anumit algoritm de planificare dă rezultate pozitive la planificarea unui set de ModX-uri pe intervalul $[0, T_{LCM})$, atunci setul respectiv este fezabil.

Indiferent de criteriul de selecție al ModX-ului care va fi planificat la un moment de timp dat, t , algoritmi de planificare dinamică, non-preemptivă, a ModX-urilor independente operează după organigrama generală reprezentată în Figura 5-1.

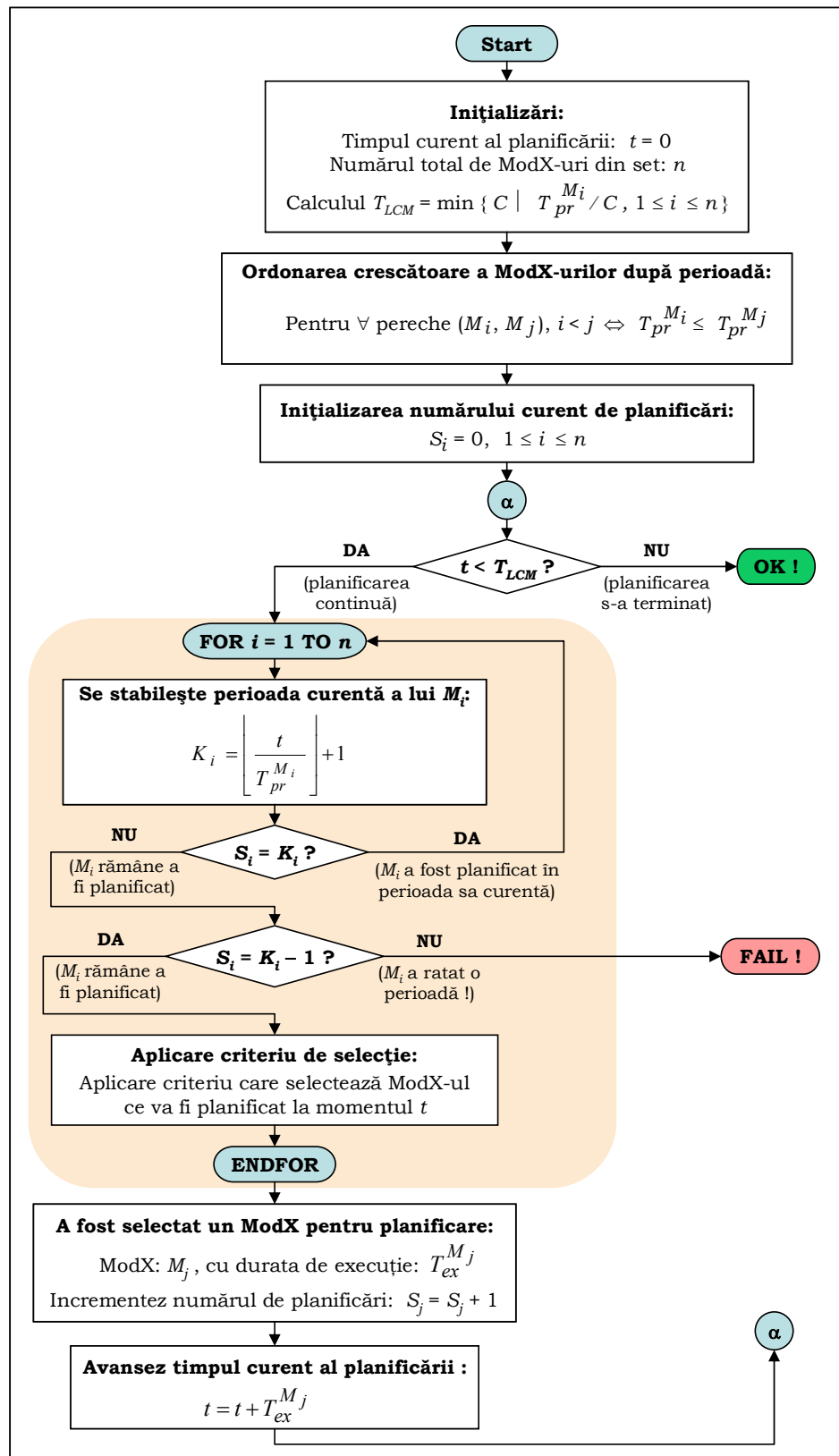


Figura 5-1. Organigrama generală a algoritmilor de planificare non-preemptivă a unui set de ModX-uri independente

5.2.2 Algoritm de planificare MLFNP

Pentru selecția ModX-ului ce va fi planificat la momentul t , algoritmul de planificare *MLFNP* (*Minimum Laxity First Non-Preemptive*) utilizează un criteriu bazat pe intervalul disponibil planificării (*Laxity interval*, vezi Tabelul 4-3).

În cazul ModX-urilor simple, intervalul disponibil planificării, notat cu $T_{lx}^{M_i}$ (pentru ModX-ul M_i) este (vezi Figura 5-2):

$$T_{lx}^{M_i} = T_{pr}^{M_i} - T_{ex}^{M_i}, \forall M_i \in \mathbf{M} \quad (5-8)$$

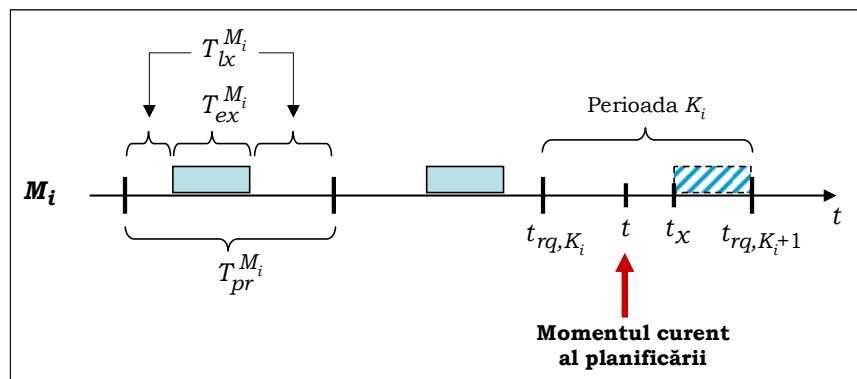


Figura 5-2. Intervalul disponibil planificării în cazul ModX-urilor simple

Din (5-8) rezultă că $T_{lx}^{M_i}$ este un parametru constant, ce nu depinde de numărul ciclului curent de execuție (planificare), K_i , a lui M_i , ci doar de perioada și de durata sa de execuție.

Criteriul de selecție al ModX-ului ce va fi planificat la un moment t se bazează pe ideea de interval de timp pe care îl mai are la dispoziție algoritmul, din momentul t , până la depășirea termenelor fiecărui ModX. Cu cât acest interval este mai mare pentru un ModX M_i , cu atât prioritatea (sau "urgența") planificării lui M_i este mai mică în comparație cu a altor ModX-uri, și reciproc.

Pentru exemplul prezentat în Figura 5-2, la momentul curent al planificării, t , ModX-ul M_i se găsește în al K_i -lea ciclu de execuție, delimitat de momentele t_{rq,K_i} și t_{rq,K_i+1} , astfel încât:

$$\left[t_{rq,K_i}, t_{rq,K_i+1} \right] = T_{pr}^{M_i} = t_{rq,K_i+1} - t_{rq,K_i}.$$

Determinarea lui K_i se face pentru fiecare ModX M_i din setul \mathbf{M} , în cadrul buclei de selecție a ModX-ului ce va fi planificat, și pentru fiecare moment t al planificării (vezi Figura 5-1):

$$K_i = \left\lceil \frac{t}{T_{pr}^{M_i}} \right\rceil + 1 \quad (5-9)$$

Pentru planificarea corectă a lui M_i în ciclul K_i , algoritmul mai are la dispoziție intervalul de timp $[t, t_x]$ (vezi Figura 5-2). Dacă notăm lungimea acestui interval cu $L_i(t)$, avem că:

$$L_i(t) = t_x - t = K_i \cdot T_{pr}^{M_i} - T_{ex}^{M_i} - t \quad (5-10)$$

și din (5-9), rezultă:

$$L_i(t) = T_{pr}^{M_i} \left(\left\lfloor \frac{t}{T_{pr}^{M_i}} \right\rfloor + 1 \right) - T_{ex}^{M_i} - t \quad (5-11)$$

Evident, $L_i(t)$ e o funcție de timp (și anume, de momentul curent al planificării) și depinde de parametrii temporali ai lui M_i . Rezultă că la un moment dat, algoritmul *MLFNP* va calcula maxim n astfel de funcții, câte una pentru fiecare ModX din \mathbf{M} (unde $n = |\mathbf{M}|$). În finalul buclei de selecție, algoritmul va planifica ModX-ul al cărui interval disponibil pentru planificare este cel mai mic:

$$M_i - \text{planificat la momentul } t \Leftrightarrow L_i(t) = \min \{L_j(t) \mid 1 \leq j \leq n\} \quad (5-12)$$

Formula (5-12) exprimă formal criteriul utilizat de algoritmul *MLFNP* pentru selecția ModX-ului ce va fi planificat la momentul curent t . După ce ModX-ul M_i a fost selectat și planificat, algoritmul incrementează variabila corespunzătoare numărului de planificări a lui M_i (vezi organigrama din Figura 5-1) și avansează momentul următoarei planificări cu durata de execuție a lui M_i :

$$\begin{cases} S_i = S_i + 1 \\ t = t + T_{ex}^{M_i} \end{cases}$$

după care reia procedura, de la noul moment t . Algoritmul se termină în momentul în care planificarea s-a desfășurat cu succes pe intervalul $[0, T_{LCM}]$.

În continuare vom exemplifica operarea algoritmului *MLFNP* pentru câteva cazuri concrete de seturi de ModX-uri.

Exemplul 5-2.

Considerăm un set $\mathbf{M} = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 2 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 9 \\ T_{ex}^{M_2} = 4 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 18 \\ T_{ex}^{M_3} = 3 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 24 \\ T_{ex}^{M_4} = 3 \end{cases}$$

Se dorește analiza offline a planificabilității setului \mathbf{M} , utilizând algoritmul *MLFNP*. Figura 5-3 ilustrează rezultatul planificării, pentru intervalul de timp $[0, T_{LCM}]$, unde $T_{LCM} = 72$.

Planificarea începe de la momentul $t = 0$, după inițializarea parametrului S_i (numărul total de planificări pentru ModX-ul M_i , până la momentul curent):

$$S_1 = S_2 = S_3 = S_4 = 0$$

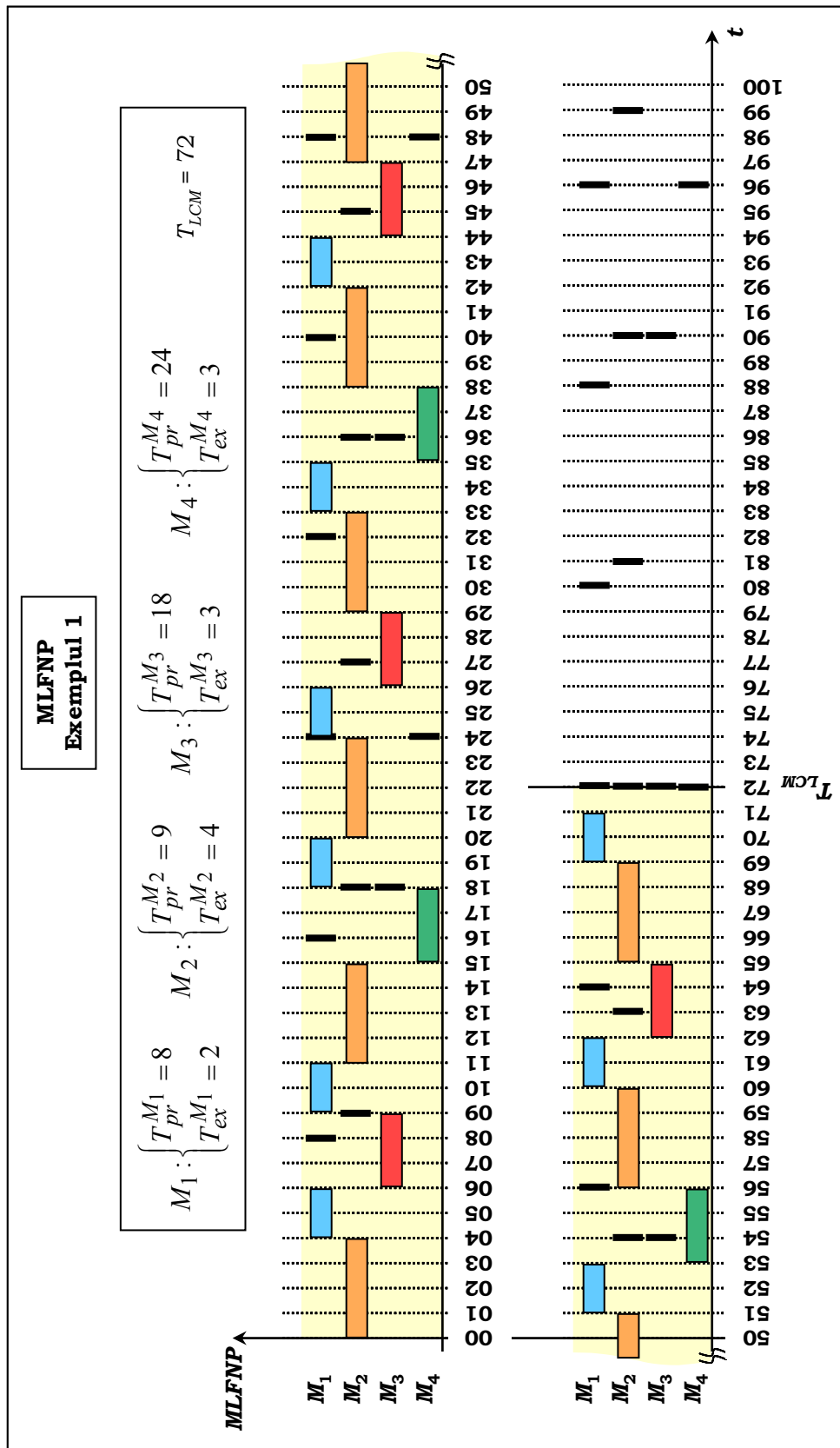


Figura 5-3. Exemplu de planificare *MLFNP* a unui set de ModX-uri

Primul pas îl reprezintă stabilirea pentru fiecare ModX a numărului ciclului de execuție K_i , corespunzător momentului curent t , aplicând (5-9):

$$K_1 = K_2 = K_3 = K_4 = 1$$

În continuare, se calculează intervalul de timp disponibil planificării fiecărui ModX, $L_i(t)$, conform relației (5-11):

$$\begin{cases} L_1(0) = 8 \cdot \left(\left\lfloor \frac{0}{8} \right\rfloor + 1 \right) - 2 - 0 = 6 \\ L_2(0) = 9 \cdot \left(\left\lfloor \frac{0}{9} \right\rfloor + 1 \right) - 4 - 0 = 5 \\ L_3(0) = 18 \cdot \left(\left\lfloor \frac{0}{18} \right\rfloor + 1 \right) - 3 - 0 = 15 \\ L_4(0) = 24 \cdot \left(\left\lfloor \frac{0}{24} \right\rfloor + 1 \right) - 3 - 0 = 21 \end{cases}$$

de unde rezultă că $L_i(0)$ este minim pentru $i = 2$, deci M_2 va fi planificat la momentul $t = 0$. Ca urmare, se incrementează S_2 ($S_2 = 1$) și se avansează timpul planificării:

$$t = t + T_{ex}^{M_2} = 4.$$

Procedura de planificare se reia de la $t = 4$, până când se ajunge la momentul $t = T_{LCM} = 72$. Planificarea setului M din acest exemplu dă rezultate pozitive, așa cum se poate vedea și în Figura 5-3. □

Exemplul 5-3.

Considerăm un set $M = \{M_1, M_2, M_3\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 3 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 10 \\ T_{ex}^{M_2} = 6 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 40 \\ T_{ex}^{M_3} = 1 \end{cases}$$

Se dorește analiza offline a planificabilității setului M , utilizând algoritmul *MLFNP*. Figura 5-4 ilustrează rezultatul planificării, pentru intervalul de timp $[0, T_{LCM})$, unde $T_{LCM} = 40$.

Algoritmul începe planificarea la $t = 0$, cu:

$$\begin{cases} L_1(0) = 8 - 3 = 5 \\ L_2(0) = 10 - 6 = 4 \\ L_3(0) = 40 - 1 = 39 \end{cases}$$

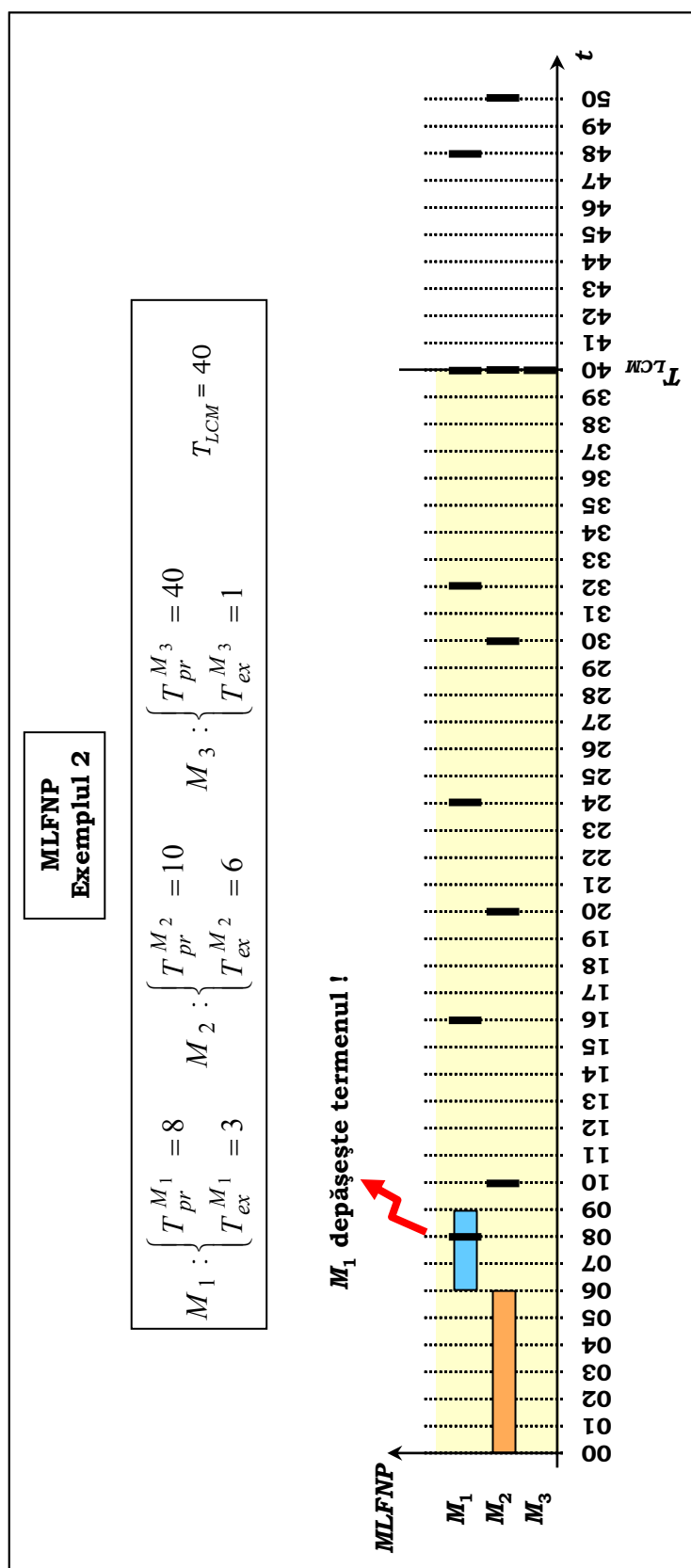


Figura 5-4. Exemplu de planificare MLFNP a unui set de ModX-uri

Așa cum rezultă din Figura 5-4, setul M de ModX-uri nu poate fi planificat utilizând algoritmul *MLFNP* de planificare. □

Setul de ModX-uri din Exemplul 5-3 nu poate fi planificat utilizând algoritmul *MLFNP* (ceea ce nu înseamnă că nu poate fi planificat cu ajutorul unui alt algoritm – așa cum se va vedea în secțiunea următoare). Vom spune că setul de ModX-uri din Exemplul 5-3 *nu este fezabil din punct de vedere al planificării* cu algoritmul *MLFNP*.

5.2.3 Algoritmul de planificare *EDFNP*

Planificarea *EDFNP* (*Earliest Deadline First Non-Preemptive*) utilizează un criteriu mai simplu pentru selectarea task-ului ce urmează a fi planificat la momentul t : se va selecta de fiecare dată task-ul cu termenul limită cel mai apropiat de t .

[Jeffay 91] demonstrează faptul că *EDFNP* este un algoritm optim de planificare, în sensul că dacă un set de ModX-uri este fezabil, atunci el poate fi planificat cu *EDFNP*. Exemplele următoare ilustrează acest lucru și, în plus, arată că *EDFNP* este mai eficient ca *MLFNP* din cel puțin două motive:

- există seturi de ModX-uri care nu pot fi planificate cu *MLFNP*, dar *EDFNP* reușește să le planifice,
- algoritmul *EDFNP* este mai simplu și necesită resurse (de calcul) mai mici ca *MLFNP*.

Se vor utiliza aceleași seturi de ModX-uri ca în Exemplul 5-2 și 5-3, pentru a ilustra inclusiv diferențele de abordare a planificării (prin criteriile de selecție a ModX-ului care urmează a fi planificat la momentul curent).

Exemplul 5-4.

Considerăm un același set $M = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, ca în Exemplul 5-2:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 2 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 9 \\ T_{ex}^{M_2} = 4 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 18 \\ T_{ex}^{M_3} = 3 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 24 \\ T_{ex}^{M_4} = 3 \end{cases}$$

Figura 5-5 ilustrează analiza offline a planificabilității setului M , utilizând algoritmul *EDFNP*, pentru intervalul de timp $[0, T_{LCM})$, unde $T_{LCM} = 72$.

La momentul $t = 0$, M_1 e ModX-ul cu termenul limită cel mai apropiat de t (intervalul minim de la t la momentul de invocare al perioadei următoare):

$t_{rq,2}^{M_1} - t = 8$; $t_{rq,2}^{M_2} - t = 9$; $t_{rq,2}^{M_3} - t = 18$; $t_{rq,2}^{M_4} - t = 24$. După planificarea

lui M_1 la momentul $t = 0$, se avansează timpul de planificare la $t = t + T_{ex}^{M_1} = 2$ și se reia procedura de selecție a ModX-urilor care nu au fost încă planificate. □

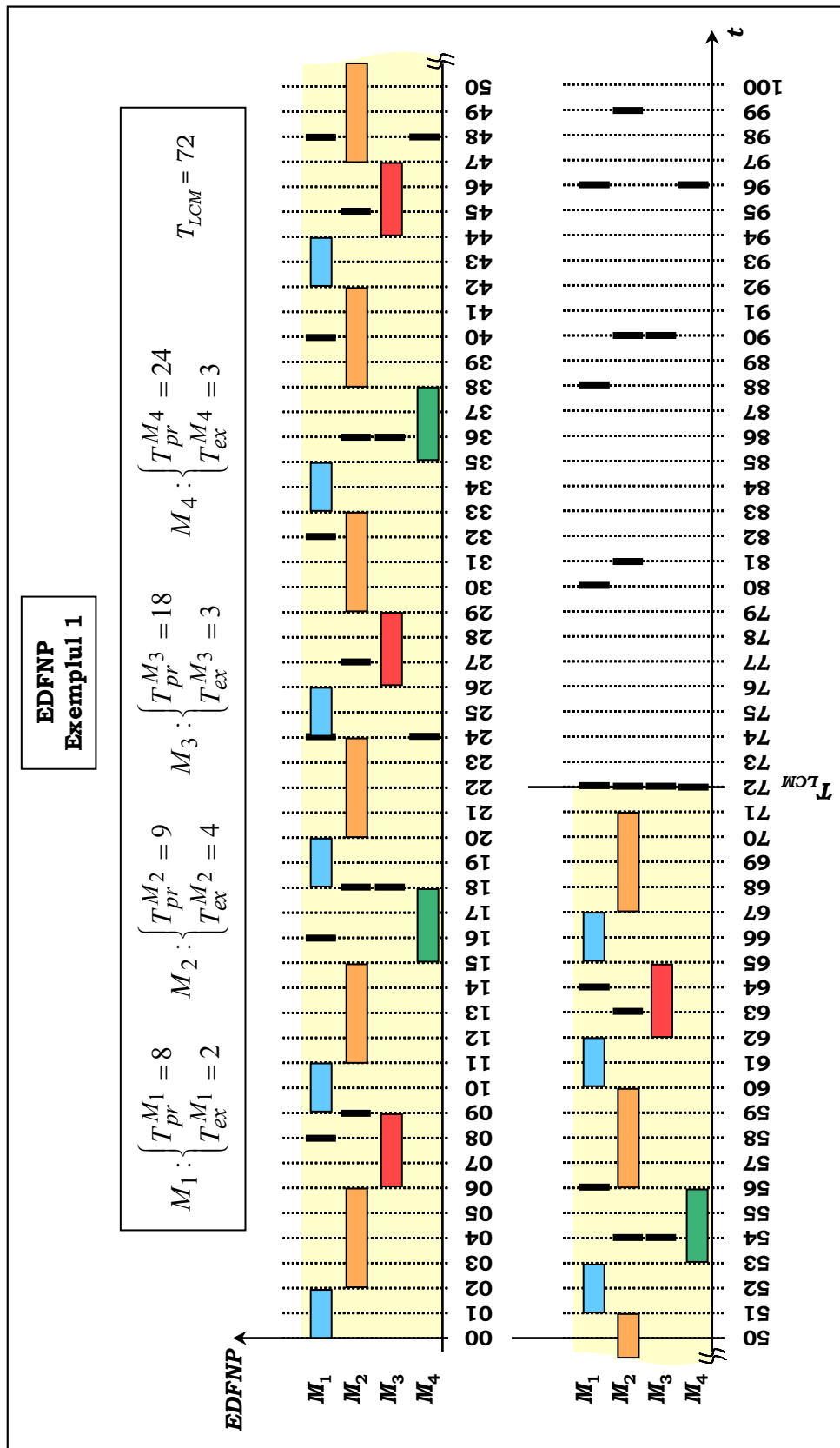


Figura 5-5. Exemplu de planificare EDFNP a unui set de ModX-uri

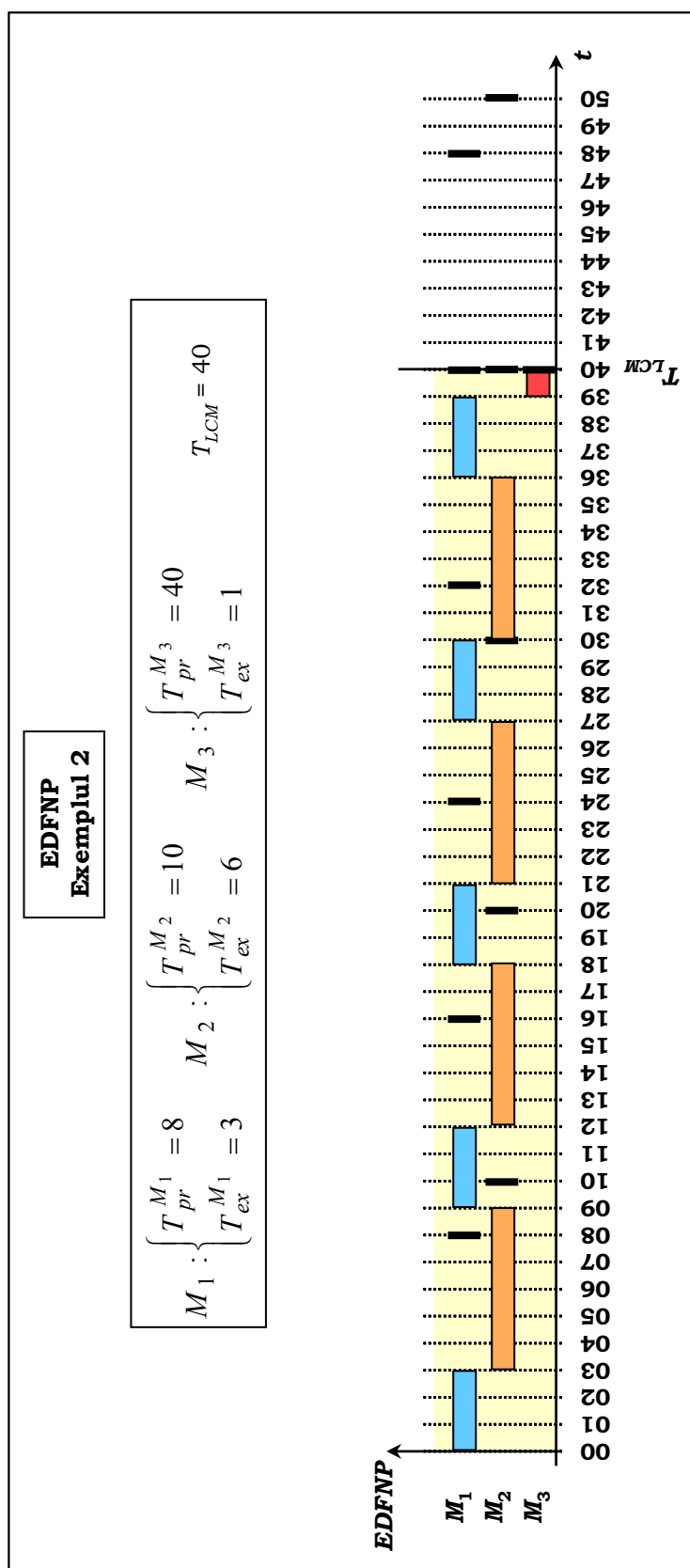


Figura 5-6. Exemplu de planificare EDFNP a unui set de ModX-uri

Exemplul 5-5.

Considerăm un același set $M = \{M_1, M_2, M_3\}$ de ModX-uri simple, independente, ca în Exemplul 5-3:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 3 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 10 \\ T_{ex}^{M_2} = 6 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 40 \\ T_{ex}^{M_3} = 1 \end{cases}$$

Așa cum rezultă și din Figura 5-6, analiza offline a planificabilității setului M utilizând algoritmul *EDFNP* pentru intervalul $[0, T_{LCM})$, unde $T_{LCM} = 40$, dă rezultate pozitive, spre deosebire de cele obținute cu *MLFNP*. □

5.2.4 Condițiile de planificabilitate

În cadrul analizei offline a unei aplicații TR, determinarea fezabilității setului de task-uri din punct de vedere al planificării cu un anumit algoritm se poate realiza, fie aplicând direct algoritmul, fie evaluând mai întâi dacă setul de task-uri îndeplinește o serie de condiții – *condițiile de planificabilitate*. Evaluarea condițiilor de planificabilitate înaintea aplicării algoritmului propriu-zis prezintă avantajul unei decizii rapide, obținute cu eforturi relativ mici (resurse de calcul, timp, etc.).

Condițiile de planificabilitate reprezintă seturi de relații definite între parametrii temporali ai task-urilor din setul analizat, și pot fi de două tipuri:

- *Condiții de necesitate*: dacă setul de task-uri nu le îndeplinește, atunci în mod sigur acesta nu este fezabil pentru planificare;
- *Condiții de suficiență*: dacă setul de task-uri le îndeplinește, atunci în mod sigur acesta este fezabil și va exista un algoritm care îl poate planifica cu succes.

Pentru cazul seturilor de task-uri TR stricte, ce vor fi planificate în context non-preemptiv, condițiile de suficiență sunt foarte dificil de determinat – dacă ele există [Kim 80, Jeffay 91]. Pot fi stabilite însă o serie de condiții de necesitate, care să fie utile în identificarea rapidă a unui set de task-uri ce nu este fezabil. O primă condiție de necesitate se referă la încărcarea procesorului [Liu 73, Kim 80, Jeffay 91, Howell 95, George 96, Kang 98], și o a doua a fost enunțată și demonstrată în [Jeffay 91]. Vom prezenta în continuare cele două condiții de necesitate a planificabilității pentru cazul general al seturilor de task-uri ce vor fi planificate în context non-preemptiv și le vom analiza sub aspectul valabilității și aplicării lor în cazul modelului propus în lucrare.

Modelul de task-uri cu ajutorul căruia [Jeffay 91] studiază planificarea non-preemptivă se aseamănă cu modelul ModX-urilor independente propus în secțiunea de față: amândouă iau în considerare task-uri periodice, caracterizate din punct de vedere temporal prin doi parametri – perioada și durata de execuție. Diferența constă din faptul că în [Jeffay 91] se tratează seturile de task-uri la modul general, făcându-se distincție dintre un "set de task-uri" și un "set concret de task-uri". Dacă

$$M = \{M_1, M_2, \dots, M_n\}$$

este un set de task-uri, atunci un set concret de task-uri este mulțimea de perechi:

$$W = \{(M_1, R_1), (M_2, R_2), \dots, (M_n, R_n)\}$$

unde $R_i = t_{rq,1}^{M_i}$ reprezintă momentul primei invocări a task-ului M_i (*release time*).

Un set de task-uri M poate genera o infinitate de seturi concrete de task-uri W . Cu aceste premise, condițiile de necesitate pentru planificarea non-preemptivă sunt enunțate în [Jeffay 91] după cum urmează:

"Fie $M = \{M_1, M_2, \dots, M_n\}$ un set de task-uri periodice sortate în ordine crescătoare a perioadelor (adică, pentru orice pereche de task-uri (M_i, M_j) , dacă $i < j$, atunci $T_{pr}^{M_i} \leq T_{pr}^{M_j}$).

Dacă M este planificabil, atunci:

$$\text{CN1) } \sum_{i=1}^n \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} \leq 1 \quad (5-13)$$

$$\text{CN2) } \forall i, 1 < i \leq n; \forall L, T_{pr}^{M_1} < L < T_{pr}^{M_i} :$$

$$L \geq T_{ex}^{M_i} + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{T_{pr}^{M_j}} \right\rfloor \cdot T_{ex}^{M_j} \quad (5-14)$$

unde fiecare task $M_i \equiv (T_{ex}^{M_i}, T_{pr}^{M_i})$ este caracterizat prin parametrii săi temporali – durata de execuție și perioada."

CN1 este o condiție de bază a oricărei planificări de task-uri pe sisteme cu un singur procesor și exprimă cerința ca suma fracțiunilor de timp procesor consumate de fiecare task din setul M să nu depășească unitatea, sau, cu alte cuvinte, *utilizarea procesor (processor utilization)* să fie mai mică sau egală cu 1. **CN1** poate fi aplicată și modelului de set de task-uri bazat pe ModX-uri, propus în secțiunea de față. Cu ajutorul acestei condiții, se pot elimina rapid acele seturi de ModX-uri pentru care utilizarea procesor e supraunitară, evitându-se astfel aplicarea algoritmului de planificare și câștigându-se timp în faza de analiză offline.

Seturile de ModX-uri din Exemplul 5-2 și 5-4, respectiv din Exemplul 5-3 și 5-5 îndeplinesc **CN1**, având utilizarea procesor egală cu 0.986, respectiv 1.

Condiția **CN2** reflectă restricția impusă de planificarea non-preemptivă și fără inserarea de timp liber pentru seturile de task-uri. Astfel, dacă un set M de task-uri este fezabil, atunci orice set concret de task-uri W generat din M poate fi planificat cu un anume algoritm non-preemptiv, și în consecință, îndeplinește relația (5-14).

Există însă cazuri în care seturi concrete de task-uri nu respectă CN2 și totuși pot fi planificate. Modelul setului de ModX-uri independente propus în lucrare este echivalentul unui set concret de task-uri – anume setul pentru care momentele

primelor invocări sunt aliniate la $t = 0$: $W_0 = \{(M_1, 0), (M_2, 0), \dots, (M_n, 0)\}$. Pentru acest model, **CN2** enunțată în [Jeffay 91] nu este aplicabilă, așa cum o arată și exemplul următor.

Exemplul 5-6.

Considerăm un set $M = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 10 \\ T_{ex}^{M_1} = 4 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 15 \\ T_{ex}^{M_2} = 8 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 90 \\ T_{ex}^{M_3} = 4 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 90 \\ T_{ex}^{M_4} = 1 \end{cases}$$

În Figura 5-7 este reprezentată analiza offline a planificabilității setului M utilizând algoritmul *EDFNP* pentru intervalul $[0, T_{LCM})$, unde $T_{LCM} = 90$, cu rezultate pozitive. În consecință setul M este fezabil.

Din punct de vedere al condițiilor de planificabilitate, setul M se conformează relației (5-13) care interzice ca utilizarea procesor (egală cu 0.9888) să fie supraunitară, deci setul M se conformează condiției de necesitate **CN1**. În schimb, relația (5-14) nu este îndeplinită, existând un ModX (M_2) și un interval temporal ($L = 11$) pentru care (5-14) e falsă:

$$\begin{aligned} L &\geq T_{ex}^{M_i} + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{T_{pr}^{M_j}} \right\rfloor \cdot T_{ex}^{M_j}, \text{ cu } i = 2, L = 11 \\ \Rightarrow 11 &\geq T_{ex}^{M_2} + \sum_{j=1}^{2-1} \left\lfloor \frac{L-1}{T_{pr}^{M_j}} \right\rfloor \cdot T_{ex}^{M_j} \\ \Rightarrow 11 &\geq 8 + \left\lfloor \frac{11-1}{10} \right\rfloor \cdot 4 = 12 \rightarrow \mathbf{FALS!} \end{aligned}$$

În consecință, condiția **CN2** nu este îndeplinită, deși setul M este totuși fezabil. □

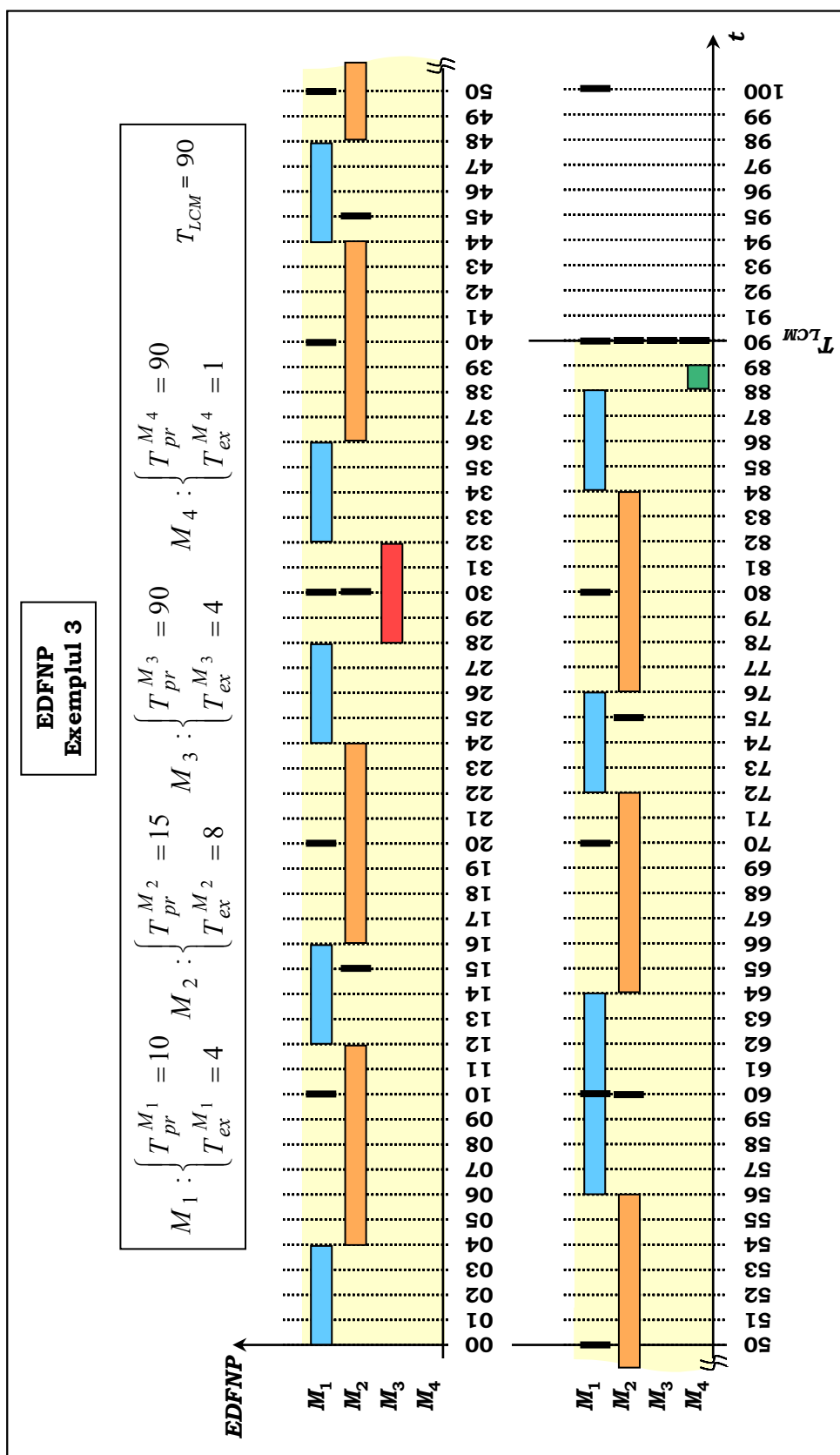


Figura 5-7. Planificarea EDFNP a setului M

Utilizarea ModX-urilor în dezvoltarea, analiza, planificarea și execuția sistemelor și aplicațiilor TR stricte necesită o condiție suplimentară de necesitate a planificabilității. Această condiție este impusă de modelul de task non-preemptiv pe care îl reprezintă ModX-ul.

Teorema 5-1. Condiția a treia de necesitate a planificării ModX-urilor

Fie $M = \{M_1, M_2, \dots, M_n\}$, un set de n ModX-uri simple, independente, caracterizate din punct de vedere temporal prin durată de execuție și perioadă:
 $M_i \equiv (T_{ex}^{M_i}, T_{pr}^{M_i}), i = 1..n.$

Dacă M este planificabil, atunci:

$$\text{CN3)} \quad T_{ex}^{M_{ex\ max}} \leq 2 \left(T_{pr}^{M_{pr\ min}} - T_{ex}^{M_{pr\ min}} \right) \tag{5-15}$$

unde $T_{ex}^{M_{ex\ max}}$ reprezintă durata de execuție a ModX-ului cu valoarea maximă pentru acest parametru din setul M , iar $T_{pr}^{M_{pr\ min}}$ și $T_{ex}^{M_{pr\ min}}$ reprezintă perioada, respectiv durata de execuție a ModX-ului cu perioada minimă din setul M .

Demonstrație

Teorema impune practic o condiție de limitare a duratei maxime de execuție a oricărui ModX din M , vis-a-vis de perioada minimă a ModX-urilor din set, și se bazează pe faptul că ModX-urile nu pot fi întrerupte pe parcursul execuției acestora (operarea non-preemptivă). Figura 5-8 prezintă cazul cel mai defavorabil care poate să apară pentru parametrii temporali ai ModX-urilor unui set M , care este totuși fezabil.

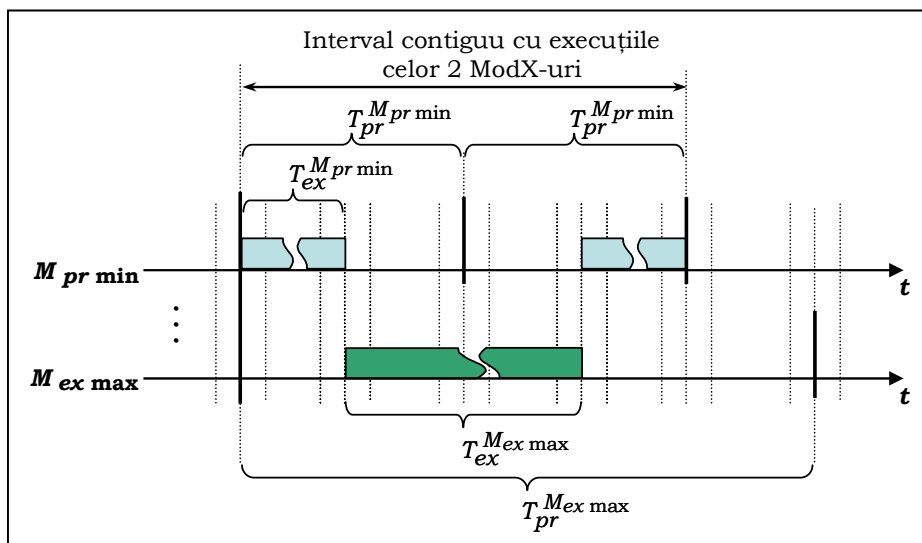


Figura 5-8. Cazul cel mai defavorabil pentru planificarea fezabilă a ModX-urilor unui set M

În situația din Figura 5-8 sunt considerate cele două ModX-uri ale căror parametri temporali pot influența planificabilitatea din punctul de vedere al acestei teoreme. Se poate observa că intervalul de timp disponibil pentru planificarea lui $M_{ex\ max}$ astfel încât să nu se depășească termenele ModX-urilor, este limitat de perioada lui $M_{pr\ min}$ și de durata acestuia de execuție, conform cu relația (5-15), care trebuia demonstrată. □

5.3 Planificarea online non-preemptivă a ModX-urilor independente

În subcapitolul precedent am tratat analiza offline a planificabilității non-preemptive a unui set de ModX-uri simple, independente, fără a lua însă în considerare modul cum poate fi implementată practic planificarea, pe un STR real. Rezultatele analizei offline constau (în cazul în care setul de ModX-uri ale aplicației TR analizate este fezabil) dintr-o listă în care fiecare element e compus din indexul ModX-ului (identificatorul acestuia) și momentul de timp la care e planificat. Astfel, pentru Exemplul 5-5, lista generată la analiza offline a planificabilității e prezentată în Tabelul 5-1, cu un total de 10 elemente, ce acoperă intervalul $[0, T_{LCM}) = [0, 40)$.

Din punct de vedere practic, al implementării planificării pe un STR real (*planificarea online*), se pune problema resurselor necesare, în speță, a spațiului de memorie necesar stocării în sistem a listei cu planificarea ModX-urilor. Dimensiunile listei de planificare variază puternic în funcție de setul de ModX-uri care se dorește a fi planificat și de caracteristicile temporale ale acestora. De exemplu, în cazul în care setul conține relativ multe ModX-uri și cu perioade puternic diferite între ele (fără divizori comuni), iar duratele de execuție sunt mici, rezultă în primul rând un interval de analiză T_{LCM} foarte mare, iar fiecare ModX din set va fi planificat de foarte multe ori în acest interval. Ca rezultat, spațiul de memorie necesar planificării acestui set poate depăși cu ușurință resursele disponibile ale sistemului.

Tabelul 5-1. Lista planificării EDFNP pentru Exemplul 5-5

Nr. element	Indice ModX	Moment planificare
1	1	0
2	2	3
3	1	9
4	2	12
5	1	18
6	2	21
7	1	27
8	2	30
9	1	36
10	3	39

Modelul de sistem TR strict propus în lucrarea de față presupune existența în memoria sistem a unei tabele de dimensiune bine stabilită și a unui task TR strict (un ModX special) cu rolul de a completa în mod ciclic tabela cu ModX-urile rezultate în urma algoritmului de planificare implementat.

Tabela ce conține ModX-urile planificate împreună cu momentele de start ale execuției acestora, și pe care o denumim *Tabela de Dispatch*, este de fapt un buffer, accesat pe de o parte de către *online scheduler* M_S , pentru a o completa ciclic, și pe de altă parte, de către executivul sistemului, *Dispatcher*, cu rolul de lansare în execuție a câte unui ModX din tabelă, la momentul specificat de planificare. Notăm cu λ dimensiunea utilă a Tabelei de Dispatch, adică numărul elemente ale tablei ce conțin ModX-uri ale aplicației, planificate de către M_S în cadrul unui *ciclu de planificare*.

În continuare vom introduce și discuta două abordări diferite ce vizează implementarea practică a algoritmilor de planificare a seturilor de ModX-uri în timpul operării STR (planificarea online).

5.3.1 Planificarea online în cicluri cu număr constant de execuții

Planificarea online în cicluri cu număr constant de execuții, *CEC-NPOS* (*Constant Execution Cycles – Non-Preemptive Online Scheduling*) presupune ca pentru fiecare ciclu de planificare să existe un același număr de planificări de ModX-uri. Prin această metodă se asigură utilizarea la maxim a capacității de memorie alocată pentru Tabela de Dispatch.

Abordarea planificării în cicluri cu număr constant de execuții presupune o structură de buffer circular pentru Tabela de Dispatch (Figura 5-9), în care prima poziție este ocupată întotdeauna de execuția viitoare a planificatorului M_S (execuția ce definește următorul ciclu de planificare), iar restul de λ locații corespund planificărilor de ModX-uri sistem și aplicație din cadrul ciclului curent. Atunci când executivul STR termină ultima execuție planificată din Tabela de Dispatch (M_k , planificat la momentul t_λ în cazul din figură), se inițiază un nou ciclu de planificare. Noul ciclu începe cu execuția planificatorului M_S , parcurgerea Tabelei de Dispatch fiind deci reluată de la prima înregistrare.

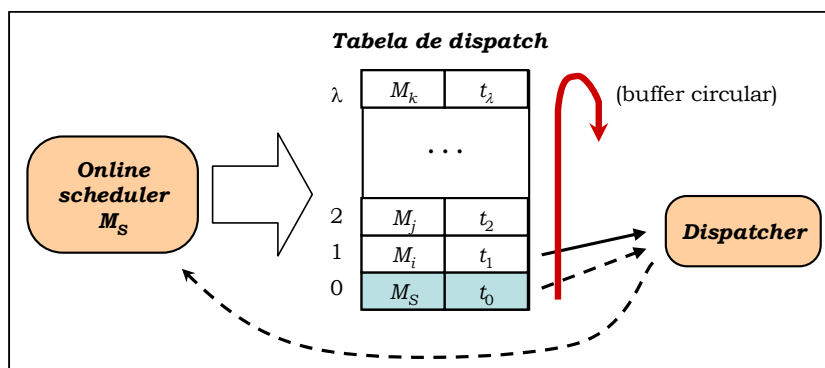


Figura 5-9. Structura de principiu a Tabelei de Dispatch

Task-ul M_S (*online scheduler*) este un caz special de ModX, cu următoarele caracteristici definitorii:

- Execuția lui M_S se desfășoară tot context non-preemptiv;
- $T_{ex}^{M_S}$ este durata de execuție;
- Perioada lui M_S , $T_{pr}^{M_S}$, nu se definește. Așadar, execuția M_S nu este periodică în timp, în schimb task-ul prezintă o execuție ce se repetă ciclic de fiecare dată ce au fost executate λ planificări de ModX-uri ale aplicației;
- Algoritmul de planificare implementat de către M_S va fi o versiune similară a algoritmului utilizat în analiza offline a planificării aplicației, pentru a avea garanția că în timpul operării sistemul se comportă absolut în conformitate cu analiza efectuată offline.

În acest context, analiza offline a planificării unui set M de ModX-uri simple, independente, va trebui să includă și ModX-ul special M_S , care nu este periodic, ci trebuie planificat ciclic, după fiecare λ planificări normale de ModX-uri din M . Algoritmul de planificare propus este o versiune a lui *EDFNP*, modificată corespunzător. A rezultat algoritmul denumit *CEC-NPOS*, cu organigrama operării de principiu prezentată în Figura 5-10. Algoritmul *CEC-NPOS* planifică prima dată (la momentul $t = 0$) pe task-ul M_S , avansează timpul de planificare cu $t = t + T_{ex}^{M_S}$, după care aplică *EDFNP* pentru următoarele λ planificări de ModX-uri ale aplicației (incrementând un contor intern de planificări). În continuare, este planificat din nou M_S , și așa mai departe până se ajunge la acoperirea intervalului de planificare, T_{LCM} .

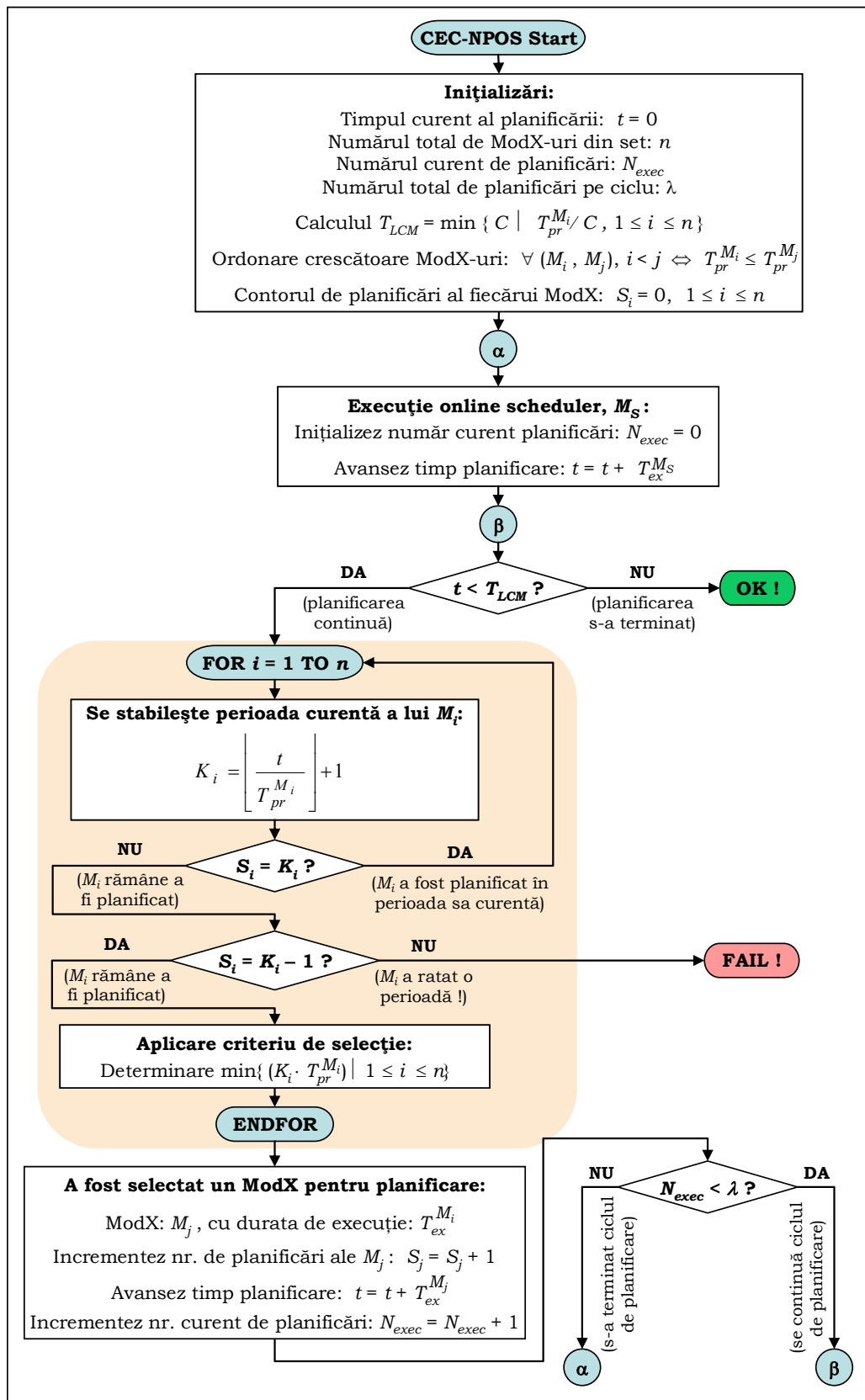


Figura 5-10. Organigrama operării de principiu a algoritmului CEC-NPOS

Introducerea ModX-ului special M_S , care nu e periodic în timp ci are o execuție ciclică, în număr de planificări ale celorlalte ModX-uri din setul M supus analizei, modifică și condițiile de planificabilitate discutate în secțiunea anterioară. Introducem următoarea teoremă:

Teorema 5-2. Condiția de necesitate a planificării *CEC-NPOS*

Fie $M = \{M_1, M_2, \dots, M_n\}$, un set de n ModX-uri simple, independente, caracterizate din punct de vedere temporal prin durată de execuție și perioadă: $M_i \equiv (T_{ex}^{M_i}, T_{pr}^{M_i})$. ModX-urile din M sunt ordonate crescător după perioadă (pentru orice pereche (M_i, M_j) , dacă $i < j$ atunci $T_{pr}^{M_i} \leq T_{pr}^{M_j}$). Considerăm un sistem TR țintă ce dispune de o Tabelă de Dispatch de dimensiune $\lambda + 1$ și de un ModX special de planificare online M_S , cu durata de execuție $T_{ex}^{M_S}$.

Dacă setul M este planificabil cu algoritmul *CEC-NPOS*, atunci:

$$\sum_{i=1}^n \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} + \left[\frac{T_{LCM} \cdot \sum_{i=1}^n \frac{1}{T_{pr}^{M_i}}}{\lambda} \right] \cdot \frac{T_{ex}^{M_S}}{T_{LCM}} \leq 1 \quad (5-16)$$

Demonstrație

Dacă setul M este planificabil, rezultă că planificarea cu *CEC-NPOS* pe intervalul T_{LCM} va avea rezultate pozitive.

Notăm cu N_S , numărul total de execuții ale ModX-ului special M_S în intervalul de lungime T_{LCM} . Dar, N_S va fi egal cu numărul total de execuții ale ModX-urilor din M pe durata T_{LCM} , raportat la λ :

$$N_S = \left[\frac{T_{LCM} \cdot \sum_{i=1}^n \frac{1}{T_{pr}^{M_i}}}{\lambda} \right] \quad (5-17)$$

Prin urmare, fracțiunea din timpul procesor ocupată cu execuția lui M_S pe durata intervalului T_{LCM} este:

$$U^{M_S}(T_{LCM}) = \frac{N_S}{T_{LCM}} \cdot T_{ex}^{M_S} = \left[\frac{T_{LCM} \cdot \sum_{i=1}^n \frac{1}{T_{pr}^{M_i}}}{\lambda} \right] \cdot \frac{T_{ex}^{M_S}}{T_{LCM}} \quad (5-18)$$

Pe de altă parte, condiția planificării pe sisteme monoprosesor specifică faptul că suma fracțiunilor din timpul procesor ocupate de execuția task-urilor planificate trebuie să nu fie supraunitară, pentru orice interval de timp. Rezultă, pentru cazul nostru:

$$\begin{aligned} U^{M+M_S}(\infty) &= U^{M+M_S}(T_{LCM}) = U^M(T_{LCM}) + U^{M_S}(T_{LCM}) = \\ &= \sum_{i=1}^n \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} + U^{M_S}(T_{LCM}) \leq 1 \end{aligned} \quad (5-19)$$

Înlocuind pe (5-18) în (5-19), obținem condiția de necesitate a planificării CEC-NPOS specificată de (5-16). □

Exemplul 5-7.

Considerăm un set $M = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 10 \\ T_{ex}^{M_1} = 2 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 15 \\ T_{ex}^{M_2} = 4 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 20 \\ T_{ex}^{M_3} = 4 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 90 \\ T_{ex}^{M_4} = 4 \end{cases}$$

Setul M urmează a fi planificat în cadrul unui STR cu ajutorul planificatorului online M_S , ce dispune de o Tabelă de Dispatch de dimensiune $10 + 1$ (deci $\lambda = 10$).

Înainte de a începe analiza offline a planificabilității setului M (pentru intervalul $[0, T_{LCM})$, unde $T_{LCM} = 180$), se va efectua verificarea condiției de necesitate a planificabilității, enunțată în Teorema 5-2. Utilizarea procesor ce corespunde setului M de ModX-uri are valoarea:

$$U^M(T_{LCM}) = U^M(180) = \sum_{i=1}^4 \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} = 0.711$$

Pe de altă parte, utilizarea procesor ce corespunde execuțiilor M_S pe intervalul $[0, T_{LCM}) = [0, 180)$, se determină aplicând (5-18):

$$U^{M_S}(T_{LCM}) = U^{M_S}(180) = \left[\frac{180 \cdot \sum_{i=1}^4 \frac{1}{T_{pr}^{M_i}}}{10} \right] \cdot \frac{7}{180} = 0.194$$

Utilizarea procesor a întregului sistem de task-uri, compus din setul M împreună cu ModX-ul special M_S , se determină ca fiind:

$$U^{M+M_S}(180) = U^M(180) + U^{M_S}(180) = 0.905 \quad (5-20)$$

Din (5-20) rezultă că suma fracțiunilor din timpul procesor ocupate de execuția task-urilor planificate nu este supraunitară, deci setul M îndeplinește condiția impusă de planificarea *CEC-NPOS* pe sistemul TR stabilit ca țintă.

În Figura 5-11 este reprezentată analiza offline a planificabilității setului M utilizând algoritmul *CEC-NPOS* pentru intervalul $[0, T_{LCM})$, unde $T_{LCM} = 180$, cu rezultate pozitive. În consecință setul M este fezabil din punct de vedere al planificării și execuției pe sistemul țintă.

□

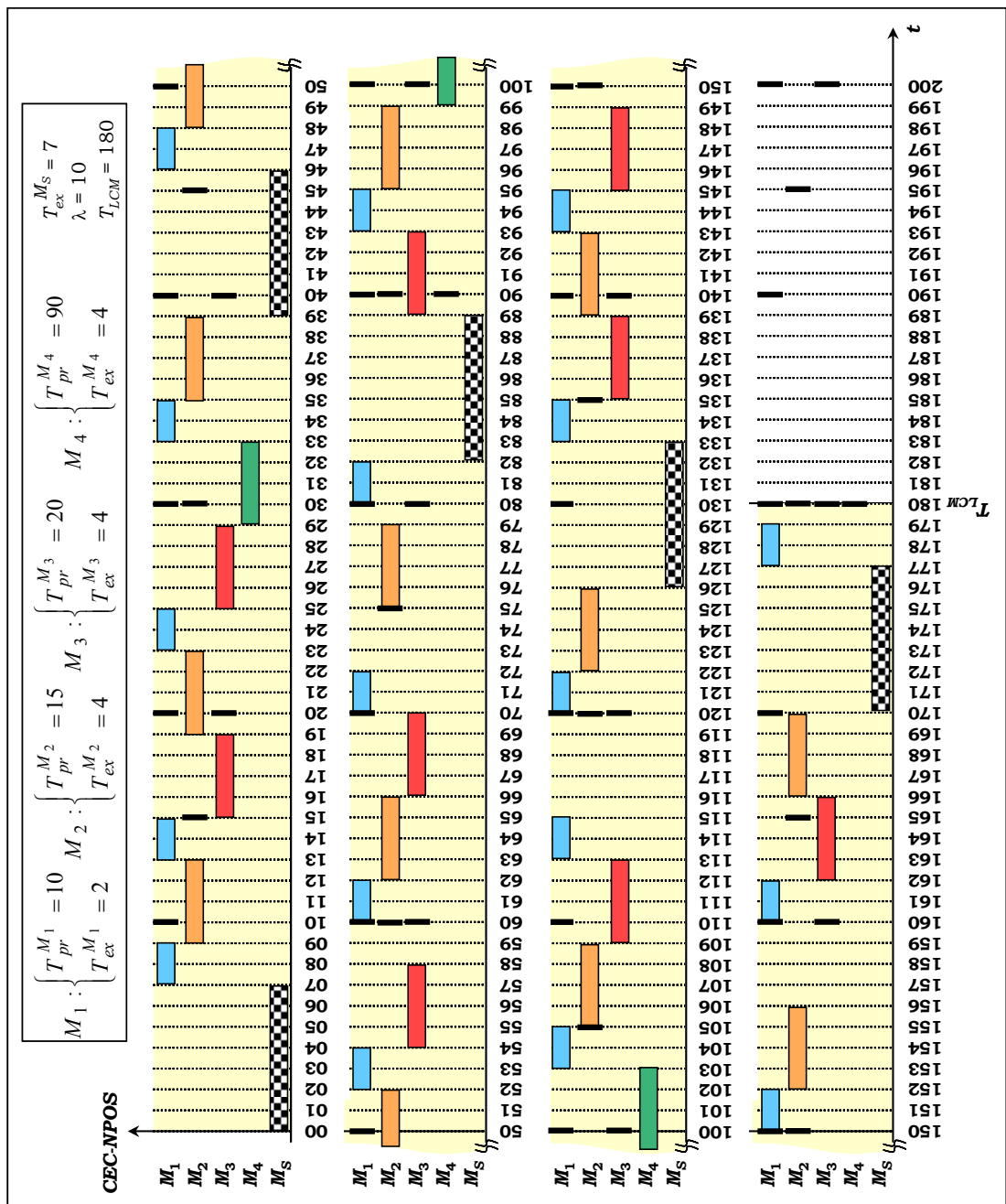


Figura 5-11. Analiza offline a planificabilității *CEC-NPOS* a setului M

5.3.2 Planificarea online în cicluri periodice

Metoda planificării online, în cicluri cu număr constant de execuții, descrisă anterior, impune o serie de restricții pentru setul de ModX-uri sistem și ale aplicației, în principal din cauză că task-ul de planificare M_S nu respectă modelul definit pentru ModX-uri (nu este un task periodic). Mai mult, așa cum se va vedea în subcapitolul următor, în cazul în care setul de ModX-uri supus planificării conține și ModX-uri cu execuție fixă în cadrul perioadei acestora (așa-numitele FModX-uri), abordarea planificării *CEC-NPOS* nu mai este fezabilă.

Rezolvarea acestor probleme pentru planificarea online poate fi găsită cu următoarele soluții:

- Existența în sistem a unui procesor dedicat planificărilor online: așa-numitul "scheduling coprocessor", propus pentru arhitectura sistemului TR distribuit "Spring" [Stankovic 91a, Stankovic 91b, Niehaus 93, Stankovic 99]. În această situație, sistemul țintă devine un sistem multiprocesor, cu un procesor destinat în exclusivitate planificărilor, și unul sau mai multe procesoare pentru execuția task-urilor aplicației. Această soluție oferă în primul rând o importantă creștere a eficienței întregului sistem, degrevându-l de încărcarea suplimentară impusă de execuția planificatorului, care, din punctul de vedere al aplicațiilor, este complet transparentă. Datorită acestei perspective, considerăm abordarea ca un subiect important pentru cercetarea viitoare din domeniu. Ca dezavantaj, se poate totuși menționa creșterea în complexitate a sistemului și algoritmilor de planificare, ce trebuie să vizeze sistemele multiprocesor și cele distribuite.
- Având în vedere o configurație monoprocessor a sistemului țintă (subiectul preocupărilor curente de cercetare), rezolvarea problemei o poate constitui introducerea unui planificator online periodic, care respectă în totalitate caracteristicile modelului de ModX. De aici rezultă abordarea *PC-NPOS* (*Periodic Cycles – Non-Preemptive Online Scheduling*).

Faptul că M_S este un ModX, deci un task periodic, executat în context non-preemptiv, relaxează restricțiile de planificare impuse celorlaltor ModX-uri din set, dar, pe de altă parte complică problema utilizării Tabelei de Dispatch, datorită faptului că numărul total de înregistrări utilizate diferă de la un ciclu de planificare la altul. Se pune problema determinării valorii perioadei lui M_S , $T_{pr}^{M_S}$, astfel încât capacitatea tabelii să nu fie depășită pentru nici un ciclu de planificare, pe toată durata operării sistemului.

Teorema 5-3. A planificării online periodice (*PC-NPOS*)

Considerăm o Tabelă de Dispatch de dimensiune totală $\lambda + 1$ înregistrări (similară ca structură a înregistrărilor cu cea prezentată în Figura 5-9). ModX-ul de planificare online este definit prin următorii parametri temporali:

$M_S \equiv \left(T_{pr}^{M_S}, T_{ex}^{M_S} \right)$. Considerăm de asemenea $M = \{M_1, M_2, \dots, M_n\}$, un set

de n ModX-uri ordonate crescător după perioadă:

$$M_i \equiv (T_{pr}^{M_i}, T_{ex}^{M_i}) \text{ și } \forall (M_i, M_j). i < j \Leftrightarrow T_{pr}^{M_i} \leq T_{pr}^{M_j}$$

Pentru a se putea realiza planificarea online a setului M împreună cu ModX-ul M_S , fără a se depăși capacitatea totală a Tabelei de Dispatch, este necesară verificarea relației:

$$\sum_{i=1}^n \left\lceil \frac{2 \left(T_{pr}^{M_S} - T_{ex}^{M_S} \right)}{T_{pr}^{M_i}} \right\rceil \leq \lambda \tag{5-21}$$

Demonstrație

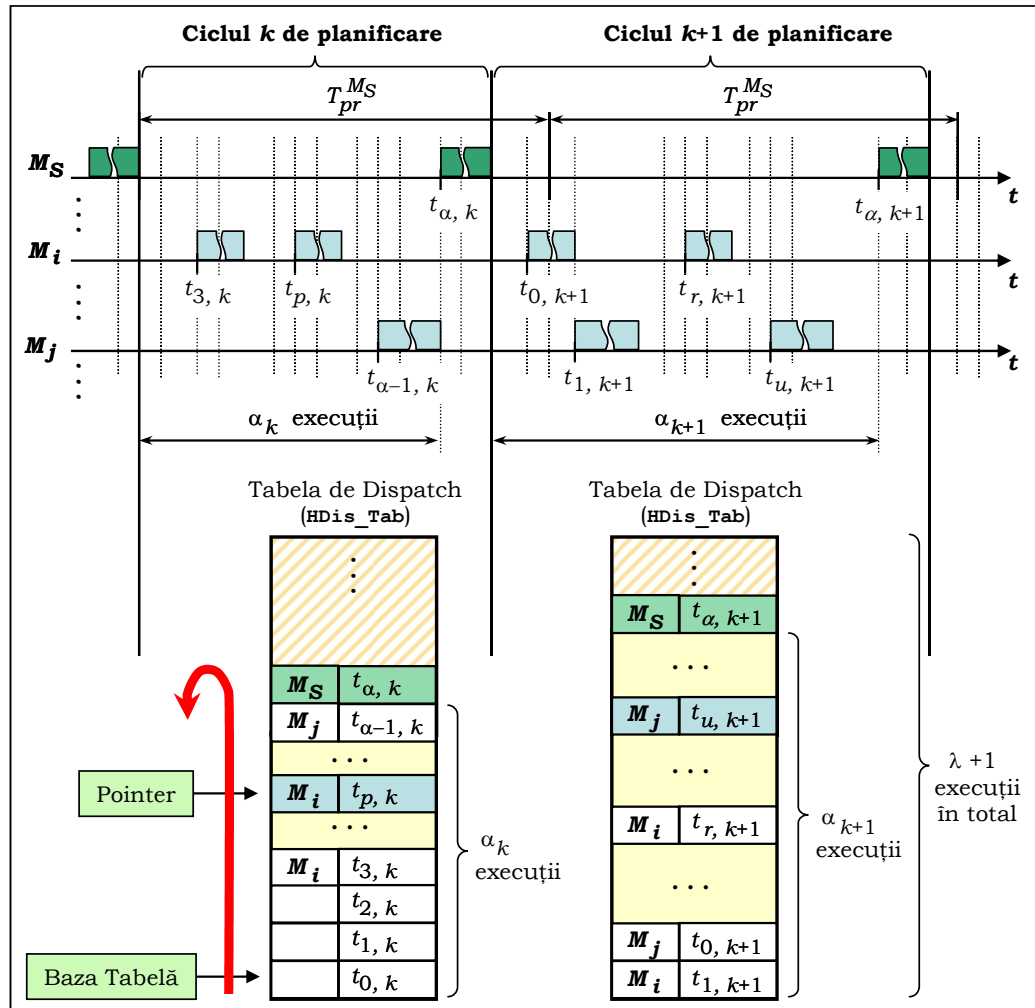


Figura 5-12. Exemplificare a două cicluri consecutive de planificare

Figura 5-12 exemplifică execuția M_S și a ModX-urilor din M pentru două cicluri consecutive de planificare, k și $k + 1$. Structura Tabelei de Dispatch pentru cele două cicluri este de asemenea ilustrată în figură. Se poate observa cum numărul de înregistrări de descriu planificările variază de la un ciclu la altul. Ciclurile de planificare sunt periodice, fiind definite de două execuții consecutive ale lui M_S (practic, un ciclu se termină cu planificarea M_S).

Pentru a nu se depăși capacitatea totală a Tabelei de Dispatch, trebuie ca numărul de planificări ale ModX-urilor din M pentru oricare ciclu, să respecte condiția: $\alpha_k \leq \lambda$, sau:

$$\alpha \leq \lambda, \text{ unde: } \alpha = \max\{\alpha_k \mid k = 1, 2, \dots\} \tag{5-22}$$

Parametrul α_k depinde în mod direct de intervalul de timp din cadrul ciclului k de planificare: cu cât L_k este mai lung, cu atât α_k poate fi mai mare. Cel mai defavorabil caz din acest punct de vedere este cel în care M_S se execută cu întârzierea cea mai mare posibilă, față de execuția precedentă (Figura 5-13).

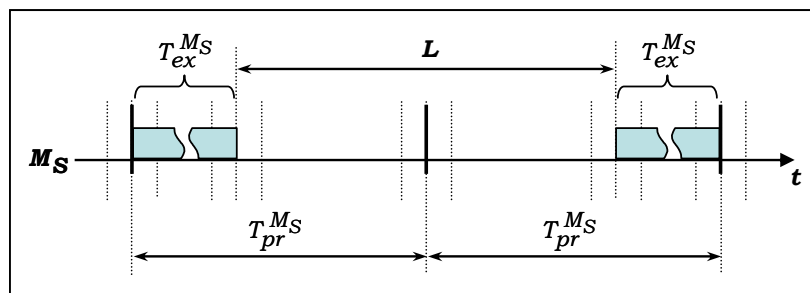


Figura 5-13. Intervalul cel mai lung dintre două execuții succesive ale M_S

$$L = \max\{L_k \mid k = 1, 2, \dots\} = 2 \left(T_{pr}^{M_S} - T_{ex}^{M_S} \right) \tag{5-23}$$

Numărul maxim de execuții ale ModX-urilor din M ce pot fi planificate într-un interval de lungime (durată) L este:

$$\alpha = \sum_{i=1}^n \left\lceil \frac{L}{T_{pr}^{M_i}} \right\rceil \tag{5-24}$$

Din (5-22), (5-23) și (5-24) rezultă relația (5-21), care exprimă restricțiile impuse celor trei parametri care definesc planificarea online periodică pe un sistem țintă: perioada și durata de execuție a planificatorului M_S , și dimensiunea maximă (utilă) a Tabelei de Dispatch. □

Considerând parametrii $T_{ex}^{M_S}$ și λ cu valori fixate/calculate apriori, pe baza relației (5-21) se poate determina o perioadă pentru M_S ($T_{pr}^{M_S}$), suficient de mică pentru a nu se depăși dimensiunea totală a Tabelei de Dispatch, dar în același timp, convenabil de mare pentru a asigura o performanță de operare a sistemului cât mai bună posibil. Termenul însumat în (5-21) este limitat, conform definiției operatorului "⌈ ⌉" ("ceiling"), astfel:

$$2 \frac{T_{pr}^{M_S} - T_{pr}^{M_S}}{T_{pr}^{M_i}} \leq \left\lceil 2 \frac{T_{pr}^{M_S} - T_{pr}^{M_S}}{T_{pr}^{M_i}} \right\rceil \leq 2 \frac{T_{pr}^{M_S} - T_{pr}^{M_S}}{T_{pr}^{M_i}} + 1 \quad (5-25)$$

Prin urmare, o valoare minimală dar validă, pentru perioada M_S se poate obține din (5-21) și (5-25):

$$\begin{aligned} \sum_{i=1}^n \left(2 \frac{\left(T_{pr}^{M_S} \right)_{\min} - T_{ex}^{M_S}}{T_{pr}^{M_i}} + 1 \right) &\leq \lambda \\ 2 \left(\left(T_{pr}^{M_S} \right)_{\min} - T_{ex}^{M_S} \right) \sum_{i=1}^n \frac{1}{T_{pr}^{M_i}} + n &\leq \lambda \\ \Rightarrow \left(T_{pr}^{M_S} \right)_{\min} &= \left\lceil \frac{\lambda - n}{2 \sum_{i=1}^n \frac{1}{T_{pr}^{M_i}}} \right\rceil + T_{ex}^{M_S} \end{aligned} \quad (5-26)$$

De la valoarea $\left(T_{pr}^{M_S} \right)_{\min}$ obținută cu (5-26), se poate determina un maxim valid pentru perioada M_S , prin incrementări succesive, astfel încât noua valoare să respecte relația (5-21). Se poate ușor observa că această metodă de determinare a perioadei planificatorului online M_S atunci când i se cunoaște durata de execuție și dimensiunea maximă a Tabelei de Dispatch, se pretează la o abordare algoritmică, automată, putând fi implementată în contextul unor unelte soft de verificare și analiză automată a planificabilității setului de ModX-uri ce compune o aplicație TR oarecare.

Dimensiunea utilă a Tabelei de Dispatch, λ , este supusă la rândul ei unor restricții de limită inferioară. Pentru ca relația (5-26) să poată fi evaluată, este necesar ca $\lambda - n > 0$, adică:

$$\lambda > n \quad (5-27)$$

Limita inferioară a lui λ depinde așadar în mod direct de numărul total de ModX-uri din M . Cu cât n este mai mare, cu atât este necesară o Tabelă de Dispatch mai mare.

Pe de altă parte, deoarece și M_S este, din punct de vedere al sistemului, tot un ModX, discuțiile legate de condițiile de planificabilitate **CN1**), **CN2**) și **CN3**) din subcapitolul anterior, rămân valabile și se aplică și pentru setul extins $M \cup \{M_S\}$. Astfel, de exemplu durata de execuție a planificatorului online trebuie să fie suficient de mică pentru a satisface **CN3**):

$$T_{ex}^{M_S} \leq 2 \left(T_{pr}^{M_{pr \min}} - T_{ex}^{M_{pr \min}} \right) \quad (5-28)$$

În concluzie, introducerea planificatorului online periodic și a Tabelei de Dispatch în analiza operării sistemului țintă, impune restricțiile (5-21), (5-27) și (5-28) pentru parametrii care definesc acest tip de planificare: $T_{pr}^{M_S}$, $T_{ex}^{M_S}$ și λ .

Exemplul 5-8.

Considerăm un set $M = \{M_1, M_2, M_3, M_4\}$ de ModX-uri simple, independente, caracterizate prin următorii parametri temporali:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 10 \\ T_{ex}^{M_1} = 1 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 15 \\ T_{ex}^{M_2} = 7 \end{cases}; \quad M_3 : \begin{cases} T_{pr}^{M_3} = 24 \\ T_{ex}^{M_3} = 2 \end{cases}; \quad M_4 : \begin{cases} T_{pr}^{M_4} = 24 \\ T_{ex}^{M_4} = 1 \end{cases}$$

Setul M urmează a fi planificat (metoda *PC-NPOS*) în cadrul unui STR cu ajutorul planificatorului online periodic M_S , ce dispune de o Tabelă de Dispatch de dimensiune $12 + 1$ (deci $\lambda = 12$). Considerăm că durata de execuție a lui M_S este $T_{ex}^{M_S} = 8$.

Prima relație care se verifică este (5-27): $\lambda = 12 > n = 4$. ModX-ul cu cea mai mică perioadă din set este M_1 , astfel încât și (5-28) este verificată:

$$T_{ex}^{M_S} \leq 2 \left(T_{pr}^{M_1} - T_{ex}^{M_1} \right) \Leftrightarrow 8 \leq 2(10 - 1) = 18$$

În continuare, se determină perioada minimă a lui M_S , cu ajutorul (5-26):

$$\left(T_{pr}^{M_S} \right)_{\min} = \left\lceil (12 - 4) / \left(2 \sum_{i=1}^n \frac{1}{T_{pr}^{M_i}} \right) \right\rceil + 8 = 24$$

De la această valoare, se determină perioada efectivă a lui M_S , prin incrementare succesivă și verificarea relației (5-21). În final, se va ajunge la $T_{pr}^{M_S} = 30$.

Figura 5-14 prezintă diagrama de analiză offline a planificabilității *PC-NPOS* a setului extins $M \cup \{M_S\}$. Încărcarea procesor implicată de ModX-urile din M este: $U^M(120) = 0.691$, iar cea a setului extins: $U^{M+M_S}(120) = 0.958$. □

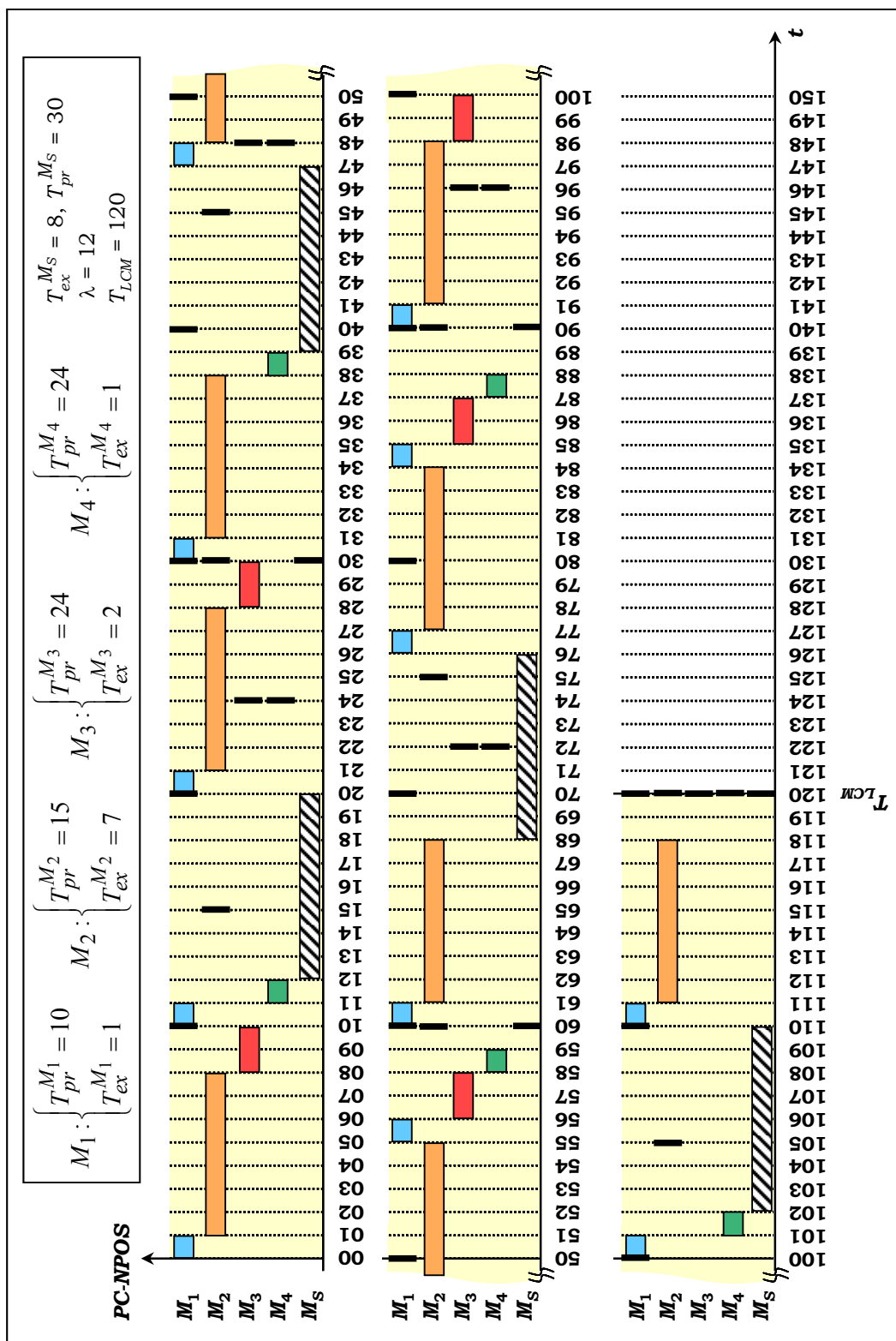


Figura 5-14. Analiza offline a planificabilității PC-NPOS a setului M

5.4 Planificarea non-preemptivă a ModX-urilor independente cu execuție fixă

5.4.1 Introducere

În subcapitolele anterioare a fost tratată problema planificării non-preemptive a unui set de ModX-uri simple, independente, pe sisteme TR stricte. Algoritmii de planificare discutați, dintre care *EDFNP* a fost găsit ca cel mai eficient, sunt capabili a planifica seturi fezabile de ModX-uri, astfel încât acestea să-și respecte termenele stricte impuse de specificațiile de comportare temporală.

Așa cum rezultă din exemplele 5-2, 5-4 până la 5-7, respectiv figurile 5-3, 5-5 până la 5-7, 5-10 și 5-14 momentul exact de planificare a fiecărui ModX în cadrul perioadei sale variază de la o perioadă la alta (de la un ciclu de execuție la altul). Există însă o clasă importantă de aplicații TR stricte, și deci un număr relativ mare de cazuri de ModX-uri, pentru care momentul execuției în cadrul fiecărei perioade trebuie să rămână constant de-a lungul operării. Exemple de aplicații cu astfel de cerințe pentru unele ModX-uri includ achizițiile periodice de date, generatoarele de semnale/de funcții programabile, sistemele de comunicație sincrone, etc.

Exemplul 5-9.

Considerăm o aplicație TR de generare a unui semnal de ieșire strict periodic, de o anumită formă. Pentru simplificare, presupunem ca necesară doar condiția ca primul eșantion al semnalului dintr-o perioadă oarecare să fie egal distanțat în timp de eșantionul corespunzător al perioadelor vecine, pe toată durata generării semnalului.

Task-ul care generează semnalul (sau doar primul eșantion al fiecărei perioade a semnalului) trebuie să fie periodic, și, mai mult, intervalele de timp ce separă începutul fiecărei execuții a task-ului trebuie să fie egale cu perioada semnalului (vezi Figura 5-15).

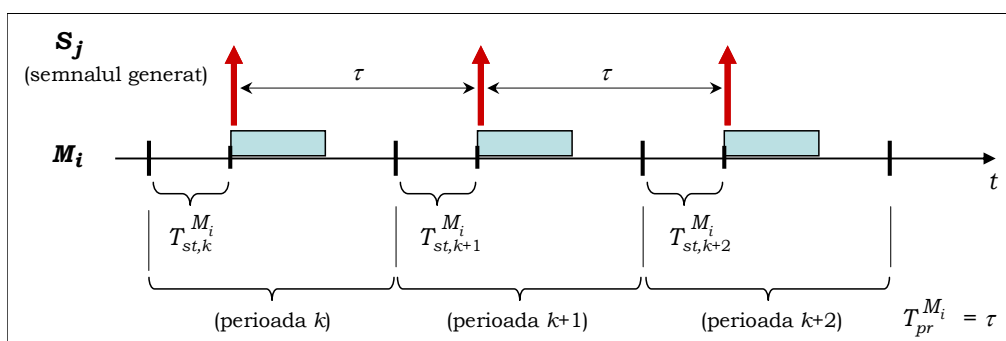


Figura 5-15. Generarea unui semnal de ieșire perfect periodic

Cea mai simplă abordare constă din utilizarea mecanismului de întreruperi și programarea unui timer pentru perioada τ . Problema acestei abordări constă însă în cazurile în care apar alte întreruperi mai prioritare în sistem, adică există alte task-uri, mai prioritare, ce întrerup execuția lui M_i , compromițând astfel generarea semnalului la perioade egale.

Utilizând modelul de ModX-uri, propus în lucrarea de față, problema întreruperilor este rezolvată. ModX-ul M_i , responsabil pentru generarea semnalului S_j , va avea perioada $T_{pr}^{M_i} = \tau$. Va mai trebui prevăzută însă o restricție în faza de specificare, anume aceea că M_i se va executa în mod obligatoriu cu aceeași întârziere față de momentul de început al fiecărei perioade, adică:

$$T_{st,k}^{M_i} = T_{st,k+1}^{M_i} = T_{st}^{M_i}, \forall k = 1, 2, \dots \quad (5-29)$$

□

Definiția 5-3.

Numim *ModX cu execuție fixă*, sau mai simplu, *FModX*, un ModX a cărui planificare și execuție se vor efectua în mod obligatoriu cu o aceeași întârziere față de momentul de început al fiecărei perioade (5-29). Cu alte cuvinte, intervalul dintre oricare două execuții efective ale unui FModX în sistem este constant și egal chiar cu perioada FModX-ului. Intervalul de timp dintre startul unei perioade oarecare a unui FModX și momentul execuției sale este denumit *interval (sau durată) de start*, și e notat cu $T_{st}^{M_i}$.

□

Ca observație, menționăm că specificarea unui ModX cu execuție fixă se poate realiza și cu ajutorul modelului de ModX descris în capitolul precedent, utilizând cei doi parametri care restricționează momentul execuției în cadrul perioadei – intervalul de întârziere ($T_{dy}^{M_i}$) și intervalul limită ($T_{dl}^{M_i}$):

$$\begin{cases} T_{dy}^{M_i} = T_{st}^{M_i} \\ T_{dl}^{M_i} = T_{st}^{M_i} + T_{ex}^{M_i} \end{cases}$$

În paragrafele următoare vom aborda problema planificării non-preemptive a ModX-urilor cu execuție fixă, independente (*FENP: Fixed Execution Non-Preemptive*). Vom propune și demonstra un model matematic pentru FModX-uri, pe care îl vom utiliza la studiul planificării non-preemptive a seturilor de FModX-uri independente.

5.4.2 Modelul matematic al ModX-urilor cu execuție fixă

Definiția 5-4.

Definim funcția $M_i(t)$ după cum urmează:

$$M_i(t): \mathbf{N} \rightarrow \{0,1\}, \text{ cu :} \\ M_i(t) = \sigma\left(t \bmod T_{pr}^{M_i} - T_{st}^{M_i}\right) - \sigma\left(t \bmod T_{pr}^{M_i} - T_{st}^{M_i} - T_{ex}^{M_i}\right) \quad (5-30)$$

unde: "mod" reprezintă *operatorul modulo*, iar $\sigma: \mathbf{Z} \rightarrow \{0,1\}$, $\sigma(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$ este *funcția treaptă unitate*.

Dacă stabilim convenția ca valoarea "1" a funcției $M_i(t)$ reprezintă starea de execuție a unui ModX M_i la momentul t , atunci funcția $M_i(t)$ modelează din punct de vedere matematic comportarea în timp a operării ModX-urilor cu execuție fixă (a FModX-urilor).

□

Pentru ilustrarea definiției modelului matematic al FModX-urilor, prezentăm următorul exemplu:

Exemplul 5-10.

Considerăm un ModX cu execuție fixă M_1 , caracterizat prin următorii parametri temporali (Figura 5-16):

$$M_1: \begin{cases} T_{pr}^{M_1} = 8 \\ T_{ex}^{M_1} = 3 \\ T_{st}^{M_1} = 2 \end{cases}$$

Operarea FModX-ului M_1 este modelată de funcția:

$$M_1(t) = \sigma(t \bmod 8 - 2) - \sigma(t \bmod 8 - 2 - 3) = \sigma(t \bmod 8 - 2) - \sigma(t \bmod 8 - 5)$$

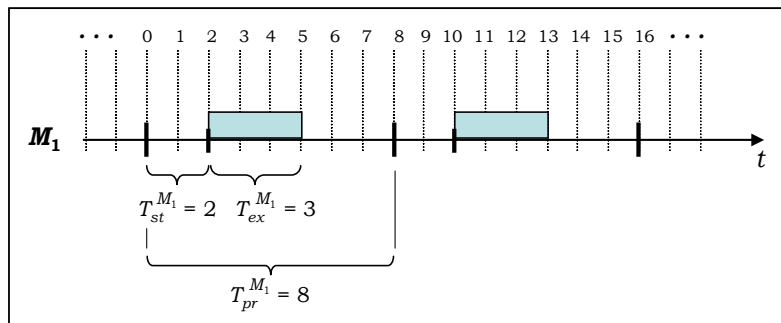


Figura 5-16. Operarea FModX-ului M1 pentru (primele) două perioade ale sale

Valorile funcției $M_1(t)$ pentru câteva instanțe de timp sunt după cum urmează:

$$M_1(1) = \sigma(1 \bmod 8 - 2) - \sigma(1 \bmod 8 - 5) = \sigma(-1) - \sigma(-4) = 0$$

$$M_1(3) = \sigma(3 \bmod 8 - 2) - \sigma(3 \bmod 8 - 5) = \sigma(1) - \sigma(-2) = 1$$

$$M_1(5) = \sigma(5 \bmod 8 - 2) - \sigma(5 \bmod 8 - 5) = \sigma(3) - \sigma(0) = 0$$

$$M_1(14) = \sigma(14 \bmod 8 - 2) - \sigma(14 \bmod 8 - 5) = \sigma(4) - \sigma(1) = 0$$

□

Modelul matematic al ModX-urilor cu execuție fixă se bazează pe operatorul "mod" și pe proprietățile "aritmeticii modulare" din teoria numerelor [Chouinard 02, Ning 02, Lerma 03]. Câteva proprietăți ale operatorului modulo, care vor fi utilizate în studiul planificării FModX-urilor, sunt enunțate în continuare.

Fie $a, b, c \in \mathbb{Z}$ și $n \in \mathbb{Z}^*$.

Proprietatea 5-1. Operatorul $(\bmod n)$ împarte \mathbb{Z} în n seturi de întregi:

$$\forall a \in \mathbb{Z}, a \in \mathbb{Z}_n = \{0, 1, \dots, (n-1)\}.$$

Proprietatea 5-2. Comutativitatea adunării și înmulțirii modulare:

$$\begin{cases} (a+b) \bmod n = (b+a) \bmod n \\ (a \cdot b) \bmod n = (b \cdot a) \bmod n \end{cases}, \forall a, b \in \mathbb{Z}_n.$$

Proprietatea 5-3. Asociativitatea:

$$\begin{cases} ((a+b)+c) \bmod n = (a+(b+c)) \bmod n \\ ((a \cdot b) \cdot c) \bmod n = (a \cdot (b \cdot c)) \bmod n \end{cases}, \forall a, b, c \in \mathbb{Z}_n.$$

Proprietatea 5-4. Distributivitatea:

$$(a \cdot (b+c)) \bmod n = (a \cdot b + a \cdot c) \bmod n, \forall a, b, c \in \mathbb{Z}_n.$$

Proprietatea 5-5. Elementul neutru:

$$\begin{cases} (0+a) \bmod n = a \bmod n \\ (1 \cdot a) \bmod n = a \bmod n \end{cases}, \forall a \in \mathbb{Z}_n.$$

Proprietatea 5-6. Elementul invers față de adunarea modulară:

$$\forall a \in \mathbb{Z}_n, \exists b \in \mathbb{Z}_n; (a+b) \bmod n = 0.$$

Proprietatea 5-7. Elementul invers față de înmulțirea modulară:

$$\forall a \in \mathbb{Z}_n, \exists b \in \mathbb{Z}_n; (a \cdot b) \bmod n = 1,$$

dacă și numai dacă a și n sunt relativ prime (cel mai mare divizor comun al lui a și n este 1).

Proprietatea 5-8. Descompunerea modulară față de adunare și înmulțire:

$$\begin{cases} (a + b) \bmod n = (a \bmod n + b \bmod n) \bmod n \\ (a \cdot b) \bmod n = (a \bmod n \cdot b \bmod n) \bmod n \end{cases}, \quad \forall a, b \in \mathbf{Z}_n.$$

Proprietatea 5-9. Congruența a două numere:

$$\forall a, b \in \mathbf{Z}_n; \left(a \stackrel{\text{not}}{\equiv} b \right) \bmod n, \text{ dacă și numai dacă } n \text{ divide pe } (a - b)$$

(adică, $n/(a - b)$), sau $(a - b) \bmod n = 0$.

5.4.3 Planificarea unui set de două FModX-uri

Fie $\mathbf{M} = \{M_1, M_2\}$, un set de două ModX-uri independente, cu execuție fixă, ordonate crescător după perioadă:

$$\begin{cases} M_1 \equiv (T_{pr}^{M_1}, T_{ex}^{M_1}, T_{st}^{M_1}) \\ M_2 \equiv (T_{pr}^{M_2}, T_{ex}^{M_2}, T_{st}^{M_2}) \end{cases}, \text{ cu } T_{pr}^{M_1} \leq T_{pr}^{M_2}.$$

Conform cu Definiția 5-4, cele două FModX-uri sunt modelate cu ajutorul următoarelor funcții:

$$\begin{cases} M_1(t) = \sigma(t \bmod T_{pr}^{M_1} - T_{st}^{M_1}) - \sigma(t \bmod T_{pr}^{M_1} - T_{st}^{M_1} - T_{ex}^{M_1}) \\ M_2(t) = \sigma(t \bmod T_{pr}^{M_2} - T_{st}^{M_2}) - \sigma(t \bmod T_{pr}^{M_2} - T_{st}^{M_2} - T_{ex}^{M_2}) \end{cases}, \quad t \in \mathbf{N}$$

Pentru studiul planificării non-preemptive a setului \mathbf{M} , considerăm următoarele notații:

- Cel mai mare divizor comun al perioadelor lui M_1 și M_2 :

$$\mathcal{D} = \text{GCD}^{\text{not}}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \max\{\delta \in \mathbf{N}^* \mid T_{pr}^{M_1} \bmod \delta = T_{pr}^{M_2} \bmod \delta = 0\} \quad (5-31)$$

- Cel mai mic multiplu comun al perioadelor lui M_1 și M_2 :

$$\mathcal{M} = \text{LCM}^{\text{not}}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \min\{\mu \in \mathbf{N}^* \mid \mu \bmod T_{pr}^{M_1} = \mu \bmod T_{pr}^{M_2} = 0\} \quad (5-32)$$

Definiția 5-5.

Definim *maparea planificării fixe a lui M_1 în perioada lui M_2* ($T_{pr}^{M_2}$), ca fiind o funcție de forma:

$$\Delta_{M_2/M_1} : \{0, 1, \dots, T_{pr}^{M_2} - 1\} \rightarrow \{0, 1\}$$

$$\Delta_{M_2/M_1}(\tau) = \bigcup_{k=0}^{\frac{T_{pr}^{M_1}}{T_{pr}^{M_2}} - 1} M_1(\tau + k \cdot T_{pr}^{M_2}) \quad (5-33)$$

□

Funcția de mapare a planificării fixe a lui M_1 în perioada lui M_2 calculează pentru orice moment de timp τ relativ la intervalul $T_{pr}^{M_2}$ o valoare ce specifică dacă M_1 se află în execuție sau nu, de-a lungul operării setului \mathbf{M} în cadrul STR. Calculul unei valori $\Delta_{M_2/M_1}(\tau)$ se bazează pe reunirea (operatorul "∪" – "sau" logic) valorilor funcției $M_1(t)$ pentru toate instanțele de timp t care se mapează în "poziția relativă" τ în cadrul perioadei lui M_2 , de-a lungul operării setului \mathbf{M} . Exemplul următor ilustrează un caz practic de calcul a funcției de mapare $\Delta_{M_2/M_1}(\tau)$.

Exemplul 5-11.

Considerăm un set $\mathbf{M} = \{M_1, M_2\}$, compus din două FModX-uri independente, cu următoarele caracteristici temporale:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 15 \\ T_{ex}^{M_1} = 3 \\ T_{st}^{M_1} = 0 \end{cases} ; M_2 : \begin{cases} T_{pr}^{M_2} = 20 \\ T_{ex}^{M_2} = 1 \\ T_{st}^{M_2} = 4 \end{cases}$$

Cei doi parametri suplimentari ce derivă din caracteristicile temporale ale setului \mathbf{M} sunt, conform (5-31) și (5-32):

$$\begin{cases} \mathcal{D} = \text{GCD} \{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \text{GCD} \{15, 20\} = 5 \\ \mathcal{M} = \text{LCM} \{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \text{LCM} \{15, 20\} = 60 \end{cases}$$

adică, cel mai mare divizor comun, respectiv cel mai mic multiplu comun al perioadelor lui M_1 și M_2 . Conform relației (5-33), funcția de mapare a planificării fixe a lui M_1 în perioada lui M_2 este de forma:

$$\Delta_{M_2/M_1}(\tau) = \bigcup_{k=0}^2 M_1(\tau + k \cdot 20), \text{ cu } \tau = 0, 1, \dots, 19.$$

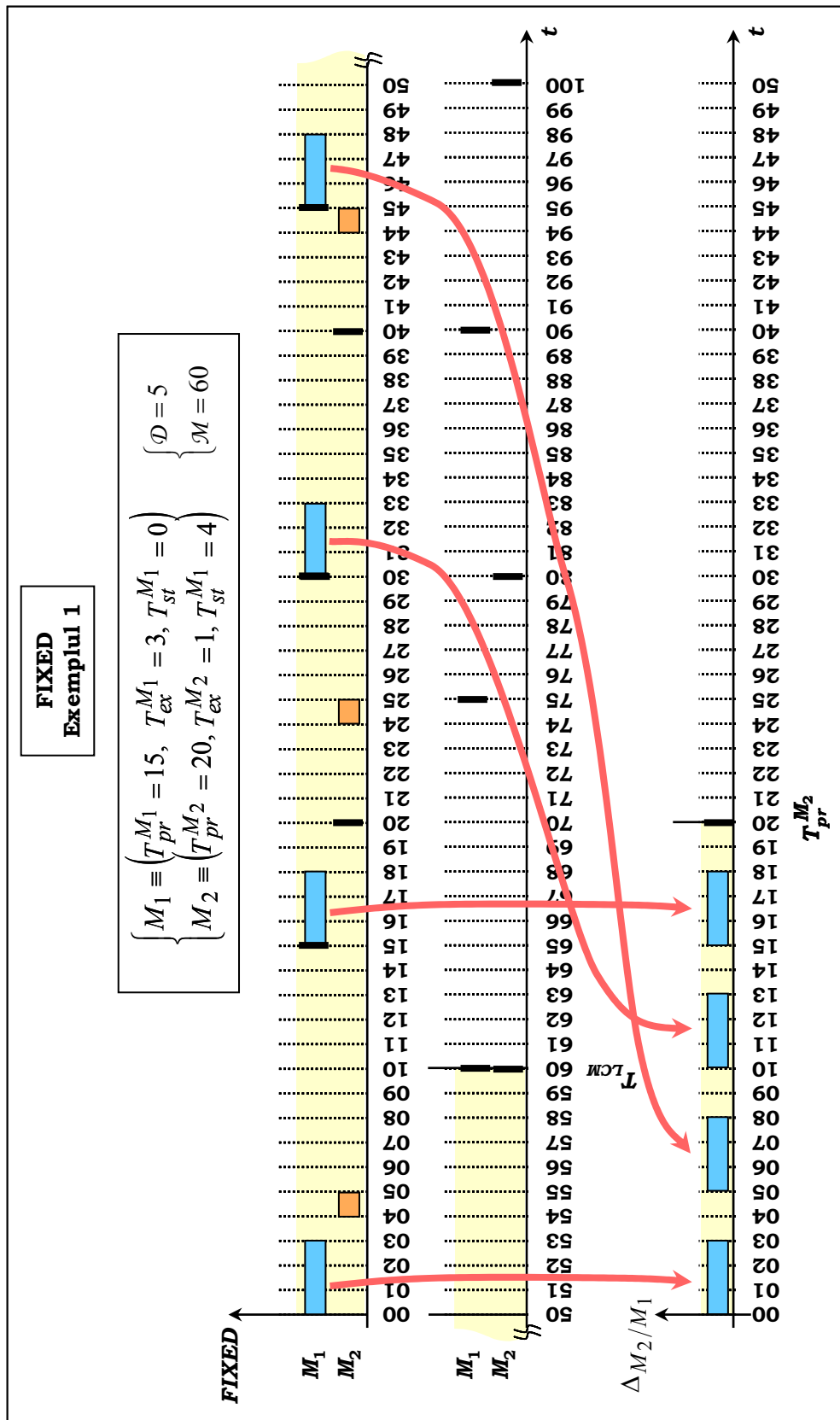


Figura 5-17. Planificarea fixă a setului M de FModX-urii și funcția de mapare

Încărcarea procesor a setului M nu este supraunitară, satisfăcând astfel condiția de necesitate **CN1**) a planificabilității (5-13):

$$U^M = \sum_{i=1}^2 \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} = 0.25 \leq 1$$

Figura 5-17 ilustrează analiza offline a planificabilității setului M de FModX-uri și funcția de mapare $\Delta_{M_2/M_1}(\tau)$. Calculul funcției de mapare pentru instanțele timpului relativ la perioada lui M_2 , τ , se efectuează utilizând relațiile (5-33) și (5-30):

$$\begin{aligned} \Delta_{M_2/M_1}(1) &= \bigcup_{k=0}^2 M_1(1+k \cdot 20) = M_1(1) \cup M_1(21) \cup M_1(41) = \\ &= (\sigma(1 \bmod 15 - 0) - \sigma(1 \bmod 15 - 3)) \cup \\ &\quad \cup (\sigma(21 \bmod 15 - 0) - \sigma(21 \bmod 15 - 3)) \cup \\ &\quad \cup (\sigma(41 \bmod 15 - 0) - \sigma(41 \bmod 15 - 3)) = \\ &= (\sigma(1) - \sigma(-2)) \cup (\sigma(6) - \sigma(3)) \cup (\sigma(11) - \sigma(8)) = 1 \cup 0 \cup 0 = 1 \end{aligned}$$

$$\begin{aligned} \Delta_{M_2/M_1}(9) &= \bigcup_{k=0}^2 M_1(9+k \cdot 20) = M_1(9) \cup M_1(29) \cup M_1(49) = \\ &= (\sigma(9 \bmod 15 - 0) - \sigma(9 \bmod 15 - 3)) \cup \\ &\quad \cup (\sigma(29 \bmod 15 - 0) - \sigma(29 \bmod 15 - 3)) \cup \\ &\quad \cup (\sigma(49 \bmod 15 - 0) - \sigma(49 \bmod 15 - 3)) = \\ &= (\sigma(9) - \sigma(6)) \cup (\sigma(14) - \sigma(11)) \cup (\sigma(4) - \sigma(1)) = 0 \cup 0 \cup 0 = 0 \end{aligned}$$

□

Înainte de a aborda planificarea unui set de ModX-uri independente cu execuție fixă utilizând funcția de mapare $\Delta_{M_2/M_1}(\tau)$, vom studia câteva proprietăți importante ale acesteia.

Teorema 5-4.

Fie $\mathbf{M} = \{M_1, M_2\}$, un set de două ModX-uri cu execuție fixă, ordonate crescător după perioadă:

$$\begin{cases} M_1 \equiv (T_{pr}^{M_1}, T_{ex}^{M_1}, T_{st}^{M_1}) \\ M_2 \equiv (T_{pr}^{M_2}, T_{ex}^{M_2}, T_{st}^{M_2}) \end{cases}, \text{ cu } T_{pr}^{M_1} \leq T_{pr}^{M_2},$$

și $\mathcal{D} \in \mathbf{N}^*$, cel mai mare divizor comun al perioadelor lui M_1 și M_2 :

$$\mathcal{D} = \text{GCD}^{\text{not}}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \max\left\{\delta \in \mathbf{N}^* \mid T_{pr}^{M_1} \bmod \delta = T_{pr}^{M_2} \bmod \delta = 0\right\}.$$

Funția de mapare a planificării fixe a lui M_1 în perioada lui M_2 , $\Delta_{M_2/M_1}(\tau)$, este periodică, de perioadă \mathcal{D} :

$$\Delta_{M_2/M_1}(t_0) = \Delta_{M_2/M_1}(t_0 + \mathcal{D}), \quad \forall t_0 \in \{0, 1, \dots, T_{pr}^{M_2} - 1\} \quad (5-34)$$

Demonstrație

Din definiția funcției de mapare (5-33) și din (5-34), va trebui demonstrat că:

$$\forall k \in \left\{0, 1, \dots, \frac{T_{pr}^{M_1}}{\mathcal{D}} - 1\right\}, \exists m \in \left\{0, 1, \dots, \frac{T_{pr}^{M_1}}{\mathcal{D}} - 1\right\}, \text{ astfel incat :} \quad (5-35)$$

$$M_1(t_0 + k \cdot T_{pr}^{M_2}) = M_1(t_0 + \mathcal{D} + m \cdot T_{pr}^{M_2})$$

Introducem notațiile:

$$\begin{cases} T_{pr}^{M_1} = \alpha \cdot \mathcal{D} \\ T_{pr}^{M_2} = \beta \cdot \mathcal{D} \end{cases}, \text{ cu } T_{pr}^{M_1} \leq T_{pr}^{M_2},$$

de unde rezultă că $\alpha \leq \beta$, și mai departe:

$$\beta = \gamma \cdot \alpha + \delta \quad (5-36)$$

unde: $\delta \in Z_\alpha = \{0, 1, \dots, \alpha - 1\}$, iar α și δ sunt relativ prime ($\text{GCD}\{\alpha, \delta\} = 1$).

Relația (5-36) se obține din faptul că $\text{GCD}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \mathcal{D}$. Presupunând că

$\text{GCD}\{\alpha, \delta\} = \xi > 1$, ar rezulta că $\text{GCD}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \mathcal{D} \cdot \xi$, fals.

Cu notațiile de mai sus și aplicând definiția funcției $M_1(t)$ conform (5-30), formula (5-35) ce trebuie demonstrată devine:

$$\forall k \in \mathbf{Z}_\alpha = \{0, 1, \dots, \alpha - 1\}, \exists m \in \mathbf{Z}_\alpha, \text{ astfel incat :} \quad (5-37)$$

$$(t_0 + k \cdot \beta \cdot \mathcal{D}) \bmod (\alpha \cdot \mathcal{D}) = (t_0 + (m \cdot \beta + 1) \cdot \mathcal{D}) \bmod (\alpha \cdot \mathcal{D})$$

Aplicând Proprietatea 5-9 a operatorului "mod", (5-37) se transformă în:

$$(t_0 + k \cdot \beta \cdot \mathcal{D} - t_0 - \mathcal{D} - m \cdot \beta \cdot \mathcal{D}) \bmod (\alpha \cdot \mathcal{D}) = 0$$

Utilizând descompunerea modulară (Proprietatea 5-8), rezultă:

$$\begin{aligned} ((k - m) \cdot \beta) \bmod \alpha &= 1 \\ ((k - m) \bmod \alpha \cdot (\gamma \cdot \alpha + \delta) \bmod \alpha) \bmod \alpha &= 1 \\ ((k - m) \bmod \alpha \cdot \delta) \bmod \alpha &= 1 \end{aligned}$$

Dar, (5-36) permite existența elementului invers față de înmulțirea modulară (Proprietatea 5-7):

$$\exists \mu \in \mathbf{Z}_\alpha, \text{ astfel încât } (\mu \cdot \delta) \bmod \alpha = 1$$

Rezultă:

$$\begin{aligned} (k - m) \bmod \alpha &= \mu, \text{ sau} \\ m &= (k - \mu) \bmod \alpha \end{aligned} \quad (5-38)$$

Deci, pentru orice $k \in \mathbf{Z}_\alpha$, $\exists m \in \mathbf{Z}_\alpha$, cu $m = (k - \mu) \bmod \alpha$ și $\mu \in \mathbf{Z}_\alpha$, astfel încât formula (5-35) este adevărată. Mai mult, m ia toate valorile din mulțimea $\mathbf{Z}_\alpha = \{0, 1, \dots, \alpha - 1\}$, pe măsură ce și k baleiază întreaga \mathbf{Z}_α . \square

Corolarul 5-1.

Funcția de mapare a planificării fixe a lui M_1 în perioada lui M_2 , $\Delta_{M_2/M_1}(\tau)$, conține exact v_{M_2/M_1} perioade, unde:

$$v_{M_2/M_1} = \frac{T_{pr}^{M_2}}{\mathcal{D}} \quad (5-39)$$

Demonstrație

Demonstrația este o consecință imediată a definiției funcției $\Delta_{M_2/M_1}(\tau)$ (Definiția 5-5) și a Teoremei 5-4. Astfel, știm că $\Delta_{M_2/M_1}(\tau)$ este definită pentru $\tau \in \{0, 1, \dots, T_{pr}^{M_2} - 1\}$, adică pentru $T_{pr}^{M_2}$ unități de timp, și este o funcție periodică, de perioadă \mathcal{D} . Rezultă că

$$v_{M_2/M_1} = \frac{T_{pr}^{M_2}}{\mathcal{D}} = \beta$$

reprezintă numărul de perioade ale lui $\Delta_{M_2/M_1}(\tau)$, ce împart un interval de durată $T_{pr}^{M_2}$. \square

Ca observație, menționăm că pe durata unui interval de planificare ($T_{LCM} = \text{LCM}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \mathcal{M}$, conform relației (5-32)), se vor mapa în perioada lui M_2 exact

$$\frac{\text{LCM}\{T_{pr}^{M_1}, T_{pr}^{M_2}\}}{T_{pr}^{M_1}} = \frac{T_{pr}^{M_2}}{\mathcal{D}} = v_{M_2/M_1}$$

execuții ale lui M_1 . De aici și din Corolarul 5-1 rezultă că funcția de mapare a planificării fixe a lui M_1 în perioada lui M_2 este o funcție definită pe $T_{pr}^{M_2}$ instanțe de timp, pe care le împarte în v_{M_2/M_1} intervale identice (perioade), de lungime \mathcal{D} , iar fiecare perioadă a lui $\Delta_{M_2/M_1}(\tau)$ include (mapează) o execuție a lui M_1 , de durată $T_{ex}^{M_1}$.

Cu ajutorul funcției de mapare și a proprietăților acesteia, studiate mai sus, putem aborda problema planificării non-preemptive a unui set oarecare de două ModX-uri cu execuție fixă.

Teorema 5-5.

Fie $\mathbf{M} = \{M_1, M_2\}$, un set de două ModX-uri cu execuție fixă, ordonate crescător după perioadă:

$$\begin{cases} M_1 \equiv (T_{pr}^{M_1}, T_{ex}^{M_1}, T_{st}^{M_1}) \\ M_2 \equiv (T_{pr}^{M_2}, T_{ex}^{M_2}, T_{st}^{M_2}) \end{cases}, \text{ cu } T_{pr}^{M_1} \leq T_{pr}^{M_2}.$$

Setul \mathbf{M} este planificabil în context non-preemptiv, dacă și numai dacă:

$$\exists t_0 \in \{0, 1, \dots, T_{pr}^{M_2} - T_{ex}^{M_2}\}, \text{ astfel incat } \bigcup_{\tau=t_0}^{t_0+T_{ex}^{M_2}-1} \Delta_{M_2/M_1}(\tau) = 0 \quad (5-40)$$

Demonstrație

a) Implicația directă: dacă (5-40) e adevărată, rezultă că setul \mathbf{M} este fezabil.

Dacă (5-40) este adevărată, rezultă că în cadrul perioadei lui M_2 există un interval de timp de durată egală cu $T_{ex}^{M_2}$, care începe de la momentul t_0 și în cadrul căruia nu se mapează execuția lui M_1 , pentru toată durata planificării. De aici rezultă faptul că intervalul de timp $[t_0, t_0 + T_{ex}^{M_2} - 1]$ poate fi alocat execuției lui M_2 în cadrul planificării setului \mathbf{M} .

Deoarece execuțiile lui M_1 și M_2 nu se vor suprapune pe durata planificării în condițiile de mai sus, rezultă că există o planificare fezabilă a setului \mathbf{M} .

b) Implicația inversă: dacă setul \mathbf{M} este fezabil, rezultă că (5-40) se verifică.

Dacă setul \mathbf{M} este fezabil, rezultă că există o planificare a lui M_1 și M_2 astfel încât execuțiile acestora să nu se suprapună pe durata operării.

Acest lucru este echivalent cu faptul că maparea execuției lui M_1 în perioada lui M_2 pe durata planificării, conține cel puțin un interval de timp care este mai mare sau egal cu durata de execuție a lui M_2 . De aici rezultă că (5-40) este adevărată. □

Corolarul 5-2.

Dacă setul $M = \{M_1, M_2\}$ de FModX-uri este planificabil, astfel încât

$$\exists t_0 \in \{0, 1, \dots, T_{pr}^{M_2} - T_{ex}^{M_2}\}, \text{ cu } \bigcup_{\tau=t_0}^{t_0+T_{ex}^{M_2}-1} \Delta_{M_2/M_1}(\tau) = 0,$$

atunci: $T_{st}^{M_2} = t_0$.

Demonstrație

Demonstrația corolarului derivă direct din demonstrația Teoremei 5-5. □

Corolarul 5-3.

Dacă setul $M = \{M_1, M_2\}$ de FModX-uri este planificabil, atunci:

$$T_{ex}^{M_1} + T_{ex}^{M_2} \leq \mathcal{D} = \text{GCD}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} \quad (5-41)$$

Demonstrație

Teorema 5-5 afirmă că pentru ca setul M să fie fezabil, trebuie să existe cel puțin un interval de valori succesive ale lui τ , pentru care funcția de mapare $\Delta_{M_2/M_1}(\tau)$ este nulă, iar intervalul respectiv va avea o lungime (durată) cel puțin egală cu $T_{ex}^{M_2}$. Notăm acest interval cu T_x :

$$T_x \geq T_{ex}^{M_2} \quad (5-42)$$

Pe de altă parte, conform Teoremei 5-4, funcția de mapare este periodică, de perioadă D . Prin urmare, T_x nu poate fi mai mare decât perioada funcției $\Delta_{M_2/M_1}(\tau)$: $T_x \leq D$.

Conform aceleiași teoreme și a Corolarului 5-1, în cadrul fiecărei perioade a funcției $\Delta_{M_2/M_1}(\tau)$ este mapată câte o execuție a lui M_1 . Cu alte cuvinte, în cadrul fiecărei perioade D , funcția de mapare are valoarea "1" pentru $T_{ex}^{M_1}$ instanțe consecutive ale lui τ . Rezultă că:

$$T_x \leq D - T_{ex}^{M_1} \quad (5-43)$$

Din (5-42) și (5-43) rezultă că relația (5-41) din enunț se verifică. □

Corolarul 5-3 exprimă practic, *condițiile de necesitate și suficiență* ale planficării non-preemptive pentru un set de două (și numai două!) ModX-uri independente, cu execuție fixă (FENP):

"Condiția necesară și suficientă pentru ca un set de două ModX-uri independente, cu execuție fixă, să poată fi planificat în context non-preemptiv este ca suma duratelor lor de execuție să nu depășească valoarea celui mai mare divizor comun al perioadelor acestora."

În cazul în care setul de FModX-uri îndeplinește condițiile de necesitate și suficiență, se poate avansa la stabilirea unei planficării fezabile, constând practic din determinarea intervalelor de start $T_{st}^{M_i}$ ale FModX-urilor, pe baza funcției de mapare $\Delta_{M_2/M_1}(\tau)$.

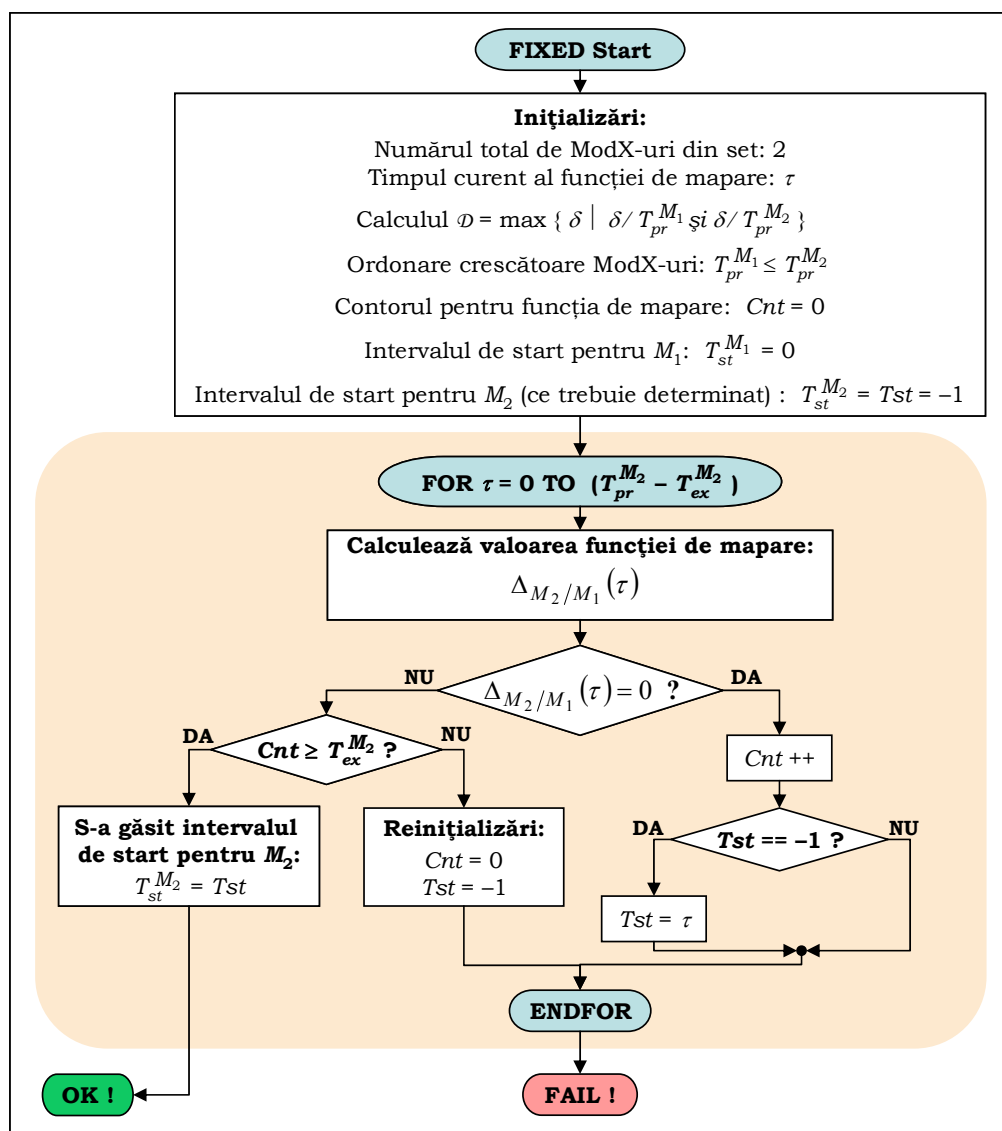


Figura 5-18. Algoritmul de planificare non-preemptivă a unui set de două FModX-uri

Algoritmul de planificare non-preemptivă a două FModX-uri independente (Figura 5-18), pornește de la ipoteza că intervalul de start al lui M_1 este cunoscut apriori (în cazul nostru, pentru simplitate, $T_{st}^{M_1} = 0$) și urmărește determinarea lui $T_{st}^{M_2}$. Pentru aceasta, utilizează o variabilă care contorizează numărul de valori nule consecutive ale funcției de mapare $\Delta_{M_2/M_1}(\tau)$.

În cazul setului de FModX-uri din Exemplul 5-11, algoritmul va obține $T_{st}^{M_2} = 3$, rezultând într-o planificare fezabilă a setului (alta decât cea ilustrată în Figura 5-17). Exemplul următor prezintă toate procedurile implicate în analiza offline a planificabilității setului de FModX-uri considerat.

Exemplul 5-12.

Considerăm un set $M = \{M_1, M_2\}$, compus din două FModX-uri independente, cu următoarele caracteristici temporale:

$$M_1 : \begin{cases} T_{pr}^{M_1} = 30 \\ T_{ex}^{M_1} = 5 \end{cases}; \quad M_2 : \begin{cases} T_{pr}^{M_2} = 40 \\ T_{ex}^{M_2} = 4 \end{cases}$$

Analiza offline a setului M presupune un interval de start nul pentru M_1 și urmărește determinarea intervalului de start pentru M_2 , în condițiile în care setul este fezabil.

Conform (5-31) și (5-32), cel mai mare divizor comun, respectiv, cel mai mic multiplu comun al perioadelor FModX-urilor, sunt:

$$\begin{cases} \mathcal{D} = \text{GCD}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \text{GCD}\{30, 40\} = 10 \\ \mathcal{M} = \text{LCM}\{T_{pr}^{M_1}, T_{pr}^{M_2}\} = \text{LCM}\{30, 40\} = 120 \end{cases}$$

Primul pas constă din aplicarea condiției de necesitate și suficiență a planificabilității non-preemptive a setului M , conform (5-41):

$$T_{ex}^{M_1} + T_{ex}^{M_2} = 9 < \mathcal{D} = 10.$$

Verificarea condiției implică existența unei planificări fezabile pentru FModX-urile setului M .

Aplicarea algoritmului de planificare non-preemptivă a FModX-urilor din setul M constă din calculul valorilor funcției de mapare $\Delta_{M_2/M_1}(\tau)$, pentru $\tau \in \{0, 1, \dots, T_{pr}^{M_2} - 1\} = \{0, 1, \dots, 39\}$ și contorizarea numărului de instanțe de timp τ pentru care $\Delta_{M_2/M_1}(\tau) = 0$. Se poate observa și din Figura 5-19 că funcția

$\Delta_{M_2/M_1}(\tau)$ este periodică, de perioadă $D = 10$, și conține $\nu_{M_2/M_1} = \frac{T_{pr}^{M_2}}{D} = 4$ perioade în care se mapează câte o execuție a lui M_1 .

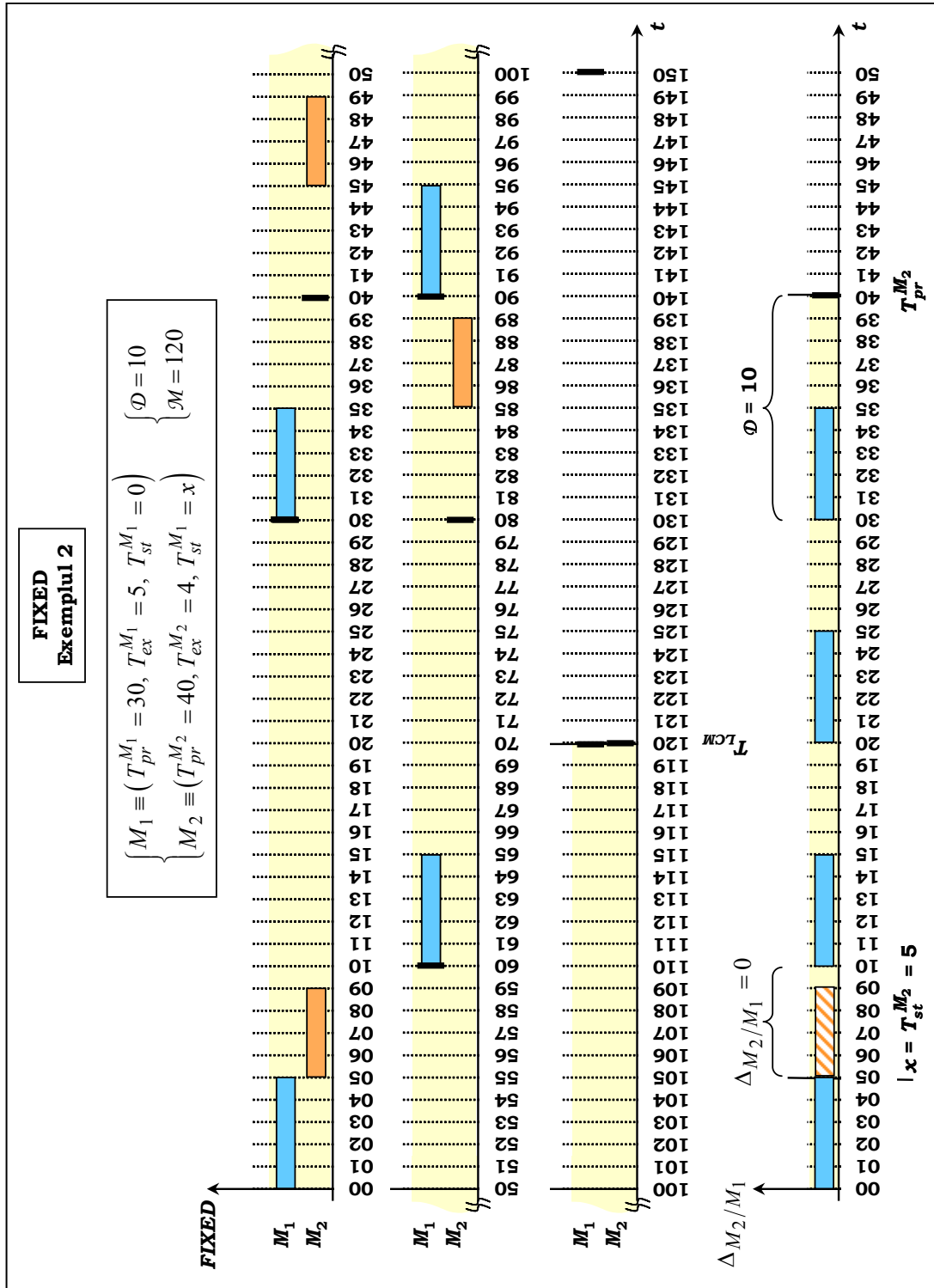


Figura 5-19. Analiza offline a planificării setului M

Cel mai lung interval de timp contiguu pentru care $\Delta_{M_2/M_1}(\tau) = 0$ are valoarea $T_x = 5$, iar prima instanță de timp care definește acest interval este $t_0 = 5$. Mai mult, $T_x = 5 > T_{ex}^{M_2} = 4$. De aici rezultă că, pentru $T_{st}^{M_2} = t_0 = 5$, vom avea o planificare fezabilă a setului \mathbf{M} (vezi Figura 5-19).

Ca observație, menționăm că încărcarea procesor a setului \mathbf{M} din acest exemplu are valoarea:

$$U^{\mathbf{M}} = \sum_{i=1}^2 \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}} = 0.266 < 1.$$

□

5.4.4 Planificarea unui set cu un număr oarecare de FModX-uri

Generalizând principiile de bază prezentate și demonstrate în paragrafele anterioare, se poate dezvolta analiza planificabilității non-preemptive a unui set ce conține un număr oarecare (mai mare ca 2) de ModX-uri independente, cu execuție fixă.

Teorema 5-6.

Fie $\mathbf{M} = \{M_1, M_2, \dots, M_n\}$, un set de n FModX-uri independente, ordonate crescător după perioadă:

$$\begin{cases} M_1 \equiv (T_{pr}^{M_1}, T_{ex}^{M_1}, T_{st}^{M_1}) \\ M_2 \equiv (T_{pr}^{M_2}, T_{ex}^{M_2}, T_{st}^{M_2}) \\ \vdots \\ M_n \equiv (T_{pr}^{M_n}, T_{ex}^{M_n}, T_{st}^{M_n}) \end{cases}, \text{ cu } T_{pr}^{M_i} \leq T_{pr}^{M_j} \text{ pentru } i < j.$$

Setul \mathbf{M} este planificabil în context non-preemptiv, dacă și numai dacă:

$$\forall M_j \in \mathbf{M} . 1 < j \leq n, \exists t_j \in \left\{ 0, 1, \dots, T_{pr}^{M_j} - T_{ex}^{M_j} \right\}, \text{ astfel incat :}$$

$$\bigcup_{\tau=t_0}^{t_j+T_{ex}^{M_j}-1} \bigcup_{i=1}^{j-1} \Delta_{M_j/M_i}(\tau) = 0 \quad (5-44)$$

Demonstrație

Teorema afirmă că setul \mathbf{M} este planificabil dacă și numai dacă orice FModX $M_j \in \mathbf{M}$ ($j > 1$) poate fi planificat împreună cu celelalte FModX-uri anterioare acestuia din punct de vedere al ordonării după perioadă (FModX-urile cu indicii mai mici decât j).

Termenul $\bigcup_{i=1}^{j-1} \Delta_{M_j/M_i}(\tau)$ din (5-44) reprezintă maparea în perioada lui M_j a

tuturor FModX-urilor din \mathbf{M} , cu perioade mai mici sau egale cu $T_{pr}^{M_j}$. Prin urmare, teorema susține că dacă această mapare prevede existența unui interval de timp rămas neocupat în care poate avea loc execuția lui M_j , atunci acest FModX poate fi planificat. Dacă acest lucru se verifică pentru toate M_j din \mathbf{M} (cu $j > 1$), atunci setul \mathbf{M} este fezabil din punct de vedere al planificării cu execuție fixă (FENP), și reciproc.

Teorema derivă direct din Teorema 5-5, ca generalizare a acesteia pentru un set \mathbf{M} compus din mai mult de două FModX-uri. Demonstrația este similară cu cea a Teoremei 5-5, considerând cazul $j = n$, adică a FModX-ului cu cea mai mare perioadă din set. Dacă M_n poate fi planificat împreună cu toate celelalte FModX-uri din \mathbf{M} , rezultă că setul este fezabil, și reciproc. \square

Corolarul 5-4.

Dacă setul $\mathbf{M} = \{M_1, M_2, \dots, M_n\}$ este planificabil conform Teoremei 5-6:

$\forall M_j \in \mathbf{M} . 1 < j \leq n , \exists t_j \in \left\{ 0, 1, \dots, T_{pr}^{M_j} - T_{ex}^{M_j} \right\}$, astfel încât :

$$\bigcup_{\tau=t_0}^{t_j+T_{ex}^{M_j}-1} \bigcup_{i=1}^{j-1} \Delta_{M_j/M_i}(\tau) = 0$$

atunci: $T_{st}^{M_j} = t_j$

Demonstrație

Demonstrația corolarului derivă direct din demonstrația Teoremei 5-6. \square

Corolarul 5-5.

Dacă setul $\mathbf{M} = \{M_1, M_2, \dots, M_n\}$ de FModX-uri este planificabil, atunci:

$$T_{ex}^{M_i} + T_{ex}^{M_j} \leq \mathcal{D}_{i,j} = \text{GCD} \left\{ T_{pr}^{M_i}, T_{pr}^{M_j} \right\}, \forall i, j = 1, 2, \dots, n \quad (5-45)$$

Demonstrație

Demonstrația corolarului derivă din cea a Corolarului 5-3, care afirmă că dacă există o pereche $\{M_i, M_j\} \subset \mathbf{M}$, care nu verifică relația (5-45), atunci cele două FModX-uri nu pot fi planificate împreună, rezultând că setul \mathbf{M} care le conține nu este fezabil (conform Teoremei 5-6).

Presupunerea făcută duce însă la contradicție cu ipoteza corolarului. \square

Corolarul 5-5 exprimă *condiția de necesitate* a planificării non-preemptive pentru un set de n FModX-uri cu execuție fixă (*FENP*):

"*Condiția necesară pentru ca un set de n FModX-uri independente să poată fi planificat în context non-preemptiv este ca suma duratelor de execuție a oricăror două FModX-uri din set să nu depășească valoarea celui mai mare divizor comun al perioadelor acestora.*"

Observație

Se poate observa că, spre deosebire de cazul particular al seturilor de două FModX-uri, pentru cazul general ($n > 2$), Corolarul 5-5 furnizează *doar condiția de necesitate*, nu și de suficiență.

Teorema 5-6, împreună cu Corolarul 5-4 și 5-5, facilitează abordarea algoritmică a analizei planificabilității non-preemptive *FENP* a unui set $\mathbf{M} = \{M_1, M_2, \dots, M_n\}$ de FModX-uri ordonate crescător după perioadă. Un astfel de algoritm (care reprezintă practic o generalizare a celui pentru două FModX-uri, ilustrat în Figura 5-18) fixează la început o valoare pentru intervalul de start al lui M_1 , $T_{st}^{M_1}$. Dacă nu există alte restricții, $T_{st}^{M_1} = 0$. În continuare se analizează pe rând planificabilitatea celorlalte FModX-uri din \mathbf{M} , cu ajutorul relației (5-44). Dacă relația se verifică pentru $M_j \in \mathbf{M}$, atunci se stabilește $T_{st}^{M_j} = t_j$, conform Corolarului 5-4. Algoritmul se termină cu succes dacă toate FModX-urile din \mathbf{M} au putut fi planificate.

Pe lângă evaluarea fezabilității setului \mathbf{M} , un alt rezultat al algoritmului de analiză offline *FENP* constă în determinarea setului $\left\{ T_{st}^{M_i} \mid M_i \in \mathbf{M}, i = 1, 2, \dots, n \right\}$, adică a setului intervalelor de start ale fiecărui FModX din \mathbf{M} , în cadrul perioadelor corespunzătoare.

Momentul efectiv de start al unui FModX M_i , corespunzător ciclului k de execuție a acestuia, poate fi calculat cu relația:

$$t_{st,k}^{M_i} = (k-1)T_{pr}^{M_i} + T_{st}^{M_i}, \quad k = 1, 2, \dots \quad (5-46)$$

5.5 Concluzii

În capitolul de față am abordat problema planificării non-preemptive din perspectiva modelului de task TR strict – ModX-ul, prezentat în detaliu în capitolul anterior.

A fost arătat faptul că introducerea etapei de analiză offline a planificabilității setului de task-uri, cu confirmarea (sau nu) a fezabilității acestuia, evită problemele de predictibilitate legate de timpii de factură non-polinomială necesari efectuării acestei analize în timpul operării efective a sistemului (analiza online).

Studiul planificării non-preemptive a seturilor de ModX-uri a fost efectuat considerându-se diferite cazuri în care s-au aplicat diverse restricții, plecând de la situațiile mai simple (și mai aproape de ideal) și avansându-se către situații mai complexe și mai realiste.

Pentru seturile de ModX-uri independente au fost introduși și studiați doi dintre cei mai eficienți algoritmi de planificare non-preemptivă: *MLFNP* (Minimum Laxity First Non-Preemptive) și *EDFNP* (Earliest Deadline First Non-Preemptive). Prin studiile și simulările efectuate și exemplificate, am ajuns la concluzia că *EDFNP* poate rezolva cu succes planificarea mai multor seturi de ModX-uri independente, comparativ cu *MLFNP*, fiind astfel mai eficient (așa cum se va vedea și în capitolul următor). În același context, a fost abordată problema evaluării fezabilității setului de task-uri utilizând condiții de necesitate și/sau suficiență. Au fost studiate trei condiții de necesitate a planificabilității non-preemptive a seturilor de ModX-uri. Am demonstrat că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele alinate la momentul inițial $t_0 = 0$), condiția a doua de necesitate enunțată în [Jeffay 91] nu este aplicabilă.

În continuare, am considerat cazurile, mai apropiate de realitate, ale sistemelor TR stricte ce execută, pe lângă setul de ModX-uri ale aplicației, și un task special – planificatorul online, ce utilizează o Tabelă de Dispatch de dimensiuni fixe, bine determinate, pentru planificarea ModX-urilor. Pentru abordarea planificării online în cicluri cu număr constant de execuții, au fost concepute și demonstrate condițiile de necesitate ale planificabilității sistemului, ținând cont și de caracteristicile de comportare temporală a planificatorului online. Algoritmul de planificare non-preemptivă pentru astfel de sisteme de task-uri (*CEC-NPOS*) a fost prezentat și verificat prin simulări și exemple. Apoi a fost studiată în detaliu abordarea planificării online în cicluri periodice (*PC-NPOS*), demonstrându-se relațiile ce se aplică parametrilor esențiali ai planificării (perioada și durata de execuție a planificatorului online, și dimensiunea maximă a Tabelei de Dispatch).

În fine, am introdus și abordat problema modelării și planificării seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora. A fost conceput și studiat modelul matematic al ModX-urilor cu execuție fixă (FModX-uri). A fost analizată planificarea non-preemptivă a seturilor compuse din două FModX-uri independente, prin introducerea și demonstrarea condițiilor de necesitate și suficiență ale planificabilității, și prin conceperea algoritmului de planificare ce utilizează funcția de mapare (*FENP*). Modul de operare al algoritmului a fost exemplificat pe diverse cazuri. Problema a fost apoi generalizată pentru un set cu un număr oarecare (mai mare ca 2) de FModX-uri.

Preocupările curente și de viitor imediat din cadrul activității noastre de cercetare în domeniu cuprind următoarele subiecte legate de planificarea non-preemptivă a seturilor de ModX-uri:

- Planificarea seturilor de ModX-uri cu dependențe de program (cu alte cuvinte, planificarea unei aplicații TR complete);
- Integrarea tuturor cazurilor de planificare non-preemptivă și conceperea unui model unificat, capabil a fi utilizat în cât mai multe aplicații reale.

6 Evaluarea performanțelor algoritmilor de planificare non-preemptivă

În cadrul capitolului anterior au fost discutați în detaliu o serie de algoritmi de planificare non-preemptivă existenți în literatura de specialitate, cu particularizările impuse de utilizarea modelului de task TR strict introdus de abordarea noastră: ModX-ul. De asemenea, în Capitolul 5 au fost introduși și studiați noi algoritmi de planificare non-preemptivă, pentru rezolvarea diverselor cazuri reale, ce se abat de la modelul abstract, general, utilizat în publicațiile ce tratează sistemele TR. De exemplu, pentru tratarea planificării online, au fost introduși algoritmi *CEC-NPOS* și *PC-NPOS*, iar pentru planificarea seturilor de ModX-uri cu execuție fixă în cadrul perioadei (FModX-uri) a fost definit algoritmul *FENP*.

Datorită importanței planificării în cadrul STR stricte și a impactului pe care aceștia îl au asupra performanței întregului sistem, o analiză amănunțită a performanțelor acestor algoritmi este obligatorie [Lehoczky 89, Panzieri 93a, Panzieri 93b, Ghosh 94]. Pentru cazurile ce pot fi tratate de mai mulți algoritmi de planificare, studiul comparativ al acestora permite luarea unor decizii de selectare a celui mai performant.

În paragrafele următoare sunt prezentate evaluările de performanță ale următorilor algoritmi de planificare non-preemptivă:

1. *EDFNP* și *MLFNP*: studiu comparativ pentru evaluarea planificării non-preemptive a seturilor de ModX-uri simple, independente.
2. *CEC-NPOS* și *PC-NPOS*: studiu comparativ pentru evaluarea planificării online non-preemptive a seturilor de ModX-uri simple, independente, utilizând o Tabelă de Dispatch de dimensiune maximă prestabilită.
3. *FENP*: analiza performanțelor planificării non-preemptive a seturilor de FModX-uri independente.

6.1 Introducere: metodele utilizate pentru analiza performanțelor

Evaluarea performanțelor algoritmilor de planificare propuși se bazează pe determinarea următorilor parametri generali:

- Succesul sau insuccesul condițiilor de planificabilitate care se aplică algoritmului studiat;
- Succesul sau insuccesul planificării efectuate cu algoritmul studiat asupra unui set dat de ModX-uri. Fezabilitatea planificării se analizează aplicând algoritmul respectiv asupra setului de ModX-uri pentru intervalul temporal $[0, T_{LCM})$, unde T_{LCM} reprezintă cel mai mic multiplu comun al perioadelor ModX-urilor din setul analizat (conform Lemei 5-1);

- Timpul efectiv necesar execuției algoritmului pentru terminarea întregii planificări a unui set de ModX-uri, pe un calculator PC tip Pentium III. Acest timp nu este un parametru esențial în ceea ce privește performanțele algoritmilor de planificare studiați, pentru că aici este tratată analiza de tip offline a seturilor de ModX-uri aparținând sistemului și aplicațiilor. Comportarea algoritmilor în timpul operării sistemului (la "run-time") vizează corectitudinea operării propriu-zise a algoritmului de planificare, durata sa de execuție pentru un singur ciclu de planificare, și (dacă e cazul) perioada acestuia. Acești parametri se determină după implementarea planificatorului online pe platforma țintă, după ce a fost selectat algoritmul potrivit (sau combinația de algoritmi).

Pentru analiza performanțelor algoritmilor de planificare studiați, se efectuează sesiuni extinse de teste asupra unor seturi de ModX-uri generate aleator în ceea ce privește parametrii temporali ai ModX-urilor. Astfel, programele de evaluare vor primi o configurație de operare, ce specifică printre altele:

- $TotMX (= n)$: numărul total de ModX-uri ce compun seturile analizate;
- $IntervTimp$: intervalul temporal în care se vor genera perioadele ModX-urilor din set;
- $GenPer$: tipul de distribuție utilizată la generarea aleatoare a valorilor perioadelor ModX-urilor;
- $[MinPU, MaxPU]$: intervalul de valori ce încadrează factorul de utilizare procesor al setului M de ModX-uri generate, $PU = U^M$. Acest parametru influențează duratele de execuție ale ModX-urilor din M ;
- $TschedMax = T_{LCM}$: intervalul temporal maxim în care se poate încadra limita de planificare;
- $NrTeste$: numărul total de teste de planificare (numărul total de seturi generate aleator pentru analiza fezabilității cu un anumit algoritm).

Exemplul 6-1.

Considerăm pentru programul de evaluare, configurația dată în Tabelul 6-1.

Tabelul 6-1. Exemplu de configurație pentru programul de evaluare

Parametru configurație	Valoare
TotMX	10
IntervTimp	310
GenPer	2
[MinPU, MaxPU]	[0.8, 0.9]
TschedMax	100000000
NrTeste	1000

Ca urmare, evaluarea algoritmului de planificare studiat se va face cu 1000 de teste individuale, în care se generează aleator câte un set de 10 ModX-uri. Parametrii temporali ai fiecărui ModX din cele 1000 de seturi ce vor fi generate,

adică perioada și durata de execuție, vor avea următoarele caracteristici: perioada se va încadra în intervalul [10, 310], fiind obținută prin aplicarea unei proceduri de generare aleatoare cu distribuție normală (Gaussiană) pentru acest interval, iar durata de execuție va respecta condiția ca utilizarea procesor a întregului set să fie cuprinsă în intervalul [0.8, 0.9]. De asemenea, cel mai mic multiplu comun al perioadelor ModX-urilor din fiecare set va fi mai mic decât 100000000.

Duratele de execuție a ModX-urilor sunt obținute de asemenea în manieră aleatoare, verificându-se de fiecare dată utilizarea procesor a întregului set:

$$U^M = \sum_{i=1}^n \frac{T_{ex}^{M_i}}{T_{pr}^{M_i}}$$

Tabelul 6-2 prezintă un exemplu de set de ModX-uri generat pe baza parametrilor de configurare din Tabelul 6-1.

Tabelul 6-2. Exemplu de set de ModX-uri generat

Element	Valoare	
	Perioada	Durata execuție
M_1	63	1
M_2	95	12
M_3	126	15
M_4	133	16
M_5	133	9
M_6	141	17
M_7	172	10
M_8	210	18
M_9	215	22
M_{10}	235	1
U^M	0.820208	
T_{LCM}	48382740	

□

Pentru generarea aleatoare a perioadelor ModX-urilor au fost utilizate două tipuri distincte de distribuție: (a) distribuția uniformă și (b) distribuția normală (Gaussiană). Procedura de generare uniform aleatoare a perioadelor ModX-urilor împarte intervalul fixat pentru acestea la configurare, [10, IntervTimp], în n (TotMX) subintervale de dimensiune egală, din care este apoi extrasă câte o valoare. Figura 6-1 prezintă histogrammele și liniile de tendință a evoluției ("trend lines") pentru patru seturi de câte 50 de ModX-uri, cu perioade generate uniform aleator în intervalul [10, 310].

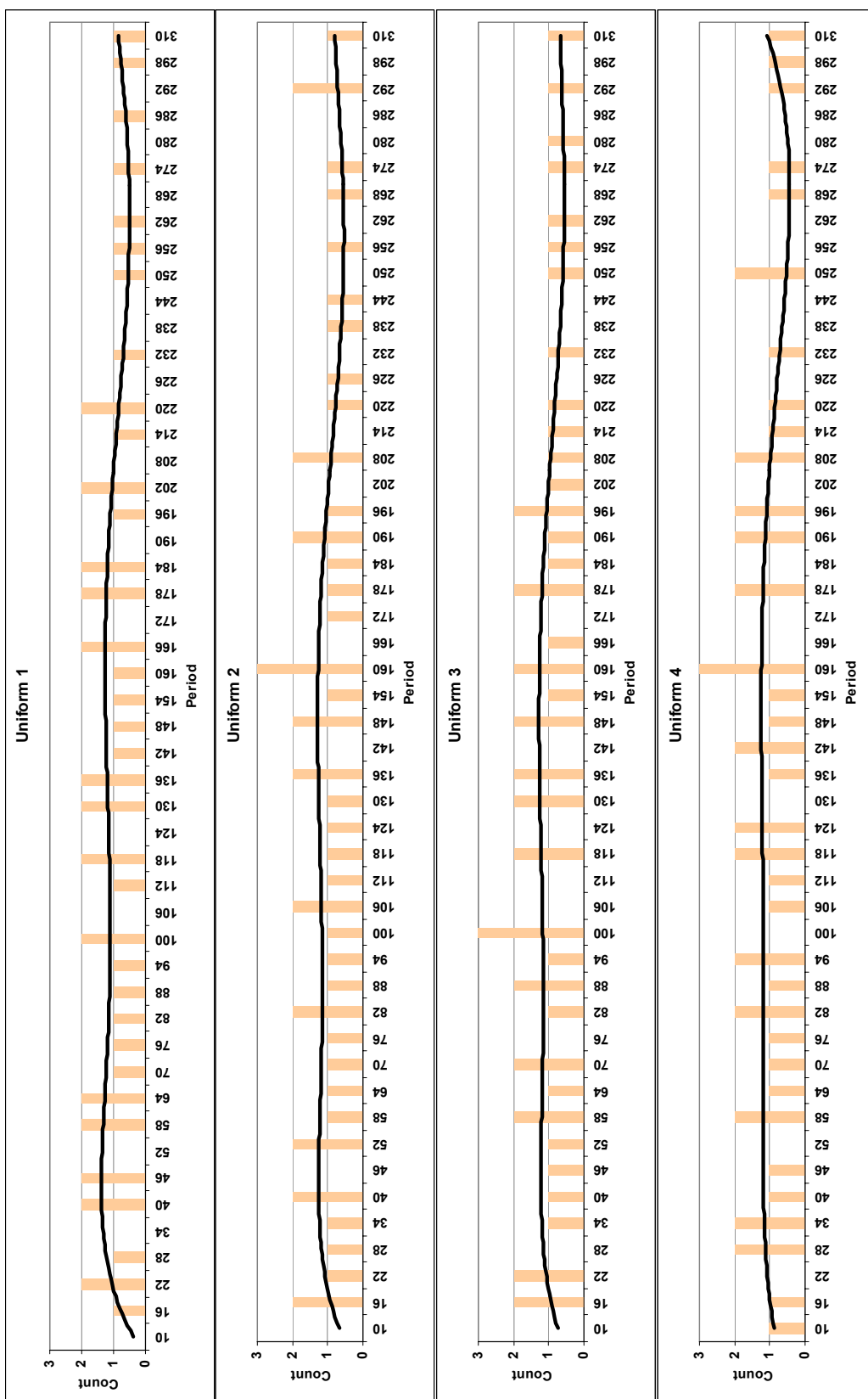


Figura 6-1. Histogramele a patru seturi de ModX-uri generate cu distribuție uniformă

Generarea de numere aleatoare cu distribuție normală (Gaussiană) se bazează pe funcția de densitate normală [Knuth 98, Ulrich 98]:

$$N(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad (6-1)$$

unde μ reprezintă media de distribuție iar σ este deviația standard. Funcția de densitate normală standard se obține pentru o medie nulă și o deviație standard unitară, și reprezintă probabilitatea de generare a valorilor numerice din intervalul $[-3, 3]$ (vezi Figura 6-2):

$$N(x, 0, 1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \quad (6-2)$$

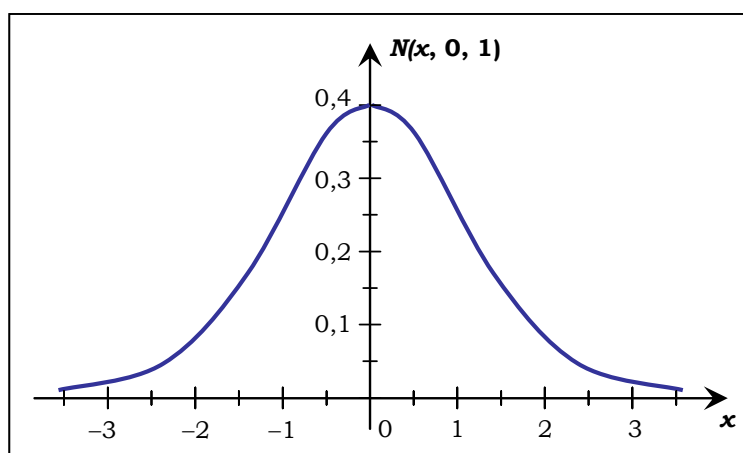


Figura 6-2. Graficul distribuției normale standard, $N(x, 0, 1)$

Generarea numerică a valorilor cu distribuție normală standard utilizează forma polară a transformării Box-Muller [Carter 04], pe baza următorului algoritm:

Programul 6-1. Procedura de generare a valorilor cu distribuție normală standard

```
float x1, x2, w, y1, y2;
do {
    x1 = 2.0 * rand() - 1.0;
    x2 = 2.0 * rand() - 1.0;
    w = x1 * x1 + x2 * x2;
} while (w >= 1.0);
w = sqrt((-2.0 * ln(w)) / w);
y1 = x1 * w;
y2 = x2 * w;
```

Cu algoritmul de mai sus se obține o pereche de valori $\{y_1, y_2\}$ normal distribuite în intervalul $[-1, 1]$. Funcția `rand()` se găsește în bibliotecile standard ANSI C, pentru generarea de numere aleatoare uniforme distribuite în intervalul $[-1, 1]$.

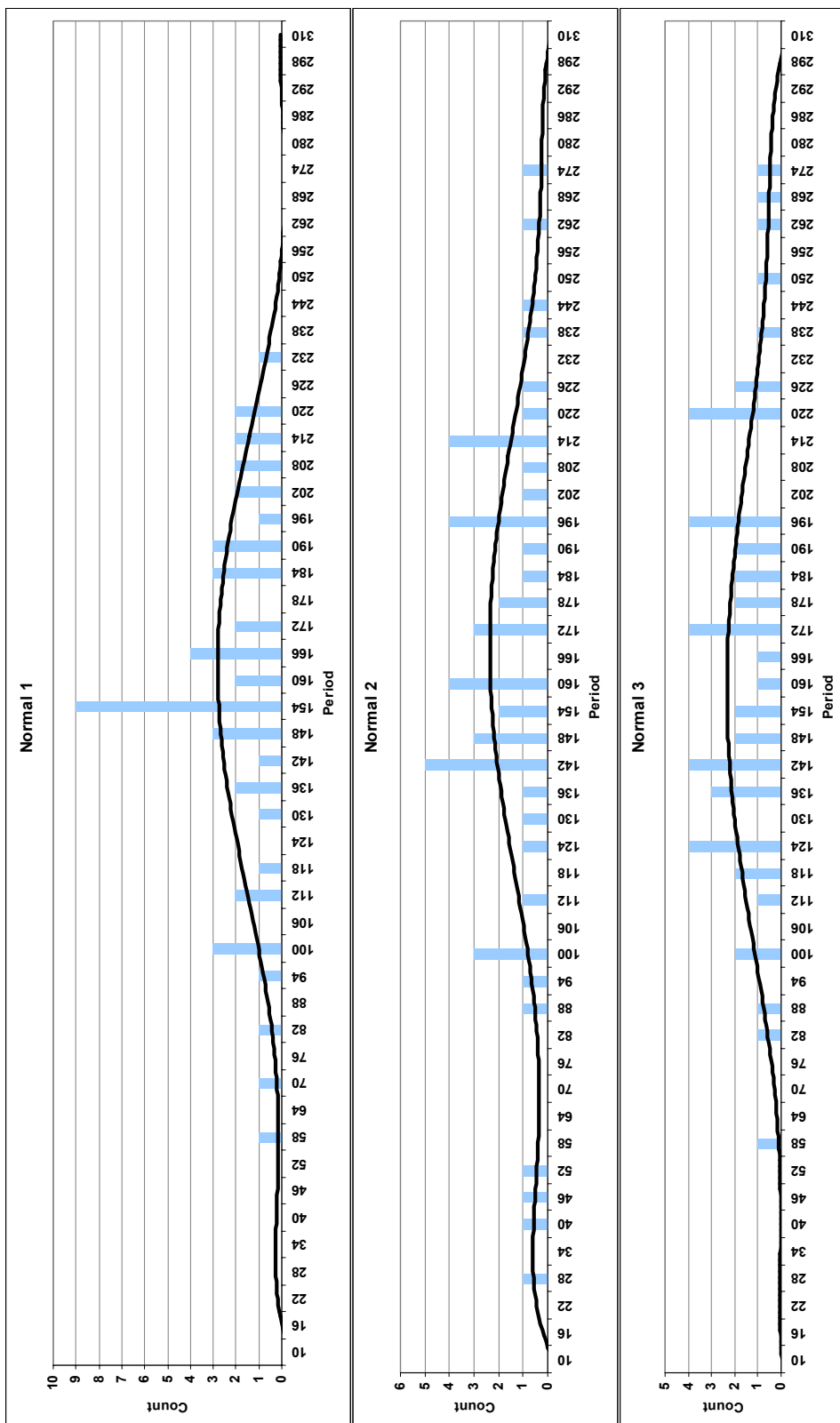


Figura 6-3. Histogramele a trei seturi de ModX-uri generate cu distribuție normală

În final, pentru generarea perioadelor ModX-urilor în intervalul specificat la configurarea programului, $[10, \text{IntervTimp}]$, se va efectua transformarea de la (6-2) la (6-1):

$$T_{pr}^{M_i} = \mu + \sigma \cdot y_1, \text{ unde: } \begin{cases} \mu = \frac{\text{IntervTimp} - 10}{2} + 10 \\ \sigma = \frac{\text{IntervTimp} - 10}{6} \end{cases} \quad (6-3)$$

Figura 6-3 prezintă histogramele și liniile de tendință a evoluției ("trend lines") pentru trei seturi de câte 50 de ModX-uri, cu perioade generate uniform aleator în intervalul $[10, 310]$. Parametrii procedurii de determinare a perioadelor corespunzătoare configurației din Figura 6-3 sunt: $\mu = 160$ și $\sigma = 50$.

Evaluarea performanțelor algoritmilor de planificare pentru modelul de STR propus și discutat în lucrarea de față presupune abordarea a două perspective: (i) selectarea unui algoritm cu performanțe optime în ceea ce privește planificarea a cât mai multe seturi de ModX-uri posibil și pentru încărcări procesor cât mai mari, și (ii) evaluarea performanțelor de operare online ale aceluiași algoritm. În cel de al doilea caz, parametrii vizați sunt în legătură directă cu modul de implementare a codului planificatorului online: durata maximă de execuție (viteza) și dimensiunea codului. În Secțiunea III sunt discutate în detaliu aceste aspecte.

Un parametru cheie de performanță în ceea ce privește analiza optimalității algoritmilor de planificare, este așa-numitul *raport de planificabilitate* (*SR*, *Schedulability Ratio*). Raportul de planificabilitate va fi evaluat în paragrafele următoare pentru diferiți algoritmi de planificare și pentru diferite încărcări procesor implicate de seturi de ModX-uri generate aleator, cu tipurile de distribuție prezentate anterior.

6.2 Evaluarea comparativă a algoritmilor EDFNP și MLFNP

Pentru evaluarea comparativă a performanțelor (optimalității) algoritmilor de planificare non-preemptivă *EDFNP* și *MLFNP* discutați în Capitolul 5, a fost dezvoltat un program specific de testare, "Evaluation1" și un program de colectare a rezultatelor, "DataAdd", care are și rolul de eliminare a rezultatelor duplicat.

Cu ajutorul programului "Evaluation1" au fost efectuate peste 24000 de teste, utilizând cele 12 stații de calcul ale rețelei de calculatoare tip PC Pentium III a laboratorului DSPLabs (<http://dsplabs.utt.ro>). Testele desfășurate au urmărit determinarea raportului de planificabilitate *SR* în funcție de următorii parametri suplimentari (combinații ale acestora):

- Numărul total de ModX-uri dintr-un set: $\text{TotMX} = 10, 15, 20$;
- Utilizarea procesor (*PU*) implicată de seturile de ModX-uri, încadrată în intervalul: $[\text{MinPU}, \text{MaxPU}] = [0.6, 0.7], [0.7, 0.8], [0.8, 0.9], [0.9, 1.0]$;
- Tipul de distribuție utilizat la generarea aleatoare a perioadelor ModX-urilor: distribuție uniformă, și distribuție normală.

Intervalul ce încadrează valorile perioadelor ModX-urilor a fost considerat constant: $\text{IntervTime} = 310$, valoarea minimă pentru perioade fiind 10. Cum valoarea minimă a duratelor de execuție a ModX-urilor generate este 1, rezultă un raport maxim de 1/310 între durata de execuție și perioada oricărui ModX. Seturile de ModX-uri utilizate în operațiile de evaluare pot fi puse în corespondență cu cazurile realiste din aplicațiile TR, prin aplicarea unor factori de scalare pentru parametrii temporali respectivi. Astfel, de exemplu, un ModX

$$M_i \equiv \left\{ T_{pr}^{M_i} = 210, T_{ex}^{M_i} = 3 \right\}$$

ce este utilizat în programul de evaluare, poate fi pus în relație directă cu ModX-ul $M'_i \equiv \left\{ 100 T_{pr}^{M_i} = 21000 \mu\text{s}, 100 T_{ex}^{M_i} = 300 \mu\text{s} \right\}$ al

unei aplicații TR reale, prin scalarea cu $100 \mu\text{s}$ a parametrilor temporali ai lui M_i .

Pe lângă evaluarea celor doi algoritmi studiați, *EDFNP* și *MLFNP*, programul "Evaluation1" aplică și testarea condiției de necesitate a planificării **CN2**) (vezi Capitolul 5), introdusă de [Jeffay 91]. Așa cum s-a demonstrat în capitolul anterior, această condiție de necesitate nu este valabilă pentru toate seturile de ModX-uri, deoarece modelul de seturi de task-uri TR utilizat aici este un caz particular față de modelul considerat în lucrarea lui Jeffay.

În Figura 6-4, Figura 6-5 și Figura 6-6 sunt prezentate graficele rezultate ca urmare a testelor de planificabilitate aplicate cu programul "Evaluation1" și procesate apoi cu "DataAdd". Din cele trei figuri se pot desprinde următoarele elemente:

- *EDFNP* se comportă mai bine ca *MLFNP* în sensul unei rate de succes a planificării mai mari, pentru toate cazurile considerate: orice total de ModX-uri din set, orice interval de încărcare procesor și orice tip de distribuție utilizată la generarea parametrilor temporali ai ModX-urilor;
- *EDFNP* reușește planificarea multor seturi de ModX-uri pentru care condiția **CN2**) (testul "Jeffay") dă rezultate negative. Acest lucru se observă mai pregnant pentru seturile cu un număr mai mic de ModX-uri;
- Pentru oricare din cazurile considerate, seturile de ModX-uri care dau rezultate pozitive la testul "Jeffay", pot fi planificate cu algoritmul *EDFNP*. Există însă și seturi particulare ce pot fi planificate cu *EDFNP* și fără a îndeplini condiția "Jeffay". Acest lucru este în concordanță cu discuția din Capitolul 5 referitoare la condiția de planificabilitate **CN2**), unde a fost evidențiată și demonstrată teoretic această problemă;
- Comportarea ambilor algoritmi studiați se îmbunătățește vizibil pe măsură ce numărul de ModX-uri din seturile supuse planificării, crește. Acest lucru se explică prin faptul că, pentru o aceeași încărcare procesor (*PU*), cu cât sunt mai multe ModX-uri într-un set, cu atât scade durata de execuție a acestora. Prin urmare planificarea non-preemptivă are șanse mai mari de reușită pentru mai multe task-uri, ce se execută fiecare într-un interval mai scurt, decât dacă sunt task-uri mai puține dar de durată mai mare;
- Rata de succes a planificării cu cei doi algoritmi scade, așa cum este de așteptat, pe măsură ce crește utilizarea procesor a seturilor de ModX-uri testate;

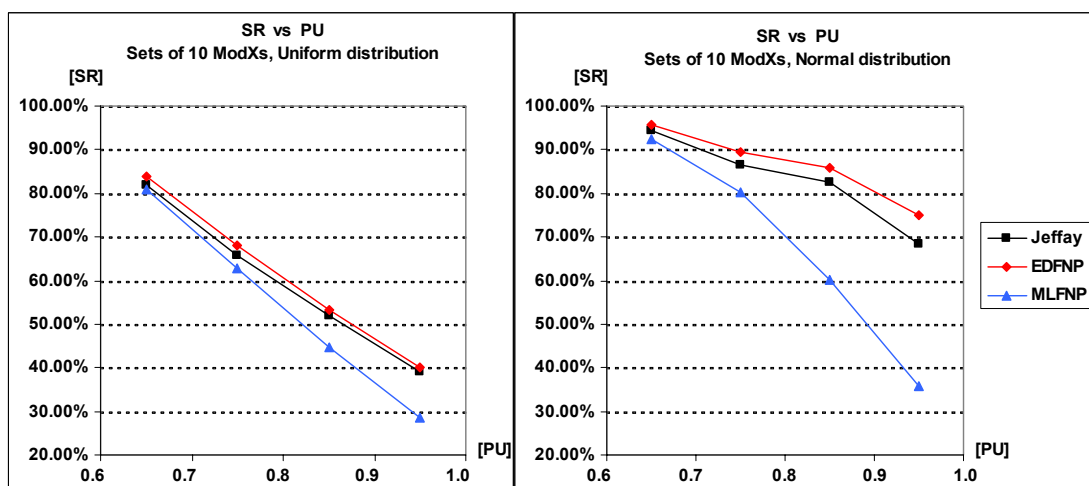


Figura 6-4. SR funcție de PU, pentru seturi de 10 ModX-uri cu distribuție uniformă (stânga) și normală (dreapta)

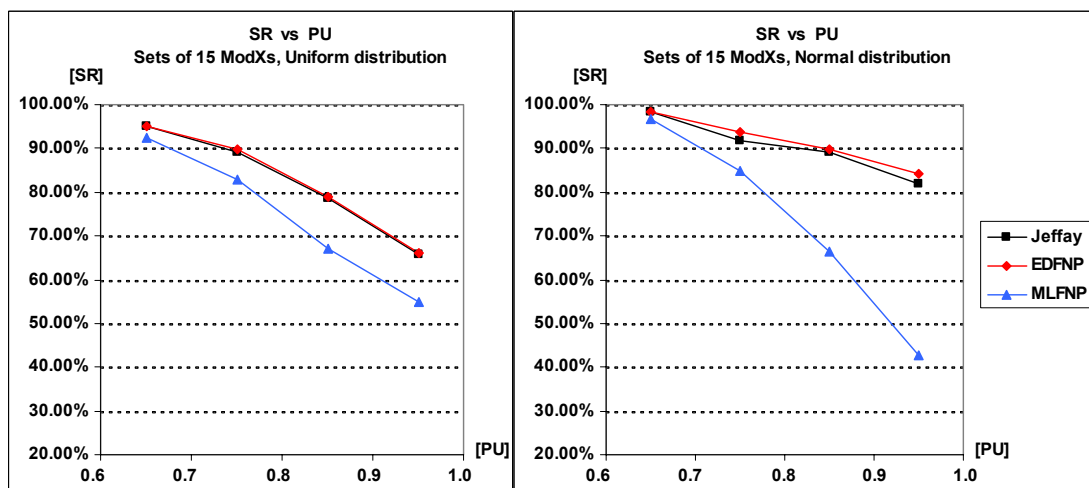


Figura 6-5. SR funcție de PU, pentru seturi de 15 ModX-uri

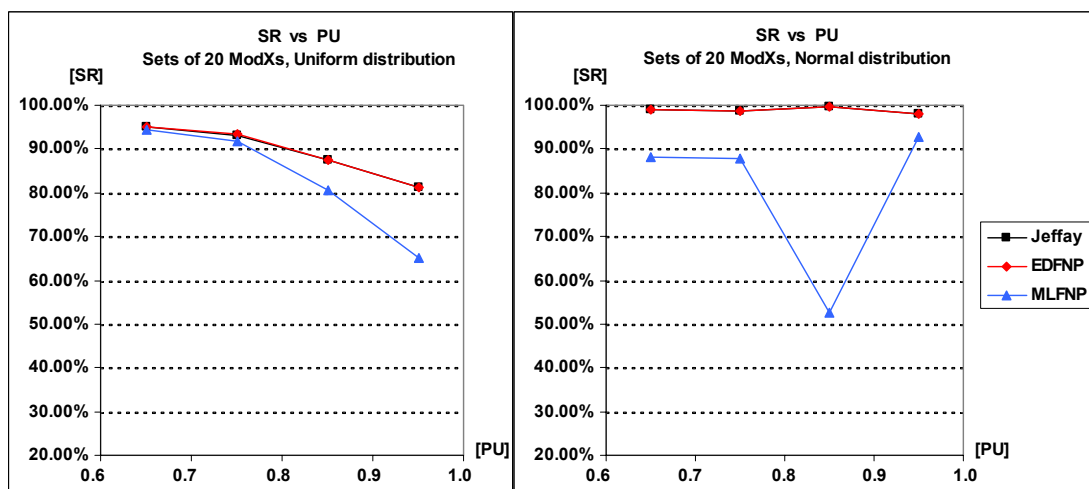


Figura 6-6. SR funcție de PU, pentru seturi de 20 ModX-uri

- În cazul seturilor de ModX-uri generate cu distribuție normală, ambii algoritmi se comportă semnificativ mai bine, comparativ cu seturile generate cu distribuție uniformă. Acest lucru se explică prin faptul că algoritmi de planificare non-preemptivă discutați dau rezultate mai bune pentru seturi de ModX-uri cu perioade având valori relativ apropiate între ele, situate în apropierea mediei intervalului considerat pentru acestea, comparativ cu cazurile în care perioadele iau valori extreme, mult diferite între ele. Distribuția normală a perioadelor în cadrul unui anumit interval corespunde de asemenea multor aplicații TR practice.

În Figura 6-6 se poate observa, cu precădere pentru distribuția normală, o deviere de la tendința generală de comportare a algoritmilor *EDFNP* și *MLFNP* (Figura 6-4 și Figura 6-5). Explicația constă în faptul că în procesul de evaluare a performanțelor cu ajutorul programului "Evaluation 1", a fost impusă o limită pentru valoarea timpului de planificare: $T_{LCM} = T_{\text{schedMax}} = 1000000000$ (pentru seturile de 20 ModX-uri). Ca rezultat, găsirea (în manieră aleatoare) a unui set de 20 de valori pentru perioadele ModX-urilor, astfel încât cel mai mic multiplu comun al acestora să se încadreze în limita impusă, este dificilă și de durată. Cu cât valorile perioadelor sunt mai mari (ca în cazul distribuției normale, Figura 6-6 dreapta, față de distribuția uniformă, Figura 6-6, stânga), cu atât mai dificilă este această problemă.

Limitarea timpului maxim de planificare (T_{LCM}) a fost impusă deoarece în caz contrar, se pot genera cu ușurință seturi de 20 de ModX-uri de exemplu, pentru care T_{LCM} să ajungă la ordinul de mărime 10^{30} , și chiar mai mult. De aici rezultă două probleme:

- (a) Necesitatea operării cu numere întregi foarte mari, care depășesc capacitățile tipurilor de date implementate pe majoritatea arhitecturilor PC de către limbajele de programare. În consecință se impune utilizarea de biblioteci specializate în operarea cu numere cu precizie variabilă [Shoup 02, Allombert 03, Shamus 03].
- (b) Timpul necesar analizei planificabilității seturilor de ModX-uri este direct proporțional cu T_{LCM} . Tabelul 6-3 prezintă timpii de planificare rezultați ca urmare a aplicării programului "Evaluation 1" pentru câteva seturi de ModX-uri. Programul a fost rulat pe o stație PC tip Pentium III, pentru cinci seturi de 18 ModX-uri, cu perioadele generate cu distribuție normală în intervalul [10, 310] astfel încât limita $T_{LCM} \leq 2000000000$. Utilizarea procesor a seturilor a fost aleasă cât mai mică (în intervalul [0.1, 0.2]) pentru ca timpii rezultați să fie maximi pentru perioadele respective ale ModX-urilor.

Cele două probleme de mai sus converg până la urmă tot la dificultăți legate de timpul de procesare. Utilizarea unei biblioteci pentru numere cu precizie variabilă implică o creștere a acestor timpi cu cel puțin un ordin de mărime.

Versiunea utilizată în final pentru programul "Evaluation 1" la efectuarea testelor comparative prezentate în diagramele de mai sus, se bazează pe o implementare Microsoft® VisualStudio™ 6.0 (Visual C++), utilizând tipul de numere întregi reprezentate pe 64 biți: int_{64} . În acest fel se pot efectua operații rapide cu numere întregi de ordinul $9 \cdot 10^{18}$.

Tabelul 6-3. Timpii de analiză a planificării pentru câteva teste cu "Evaluation1"

Valoare T_{LCM}	Timpii de planificare [sec]	
	<i>MLFNP</i>	<i>EDFNP</i>
145044900	476	469
325155600	1060	1052
149189040	483	481
681912000	2214	2212
1730907360	5698	5601
Medieri		
1000000000	3275	3237

6.3 Evaluarea comparativă a algoritmilor de planificare online non-preemptivă

Comportarea algoritmilor de planificare online non-preemptivă introduși și discutați în Subcapitolul 5.3, *CEC-NPOS* și *PC-NPOS*, a fost testată și analizată cu ajutorul unui program special conceput și dezvoltat cu mediul Microsoft® VisualStudio™ 6.0 (Visual C++) pentru platforme PC: "OnlineScd". Principiile de evaluare implementate de "OnlineScd" sunt similare cu cele ale lui "Evaluation 1": efectuarea unui număr prestabilit de teste de planificare pentru algoritmi *CEC-NPOS* și *PC-NPOS*, asupra unor seturi cu număr dat de ModX-uri, ai căror parametri temporali sunt generați în manieră aleatoare după distribuții uniforme sau normale.

Elementele suplimentare ce apar în acest caz sunt ModX-ul sistem de planificare online, M_S , și Tabela de Dispatch a sistemului TR, modelată prin dimensiunea maximă a înregistrărilor sale, $\lambda + 1$.

Procedura generală de operare a programului "OnlineScd" cuprinde următorii pași:

1. Se generează perioadele ModX-urilor normale $M_i \in \mathbf{M} = \{M_1, M_2, \dots, M_n\}$, unde: $n = \text{TotMX}$ este totalul ModX-urilor din \mathbf{M} , perioadele sunt cuprinse în intervalul $[10, \text{IntervTimp}]$ și sunt generate aleator, cu distribuție uniformă sau normală, astfel încât cel mai mic multiplu comun al lor T_{LCM} să nu depășească valoarea T_{schedMax} .
2. Se sortează ModX-urile din setul \mathbf{M} în ordinea crescătoare a perioadelor.
3. Se generează duratele de execuție ale ModX-urilor din \mathbf{M} , astfel încât utilizarea procesor a setului să fie cuprinsă în intervalul: $PU_M \in \{\text{MinPU}, \text{MaxPU}\}$.
4. Se generează durata de execuție a planificatorului online M_S în funcție de perioada și durata de execuție a ModX-ului cu cea mai mică perioadă din set:

$$T_{ex}^{M_S} = 2 \left(T_{pr}^{M_{pr\ min}} - T_{ex}^{M_{pr\ min}} \right) \quad (6-4)$$

La valoarea obținută pentru durata de execuție a M_S , se aplică un factor de scalare (subunitar), specificat la configurarea programului: $Scale_TexMs$.

5. În continuare, în funcție de tipul algoritmului de planificare aplicat, se parcurg următoarele etape:

5.1. Pentru *CEC-NPOS*:

- se verifică și se contorizează condiția de planificabilitate *CEC-NPOS* (Teorema 5-2);
- se aplică și se contorizează algoritmul de planificare, indiferent de rezultatele condiției de necesitate.

5.2. Pentru *PC-NPOS*:

- se determină perioada ModX-ului de planificare online M_S , cu relația (5-26) și procedura descrisă în Subcaptioul 5.3.2;
- se re-sortează ModX-urile din set, împreună cu M_S , după perioade;
- se aplică și se contorizează testul "Jeffay" pentru setul extins de ModX-uri: $M \cup \{M_S\}$;
- se aplică și se contorizează algoritmul de planificare *PC-NPOS*, indiferent de rezultatul testului "Jeffay".

Cu ajutorul programului "OnlineScd" au fost efectuate peste 35000 de teste, utilizând cele 12 stații ale rețelei de calculatoare tip PC Pentium III a laboratorului DSPLabs. Testele desfășurate au urmărit determinarea raportului de planificabilitate *SR* în funcție de următorii parametri suplimentari (combinații ale acestora):

- Numărul total de ModX-uri dintr-un set: $TotMX = 10, 15$;
- Utilizarea procesor (*PU*) implicată de seturile de ModX-uri, încadrată în intervalul: $[MinPU, MaxPU] = [0.5, 0.6], [0.6, 0.7], [0.7, 0.8], [0.8, 0.9]$, și $[0.9, 1.0]$;
- Dimensiunea maximă utilă a Tabelei de Dispatch: $lambda = 100, 200$;
- Factorul de scalare a duratei de execuție a M_S : $Scale_TexMs = 0.3, 0.5, 0.7$.

Intervalul ce încadrează valorile perioadelor ModX-urilor a fost considerat constant: $IntervTimp = 310$, valoarea minimă pentru perioade fiind 10. În Figura 6-7 la Figura 6-12 sunt prezentate diagramele rezultatelor testelor efectuate cu "OnlineScd". Din aceste diagrame rezultă ca evidentă comportarea mult mai bună a algoritmului de planificare online periodică non-preemptivă, *PC-NPOS*, pentru toate cazurile considerate. Pe de altă parte, atunci când durata de execuție a planificatorului M_S se apropie de valoarea maximă admisă de condiția de necesitate **CN3**) (relația (6-4)), de exemplu pentru un factor de scalare de 0.7 (Figura 6-9), rata de succes a algoritmilor online este foarte mică.

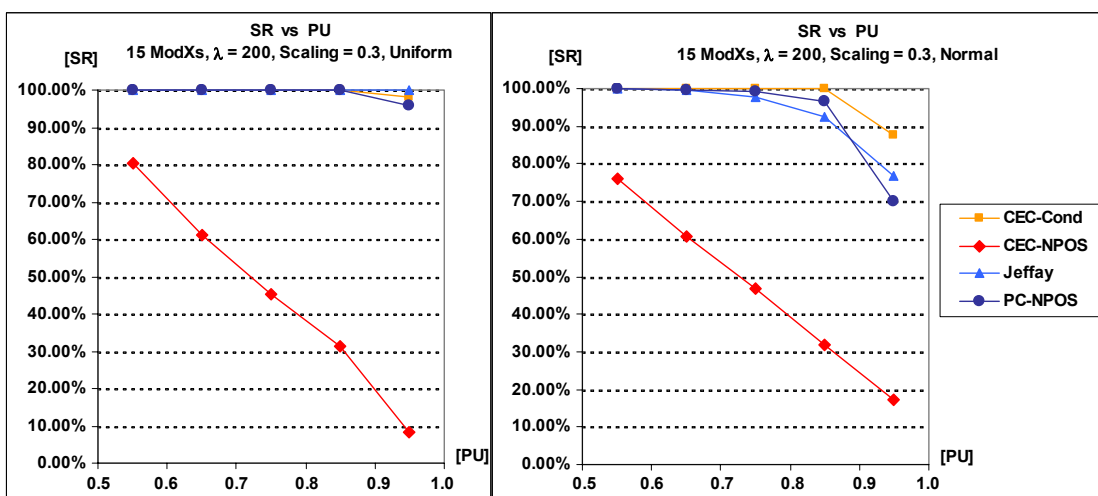


Figura 6-7. SR funcție de PU, pentru 15 ModX-uri, $\lambda = 200$ și factorul de scalare 0.3, distribuție uniformă (stânga) și normală (dreapta)

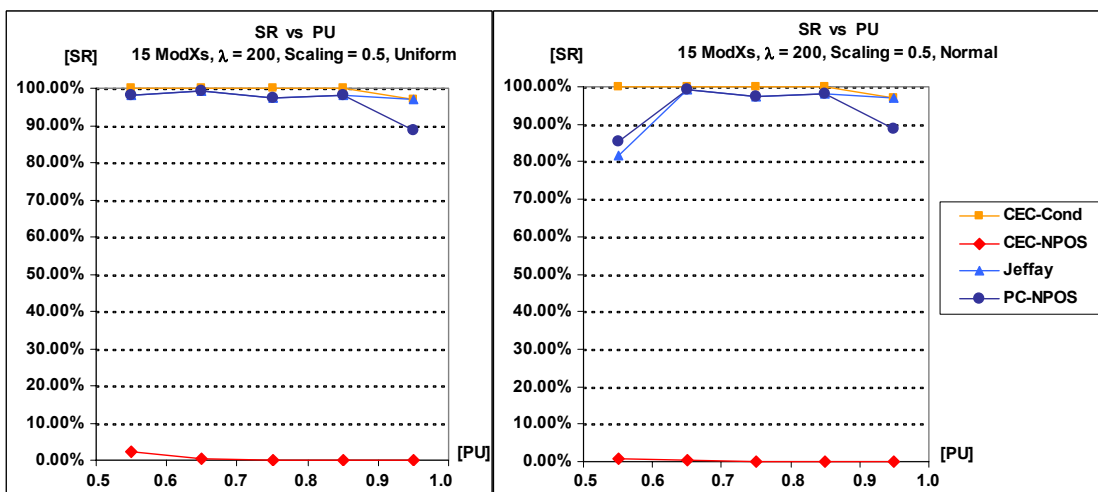


Figura 6-8. SR funcție de PU, pentru 15 ModX-uri, $\lambda = 200$ și factorul de scalare 0.5

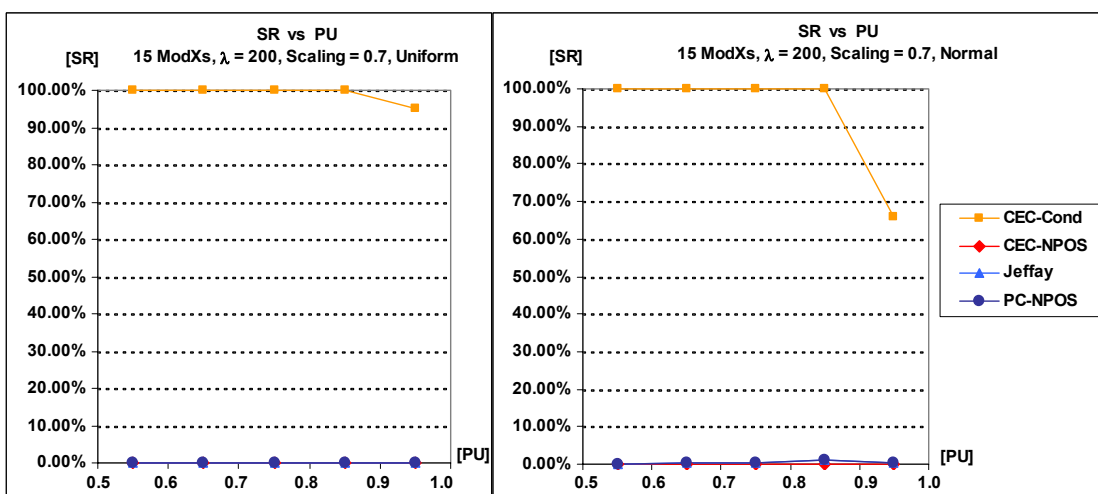


Figura 6-9. SR funcție de PU, pentru 15 ModX-uri, $\lambda = 200$ și factorul de scalare 0.7

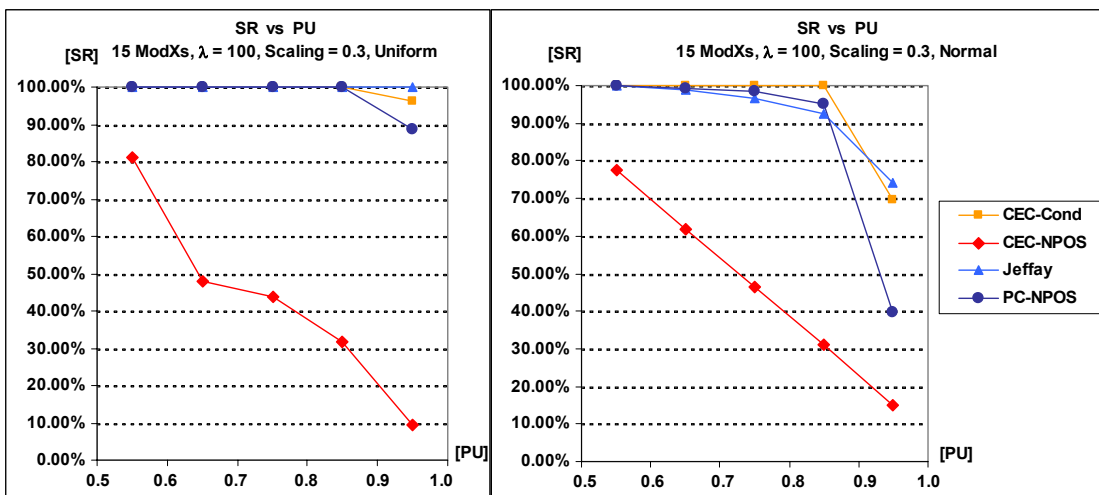


Figura 6-10. SR funcție de PU, pentru 15 ModX-uri, $\lambda = 100$ și factor de scalare 0.3

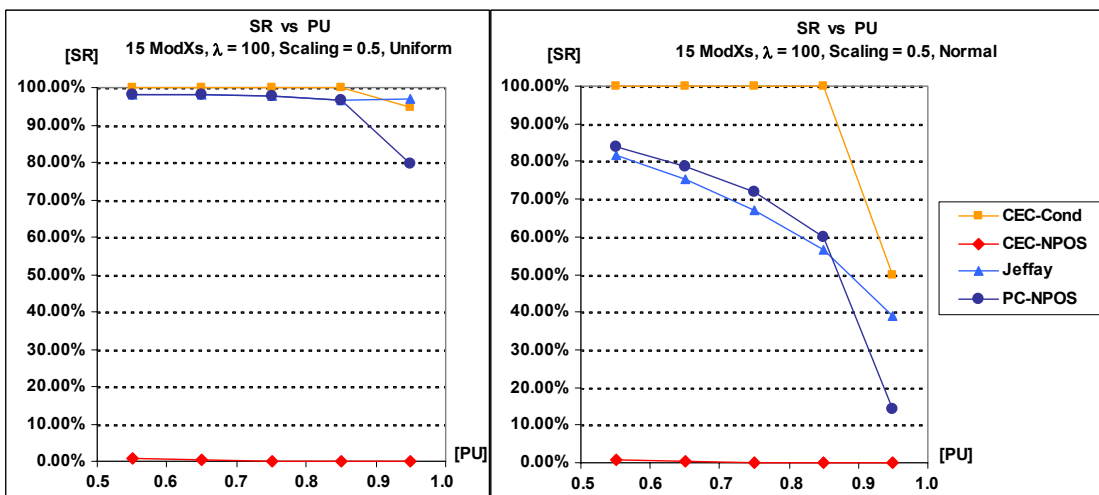


Figura 6-11. SR funcție de PU, pentru 15 ModX-uri, $\lambda = 100$ și factor de scalare 0.5

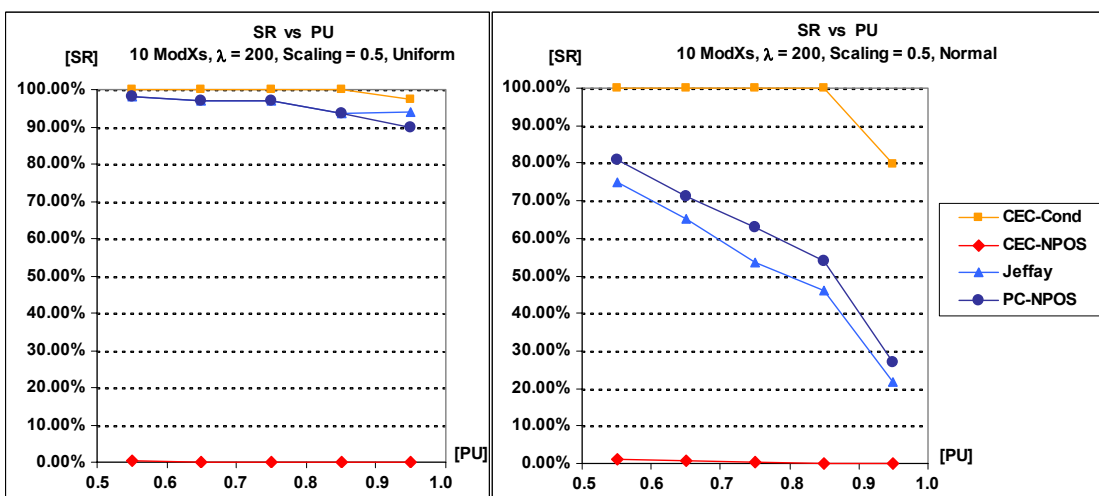


Figura 6-12. SR funcție de PU, pentru 10 ModX-uri, $\lambda = 200$ și factor de scalare 0.5

6.4 Evaluarea algoritmului FENP

Testarea și analiza performanțelor algoritmului de planificare non-preemptivă a ModX-urilor cu execuție fixă în cadrul perioadei (FModX-uri), independente, a fost realizată cu ajutorul programului "Fixed", dezvoltat ca și cele anterioare în mediul Microsoft® VisualStudio™ 6.0 (Visual C++) pentru platforme PC. Configurația de operare a programului este de asemenea similară cu cea a celorlalte unele soft utilizate, fiind specificată printr-un fișier de configurare.

Înainte de aplicarea propriu-zisă a algoritmului de planificare *FENP* asupra seturilor de FModX-uri cu parametrii temporali generați aleator, se testează condițiile de necesitate ale planificabilității: (a) testul "Jeffay", pentru evaluări comparative ale comportării algoritmului *FENP* față de ceilalți algoritmi studiați, și (b) condiția de necesitate *FENP* (Corolarul 5-5):

$$T_{ex}^{M_i} + T_{ex}^{M_j} \leq \mathcal{D}_{i,j} = \text{GCD}\left\{T_{pr}^{M_i}, T_{pr}^{M_j}\right\}, \quad \forall i, j = 1, 2, \dots, n \quad (6-5)$$

unde: $M_i \equiv \left\{T_{ex}^{M_i}, T_{pr}^{M_i}\right\} \in \mathbf{M} = \left\{M_1, M_2, \dots, M_n\right\}$

Pentru ca programul "Fixed" să poată genera cât mai multe seturi cu șanse de a fi planificate, trebuie ca generarea aleatoare a parametrilor temporali ai FModX-urilor din fiecare set să fie oarecum forțată în sensul apropierii de condițiile specificate în (6-5):

- Perioadele FModX-urilor, deși generate aleator, cu distribuție normală sau uniformă, trebuie să aibă un divizor comun cât mai mare, două câte două;
- Duratele de execuție, deși generate aleator, trebuie să fie suficient de mici, comparativ cu divizorul comun al perioadelor, luate două câte două.

Ca urmare, a fost introdus un parametru suplimentar în configurarea sesiunilor de testare aplicate cu programul "Fixed": *GCD_Seed*, care va fi un divizor pentru toate perioadele FModX-urilor, și de care se ține cont și la generarea duratelor de execuție.

Utilizând programul "Fixed" au fost efectuate peste 90000 de teste, utilizând cele 12 stații ale rețelei de calculatoare tip PC Pentium III din laboratorul DSPLabs. Testele desfășurate au urmărit determinarea raportului de planificabilitate *SR* în funcție de următorii parametri suplimentari (combinații ale acestora):

- Numărul total de FModX-uri dintr-un set: $\text{Tot}_{MX} = 10, 15, 20$;
- Utilizarea procesor (*PU*) implicată de seturile de FModX-uri, încadrată în intervalul: $[\text{Min}_{PU}, \text{Max}_{PU}] = [0.2, 0.3], [0.3, 0.4], [0.4, 0.5], [0.5, 0.6],$ și $[0.6, 0.7]$;
- Divizorul comun al perioadelor FModX-urilor: $\text{GCD_Seed} = 6, 10, 30$.

Intervalul ce încadrează valorile perioadelor ModX-urilor a fost considerat constant: $\text{Interv}_{Timp} = 510$, valoarea minimă pentru perioade fiind 10. Limita maximă pentru intervalul de analiză a planificării a fost configurată pentru $T_{LCM} = 1000000000$.

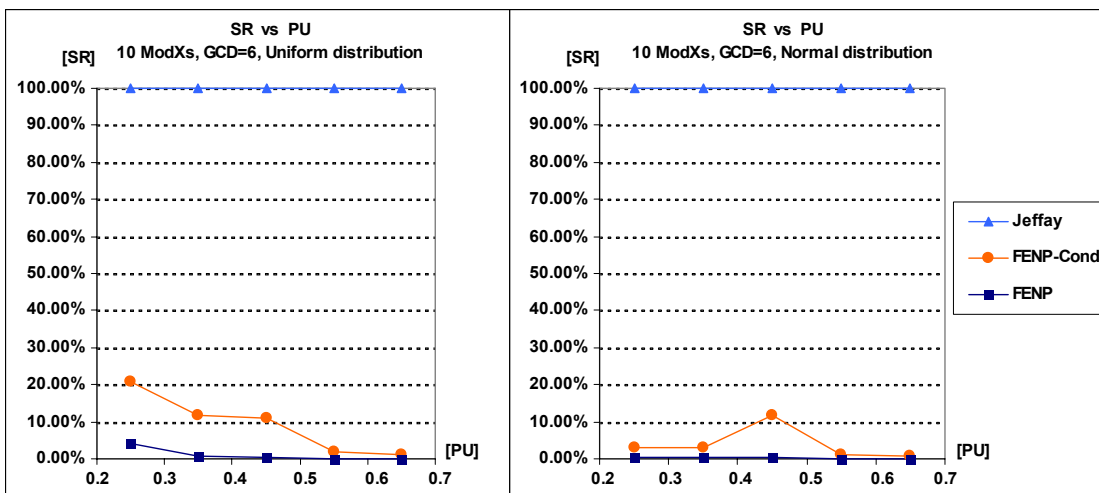


Figura 6-13. SR funcție de PU, pentru 10 ModX-uri, cu GCD_{Seed} = 6, distribuție uniformă (stânga) și normală (dreapta)

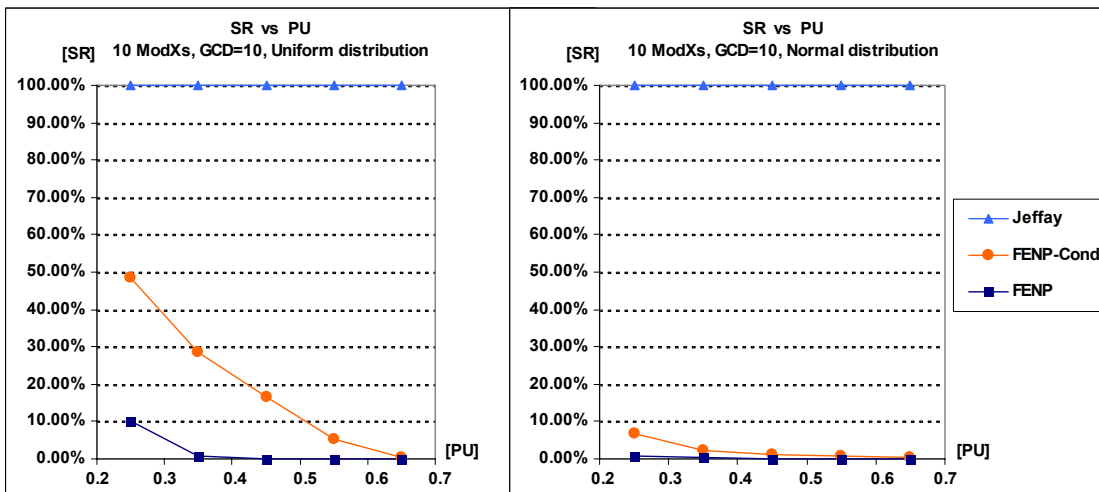


Figura 6-14. SR funcție de PU, pentru 10 ModX-uri, cu GCD_{Seed} = 10

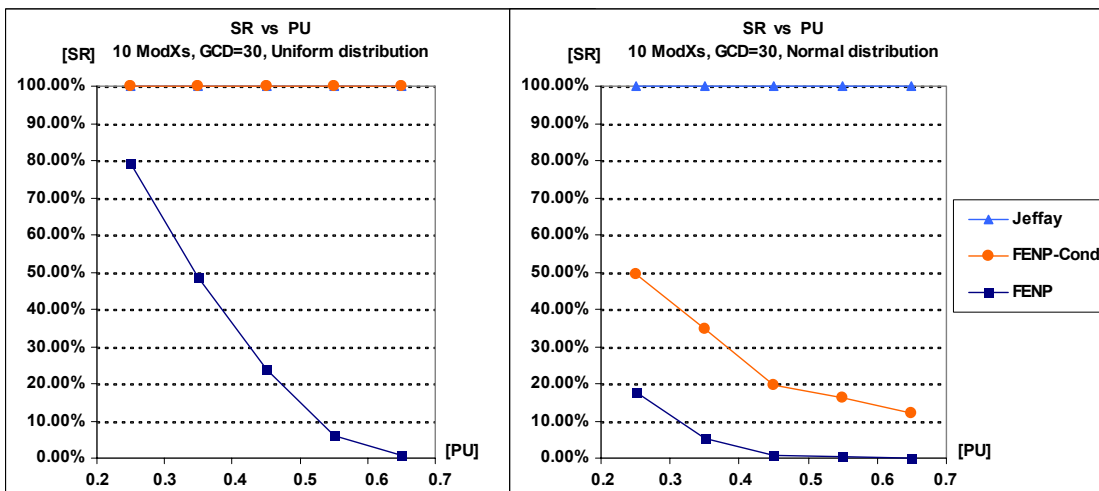


Figura 6-15. SR funcție de PU, pentru 10 ModX-uri, cu GCD_{Seed} = 30

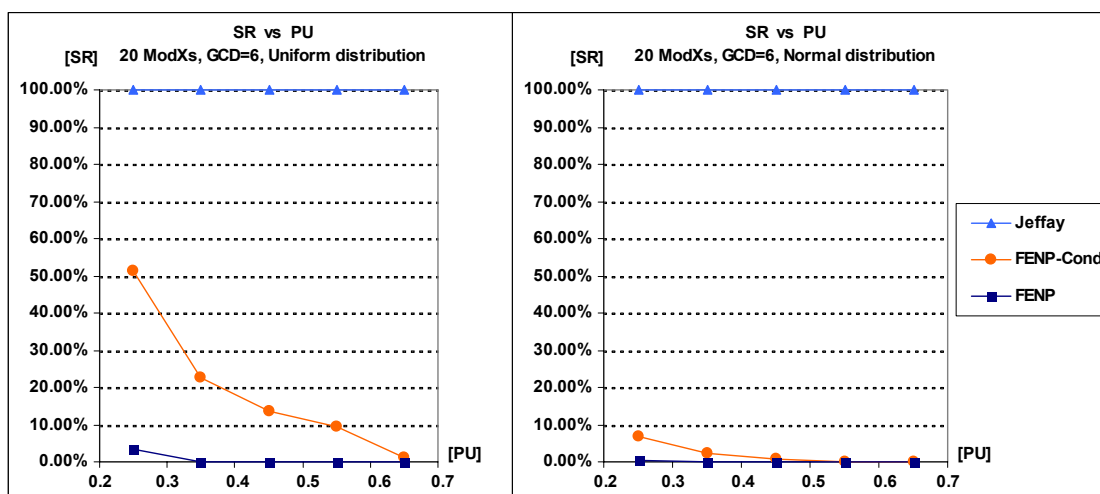


Figura 6-16. SR funcție de PU, pentru 20 ModX-uri, cu GCD_seed = 6

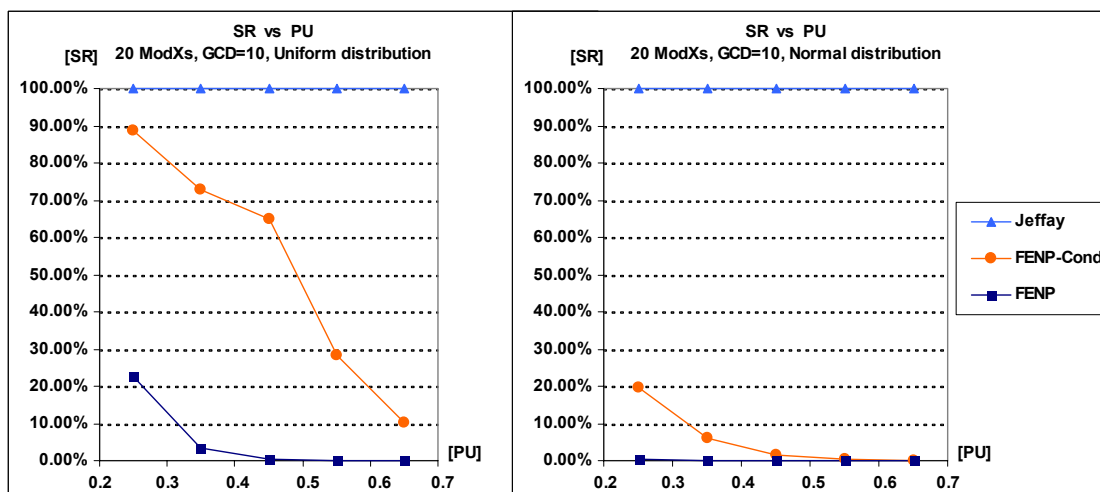


Figura 6-17. SR funcție de PU, pentru 20 ModX-uri, cu GCD_seed = 10

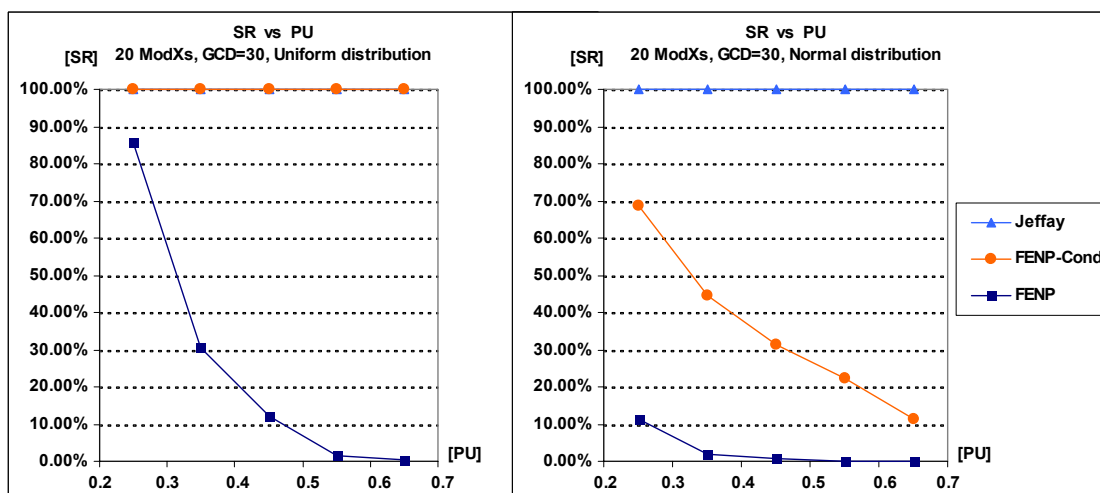


Figura 6-18. SR funcție de PU, pentru 20 ModX-uri, cu GCD_seed = 30

Rezultatele obținute cu ajutorul programului "Fixed" pentru seturi de 10 FModX-uri sunt prezentate în Figura 6-13 la Figura 6-15, iar cele pentru seturi de 20 FModX-uri, în Figura 6-16 la Figura 6-18. Din analiza efectuată asupra algoritmului *FENP*, se pot desprinde următoarele observații:

- Creșterea valorii divizorului comun al perioadelor FModX-urilor din set mărește semnificativ rata de succes a planificării *FENP*. Acest lucru este de așteptat, ținând cont de condiția de necesitate *FENP*, (6-5);
- Pentru seturile de FModX-uri cu perioadele generate aleator cu distribuție uniformă, rata de succes a planificării este mult îmbunătățită, comparativ cu distribuția normală. Explicația derivă din faptul că într-o distribuție uniformă, unde valorile perioadelor FModX-urilor "acoperă" întreg intervalul considerat, (a) există o probabilitate mai bună ca divizorul comun al oricărei perechi de perioade să fie relativ mare în comparație cu perioadele respective, și (b) seturile de FModX-uri ale căror perioade au valori "distanțate" una față de cealaltă pot fi planificate mai ușor decât în cazul în care perioadele iau valori apropiate și concentrate într-un subinterval central din domeniul de valori considerat (ca în cazul distribuției normale). Această caracteristică a planificării *FENP* vine oarecum în contradicție cu tipul de distribuție găsit la parametrii temporali ai task-urilor din aplicațiile practice, mai apropiat de distribuția normală decât de cea uniformă;
- Creșterea numărului de FModX-uri din set are ca efect și creșterea ratei de succes a planificării acestora cu *FENP* (din aceleași motive ca cele prezentate la evaluarea algoritmilor *EDFNP* și *MLFNP*);
- Comparând graficul testului "Jeffay" și valoarea "de tăiere" a utilizării procesor (*PU* pentru care rata planificabilității *SR* se anulează) din figurile de mai sus, cu cele ale celorlalți algoritmi studiați, se poate observa că performanțele planificării *FENP* sunt foarte slabe. Acest fapt este motivat de caracteristicile speciale ale FModX-urilor, în special de restricția ca execuția acestora are loc la un același interval fixat în cadrul perioadei.

6.5 Concluzii

Pentru testarea și evaluarea performanțelor algoritmilor de planificare non-preemptivă a seturilor de ModX-uri, introduși și studiați în Capitolul 5, au fost dezvoltate o serie de programe specifice:

- "Evaluation 1" și "DataAdd", pentru testarea comparativă a algoritmilor *EDFNP* și *MLFNP*;
- "OnlineScd", pentru testarea comparativă a algoritmilor de planificare online *CEC-NPOS* și *PC-NPOS*;
- "Fixed" pentru testarea și evaluarea algoritmului *FENP*.

Procedurile de evaluare a algoritmilor de planificare se bazează pe generarea de seturi de ModX-uri, în manieră aleatoare, cu două din tipurile de distribuție considerate de bază: distribuția uniformă și normală. Metodele de generare ale parametrilor temporali ai ModX-urilor cu cele două tipuri de distribuție au fost atent

studiate și implementate. După obținerea fiecărui set de ModX-uri, au fost aplicate și contorizate condițiile de necesitate ale planificării pentru algoritmi studiați și apoi s-a aplicat algoritmul propriu-zis, cu contorizarea rezultatului.

Programele de testare au fost executate în rețeaua de calcul a laboratorului DSPLabs, compusă din 12 stații PC tip Pentium III, pe o durată totală de peste 16 săptămâni. În acest interval au fost realizate peste 150000 de teste.

Rezultatele testelor efectuate au fost centralizate, analizate și reprezentate grafic. Diagramele din acest capitol ilustrează câteva din cele mai semnificative rezultate obținute. Concluziile generale ce se desprind sunt următoarele:

- Algoritmul de planificare non-preemptivă *EDFNP* (Earliest Deadline First Non-Preemptive) este în mod clar mai eficient ca *MLFNP*, și va fi deci preferat în variantele ulterioare de algoritmi de planificare non-preemptivă, care vor trata modele mai complexe de seturi de ModX-uri (de exemplu, ModX-uri cu dependențe de program, etc.).
- Planificarea în timpul operării sistemului (planificarea online) impune o serie de restricții pentru task-ul cu acest rol și pentru întregul set de ModX-uri ce urmează a fi executat în sistem. În mod evident, abordarea planificării online în cicluri periodice (*PC-NPOS*) este de preferat față de cea în cicluri cu număr fix de execuții (*CEC-NPOS*). Din punct de vedere practic, algoritmul *PC-NPOS* este similar cu *EDFNP*, diferența constând nu la nivel de algoritm, ci la nivelul setului de task-uri ce urmează a fi planificate: *PC-NPOS* include și task-ul de planificare online M_S , pe lângă ModX-urile normale.
- Dacă există aplicații TR ce necesită generarea sau tratarea de semnale la intervale fixe, exacte de timp, trebuie luat în considerare modelul de FModX, cu algoritmul de planificare corespunzător: *FENP*. Pe de altă parte, includerea acestui algoritm implică o scădere a performanțelor întregului sistem, impunând de asemenea o serie de restricții suplimentare, relativ severe, pentru parametrii temporali ai task-urilor aplicației.

7 Modelul STR strict pentru aplicații critice APNS și de control încorporat

Capitolul de față prezintă unificarea într-un sistem omogen a modelelor dezvoltate în capitolele anterioare pentru reprezentarea timpului, a semnalelor și a task-urilor TR. Se va introduce un model de STR conceput pentru a oferi un mediu integrat, unitar, de specificare, programare, analiză și execuție a aplicațiilor critice din domenii ca achiziția și prelucrarea numerică a semnalelor, orientat pe arhitecturi compacte, de control digital încorporat, cum ar fi de exemplu sistemele bazate pe procesoare numerice de semnal.

7.1 Descrierea generală și caracteristicile sistemului OPEN-HARTS

Una din observațiile esențiale desprinse din studiul sistemelor TR stricte și a algoritmilor de planificare a ModX-urilor este că faza de analiză offline, apriori execuției în sistem a setului de task-uri TR stricte, reprezintă o cerință obligatorie. În consecință, modelul STR strict conceput cuprinde două elemente principale: un mediu integrat, destinat dezvoltării și analizei offline a aplicațiilor și un nucleu de operare TR strict pe care se încarcă și se execută aplicațiile validate.

OPEN-HARTS (Operating Environment for Hard Real-Time Systems) este conceput ca un mediu de operare pentru sisteme și aplicații TR stricte, având două componente majore (Figura 7-1):

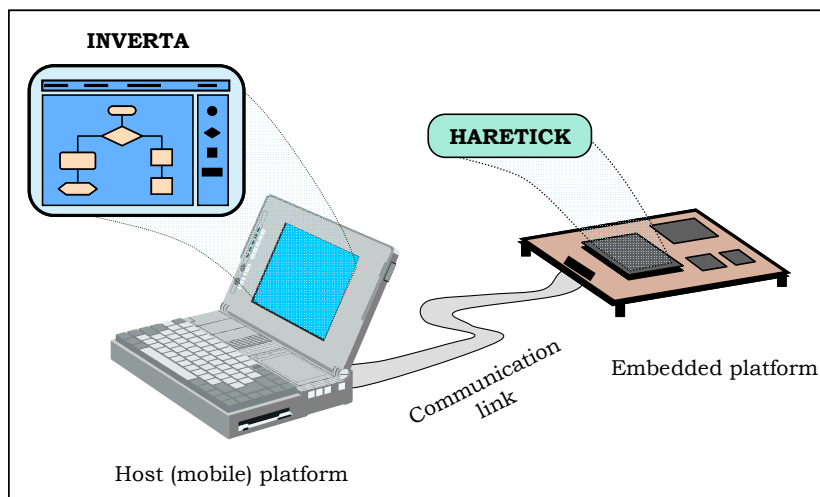


Figura 7-1. Arhitectura generală a sistemului OPEN-HARTS

- 1) Mediul integrat de dezvoltare și analiză a aplicațiilor TR stricte pe sisteme DSP și de control digital încorporat, INVERTA (Integrated Visual Environment for Real-Time Application Analysis and Development). INVERTA oferă programatorului de sistem și de aplicații TR un mediu vizual integrat, destinat proiectării, specificării și verificării formale a aplicațiilor (cu accent pe comportamentul temporal al task-urilor – obligatoriu la cele TR stricte). INVERTA permite de asemenea, programarea funcțională a fiecărui task al aplicației, analiza temporală a task-urilor (extragerea WCET) și analiza planificabilității. Toate operațiile se realizează în regim "off-line" vis-a-vis de operarea sistemului TR propriu-zis, pe o platformă de calcul ("Host platform"), care poate fi statică (workstation), sau mobilă (Laptop, PDA, Tablet PC, etc.).
- 2) Nucleul de operare TR hibrid pentru sisteme încorporate, HARETICK (Hard Real-Time Compact Kernel). HARETICK este instalat pe platforma de control digital încorporat țintă, și permite încărcarea și execuția task-urilor TR stricte (Hard Real-Time tasks) în regim non-preemptiv și a celor lejere (Soft Real-Time tasks) în regim preemptiv.

Cele două componente ale sistemului OPEN-HARTS sunt interconectate printr-o legătură și un protocol de comunicații, permițând schimbul de date, comenzi și informații de stare între INVERTA (programatorul de sistem/aplicații) și HARETICK (sistemul TR care operează pe o platformă țintă).

Întregul sistem se bazează pe o concepție puternic modulară, atât în ceea ce privește arhitectura generală cât și modul de abordare a proiectării, specificării, analizei și execuției aplicațiilor TR.

Mediul integrat INVERTA permite programatorului de sistem/aplicații dezvoltarea unei aplicații TR în regim offline, fără a interfera deci, în această fază, cu sistemul țintă, pe care poate opera la un moment dat o versiune anterioară a aplicației. Dezvoltarea aplicației începe prin construirea într-un mediu vizual și într-o manieră modulară, a diagramei de task-uri (ModX-uri și task-uri TR lejere), rezultând o rețea vizuală, echivalentă cu graful dependențelor de program (*PDG – Program Dependence Graph*). În continuare, pentru fiecare nod al grafului (reprezentând de fapt un task – ModX sau task TR lejer), se vor specifica următoarele elemente caracteristice:

- Setul parametrilor de intrare și de ieșire (dependența de date față de celelalte module).
- Setul variabilelor globale ale aplicației, utilizate de către task.
- Setul semnalelor de intrare și de ieșire cu care interacționează task-ul. Pentru fiecare semnal, programatorul poate modela comportarea acestuia în timp, prin intermediul unui set minimal și complet de parametri temporali (vezi Capitolul 4). Pentru semnalele de intrare, specificarea comportării temporale este obligatorie.
- Prin intermediul unui set minimal și complet de parametri temporali, programatorul poate modela comportarea în timp a task-urilor TR stricte – ModX-urile (vezi Capitolul 4).

- Comportarea funcțională a fiecărui task se va specifica prin intermediul unui limbaj de programare specific aplicațiilor TR (limbaj de asamblare, C modificat, etc.).

După proiectarea și specificarea aplicației, INVERTA va trece în mod automat la faza de verificare/validare a specificațiilor. Sistemul va utiliza seturi de formule bine stabilite, între parametrii temporali ai semnalelor și ai task-urilor aplicației (prezentate în Capitolul 4), atât pentru a verifica validitatea specificațiilor, cât și pentru a stabili în mod automat valorile parametrilor ce nu au fost specificați explicit de programator. INVERTA verifică, de asemenea, dependențele de control și de date ale aplicației, ținând cont și de specificarea funcțională (codul program) al fiecărui task, și modifică acolo unde e cazul, afișarea acestor dependențe în mod corespunzător.

După validarea întregii aplicații, atât din punct de vedere funcțional, cât și al comportamentului în timp, INVERTA va compila codurile sursă ale fiecărui task și va efectua analiza temporală a programelor, pentru obținerea timpilor maximi de execuție ai task-urilor (*WCET*). Tot în cadrul fazei de analiză a aplicației, INVERTA aplică algoritmi de planificare special concepuți pentru modelul de sisteme TR stricte considerate în lucrare (Capitolul 5), pentru a testa planificabilitatea setului de ModX-uri ale aplicației.

În cazul în care INVERTA a găsit o planificare fezabilă a aplicației, se efectuează pregătirile finale pentru încărcarea ei pe platforma țintă, unde va fi executată sub nucleul de operare HARETICK: link-editare, generarea tabelii de procese (ce conține și parametrii temporali ai task-urilor) și a grafului aplicației, generarea tabelilor de simboluri și a hărții de memorie, etc.

Legătura de date și comenzi (DATALINK) dintre INVERTA și HARETICK are următoarele caracteristici principale:

- Implementează un protocol de comunicații simplu, dar flexibil și scalabil.
- Permite transmiterea de comenzi utilizator prin intermediul INVERTA, către nucleul de operare HARETICK, care execută o aplicație TR oarecare.
- Permite citirea stării de operare a sistemului țintă, prin comandarea generării de diverse rapoarte de execuție.
- Permite încărcarea unei aplicații noi (sau modificate) în sistemul țintă, sau descărcarea informațiilor complete despre o aplicație ce se execută pe sistemul țintă. Datele ce caracterizează execuția unei aplicații TR oarecare pe platforme HARETICK, cuprind printre altele, următoarele elemente:
 - sursele de cod compilat ale task-urilor aplicației;
 - tabela cu descriptorii de procese (incluzând parametrii temporali ai task-urilor);
 - tabela ce descrie graful aplicației;
 - harta de memorie;
 - tabela de simboluri.

Așadar, prin intermediul legăturii de date, a protocolului de comunicație și a interfețelor corespunzătoare din INVERTA și HARETICK, programatorul de aplicații poate să citească la orice moment starea sistemului țintă (în timpul operării acestuia, și fără a afecta operarea sistemului) – incluzând informațiile complete

despre aplicația executată curent, poate să trimită diverse comenzi utilizator către sistem și să încarce o nouă aplicație pentru a o înlocui pe cea curentă.

La terminarea cu succes a încărcării unei aplicații în cadrul sistemului țintă, nucleul de operare HARETICK va începe execuția acesteia în timp real. Acest moment definește practic începerea operării efective a aplicației în cadrul sistemului, și predictibilitatea execuției, asupra căreia se concentrează materialul din lucrarea de față, este garantată începând din acest moment.

HARETICK este un nucleu TR hibrid, multi-tasking și single-user, pentru sisteme de control încorporat (*embedded systems*), conceput pentru a oferi predictibilitate maximă de execuție aplicațiilor TR critice. Caracteristica de nucleu TR hibrid se referă la faptul că HARETICK oferă mecanisme pentru existența a două regimuri "simultane" de execuție: contextul non-preemptiv, destinat execuției task-urilor TR stricte (ModX-urile), cu garantarea respectării termenelor specificate (oferind deci, o predictibilitate maximă), și contextul preemptiv destinat execuției task-urilor lejere din punct de vedere al specificațiilor de comportament temporal. Mecanismele implementate se bazează pe limitarea (sau chiar eliminarea) întreruperilor din sistem, cu excepția întreruperii de ceas (*RTC – Real-Time Clock*), care are prioritatea maximă.

Observație

Modelul nucleului hibrid de operare TR descris mai sus a fost propus și introdus în contextul implementării HARETICK, ca soluție (cel puțin parțială) la problema eficienței scăzute pe care o prezintă sistemele de task-uri ce operează sub planificare non-preemptivă, pe de o parte, și pe de alta, pentru a compensa pierderile din timpul de operare introduse de analiza de tip "durată maximă de execuție" (WCET) a task-urilor sistemului.

Conceptele de arhitectură și implementare introduse în dezvoltarea nucleului TR hibrid HARETICK constituie subiectul secțiunii următoare.

7.2 Principiile de operare ale sistemului propus

Principiile de operare ale sistemului OPEN-HARTS au fost prezentate în esență în paragraful anterior. Pentru o exemplificare și mai concretă a capabilităților sistemului, vom considera un scenariu ipotetic de utilizare a OPEN-HARTS.

Presupunem că, la un moment dat, există un sistem digital încorporat cu rolul de control al unui echipament cu cerințe critice de operare. Pe platforma sistemului rulează nucleul HARETICK, ce execută o anumită aplicație TR. Programatorul sistem dorește modificarea aplicației care rulează în mod curent pe platforma țintă, mai precis, dorește adăugarea a încă unui ModX în aplicație, de exemplu. Secvența de pași și evenimente ce vor apărea în acest scenariu, se desfășoară după cum urmează:

1. Programatorul sistem conectează sistemul de calcul propriu (presupunem că e vorba de un sistem portabil, tip Laptop), pe care este instalat mediul de dezvoltare INVERTA, la platforma țintă, prin intermediul legăturii DATALINK.

2. Din mediul INVERTA, utilizatorul selectează opțiunea de generare a rapoartelor de stare, care este trimisă ca o comandă sistem către HARETICK, prin intermediul legăturii de comunicație.
3. Comanda de generare a rapoartelor de stare este recepționată de nucleul HARETICK de pe sistemul țintă, interpretată și executată (în regim TR lejer). Ca urmare, HARETICK transmite informația completă de stare curentă către INVERTA. Informația de stare cuprinde elemente legate de aplicația ce se execută curent pe sistemul țintă (tabela cu descriptorii de procese, graficul aplicației, sursele de cod ale task-urilor, etc.), cât și elemente legate de caracteristicile temporale ale componentelor de bază ale nucleului (dispatcher, online scheduler, etc.), și timpii și evenimentele desfășurate în cadrul operării sistemului.
4. Informația de stare recepționată este interpretată în cadrul mediului INVERTA, rezultând, printre altele, o afișare sub formă de graf vizual a aplicației ce rulează pe sistemul țintă, împreună cu valorile tuturor parametrilor acestora și cu codul program (specificarea funcțională a task-urilor).
5. Programatorul de aplicație poate în acest moment să modifice aplicația respectivă (prin adăugarea unui nou ModX de exemplu, așa cum am presupus în scenariu).
6. După adăugarea ModX-ului nou în cadrul aplicației, și după specificarea lui completă (stabilirea valorilor pentru parametrii temporali, scrierea codului program, stabilirea legăturilor de control și date cu celelalte task-uri, etc.), mediul INVERTA inițiază în mod automat verificarea și validarea întregii aplicații.
7. În momentul în care specificațiile aplicației au fost validate de către INVERTA, se activează faza de analiză temporală a programului (compilarea codului noului ModX, calculul WCET), și analiză a planificabilității setului de ModX-uri.
8. Găsirea unei planificări fezabile a noii aplicații și pregătirile finale pentru încărcarea acesteia în sistemul țintă, încheie setul de operații efectuate în mod offline de către INVERTA.

Ca observație, menționăm că toate operațiile offline pe care le desfășoară mediul integrat de dezvoltare INVERTA (pașii 4 – 8) pot fi efectuate și cu platforma gazdă (laptop-ul programatorului de sistem) deconectată de la sistemul țintă, pe care încă rulează aplicația curentă. De asemenea, timpul nu este o coordonată esențială pentru efectuarea acestor operații.

În cazul în care calculatorul gazdă a fost deconectat (și deplasat) de la sistemul țintă, va trebui reconectat la acesta, pentru etapele următoare.
9. Noua aplicație, dezvoltată și analizată cu ajutorul sistemului INVERTA este încărcată pe sistemul țintă, prin intermediul unor comenzi specifice, transmise către HARETICK.

10. Întreaga procedură se termină în momentul încărcării cu succes pe platforma țintă a tuturor datelor necesare planificării și executării online a noii aplicații. După acest moment, aplicația va opera în timp-real, cu garantarea respectării tuturor specificațiilor temporale pentru task-urile TR stricte (ModX-urile aplicației).

De remarcat faptul că, până la terminarea pasului 10 din secvența de mai sus, cu alte cuvinte, până când noua aplicație este complet încărcată în sistemul țintă, acesta va continua să execute aplicația veche. Comutarea execuției dintre cele două aplicații existente în sistem (echivalentă cu o sincopă în operarea sistemului) se face într-o manieră controlată și predictibilă. Această întrerupere a operării este singurul eveniment care poate perturba interacțiunea sistemului cu mediul înconjurător.

SECȚIUNEA III. CONTRIBUȚII LA IMPLEMENTAREA ȘI EVALUAREA STR STRICTE

În capitolele secțiunii de față sunt prezentate detaliile de implementare ale unui sistem de operare (nucleu) timp-real cu caracteristici care îi permit garantarea respectării termenelor de timp pentru task-urile critice ale aplicațiilor de achiziție și prelucrare numerică de semnal sau de control încorporat.

Astfel, nucleul de operare HARETICK ("Hard Real-Time Compact Kernel") reprezintă un studiu de caz, complet și consistent, pentru implementarea practică, testarea și evaluarea principiilor, modelelor teoretice și algoritmilor dezvoltați și discutați în cadrul secțiunii anterioare, cu privire la specificarea, analiza și planificarea aplicațiilor critice.

Nucleul HARETICK a fost parțial implementat pe platforma de prelucrare numerică de semnal Motorola DSP56307EVM, astfel încât să se poată testa și evalua principalele componente funcționale ale nucleului și să se poată verifica dacă operarea acestuia corespunde îndeosebi modelelor teoretice dezvoltate pentru clasa task-urilor cu specificații stricte de comportare temporală (HRT – Hard Real-Time) ale aplicațiilor critice.

8 Nucleul de operare HARETICK: caracteristici generale, componente și relația dintre acestea

8.1 Caracteristici generale

Conceperea, proiectarea și implementarea nucleului HARETICK are la bază faptul că sunt vizate sistemele de control digital încorporat (embedded systems). Prin urmare, HARETICK este un nucleu de operare timp-real de tip *single-user*: programatorul de sistem/aplicații poartă întreaga responsabilitate a proiectării, programării și utilizării sistemului.

De asemenea, HARETICK este un nucleu *multitasking*, permițând execuția concurentă, planificată a unor seturi compuse din mai multe task-uri. Nucleul oferă facilități pentru operarea cu predictibilitate maximă a aplicațiilor critice, garantând respectarea termenelor stricte de timp impuse de acestea. În continuare vom prezenta o serie de alte caracteristici și proprietăți importante ce definesc arhitectura și operarea nucleului HARETICK.

Arhitectura modulară

Sistemul HARETICK este conceput și proiectat într-o manieră puternic modularizată, vizându-se simplitatea și reducerea pe cât posibil a dimensiunilor acestuia. Pentru contextul strict de execuție a task-urilor, sistemul nu efectuează nici un fel de analize de planificabilitate și fezabilitate, ci implementează algoritmi de planificare, cu comportare temporală cunoscută, asupra aplicației analizate și optimizate apriori, offline.

Componentele software cu specificații temporale stricte (HRT) ale sistemului și ale aplicațiilor dezvoltate pentru acesta, sunt proiectate și implementate sub formă de module software independente, care comunică între ele prin intermediul parametrilor de intrare/ieșire și al variabilelor globale.

Task-urile existente la un moment dat în cadrul HARETICK sunt împărțite în două categorii: task-uri sistem, care definesc practic nucleul de operare și sunt încărcate în faza de inițializare a sistemului, și task-uri aplicație, care sunt încărcate la un moment dat în sistem. Nucleul HARETICK permite încărcarea, descărcarea sau înlocuirea unei aplicații TR la orice moment, prin intermediul unor comenzi operator specifice.

Gestionarea timpului

Timpul, ca parametru esențial de operare al sistemului, este reprezentat și gestionat cu ajutorul unor structuri și mecanisme dedicate. *Baza de timp* a sistemului

este implementată de *ceasul timp-real* (RTC) – elementul activ de gestionare a timpului în HARETICK. RTC are două roluri principale: (i) contor de timp, utilizat pentru referiri ale componentelor sistemului la instanța curentă de timp, și (ii) singura sursă (predictibilă) de întreruperi. *RTC este o sursă predictibilă de întreruperi în sensul că se poate determina oricând, și de către orice task, momentul apariției următoarei întreruperi.*

Pe lângă RTC, nucleul HARETICK gestionează încă două structuri de timp. *Timpul absolut sistem*, este utilizat pentru referiri ale operatorului și ale task-urilor din sistem la timpul absolut, și poate fi setat (modificat) de către operatorul sistemului în orice moment, prin intermediul unor comenzi specifice. *Timpul de planificare* este utilizat și gestionat de către planificatorul contextului strict de execuție al sistemului pentru calcularea și specificarea momentelor la care sunt planificate execuțiile task-urilor TR stricte.

Spre deosebire de timpul absolut sistem, celelalte două structuri – *ceasul timp-real* și *timpul de planificare*, lucrează practic în domeniu temporal relativ. Pe de altă parte, toate cele trei tipuri de timp cu care lucrează nucleul HARETICK sunt sincronizate între ele.

Gestionarea proceselor

Procesele (task-urile) cu care operează sistemul HARETICK sunt organizate, reprezentate și gestionate în manieră modulară, utilizând structuri de memorie și mecanisme specifice pentru fiecare task în parte: zonă dedicată pentru cod, zonă pentru parametrii de ieșire, zonă pentru variabilele globale, zonă comună pentru parametrii de intrare și variabilele locale, utilizată de către task-ul aflat în execuție curentă, tabela cu graful aplicației, tabela cu identificatorii de procese, tabela de simboluri, etc.

Task-urile cu care operează nucleul HARETICK sunt împărțite în două categorii principale: (1) task-uri TR stricte (task-uri HRT – Hard Real-Time) denumite ModX-uri (Module eXecutabile), și (2) task-uri lejere (task-uri SRT – Soft Real-Time). În consecință, sistemul HARETICK are două contexte de execuție complementare: contextul strict (sau contextul HRT) în care ModX-urile se execută în regim non-preemptiv, și contextul lejer (sau contextul SRT), pentru execuția task-urilor lejere în regim preemptiv.

Din acest punct de vedere, HARETICK este un *nucleu de operare hibrid*, comutarea celor două contexte de execuție fiind realizată de un task sistem special – executivul TR al HARETICK.

Gestionarea memoriei

Principalul deziderat urmărit la proiectarea și implementarea sistemului HARETICK este operarea predictibilă, cu garantarea respectării termenelor stricte de timp pentru task-urile care au aceste cerințe. Prin urmare toate structurile și mecanismele implementate de nucleu au o arhitectură și o comportare statică, bine determinată și limitată în timp și spațiu.

Memoria este gestionată în mod liniar de către nucleu, și este împărțită în două secțiuni – memoria sistem și memoria aplicație. Cu toate acestea, accesul task-urilor

la toate zonele de memorie este permis, nefiind implementat nici un mecanism care să le restricționeze și să definească zone de memorie "protejate". Motivația abordării este dată de caracteristica principală pentru care este proiectat sistemul: nucleu de operare TR pentru sisteme încorporate (embedded systems), în cadrul căruia programatorul/operatorul are întregul control – și de asemenea, întreaga responsabilitate.

Mecanismele IPC

Comunicația dintre task-uri (IPC – Inter-Process Communication) este realizată prin intermediul parametrilor de intrare/ieșire și al variabilelor globale. ModX-urile, ca task-uri stricte, periodice, care se execută în context non-preemptiv, implementează acțiuni de tip atomic.

Astfel, se elimină necesitatea implementării de tehnici pentru sincronizarea și controlul acceselor concurente la resursele partajate ale nucleului (inclusiv la variabilele globale). De asemenea, sunt evitate problemele de planificare de tip inversare a priorității.

Interfața cu utilizatorul

HARETICK pune la dispoziția utilizatorului/operatorului o interfață cu sistemul. Interfața este realizată printr-o serie de componente sistem care implementează un protocol simplu și flexibil de comunicații de date, un protocol de comandă și interogare a stării sistemului, și un set de proceduri de încărcare a unei aplicații noi în sistem.

8.2 Componentele HARETICK

Arhitectura generală a nucleului HARETICK (vezi Figura 8-1) este proiectată ținând cont de caracteristica esențială a operării sistemului – existența celor două contexte de operare: contextul lejer (preemptiv) și contextul strict (non-preemptiv). În consecință, toate elementele sistemului (task-urile sistem și ale aplicației) se vor clasifica după operarea într-unul din cele două contexte. Sistemul va cuprinde așadar, ModX-uri (task-uri TR stricte) și task-uri lejere.

În continuare vom prezenta pe scurt componentele principale ale HARETICK, specificând tipul lor (task-uri lejere sau ModX-uri) și funcțiile pe care le îndeplinesc. O reprezentare intuitivă a componentelor HARETICK, precum și a relațiilor stabilite între acestea, este redată în Figura 8-1. Modulele în stadiu de implementare completă sunt reprezentate cu blocuri complet colorate, cele hașurate sunt parțial implementate (funcționalitate de bază), iar cele albe urmează a fi dezvoltate în continuare.

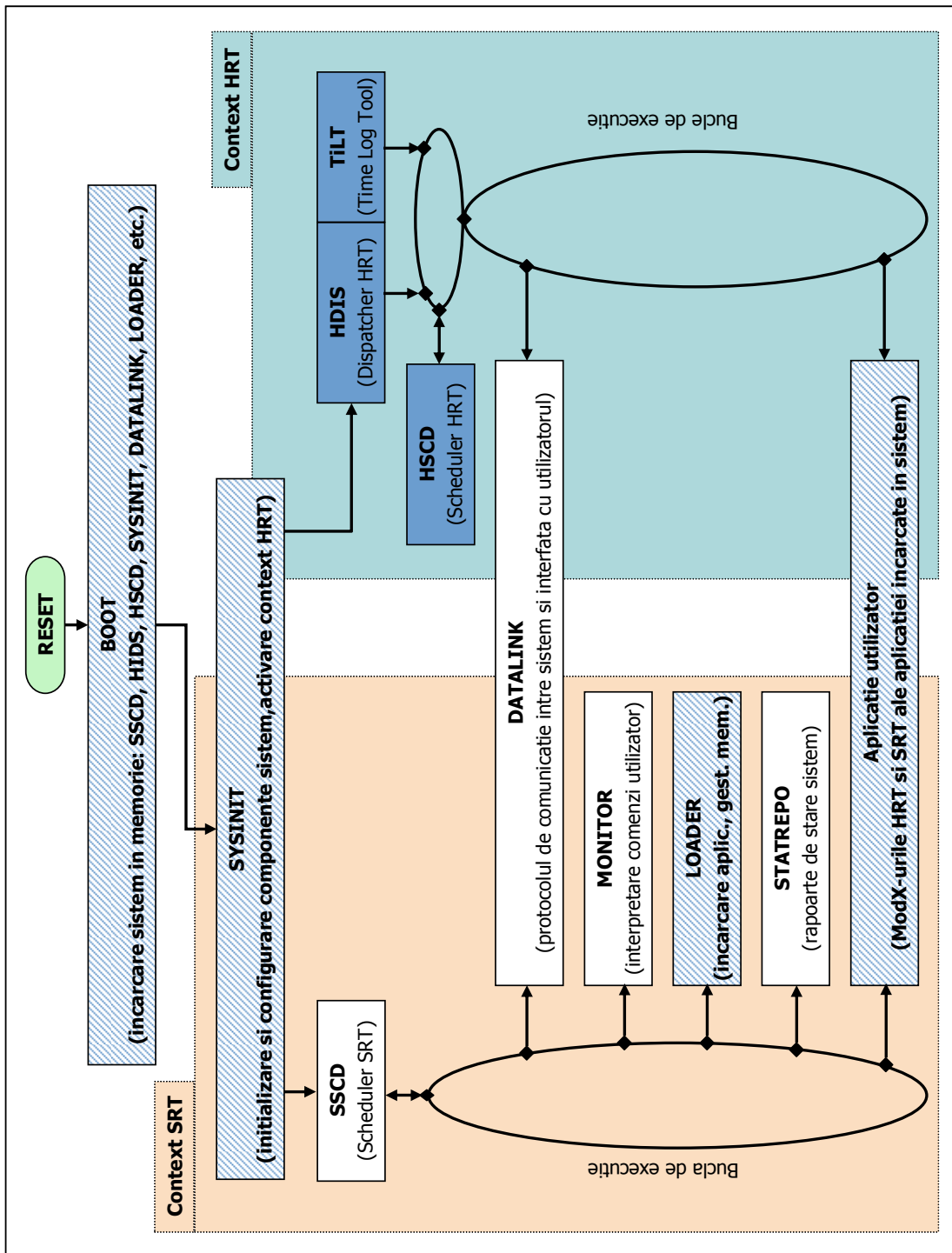


Figura 8-1. Componentele principale HARETICK și relația dintre ele

8.2.1 Încărcarea și lansarea sistemului: BOOT

Secvența (task-ul) de încărcare și startare a sistemului HARETICK este denumită "BOOT". Are rolul de a încărca în memorie structurile de date și codul tuturor componentelor de bază ale HARETICK: SYSINIT, SSCD, HDIS, TiLT, HSCD, DATALINK, MONITOR, LOADER, STATREPO, etc.

Task-ul BOOT este apelat și executat de către procesorul sistemului în momentul alimentării sau după o operație de RESET, și se găsește înscris în memoria EPROM internă sau externă a procesorului țintă.

Task-ul BOOT nu poate fi clasificat nici ca task lejer, nici ca ModX, deoarece în această etapă de operare a HARETICK, încă nu au fost inițiate cele două contexte de operare.

8.2.2 Inițializarea sistemului: SYSINIT

Task-ul de inițializare a sistemului este denumit "SYSINIT". Este task-ul lansat în execuție de către BOOT, după ce acesta a încărcat cu succes în memorie sistemul.

Rolul lui SYSINIT este de a inițializa parametrii de operare ai sistemului: variabilele globale sistem, variabilele și parametrii celorlalte componente, cum ar fi pointerii către tabela cu descriptorii de procese și tabela de dispatch, etc. După efectuarea inițializărilor, SYSINIT activează contextul strict de execuție prin startarea ceasului TR (*RTC – Real-Time Clock*), care în mod automat, prin mecanismul întreruperilor de timer, apelează executivul sistemului, HDIS (Hard real-time Dispatcher).

SYSINIT este un task de tip lejer, deci practic, se poate considera că, odată cu lansarea în execuție a lui SYSINIT de către BOOT, se activează contextul lejer de execuție. La finalul secvenței lui SYSINIT este lansat planificatorul task-urilor lejere, SSCD, care gestionează în continuare operarea acestora în contextul lejer de execuție.

8.2.3 Comunicația de date: DATALINK

Comunicația de comenzi și date dintre nucleul HARETICK și mediul de dezvoltare INVERTA de pe un calculator gazdă este realizată cu ajutorul setului de task-uri DATALINK, care implementează o interfață simplă și flexibilă de comunicații bidirecționale cu următoarele caracteristici:

- (→) Încărcarea de comenzi utilizator, ce urmează apoi a fi interpretate și executate de către HARETICK;
- (→) Încărcarea de aplicații și date în sistem;
- (←) Descărcarea de rapoarte legate de starea curentă de execuție a sistemului;
- (←) Descărcarea periodică de rapoarte de timp.

În notațiile de mai sus a fost folosit simbolul (→) pentru a indica un sens al comunicației dinspre calculatorul gazdă către platforma țintă (cu HARETICK), iar (←) pentru sensul invers.

Grupul de task-uri DATALINK este compus atât din ModX-uri, cât și din task-uri lejere (vezi și Figura 8-1). ModX-urile DATALINK sunt responsabile pentru implementarea operațiilor de transmisie/recepție a biților de date pe interfața de comunicații a sistemului. Aceste operații sunt de natură periodică și strictă, trebuind să respecte o rată (tact de bit) bine stabilită. ModX-urile DATALINK se vor executa așadar în contextul strict, asigurând respectarea debitului stabilit pentru comunicație (respectarea ratei de comunicație). Ele vor fi planificate și executate în mod continuu, periodic, chiar dacă, la un moment dat, nu există informație ce trebuie transmisă/recepționată.

Task-urile lejere DATALINK au ca scop prelucrarea la un nivel mai superior a informațiilor vehiculate pe legătura de date a sistemului. Ele utilizează zone de memorie tampon (buffere) de transmisie/recepție pentru a dezasambla/asambla cuvintele de date vehiculate în timpul comunicațiilor. Task-urile lejere DATALINK se vor executa în contextul corespunzător, putând fi astfel întrerupte de către ModX-urile ce rulează în cadrul sistemului. Execuția task-urilor lejere DATALINK nu prezintă un tipar periodic și nici specificații stricte, ele fiind apelate de către alte task-uri sistem ce necesită la un moment dat utilizarea legăturii de date cu calculatorul gazdă.

8.2.4 Interfațarea cu operatorul: MONITOR

Task-ul MONITOR are rolul de interpretor de comenzi utilizator, transmise de către mediul de dezvoltare INVERTA de pe calculatorul gazdă, prin intermediul interfeței de comunicații.

De asemenea, MONITOR determină lansarea acțiunilor de încărcare a unei aplicații, descărcarea raportului de stare sistem, etc., prin invocarea task-urilor sistem corespunzătoare.

MONITOR este un task de tip lejer, implementând interacțiuni cu utilizatorul ce nu sunt critice pentru operarea sistemului țintă.

8.2.5 Încărcarea unei aplicații: LOADER

Încărcarea unei aplicații în sistemul țintă este realizată prin intermediul task-ului LOADER, care este invocat de către MONITOR ca urmare a interpretării comenzii aferente primite de la operator. Utilizând interfața de comunicații DATALINK, task-ul LOADER recepționează de la mediul de dezvoltare INVERTA de pe calculatorul gazdă toate datele și secvențele de cod ce definesc noua aplicație, printre care:

- tabela cu identificatorii de proces (PDT – Process Descriptor Table) corespunzătoare aplicației, în care se regăsesc principalii parametri ai task-urilor, incluzând pe cei ce descriu comportarea în timp (e.g. perioada, intervalul limită, durata maximă de execuție – WCET, contorul de execuții, și așa mai departe);
- tabela grafului direcționat aciclic de program al aplicației (PDAGT – Program Directed Acyclic Graph Table) ce descrie dependențele de control și de date ale task-urilor;
- codul task-urilor aplicației;

- listele parametrilor de intrare, de ieșire și a celor globali pentru fiecare task;
- alte attribute ale task-urilor.

După recepționarea aplicației prin DATALINK, LOADER creează structurile de date și tabelele aferente, și configurează memoria sistemului prevăzută pentru încărcarea aplicației:

- înscrie codul fiecărui task în zona de memorie corespunzătoare codului aplicației (zona de text a aplicației, "CODE");
- creează zonele de memorie corespunzătoare parametrilor de ieșire ai task-urilor ("Data Out");
- generează tabela de simboluri a aplicației, în care se face corespondența dintre numele fiecărui parametru de intrare sau ieșire al task-urilor, cu adresa din memorie unde este localizat; etc.

LOADER are așadar și rolul de gestionare a memoriei, respectiv a structurilor din memoria sistemului țintă, rezervate aplicațiilor. Datorită faptului că HARETICK implementează un nucleu de operare pentru sisteme încorporate (embedded systems), pe de o parte zona structurilor de memorie proprii nucleului este gestionată practic de elementele acestuia și este configurată din etapa de proiectare și implementare, iar pe de altă parte nu există mecanisme care să controleze/restricționeze accesul task-urilor la diversele zone de memorie – acest lucru fiind responsabilitatea programatorului de sistem/aplicație.

La finalul secvenței LOADER, aplicația încărcată cu succes este lansată în execuție. Task-urile lejere ale aplicației vor fi planificate și activate de către planificatorul contextului lejer de execuție (SSCD), iar ModX-urile aplicației vor fi planificate de către planificatorul contextului strict de execuție (HSCD), fiind activate de către executivul sistemului (HDIS). Momentul lansării în execuție a aplicației este considerat ca fiind startul operării TR a aplicației în cadrul sistemului.

LOADER este un task de tip lejer, întregul set de operații, de la activarea și până la lansarea acestuia în execuție fiind considerat în afara contextului de operare timp-real a sistemului. Până la finalizarea încărcării și configurării noii aplicații în cadrul sistemului, HARETICK continuă să execute vechea aplicație, în regim timp-real ce respectă toți parametrii de comportare temporală specificați pentru aceasta.

8.2.6 Planificatorul task-urilor TR lejere: SSCD

Task-ul SSCD (Soft real-time Scheduler) are rolul de planificare a task-urilor lejere ale sistemului și aplicației, pentru execuție în context lejer pe platforma HARETICK.

SSCD este la rândul său un task lejer, execuția sa fiind întreruptă când este cazul de către ModX-urile aparținând contextului strict al sistemului.

8.2.7 Planificatorul ModX-urilor: HSCD

Task-ul HSCD (Hard real-time Scheduler) are rolul de planificare a task-urilor TR stricte (ModX-uri) ale sistemului și aplicației încărcate pe acesta, pentru execuție în contextul strict al HARETICK.

HSCD gestionează o tabelă de dimensiuni fixe (Tabela de Dispatch) în care înregistrează pe rând, ModX-urile ce urmează a fi executate în ciclul curent de planificare în contextul strict al HARETICK. O înregistrare din tabelă conține identificatorul ModX-ului corespunzător și instanța timpului sistem (valoarea RTC – ceasul timp-real) la care este planificată execuția acestuia.

Un ciclu de planificare este definit prin intervalul de timp corespunzător a două execuții consecutive ale HSCD. Astfel, Tabela de Dispatch conține la orice moment, planificările ModX-urilor ce constituie ciclul curent de planificare.

Pe lângă înregistrările normale, pentru ModX-urile sistem, Tabela de Dispatch conține o înregistrare specială, ce identifică pe HSCD.

HSCD este un ModX, fiind astfel executat de către HARETICK în contextul strict de operare.

8.2.8 Comutarea execuției ModX-urilor: HDIS

Comutarea execuției ModX-urilor în cadrul contextului strict al HARETICK este gestionată de către executivul TR al sistemului, HDIS (Hard real-time Dispatcher).

HDIS utilizează Tabela de Dispatch gestionată de HSCD, pe care o accesează în citire, înregistrare cu înregistrare. Accesarea a tabelii se face în mod circular, astfel încât după lansarea în execuție a ultimului ModX din ciclul curent de planificare, HDIS va reveni la baza tabelii, unde găsește înregistrarea corespunzătoare primei planificări din următorul ciclu.

Un alt rol important al HDIS este acela de salvare/restaurare a contextului task-urilor lejere a căror execuție este întreruptă de către contextul strict de operare a nucleului.

HDIS este apelat în mod automat în cadrul sistemului HARETICK, de către întreruperea de ceas timp-real (RTC), la momentul de timp ce specifică planificarea unui ModX. Odată activat, HDIS efectuează următoarele acțiuni:

- verifică dacă a fost întrerupt contextul lejer. În caz afirmativ, se efectuează salvarea contextului de execuție lejeră a task-ului întrerupt într-o zonă dedicată;
- identifică ModX-ul ce urmează a fi lansat în execuție la momentul curent, din Tabela de Dispatch;
- citește următoarea înregistrare din Tabela de Dispatch, corespunzând ModX-ului ce urmează ca planificare celui curent, și extrage momentul de start al execuției acestuia;
- programează timerele ceasului de timp-real (RTC) pentru a genera întreruperi la momentul de start al ModX-ului următor;
- decrementează contorul de execuții al ModX-ului curent (dacă acesta are valoare finită);
- trece ModX-ul curent din starea "planificat" ("SCD"), fie în starea "de execuție" ("RUN") – dacă valoarea contorului de execuție era nenulă înaintea decrementării, fie în starea "ModX fantomă" ("GST");
- dacă ModX-ul curent este în starea "RUN", îi cedează procesorul și îl apelează pentru execuție.

La terminarea execuției ModX-ului curent, acesta apelează din nou HDIS. În această situație HDIS efectuează următoarele acțiuni:

- trece ModX-ul curent din starea "de execuție" (RUN) sau "ModX fantomă" (GST), înapoi în "planificat" (SCD) sau în "gata de planificare" (RDY);
- avansează referința (pointerul) de citire al Tabelei de Dispatch către ModX-ul următor;
- verifică dacă există suficient timp pentru a comuta contextul strict cu cel lejer, permițând execuția task-urilor lejere ale sistemului și aplicației. Dacă nu este timp suficient, HDIS va iniția o buclă infinită în care va aștepta apariția întreruperii RTC corespunzătoare planificării ModX-ului următor. Dacă se decide comutarea controlului către contextul lejer, se va restaura contextul task-ului lejer a cărui execuție a fost întreruptă.

Din cele de mai sus rezultă că HDIS are două componente: HDIS Prefix (HDIS_PRE) care este apelat de întreruperea RTC și lansează în execuție ModX-uri, și HDIS Suffix (HDIS_SUF) care este apelat la terminarea fiecărui ModX.

HDIS este, la rândul său, un ModX, fiind astfel executat în context strict în cadrul HARETICK.

8.2.9 Raportarea stării curente a sistemului: STATREPO

Task-ul de raportare a stării curente a sistemului este denumit STATREPO. Rolul acestuia este de a trimite către interfața utilizator de pe calculatorul gazdă (INVERTA), rapoarte la cerere, utilizând interfața de comunicație DATALINK. Conținutul rapoartelor generate de STATREPO cuprinde:

- momentul curent al generării raportului;
- tabela cu identificatorii de proces (PDT – Process Descriptor Table) corespunzătoare aplicației executate curent, în care se regăsesc principalii parametri ai task-urilor, incluzând pe cei ce descriu comportarea în timp (e.g. perioada, intervalul limită, durata maximă de execuție – WCET, contorul de execuții, și așa mai departe);
- graficul dependențelor de program (PDG – Program Time Graph) ale task-urilor aplicației (descrie, cu ajutorul unei structuri unice de graf aciclic, direcționat, dependențele de control și de date);
- codul task-urilor aplicației;
- listele parametrilor de intrare, de ieșire și a celor globali pentru fiecare task;
- alte attribute ale task-urilor;
- harta memoriei;
- conținutul curent al Tabelei de Dispatch;
- tabela de simboluri a aplicației; etc.

Așa cum a fost descris, în cadrul referatului precedent, sistemul OPEN-HARTS împreună cu componenta sa dedicată dezvoltării și analizei aplicațiilor TR stricte INVERTA, informațiile legate de starea curentă a sistemului HARETICK și a aplicației ce rulează pe acesta sunt necesare mediului INVERTA pentru analiza unei noi aplicații sau pentru modificarea celei curente.

STATREPO mai poate fi configurat pentru a genera și transmite rapoarte periodice privind timpii importanți ai execuției sistemului (istoria execuției).

STATREPO este un task de tip lejer, fiind activat doar la cererea operatorului și putând fi întrerupt de către contextul strict de execuție al HARETICK de câte ori este necesar.

8.2.10 Subrutina de raportare a evenimentelor sistem: TiLT

TiLT ("Time Log Tool") este o subrutină aparținând task-ului TR strict (ModX-ului) HDIS. TiLT este apelată atât de HDIS_PRE înainte de lansarea în execuție a unui ModX, cât și de HDIS_SUF la terminarea execuției ModX-ului curent.

Rolul TiLT este de a stoca într-o zonă dedicată de memorie tampon circulară (buffer circular), informații legate de evenimentele lansării sau terminării execuției ModX-urilor sistemului:

- valoarea ceasului timp-real (RTC) în momentul apariției evenimentului;
- starea HDIS din acel moment;
- valoarea contorului de execuție.

Fiind o subrutină a ModX-ului HDIS, TiLT este executat, la rândul său, în context strict, non-preemptiv.

8.3 Relația dintre componentele nucleului HARETICK

Figura 8-1 reprezintă intuitiv atât componentele de bază ale nucleului TR HARETICK, cât și relația dintre acestea. În figură sunt evidențiate cele două contexte de execuție ce constituie caracteristica distinctivă a HARETICK: contextul strict (HRT) și contextul lejer (SRT).

Relația dintre componentele HARETICK poate fi prezentată și prin scenariul ce presupune la un moment dat pornirea platformei țintă (sistemul de control încorporat) ce operează sub nucleul HARETICK, după care se încarcă în sistem o aplicație oarecare pentru a fi executată. Scenariul evidențiază următoarele evenimente:

1. În urma startării platformei țintă (sau după o operație de RESET), execuția procesorului este redirectată către zona de start a codului secvenței BOOT din memoria EPROM a sistemului.

Odată activat, task-ul BOOT încarcă în memoria sistemului (RAM) structurile de date și codul celorlalte componente ale nucleului, după care lansează inițializarea sistemului (apelează task-ul SYSINIT).

2. În faza de inițializare a sistemului, task-ul lejer SYSINIT setează valorile parametrilor de operare ai nucleului, variabilele globale, parametrii și variabilele celorlalte componente ale nucleului.

Lansarea task-ului SYSINIT corespunde practic cu activarea contextului lejer de operare al HARETICK. După inițializarea parametrilor sistemului, SYSINIT configurează și pornește ceasul de timp-real (RTC) care, prin intermediul întreruperii de timer apelează executivul TR al HARETICK, HDIS. Prin aceasta, contextul strict de operare al sistemului este, la rândul lui, activat.

3. Odată activat contextul strict, execuția SYSINIT este întreruptă de către HDIS, care identifică din Tabela de Dispatch, ModX-ul care urmează a fi lansat în execuție. În această fază de operare a sistemului, singurul ModX prezent în tabelă este planificatorul pentru contextul strict, HSCD. Prin urmare, HSCD este apelat de către HDIS.
Înainte de a ceda procesorul către HSCD, executivul HDIS apelează subrutina de raportare a evenimentelor sistem, TiLT, pentru a marca lansarea în execuție a lui HSCD.
4. Planificatorul de ModX-uri, HSCD, va aplica algoritmi de planificare non-preemptivă special concepuți, asupra setului de ModX-uri existente în momentul curent în cadrul sistemului (referiți în PDT – tabela cu descriptorii de proces). Planificarea ModX-urilor durează până în momentul în care HSCD a completat toate înregistrările din Tabela de Dispatch cu indentificatorii și momentele de start corespunzătoare ciclului curent de planificare.
Ca observație, menționăm ca HSCD va modifica și înregistrarea proprie din Tabela de Dispatch, setând pentru momentul de start, noua valoare la care este planificat HSCD după execuția ultimului ModX în cadrul ciclului de planificare.
5. Contextul de execuție strictă se va continua după terminarea execuției planificatorului HSCD, prin lansarea celorlalte ModX-uri din Tabela de Dispatch la momentele de timp corespunzătoare.
Printre aceste ModX-uri, aparținând configurației inițiale de operare a nucleului HARETICK, se regăsesc ModX-urile componentei DATALINK, ce asigură comunicația de comenzi și date cu un calculator gazdă.
În Figura 8-1 sunt reprezentate două bucle de execuție pentru contextul strict, subliniind faptul că fiecare ModX este lansat în execuție de către HDIS și că planificarea acestora se face ciclic, pe baza Tabelei de Dispatch, de către HSCD care se execută odată la fiecare ciclu.
6. În intervalele de timp neocupate de execuția contextului strict, se vor executa task-urile contextului lejer. Prima dată când controlul execuției revine contextului lejer, se continuă execuția task-ului SYSINIT care termină inițializările celorlalte task-uri (aparținând contextului lejer) și lansează planificatorul task-urilor lejere, SSCD.
7. SSCD are rolul de a planifica, în intervalele de timp neocupate de execuția contextului strict al HARETICK, task-urile lejere ale sistemului și ale aplicației (după ce se încarcă o anumită aplicație în sistem). În această fază a scenariului, task-urile lejere considerate de algoritmul de planificare preemptivă al SSCD sunt: task-urile lejere ale DATALINK, task-ul MONITOR, LOADER și STATREPO. Acestea sunt planificate și executate după un algoritm de planificare de tip "time-sharing", preemptiv, pe bază de priorități.
În Figura 8-1 este reprezentată bucla de execuție corespunzătoare contextului lejer al HARETICK, definită de task-ul de planificare SSCD.

9 Reprezentarea și gestionarea timpului și a ModX-urilor în cadrul sistemului

9.1 Timpul ca parametru esențial de operare al HARETICK

Timpul, ca și coordonată esențială a sistemelor TR stricte, trebuie să fie implicat în toate fazele de dezvoltare a acestora: specificare și verificare formală, programare, analiză, planificare și execuție, și implementare (așa cum se subliniază în secțiunea anterioară). În consecință, întreaga concepție și implementare a nucleului HARETICK ține cont de acest obiectiv principal.

În cele ce urmează vom discuta aspectele legate de reprezentarea și gestionarea timpului în cadrul nucleului TR HARETICK pentru sisteme de control încorporat și DSP. Acolo unde se impun particularizări, se vor exemplifica abordările din cadrul implementării HARETICK pe platforma Motorola DSP56307EVM [Motorola 95, Motorola 98, Motorola 99, Motorola 00].

Frecvența tactului de core a procesorului DSP56307 a fost setată la valoarea:

$$CoreCLK = 32 \text{ MHz.}$$

În cazul acestei arhitecturi, ciclul procesor ce definește execuția instrucțiunilor este egal cu o perioadă de tact, rezultând astfel granularitatea de timp minimă ca fiind egală cu durata ciclului procesor (vezi Capitolul 4):

$$\Delta t_{CLK} = 31,25 \text{ ns.}$$

Pe de altă parte, se va utiliza un ceas de timp-real (RTC) ce aplică tactului intern al DSP un factor de divizare (prescalare) egal cu 4. Rezultă pentru unitatea de timp sistem (unitatea de timp măsurată de RTC), valoarea:

$$\Delta t_{RTC} = 4 \cdot \Delta t_{CLK} = 125 \text{ ns.}$$

Astfel, HARETICK poate determina evenimentele din cadrul sistemului cu o frecvență maximă de $8 \text{ MHz} = CoreCLK / 4$, echivalentă cu 4 cicluri procesor.

Observație

Frecvența de lucru a procesorului și unitatea de timp sistem au fost stabilite la valorile de mai sus în implementarea HARETICK din considerente practice.

Astfel, pentru frecvența de lucru a DSP s-a căutat o valoare comodă din punctul de vedere al calculelor legate de timp. Inițial a fost încercată programarea blocului PLL (care generează practic frecvența de lucru internă a DSP pe baza unui oscilator extern, de frecvență $F_{ext} = 12.288 \text{ MHz}$), pentru valoarea $CoreCLK = 80 \text{ MHz}$. Arhitectura internă și parametrii de operare ai PLL nu au permis însă o astfel de setare.

Unitatea de timp sistem Δt_{RTC} a fost stabilită la 4 cicluri instrucțiune din considerente legate de relaxarea preciziei cu care se calculează timpii de execuție ai ModX-urilor, oferindu-se astfel un interval de toleranță destinat acoperii eventualelor erori și imprecizii ce derivă din calculul acestor timpuri.

9.2 Structurile destinate reprezentării și gestionării timpului în HARETICK

Timpul sistem este reprezentat și gestionat în cadrul HARETICK utilizând un set de structuri și mecanisme hardware și software, ce vor fi descrise pe scurt în continuare.

9.2.1 Timpul absolut sistem

Nucleul HARETICK implementează o structură pentru gestionarea timpului absolut, utilizând timpul sistem intern și setări ale operatorului de sistem, provenite din exterior (de la interfața INVERTA), prin intermediul unor comenzi specifice.

Timpul absolut sistem este stocat într-o variabilă sistem, `Sys_AbsTime`, de dimensiune egală cu 3 cuvinte procesor:

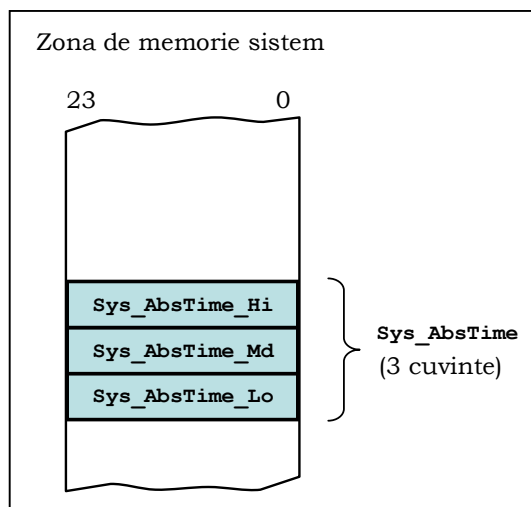


Figura 9-1. Reprezentarea timpului absolut sistem

Ținând cont că Motorola DSP56307 este o arhitectură pe 24 biți, timpul absolut sistem este exprimat pe 72 biți. În aceste condiții, și considerând setările pentru unitatea de timp sistem ($\Delta t_{RTC} = 125$ ns), timpul absolut pe care îl poate reprezenta și gestiona sistemul HARETICK se întinde pe o durată maximă:

$$\max \{ \text{Sys_AbsTime} \} = 18.718.157,355 \text{ ani.}$$

Parametrul sistem `Sys_AbsTime` poate fi setat (actualizat) de către operator prin intermediul unor comenzi specifice, lansate din mediul INVERTA, la valoarea timpului curent sau după necesități. În timpul operării sistemului HARETICK,

`Sys_AbsTime` este actualizată în mod constant, ciclic, de către ModX-ul HSCD (planificatorul contextului strict de execuție). Actualizarea timpului absolut sistem se face în cadrul fiecărei execuții a HSCD, cu valoarea acumulată a intervalului de timp corespunzător ultimului ciclu de planificare.

9.2.2 Ceasul timp-real

Ceasul timp-real (RTC) este practic elementul activ de gestionare a timpului în sistemul HARETICK. RTC are două roluri esențiale:

- (a) Contor de timp, utilizat pentru referiri ale componentelor sistemului la instanța curentă de timp sistem. Orice task, fie că aparține nucleului HARETICK, fie aplicației ce rulează la un moment dat în sistem, poate citi valoarea curentă a contorului RTC, având astfel la dispoziție o bază de timp real. Fiind vorba de un sistem TR (strict) destinat aplicațiilor critice, accesul oricărui task la baza de timp a sistemului reprezintă o facilitate extrem de utilă și necesară.
- (b) Singura sursă (predictibilă) de întreruperi.

Așa cum a fost subliniat în cadrul secțiunilor anterioare, abordarea pe care o propunem și o adoptăm în ceea ce privește dezvoltarea sistemelor TR stricte presupune restricționarea pe cât posibil a mecanismului de întreruperi.

În implementarea curentă a nucleului HARETICK, ceasul timp-real reprezintă singura sursă de întreruperi din cadrul sistemului. Timerele RTC sunt programate de către executivul TR al nucleului, HDIS, pentru a genera câte o întrerupere în momentul corespunzător lansării în execuție a fiecărui ModX, conform planificării calculate de HSCD și înscrise în Tabela de Dispatch.

Mai mult, RTC reprezintă o sursă predictibilă de întreruperi, în sensul că se poate determina oricând, și de către orice task, momentul apariției următoarei întreruperi. Acest lucru se poate realiza în două moduri: (i) prin citirea valorii înscrise la un moment dat în regiștrii de comparare ai RTC, sau (ii) prin citirea înregistrării din Tabela de Dispatch, corespunzătoare ModX-ului ce urmează a fi executat după cel curent. Înregistrarea respectivă conține informația de timp referitoare la momentul începerii execuției ModX-ului, corespunzând următoarei întreruperi RTC.

Pentru a ușura operarea și utilizarea ceasului timp-real, este de dorit o variantă bazată pe un contor de timp cu capacitate cât mai mare. Implementarea curentă a sistemului HARETICK se bazează pe arhitectura procesorului Motorola DSP56307 [Motorola 98], care prevede existența a 3 timere interne, pe 24 biți fiecare. Astfel, RTC poate fi implementat cu ajutorul a două timere cascade, capacitatea totală de contorizare a acestuia fiind prin urmare de 48 biți.

Figura 9-2 prezintă arhitectura RTC din HARETICK, bazată pe cascada a două timere pe 24 biți fiecare, `Timer0` și `Timer1`, cu numărare crescătoare. În aceste condiții, și considerând setările pentru unitatea de timp sistem ($\Delta_{RTC} = 125$ ns), timpul sistem pe care îl poate reprezenta și gestiona RTC se întinde pe o durată maximă:

$$\max \{RTC\} = 407,226 \text{ zile,}$$

iar capacitatea maximă a contorului `Timer0` al RTC este de:

$$\max \{Timer0\} \cong 2 \text{ sec.}$$

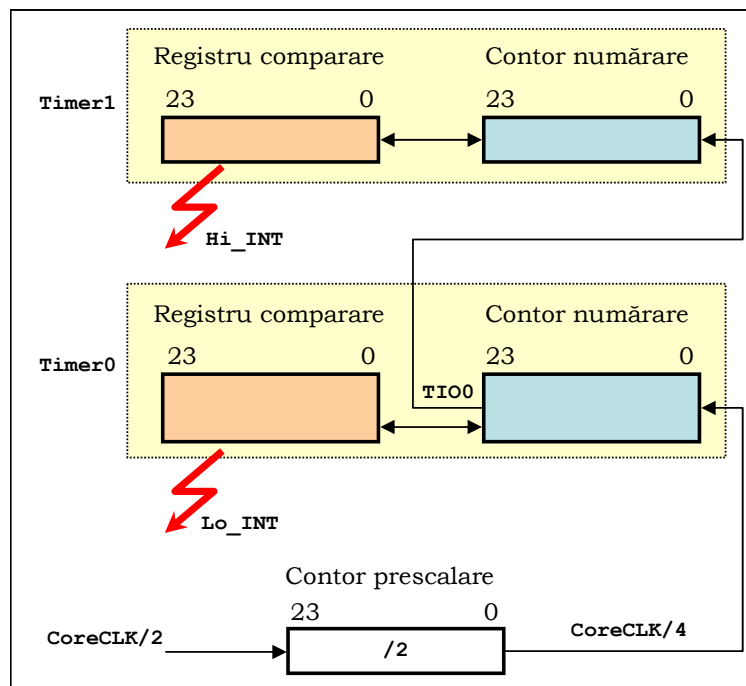


Figura 9-2. Ceasul timp-real (RTC) în HARETICK (variantea cu timere cascade)

Cascadarea celor două timere ale DSP56307 pentru obținerea unei capacități suficiente de contorizare a timpului real complică mecanismul de întreruperi al HARETICK, în sensul că introduce două întreruperi RTC în loc de una: `Lo_INT`, și `Hi_INT`. Întreruperile se activează atunci când valoarea contorului de numărare ajunge la valoarea din registrul de comparare corespunzător, pentru fiecare timer în parte. Mecanismul de întreruperi RTC va fi detaliat în cadrul prezentării executivului TR al HARETICK, HDIS, acesta fiind ModX-ul care lucrează în mod direct cu ceasul timp-real (Capitolul 10).

Pentru situațiile în care timerele procesorului țintă nu pot fi cascade sau nu sunt disponibile mai multe timere, o a doua variantă de implementare a RTC se poate lua în considerare (vezi Figura 9-3). Cel mai puțin semnificativ cuvânt (LSW) al RTC este reprezentat de contorul unui timer al procesorului, iar cel mai semnificativ cuvânt (MSW) este reprezentat printr-o variabilă dedicată, `Sys_RTC_MSW`, alocată în memoria de date sistem. Contorul `Timer0` este configurat pentru numărare în sens crescător, intrarea de tact fiind dată direct de tactul intern al procesorului (`CoreCLK`), căruia i se poate aplica un factor de divizare. În momentul în care are loc o depășire a capacității contorului `Timer0`, se generează o întrerupere, `RTC_OVR`, a cărei subrutină de tratare incrementează valoarea variabilei sistem `Sys_RTC_MSW`. Întreruperea RTC a HARETICK, utilizată pentru apelarea executivului este `RTC_INT`, care se activează în condițiile în care valoarea variabilei `Sys_RTC_MSW` a ajuns egală cu partea cea mai

semnificativă a instanței de start pentru următoarea planificare și, în același timp, contorul `Timer0` a ajuns egal cu partea cea mai puțin semnificativă a momentului următoarei planificări. `RTC_INT` este astfel, o întrerupere de comparare generată de `Timer0`. Și în această variantă de implementare, apare problema tratării unei întreruperi suplimentare, anume `RTC_OVR`.

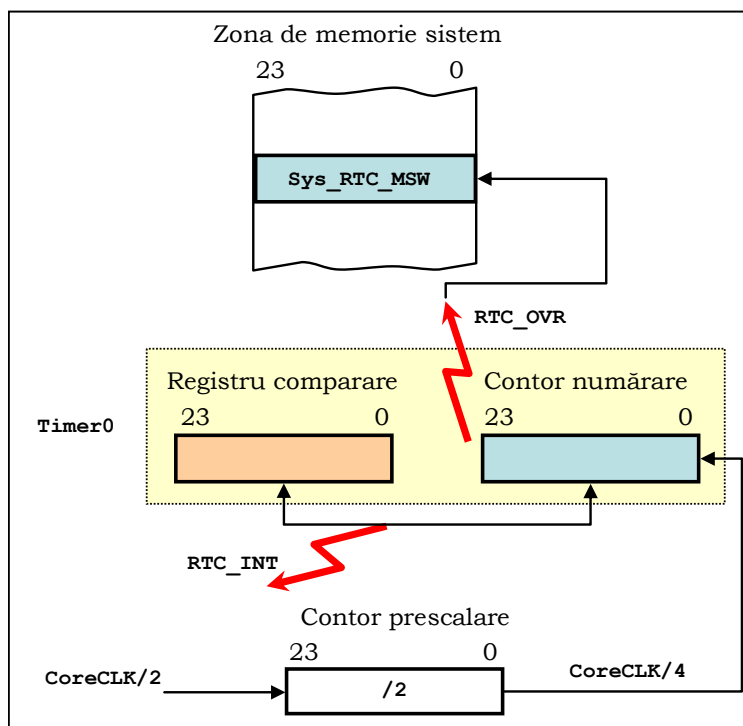


Figura 9-3. Ceasul timp-real (RTC) în varianta cu un singur timer

Gestionarea timpului în cadrul RTC începe în momentul activării contextului strict de execuție al HARETICK, operație inițiată de către task-ul SYSINIT. În acel moment valoarea timpului real sistem este:

$$t_{\text{RTC}} = t_0 = 0$$

Pe lângă executivul TR al nucleului (HDIS), ceasul timp-real mai este utilizat de către planificatorul contextului strict, HSCD, pentru calcularea și exprimarea timpilor de start ai ModX-urilor planificate. HSCD nu folosește însă în mod direct RTC și nu-i modifică valoarea, ci utilizează structuri proprii pentru exprimarea timpului de planificare. Acestea vor fi descrise în paragraful următor.

9.2.3 Timpul de planificare

Planificatorul contextului strict de execuție al HARETICK, anume ModX-ul HSCD, gestionează, la rândul său, timpul, dispunând de structuri și mecanisme proprii pentru aceasta.

Așa cum a fost detaliat în secțiunea anterioară, algoritmi de planificare propuși calculează planificarea ModX-urilor pe un interval de timp de forma:

$$T_{SCHED} = [0, T_{LCM}),$$

unde T_{LCM} reprezintă cel mai mic multiplu comun al perioadelor ModX-urilor din setul supus planificării.

În consecință, ModX-ul de planificare HSCD, utilizează așa-numitul *timp absolut de planificare*, t_{SCHED} , care este definit pe intervalul $[0, T_{LCM})$. Dat fiind faptul că lungimea intervalului de planificare (valoarea lui T_{LCM}) variază puternic cu perioadele și numărul ModX-urilor, putând ajunge la valori foarte mari, în implementarea HARETICK se utilizează o variabilă de capacitate suficientă (vezi Figura 9-4).

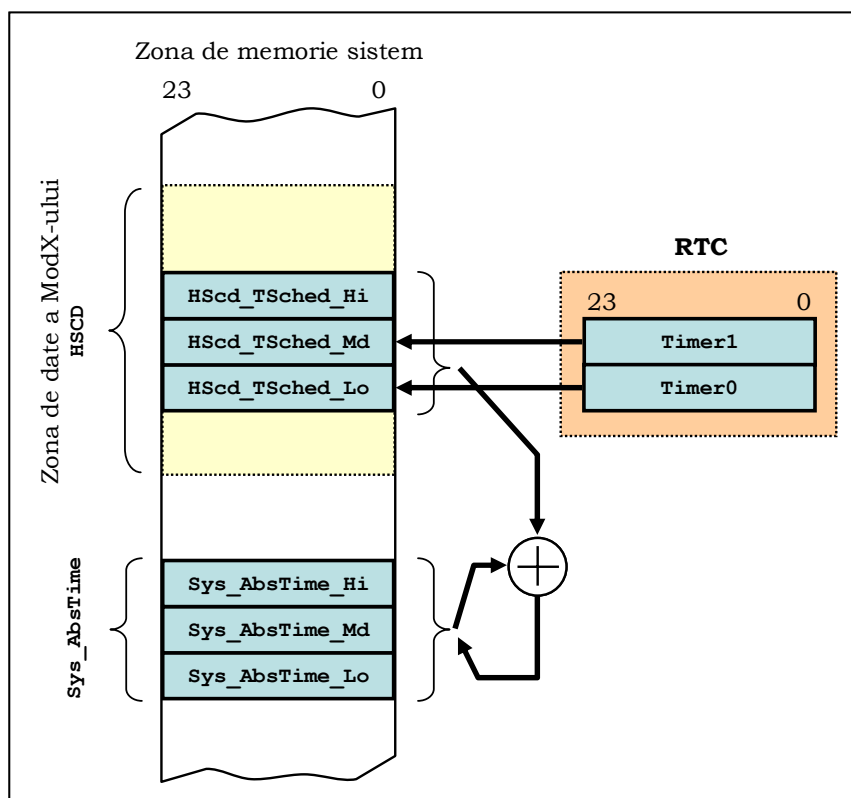


Figura 9-4. Structurile și mecanismele de gestionare a timpului absolut de planificare

Cum rezultă și din figură, timpul absolut de planificare (sau, simplu – *timpul de planificare*) este exprimat pe 3 cuvinte a câte 24 biți, rezultând o capacitate totală de 72 biți (identică cu cea corespunzătoare timpului absolut sistem). Figura 9-4 indică și faptul că timpul de planificare este sincronizat cu cele două timere ale RTC.

Timpul de planificare calculat de către algoritmul de planificare implementat de HSCD este apoi utilizat pentru completarea înregistrării corespunzătoare din Tabela de Dispatch. În tabelă, HDIS va înscrie, pe lângă identificatorul ModX-ului planificat, și valoarea timpului real sistem la care va fi lansat în execuție ModX-ul respectiv de către HDIS. Astfel, componenta temporală a unei înregistrări din Tabela de Dispatch va avea valoarea $HScd_TSched_Md:HScd_TSched_Lo$.

9.3 Mecanismele de sincronizare a structurilor de gestionare a timpului în HARETICK

Relațiile dintre cele trei tipuri de timpi utilizați în cadrul sistemului HARETICK, adică timpul absolut sistem, timpul real sistem (RTC) și timpul de planificare, sunt bine definite:

- Există o singură structură activă de generare și măsurare a timpului în cadrul sistemului: RTC, care gestionează timpul real sistem – baza de timp a sistemului.
- Timpul real sistem (t_{RTC}) este generat (modificat) doar de timerele procesorului și este referit de componentele de bază ale nucleului HARETICK: HDIS, HSCD, etc. El poate fi de asemenea citit de către alte task-uri (stricte sau lejere) aparținând nucleului sau aplicației curente.
- Inițializarea RTC este efectuată de către task-ul SYSINIT, o singură dată, la pornirea sistemului
- Timpul absolut sistem poate fi setat și referit la nivelul operatorului, prin intermediul unor comenzi specifice, și nu este utilizat de nucleu la operații de sincronizare a execuțiilor sau pentru planificarea ModX-urilor.
- ModX-ul de planificare a contextului strict de execuție, HSCD, utilizează timpul de planificare (t_{SCHED}), sincronizându-l cu timpul real sistem (RTC) la fiecare execuție. La începutul fiecărui ciclu de planificare, cu ocazia lansării sale în execuție, HSCD va efectua următoarele acțiuni:
 - Verifică relația de corespondență dintre timpul de planificare și RTC, și în caz că e necesar, efectuează sincronizarea:

$$\begin{cases} \text{HScd_TSched_Md} \leftarrow \text{Timer1} \\ \text{HScd_TSched_Lo} \leftarrow \text{Timer0} \end{cases}$$

- Sincronizează timpul absolut sistem cu timpul de planificare:

$$\begin{cases} \text{Sys_AbsTime_Hi} \leftarrow \text{Sys_AbsTime_Hi} + \text{HScd_TSched_Hi} \\ \text{Sys_AbsTime_Md} \leftarrow \text{Sys_AbsTime_Md} + \text{HScd_TSched_Md} \\ \text{Sys_AbsTime_Lo} \leftarrow \text{Sys_AbsTime_Lo} + \text{HScd_TSched_Lo} \end{cases}$$

- Aplică algoritmul de planificare non-preemptivă a setului de ModX-uri existent în sistem, pentru completarea înregistrărilor din Tabela de Dispatch cu planificările ciclului curent. Astfel, dacă la momentul curent al planificării, $t_{SCHED,i}$ a fost selectat pentru execuție ModX-ul M_j , înregistrarea curentă din Tabela de Dispatch va conține următoarele elemente (MSW semnifică cel mai semnificativ cuvânt, iar LSW – cel mai puțin semnificativ):

1. Identificatorul ModX - ului : M_j
2. MSW al momentului planificării $\left(t_{st}^{M_j}\right)$: $\left(t_{SCHED,i}\right) \cdot HScd_TSched_Md$
3. LSW al momentului planificării $\left(t_{st}^{M_j}\right)$: $\left(t_{SCHED,i}\right) \cdot HScd_TSched_Lo$

- După completarea Tabelei de Dispatch, HSCD configurează înregistrarea specială din tabelă ce face referire la acesta, cu timpul de start al următoarei sale execuții – specificând începutul unui nou ciclu de planificare.
- Momentele de start ale execuției ModX-urilor planificate în cadrul unui ciclu de planificare vor fi referite pe rând, din Tabela de Dispatch, de către executivul nucleului, HDIS. Înaintea lansării ModX-ului curent, HDIS programează registrele de comparare ale RTC cu cele două cuvinte (MSW, respectiv LSW) ale timpului de start corespunzător ModX-ului ce urmează celui curent.

Operațiile referitoare la timp, desfășurate de componentele HARETICK, vor fi detaliate în cadrul secțiunilor ce descriu componentele respective în cadrul lucrării.

9.4 Reprezentarea ModX-urilor în cadrul HARETICK

Task-urile cu specificații temporale stricte ale unui sistem sau aplicații timp-real trebuie tratate cu atenție specială. Modelul propus pentru astfel de task-uri în cadrul lucrării de față este *ModX-ul (Modul eXecutabil)*, ale cărui caracteristici au fost studiate în secțiunea precedentă: un task TR periodic, modular, cu specificații complete și stricte de comportare în timp și cu planificare și execuție în context non-preemptiv.

Nucleul de operare HARETICK implementează un set de structuri și mecanisme destinate reprezentării, planificării și execuției ModX-urilor. Dat fiind tipul sistemului considerat – nucleu de operare TR strict pentru platforme bazate pe DSP sau platforme de control digital încorporat (embedded systems), reprezentarea ModX-urilor se face pe baza unei arhitecturi pe cât posibil statice.

ModX-urile gestionate la un moment dat de nucleul HARETICK se împart în două categorii: ModX-uri sistem și ModX-uri aplicație. HARETICK utilizează structuri distincte pentru cele două categorii de ModX-uri, deși din punctul de vedere al tipului de sistem implementat (sistem încorporat), această diferențiere nu este obligatorie (de exemplu, din acest motiv nu a fost prevăzut nici un mecanism de control al accesului task-urilor la memoria sistemului, rezolvarea acestei chestiuni fiind responsabilitatea programatorului de sistem/aplicație).

ModX-urile sistem sunt încărcate de către task-ul BOOT la pornirea platformei, alocându-se pentru ele un set de resurse distincte (de exemplu, în cazul memoriei, există o zonă dedicată ModX-urilor sistem – zona sistem). ModX-urile sistem vor fi discutate separat în cadrul lucrării de față, făcând parte integrantă din nucleul HARETICK.

În continuare vom discuta cazul general al unui ModX ce aparține unei aplicații oarecare, încărcate pe platforma țintă ce operează sub nucleul HARETICK.

O primă structură implicată în gestionarea ModX-urilor aplicației este tabela ce conține graful direcționat aciclic de program al aplicației (*PDAGT – Program Directed Acyclic Graph Table*). Fiecare înregistrare a tabelii constituie o reprezentare a unui nod al grafului (adică a unui ModX al aplicației):

- identificatorul ModX-ului: `PID`;
- numărul de nivel în cadrul grafului;
- numărul de părinți ai nodului;
- referință către părintele (părinții) nodului;
- numărul de fii ai nodului;
- referință către fiul (fiii) nodului.

Până în momentul de față a fost studiată reprezentarea aplicațiilor ce rezultă în grafuri de program de tip arbore (fiecare nod are un singur părinte și unul sau mai mulți fii). Ca preocupare de viitor, avem în vedere conceperea de structuri (statice) care să poată reprezenta grafuri de program complexe (un nod poate avea mai mulți părinți și mai mulți fii).

Tabela PDAGT este creată de către task-ul LOADER al nucleului HARETICK, în faza de încărcare a aplicației în cadrul sistemului țintă, prin intermediul interfeței de comunicație cu mediul INVERTA, implementată de DATALINK. Translatarea structurii aplicației, dintr-o reprezentare dinamică de tip graf, într-una statică de tip tabelă este posibilă datorită operațiilor de analiză a aplicației dezvoltate în mediul INVERTA, pe calculatorul gazdă. Astfel, printre informațiile suplimentare furnizate de INVERTA la încărcarea aplicației în sistemul HARETICK, se găsește și un set de parametri ce definesc cantitativ graful aplicației (numărul total de noduri – ModX-uri, numărul de nivele ale grafului, etc.).

9.4.1 Reprezentarea ModX-urilor în memorie

Tot în cadrul operațiilor de încărcare a aplicației în sistem, task-ul LOADER alocă și completează, pentru fiecare ModX, un set de structuri de memorie, într-o zonă dedicată: zona aplicație.

Pentru fiecare ModX, sunt prevăzute următoarele structuri de memorie:

- Zona de cod (text), în care se încarcă secvența de cod compilat a ModX-ului;
- Zona parametrilor de ieșire, ce conține valorile fiecărui parametru de ieșire al ModX-ului.

De asemenea, pentru întreaga aplicație, este alocată o zonă de memorie pentru variabilele globale, și o zonă de memorie comună pentru parametrii de intrare și variabilele locale ale ModX-urilor, denumită *Heap*.

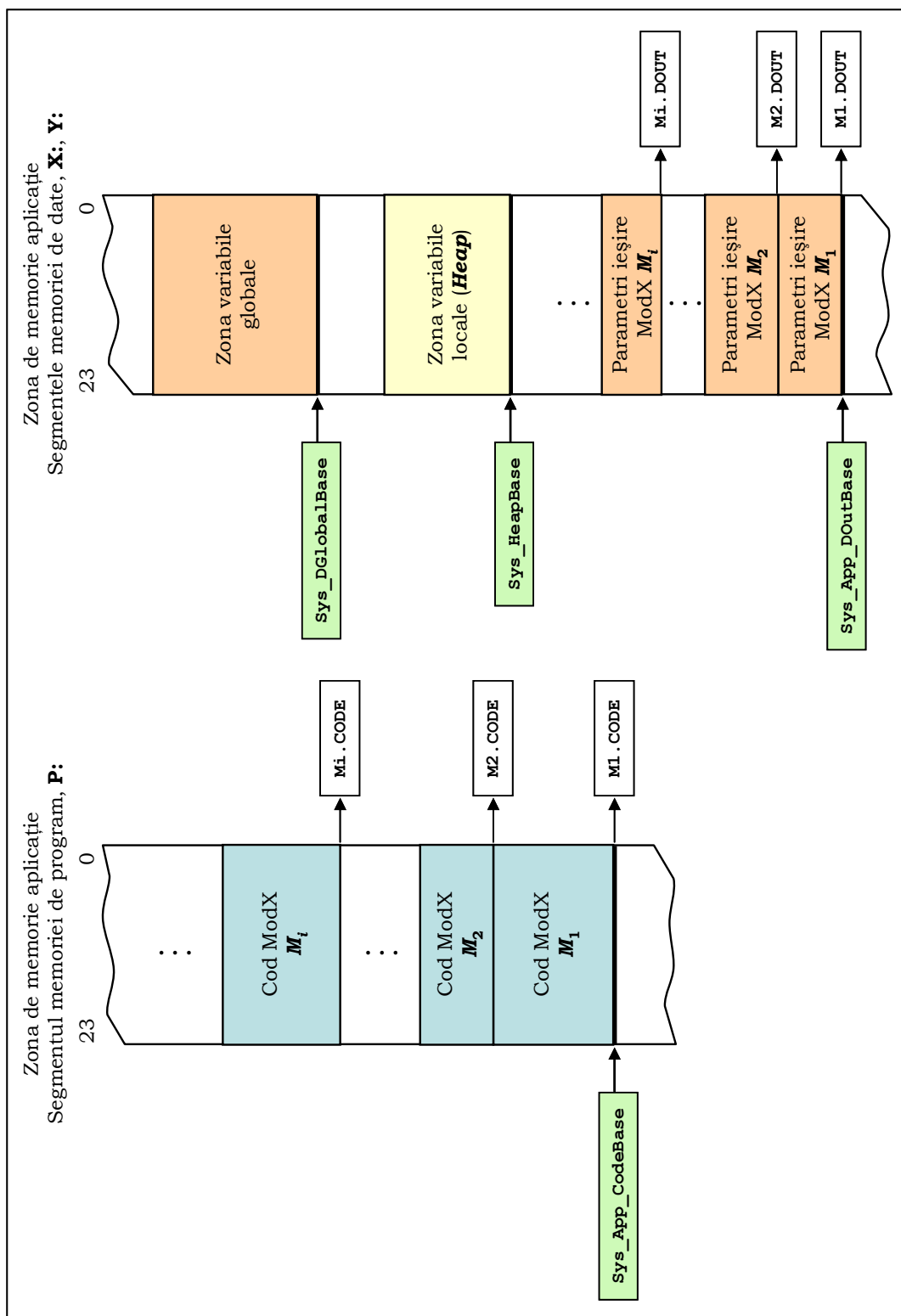


Figura 9-5. Reprezentarea ModX-urilor în memorie

La fel ca în cazul tabelii PDAGT, toate structurile utilizate de sistem pentru reprezentarea în memorie a ModX-urilor sunt definite și alocate static, pe baza informațiilor suplimentare legate de dimensiunile maxime ale fiecărei structuri, primite de HARETICK de la mediul de dezvoltare INVERTA, la încărcarea aplicației în sistem. De exemplu, pentru zona *Heap*, LOADER-ul va alocă o zonă de memorie de dimensiune cel puțin egală cu cea maximă necesară oricărui ModX din cele ale aplicației pentru lucrul cu variabilele sale locale (și parametrii de intrare).

Figura 9-5 prezintă structurile necesare nucleului HARETICK pentru reprezentarea în memorie a ModX-urilor aplicației. Codul fiecărui ModX este înscris în segmentul de memorie program al procesorului DSP56307 (memoria **P:**), începând de la o adresă de bază, `Sys_App_CodeBase`, care constituie un parametru sistem. În același timp, LOADER-ul reține adresele de start ale zonei de cod pentru fiecare ModX ($M_i.CODE$), pentru a le înscrie în celelalte tabele care servesc nucleului la gestionarea ModX-urilor, și care vor fi discutate în paragrafele următoare.

Zona parametrilor de ieșire ai fiecărui ModX este alocată în segmentul de memorie de date a DSP56307 (memoria **X:** sau **Y:**), începând de la o adresă de bază, `Sys_App_DOutBase`, care este, de asemenea, un parametru sistem.

Zona *Heap* este alocată de către LOADER, pe considerentele de dimensiune prezentate mai sus, începând de la adresa `Sys_HeapBase`, și, în manieră similară se procedează cu zona variabilelor globale ale aplicației, folosind ca adresă de bază, `Sys_DGlobalBase`.

9.4.2 Tabela de simboluri

În aceeași fază de încărcare a aplicației în cadrul sistemului HARETICK, task-ul LOADER creează tabela de simboluri, în care este înscrisă corespondența dintre fiecare parametru sau variabilă a ModX-urilor și locația de memorie corespunzătoare.

Astfel, de exemplu, pentru primul parametru de ieșire al unui ModX oarecare M_i (adică pentru parametrul alocat la startul zonei $M_i.DOUT$), LOADER-ul va trece în tabela de simboluri numele parametrului și adresa corespunzătoare valorii $M_i.DOUT$.

Pentru parametrii de intrare sau variabilele interne ale unui ModX, tabela de simboluri va conține numele variabilei și deplasamentul stabilit de către LOADER în cadrul zonei *Heap* pentru variabila respectivă. Va rezulta astfel o suprapunere a adreselor din tabela de simboluri pentru aceste variabile, prin maparea lor într-o aceeași zonă fizică de memorie.

9.4.3 Tabela descriptorilor de procese

Tabela descriptorilor de procese (*PDT – Process Descriptor Table*) este creată și completată tot de către LOADER, în faza de încărcare a aplicației în sistem. PDT conține identificarea fiecărui ModX, împreună cu caracteristicile sale (atribute, parametri temporali, etc.).

Figura 9-6 prezintă structura PDT și a unei înregistrări, care are o dimensiune de 13 cuvinte (de 24 biți în cazul arhitecturii DSP56307), și corespunde unui ModX oarecare. Semnificația elementelor înregistrării PDT este prezentată în Tabelul 9-1.

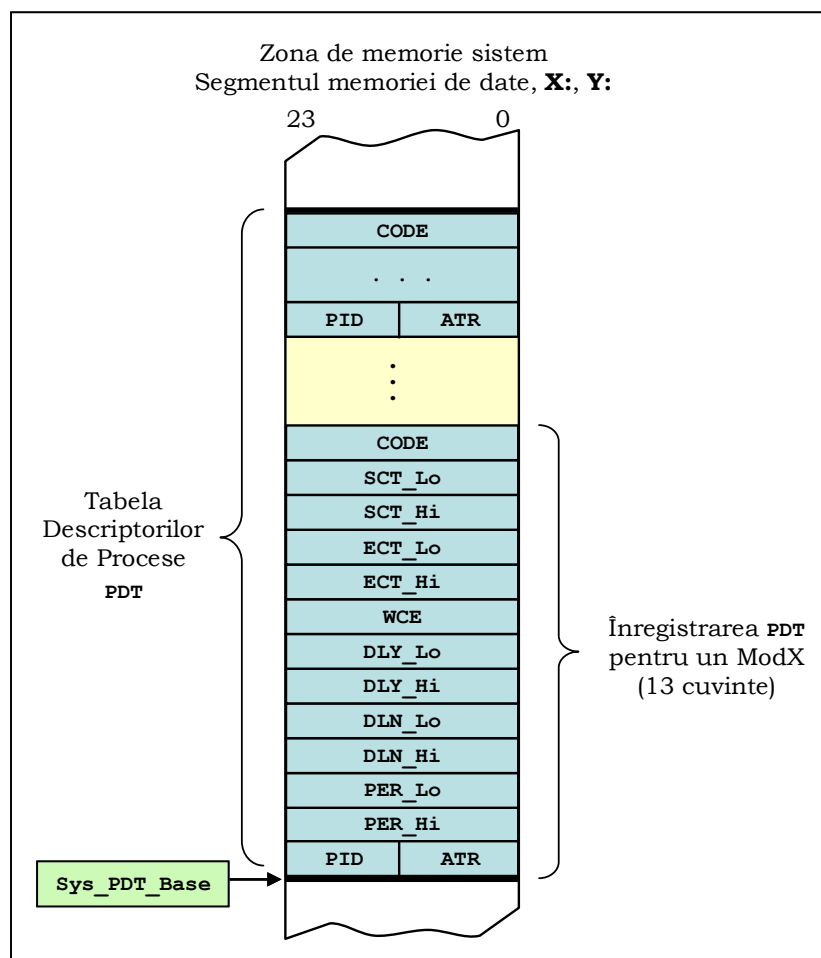


Figura 9-6. Tabela Descriptorilor de Procese (PDT)

Tabelul 9-1. Semnificația elementelor unei înregistrări din PDT (vezi și Capitolul 4)

Offset	Denumire	Semnificație
0	PID (12 biți)	Identificatorul de proces, pentru ModX-ul M_i .
	ATR (12 biți)	Atribute suplimentare ModX (e.g. "FIXED", etc.).
1:2	PER_Hi:PER_Lo	Perioada ModX-ului (period), $T_{pr}^{M_i}$.
3:4	DLN_Hi:DLN_Lo	Intervalul limită (deadline), $T_{dl}^{M_i}$.
5:6	DLY_Hi:DLY_Lo	Intervalul de întârziere (delay), $T_{dy}^{M_i}$.
7	WCE	Durata maximă de execuție (execution time), $T_{ex}^{M_i}$.
8:9	ECT_Hi:ECT_Lo	Contorul de execuții (execution count), N^{M_i} .
10:11	SCT_Hi:SCT_Lo	Contorul de planificări (scheduling count) – utilizat de algoritmul de planificare al HSCD.
12	CODE	Adresa de start a zonei de cod a ModX-ului M_i .

Majoritatea parametrilor din PDT corespunzători unui ModX sunt utilizați de către HSCD, în cadrul algoritmului de planificare non-preemptivă a contextului strict de execuție. PDT este referită de asemenea de către executivul TR al HARETICK, HDIS, pentru identificarea ModX-ului ce urmează a fi lansat în execuție (PID), pentru încărcarea adresei de start a zonei de cod a ModX-ului ($CODE$) și pentru accesarea (în citire-scriere) a contorului de execuții (ECT).

Din punct de vedere particular, al implementării HARETICK pe platforma Motorola DSP56307, structura PDT permite următoarele valori maxime pentru parametrii sistem și ai ModX-urilor:

- Dimensiune PID : 12 biți, rezultând un total de 4 K (4096) ModX-uri ce pot fi gestionate de către HARETICK;
- Dimensiune totală PDT: 100 înregistrări, rezultând un total de 100 ModX-uri ce pot fi gestionate de către sistem;
- Perioada unui ModX, PER , e exprimată pe 2 cuvinte (48 biți), rezultând un interval maxim de timp de 407,226 zile;
- Același interval maxim de timp poate fi exprimat pentru intervalul limită al planificării (DLN) și pentru intervalul de întârziere (DLY) ale ModX-ului: 407,226 zile;
- Durata maximă de execuție a ModX-ului (WCE) poate ajunge la 2 sec (aproximativ), parametrul fiind exprimat pe un cuvânt procesor (24 biți);
- Contorul de execuții al ModX-ului (ECT) are o dimensiune de 2 cuvinte (48 biți), rezultând un număr maxim de peste $281 \cdot 10^{12}$ execuții ale ModX-ului ce pot fi gestionate de sistem;
- Contorul de planificări (SCT) poate exprima, la rândul lui, peste $281 \cdot 10^{12}$ planificări ale ModX-ului.

Atributul "FIXED" specificat în Tabelul 9-1 ca exemplificare pentru parametrul ATR al ModX-urilor se referă la cerința de execuție fixă a unui ModX în cadrul perioadelor acestuia (cu alte cuvinte, față de momentul de început al fiecărei perioade, ModX-ul respectiv se va executa cu o aceeași întârziere). Această problemă a fost tratată din punct de vedere teoretic și exemplificată, în Capitolul 5.

9.5 Operarea de principiu a ModX-urilor în cadrul HARETICK

Ca linii generale, pentru satisfacerea cerințelor de predictibilitate impuse de aplicațiile critice, HARETICK utilizează structuri statice, limitate în timp și spațiu. Pentru fiecare aplicație încărcată în sistem se vor cunoaște parametrii maximi de operare ai acesteia: timp, resurse sistem, memorie ocupată, etc.

În cele ce urmează, vom exemplifica operarea sistemului pentru cazul a două ModX-uri, M_i și M_j , care aparțin unei aplicații oarecare, încărcată pe platforma sistemului. Presupunem că M_i și M_j se află într-o relație de precedență de forma: $M_j \rightarrow M_i$, precedență rezultată ca urmare a dependenței de control și de date a aplicației. Figura 9-7 ilustrează acest scenariu, reprezentând cele două noduri (ModX-uri) ale grafului aplicației, împreună cu dependențele de program corespunzătoare. Fiecare nod al grafului va avea specificații complete: numele (M_j ,

M_i), parametrii de intrare (paranteza superioară denumirilor ModX-urilor), parametrii de intrare (paranteza inferioară), parametrii de comportare temporală, etc. De exemplu, ModX-ul M_i are 3 parametri de intrare ($param1$, $param2$ și $param3$) și un parametru de ieșire ($param3$). Primele două intrări ale lui M_i provin de la parametrii de ieșire ai lui M_j (de unde rezultă și dependența de date dintre cele două), iar $param3$ reprezintă în același timp, parametru de intrare și de ieșire, pentru M_i .

Posibilitatea ca un ModX oarecare să aibă parametri care constituie în același timp intrare și ieșire, derivă din modelul conceput pentru task-urile TR stricte: prin acest mecanism ele își pot transmite informații de stare, de la o execuție la alta (adică de la o perioadă la alta).

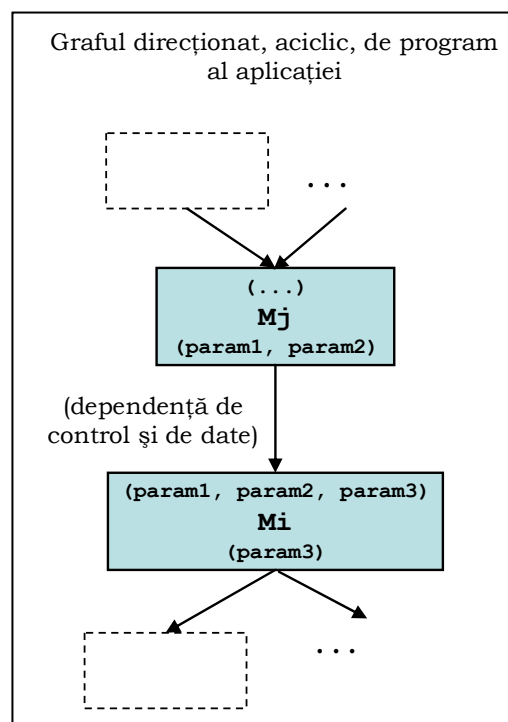


Figura 9-7. Exemplificare cu porțiunea de graf al aplicației ce conține cele două ModX-uri, M_j și M_i

Presupunem de asemenea, în cadrul acestui scenariu, că secvența de program a lui M_i lucrează cu un număr oarecare de variabile locale (de exemplu, 10), M_i ocupând, din acest punct de vedere, spațiul maxim de memorie dintre toate ModX-urile aplicației. Următorii pași generali vor fi întâlniți în operarea nucleului HARETICK, cu privire la execuția celor două ModX-uri din Figura 9-7:

- 0) Aplicația a fost încărcată în sistem, structurile de date corespunzătoare au fost alocate și inițializate (PDAGT, PDT, zonele de cod și "Data Out" pentru M_j și M_i , Heap-ul pentru variabilele locale și parametrii de intrare, zona de variabile globale, tabela de simboluri, etc.).

Pentru zona de Heap, task-ul LOADER al nucleului va alocă, pentru toate ModX-urile aplicației, o dimensiune de memorie de 13 locații: 10 cuvinte pentru

variabilele locale ale lui M_i și 3 cuvinte pentru parametrii săi de intrare. Această decizie a LOADER-ului rezultă din ipoteza că M_i ocupă spațiul maxim de memorie pentru variabilele locale și parametrii de intrare, dintre toate ModX-urile aplicației.

În cadrul contextului strict de execuție, HSCD a planificat cele două ModX-uri, M_j și M_i , pentru execuție la momentele $t_{st,k}^{M_j}$, respectiv, $t_{st,m}^{M_i}$, unde k reprezintă perioada execuției lui M_j și m perioada execuției lui M_i . Ca rezultat al planificării, Tabela de Dispatch va avea o configurație de tipul celei din Figura 9-8. Ca observație, momentul de start al ModX-ului M_i nu poate apărea mai devreme de terminarea execuției lui M_j , adică:

$$t_{st,m}^{M_i} \geq t_{st,k}^{M_j} + T_{ex}^{M_j} = t_{st,k}^{M_j} + M_j \cdot WCE$$

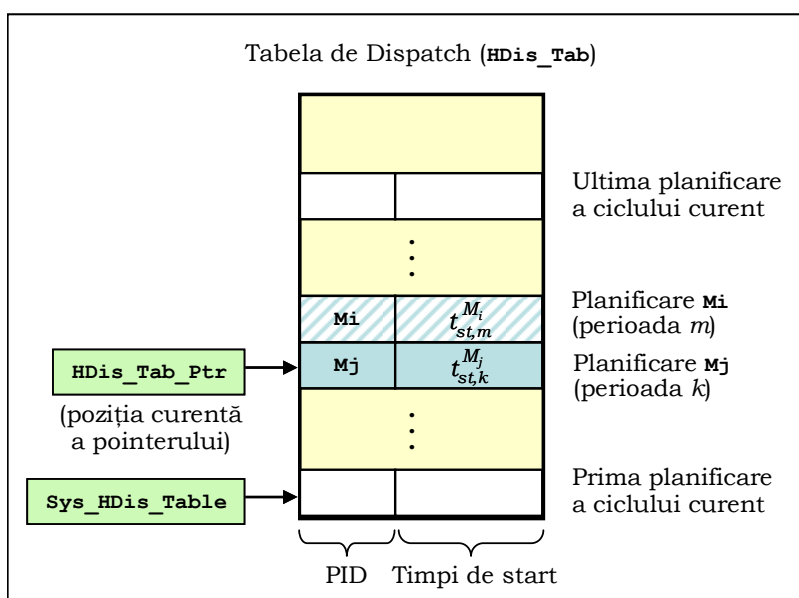


Figura 9-8. Configurația Tabelei de Dispatch pentru scenariul considerat

Presupunem că sistemul tocmai termină execuția ModX-ului M_j . Prin urmare, pointerul Tabelei de Dispatch ($HDis_Tab_Ptr$) indică spre M_j (ModX-ul curent în execuție).

- 1) Secvența de instrucțiuni a lui M_j se termină în mod obligatoriu cu apelarea componentei sufix a executivului TR al HARETICK, $HDis_SUF$ (vezi Subcapitolul 8.2.8 și Capitolul 10). Instrucțiunea de apel la $HDis_SUF$ înlocuiește astfel, instrucțiunile de tip `return` sau `RTS` ("ReTurn from Subroutine"), care încheie în mod uzual subrutinele de program (procedurile, funcțiile sau programele principale).

Ca urmare, se activează HDIS, care actualizează pointerul Tabelei de Dispatch, $HDis_Tab_Ptr$, astfel încât acesta va indica spre M_i . De asemenea, $HDis_SUF$ verifică dacă mai este timp suficient, din momentul curent (al

terminării execuției efective a lui M_j) până la momentul $t_{st,m}^{M_i}$, pentru a reda controlul contextului lejer de execuție. În caz contrar, $HDIS_SUF$ inițiază o buclă de așteptare a întreruperii RTC pentru $t_{st,m}^{M_i}$.

- 2) La momentul $t_{st,m}^{M_i}$, RTC activează întreruperea de timer, având ca rezultat lansarea lui $HDIS_PRE$ în execuție. Acesta programează regiștrii de comparare ai timerelor RTC pentru momentul startării execuției ModX-ului ce urmează lui M_i ca planificare, în Tabela de Dispatch. În continuare $HDIS_PRE$ verifică valoarea contorului de execuții ($M_i.ECT$) al lui M_i , existând trei posibilități:

- (2.1) $M_i.ECT = ECT_CONT$: ModX-ul M_i are specificat (momentan) un număr continuu de execuții. Ca rezultat, HDIS nu va modifica acest parametru și va lansa în execuție pe M_i .
- (2.2) $M_i.ECT$ are o valoare finită. În acest caz, HDIS va decrementa cu o unitate contorul de execuții și apoi va lansa în execuție pe M_i .
- (2.3) $M_i.ECT = 0$: ModX-ul M_i este un *ModX fantomă* (vezi și discuția din secțiunea anterioară). Ca rezultat, HDIS lasă nemodificat contorul de execuții și nu va lansa în execuție pe M_i , ci va apela pe $HDIS_SUF$ (ca și când s-ar fi terminat execuția lui M_i).

- 3) Odată activat ModX-ul M_i , înainte de a începe execuția instrucțiunilor din zona de cod proprie, acesta va copia în locațiile din zona Heap corespunzătoare parametrilor săi de intrare, valorile parametrilor de ieșire ai ModX-urilor față de care M_i prezintă dependențe de date. Referirea parametrilor de ieșire se face pe baza tabelii de simboluri a HARETICK, care indică pentru fiecare parametru, adresa din zona "Data Out" corespunzătoare ModX-ului de care aparține.

Astfel, în scenariul de față, M_i va copia în cele trei locații de memorie asignate de LOADER în Heap pentru parametrii săi de intrare ($Heap.param1$, $Heap.param2$ și $Heap.param3$), valorile $M_j.param1$, $M_j.param2$, respectiv, $M_i.param3$. Cele trei valori sunt regăsite de M_i , prin referirea la zona $M_j.DOut$ pentru primii doi parametri și, pentru cel de-al treilea, la zona $M_i.DOut$, utilizând adresele corespondente din tabela de simboluri a HARETICK.

Observație

Motivația acestui mecanism de copiere a parametrilor de intrare ai unui ModX înaintea execuției secvenței sale propriu-zise de cod, derivă din necesitatea obținerii unei predictibilități maxime de operare a sistemului HARETICK.

Cum numărul parametrilor de intrare/ieșire diferă de la un ModX la altul într-un mod transparent executivului TR al sistemului, a-l implementa pe acesta să efectueze operațiile de copiere a valorilor momentane, sau utilizarea unei stive, reprezintă soluții extrem de ineficiente și surse de impredictibilitate majoră în cadrul sistemului.

Pe de altă parte, includerea acestor operații în codul ModX-urilor, cu calcularea corespunzătoare a duratei maxime de execuție ($M_i.WCE$) astfel încât să cuprindă și timpii necesari copierii parametrilor, conferă sistemului o mai mare flexibilitate, și, mai ales, *nu îi afectează predictibilitatea*.

Operațiile de copiere a valorilor parametrilor de intrare pentru un ModX se pot specifica, fie explicit – prin scrierea secvenței respective de instrucțiuni chiar la începutul codului program al ModX-ului, fie implicit, de către compilator (de exemplu) – prin inserarea instrucțiunilor la începutul secvenței de cod compilat al ModX-ului.

Din cele de mai sus se desprinde și motivația structurilor de memorie care reprezintă zonele cu parametrii de ieșire ai fiecărui ModX, "Data Out", propuse în proiectul HARETICK. Mai mult, cu ajutorul acestor structuri se garantează păstrarea valorilor pe care un ModX dorește să le comunice (altor ModX-uri, sau lui însuși), de la execuția curentă, până la următoarea execuție (perioadă) a sa.

- 4) Execuția lui M_i se continuă, după copierea parametrilor săi de intrare în Heap, cu secvența program proprie, stocată în zona de cod ($M_i.CODE$) de către LOADER, și referită în cadrul înregistrării corespunzătoare din tabela descriptorilor de procese (PDT).

La sfârșitul execuției, parametrul `param3`, alocat în zona "Data Out" a lui M_i ($M_i.param3$) va conține noua valoare, rezultată în urma calculului efectuate de M_i . Execuția lui M_i se finalizează prin apelarea directă a lui `HDIS_SUF`.

9.6 Concluzii

Importanța coordonatei temporale pentru sistemele TR ce implementează aplicații critice este subliniată în cadrul nucleului HARETICK prin utilizarea și gestionarea a trei structuri distincte de reprezentare a timpului: `Sys_AbsTime` pentru timpul absolut sistem, ceasul timp-real (RTC) bazat pe contoare ale timerelor procesor, și timpul de planificare, `HScd_TSched`. Cele trei structuri sunt sincronizate între ele cu ajutorul ModX-urilor sistem (HDIS și HSCD), și sunt în strictă dependență de tactul intern al procesorului țintă.

Întreruperea de comparare RTC este, în stadiul actual de dezvoltare a sistemului, singura sursă admisă de întreruperi, fiind de asemenea predictibilă. Întreruperea RTC apelează în mod direct executivul HARETICK (mai precis, componenta prefix, `HDIS_PRE`), pentru lansarea în execuție a ModX-ului planificat la momentul apariției întreruperii.

În cadrul nucleului HARETICK, ModX-urile sunt reprezentate cu ajutorul unor structuri și mecanisme specifice:

- "PDAGT" (Program Directed Acyclic Graph Table), tabela ce conține graful direcționat, aciclic, de program al aplicației;
- Zona de cod ("CODE") a ModX-ului;
- Zona parametrilor de ieșire ("Data Out");
- Zona variabilelor globale ale aplicației;

- Zona variabilelor locale și ai parametrilor de intrare ai ModX-ului în execuție ("Heap");
- Tabela de simboluri;
- Tabela descriptorilor de procese ("PDT").

ModX-urile sunt executate în cadrul contextului strict al HARETICK, fiind planificate de către ModX-ul sistem HSCD. Activarea și terminarea ModX-urilor este gestionată de către executivul TR al nucleului, HDIS.

La începutul execuției fiecărui ModX, are loc în mod obligatoriu operația de copiere a valorilor parametrilor de ieșire, din zonele "Data Out" ale ModX-urilor de care depinde cel curent, în locațiile parametrilor de intrare, prevăzute în "Heap". Timpul de execuție a instrucțiunilor implicate de această operație se vor adăuga la WCET-ul ModX-ului.

Comunicația inter-task-uri este realizată prin transmiterea parametrilor de intrare/ieșire, cu mecanismul descris anterior. Datorită structurii și reprezentării ModX-urilor (care, practic, implementează operații atomice), și a mecanismului de comunicație dintre ele, controlul și sincronizarea acceselor la resurse concurente nu mai reprezintă o problemă. Un alt aspect interesant legat de acest subiect îl constituie facilitatea de transmitere a stării unui ModX, de la o execuție a sa la alta, prin intermediul unui același parametru care este și de intrare, și de ieșire.

La sfârșitul execuției fiecărui ModX, se găsește instrucțiunea de apel la HDIS, care, practic înlocuiește în mod echivalent instrucțiunile utilizate uzual în acest caz (`return`, `exit`, `end`, `RTS`, etc.).

Din punctul de vedere al cerințelor impuse de arhitectura nucleului HARETICK, calcularea timpului maxim de execuție a ModX-urilor (WCET) implică următoarele elemente:

- durata de execuție a celei mai lungi căi de program din codul propriu-zis al ModX-ului;
- durata secvenței de instrucțiuni, de la începutul execuției ModX-ului, care efectuează operațiile de copiere a parametrilor de ieșire;
- durata instrucțiunii de apel la `HDIS_SUF`, de la sfârșitul secvenței de cod a ModX-ului.

10 Funcționarea de principiu a sistemului

10.1 Contextele de execuție sistem și comutarea acestora

Nucleul HARETICK pune la dispoziția aplicațiilor TR o serie de structuri și mecanisme, menite să maximizeze predictibilitatea operării și să garanteze respectarea termenelor stricte impuse task-urilor în faza de proiectare/specificare:

- ModX-urile sunt modele ale task-urilor TR stricte, periodice, modulare;
- Planificarea ModX-urilor se efectuează într-o manieră complet predictibilă, aplicând algoritmi non-preemptivi, special concepuți;
- Execuția ModX-urilor are loc într-un context strict, conform planificării, fiind controlată de către executivul TR al nucleului;
- Operarea executivului TR (HDIS) se bazează pe întreruperea de ceas sistem (RTC) – singura întrerupere existentă în sistem.

Contextul de execuție descris mai sus este denumit *contextul strict* al nucleului HARETICK (sau, *contextul HRT* – Hard Real-Time), și reprezintă modalitatea de bază sub care nucleul tratează operarea task-urilor TR stricte ale unei aplicații.

Pe de altă parte, o serie de considerente au determinat introducerea și implementarea unui al doilea context de execuție în cadrul sistemului HARETICK:

- Majoritatea aplicațiilor TR, indiferent de tipul sau complexitatea acestora, conțin, pe lângă task-urile TR stricte, și task-uri cu specificații lejere de comportare în timp, sau chiar task-uri pentru care coordonata temporală nu e deloc importantă.
- Planificarea și execuția în context strict, non-preemptiv a ModX-urilor este foarte puțin flexibilă și implică o eficiență scăzută de operare a sistemului.
- Planificarea și execuția ModX-urilor se bazează pe timpii maximi de execuție ai acestora, care se determină într-o manieră pesimistă (calea de execuție cu durata cea mai mare din secvența de instrucțiuni a ModX-ului).
- Existența ModX-urilor fantomă (vezi capitolul anterior).

Pe baza acestor considerente, a fost introdus un context de execuție complementar cu cel strict, anume *contextul lejer* al nucleului HARETICK (sau, *contextul SRT* – Soft Real-Time). În cadrul contextului lejer, sunt planificate și executate task-urile sistem și ale aplicației, care nu impun cerințe stricte de comportare temporală, și pentru care garantarea respectării termenelor nu reprezintă o necesitate.

Task-urile lejere, notate cu L_i (spre deosebire de ModX-uri, care sunt notate cu M_i), se execută în regim preemptiv, în intervalele de timp rămase libere din cadrul contextului strict. Planificarea task-urilor lejere este efectuată de către un task sistem,

SSCD (Soft real-time Scheduler – la rândul lui, un task lejer), pe baza unor algoritmi ce utilizează priorități și mecanisme de tip *time-sharing*.

Observație

Contextul strict de execuție poate fi privit ca o "întrerupere prelungită" a celui lejer, conform unei planificări riguroase, stricte, analizată apriori din punctul de vedere al fezabilității (analiză "offline").

Un alt mod de a vedea problema este acela că, deasupra contextului lejer (reprezentând modul de operare tradițional, preemptiv, al sistemelor de calcul) se poate implementa, ca o facilitate de execuție cu garantarea respectării termenelor stricte și cu predictibilitate maximă, contextul strict, a cărui întrerupere (RTC) este cea mai prioritară în sistem. În plus, pe durata execuției în contextul strict, celelalte întreruperi sunt suspendate.

Componenta sistem care asigură comutarea operării între cele două contexte de execuție, este executivul TR al HARETICK, HDIS. Principiile operațiilor de comutare între contextul strict și cel lejer, aplicate de HDIS, sunt:

- În momentul apariției unei întreruperi RTC, executivul activează contextul strict de execuție, lansând ModX-ul planificat la acel moment;
- După terminarea execuției unui ModX (sau în cazul unui ModX fantomă), HDIS verifică intervalul de timp rămas disponibil până când este planificat următorul ModX, și în caz afirmativ, restaurează contextul de execuție al ultimului task lejer întrerupt de sistem.

O exemplificare detaliată a modului de operare al HARETICK, cu cele două contexte de execuție, este prezentată în Figura 10-1. Utilizând aceeași axă a timpului, în figură sunt reprezentate graficele de operare ale contextului strict (HRT) și lejer (SRT), din punctul de vedere al planificării (partea superioară) și al execuției efective în sistem (partea inferioară).

Notațiile utilizate în Figura 10-1 pentru componentele HARETICK sunt după cum urmează:

- "P-HSCD" – execuția în regim de pre-planificare a planificatorului HSCD;
- "PD" – componenta prefix a executivului TR, HDIS_PRE;
- "SD" – componenta sufix a executivului TR, HDIS_SUF;
- "SS" – planificatorul task-urilor contextului lejer de execuție, SSCD;
- "Li" – task-ul lejer *i* al sistemului/aplicației;
- "Mi" – ModX-ul *i* al sistemului/aplicației;
- "t_{sk}" – momentul *k* al planificării (scheduling time), echivalent cu momentul de start sau de terminare a planificării unui ModX.

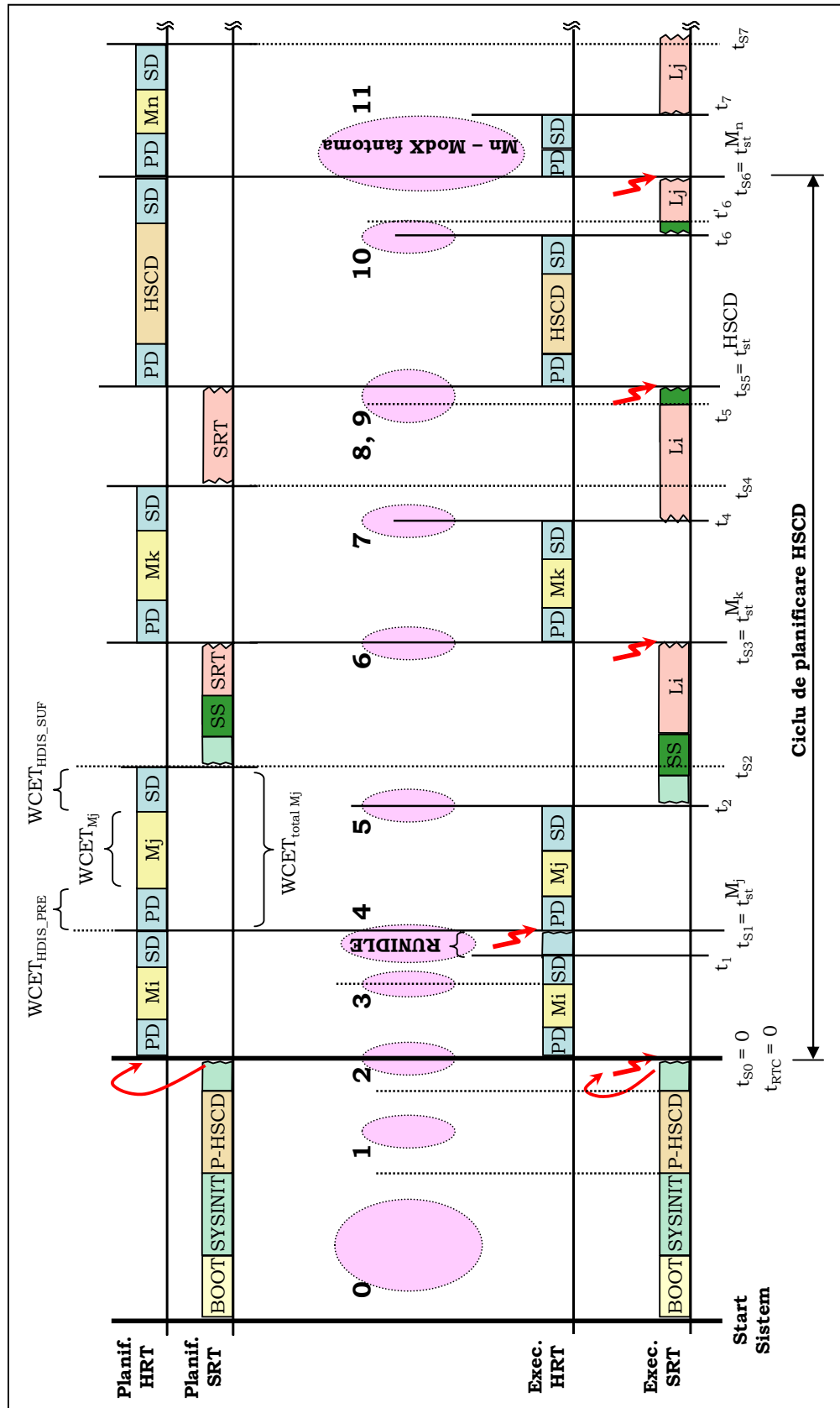


Figura 10-1. Planificarea și execuția task-urilor în HARETICK

Pe parcursul operării nucleului HARETICK, se remarcă următoarele evenimente (situații) speciale, marcate în ordinea corespunzătoare pe diagrama din Figura 10-1:

- 0) Inițial, după startarea sistemului, rulează secvența BOOT care, după încărcarea celorlalte componente de bază ale HARETICK în memorie, activează task-ul lejer SYSINIT.

SYSINIT efectuează inițializările necesare, după care lansează planificatorul contextului strict, HSCD, în regim de *pre-planificare*.

- 1) Execuția planificatorului în regim de pre-planificare (P-HSCD) este similară din punct de vedere funcțional cu o execuție normală a HSCD, singura diferență fiind că în acest caz, contextul strict al HARETICK nu este încă activat. Așadar, P-HSCD se execută în context lejer (SRT).

Odată lansat în execuție, HSCD realizează planificarea non-preemptivă a setului de ModX-uri din sistem, pentru un ciclu. Ciclul de planificare este definit ca seria ordonată în timp a planificărilor de forma

$$\{ \langle \text{identificator ModX} \rangle, \langle \text{moment start execuție} \rangle \}$$

și care este încadrată de două execuții consecutive ale HSCD.

Pentru cazul considerat în exemplul nostru, după execuția P-HSCD, Tabela de Dispatch va conține patru planificări, ce definesc practic primul ciclu de planificare: trei execuții de ModX-uri și execuția HSCD care încheie ciclul (și, totodată, startează ciclul următor). Astfel, la momentul inițial al contextului HRT, t_0 , este planificată execuția ModX-ului M_i , la terminarea căruia (momentul t_{S_1}), va fi lansat în execuție ModX-ul M_j , a cărui execuție va dura până la momentul t_{S_2} . Mai departe, la momentul t_{S_3} , va fi executat ModX-ul M_k , care se termină la t_{S_4} . Execuția lui HSCD la t_{S_5} termină ciclul curent de planificare și va iniția următorul ciclu (aici – al doilea).

Figura 10-2 ilustrează modul în care P-HSCD modifică conținutul Tabelei de Dispatch, de la starea inițială (listă vidă, setată de SYSINIT), la starea corespunzătoare terminării algoritmului de planificare pentru ciclul curent.

După terminarea execuției P-HSCD, este apelat din nou task-ul SYSINIT, care activează contextul HRT al HARETICK, prin setarea ceasului timp-real (RTC) al sistemului pe 0 ($t_0 = 0$) și a registrelor de comparare ale timerelor RTC, de asemenea pe 0. Apoi este activată întreruperea RTC.

Observație

Algoritmii de planificare non-preemptivă considerați pentru implementarea lui HSCD operează pe baza timpului maxim de execuție al ModX-urilor (WCET). Așa cum se vede și în Figura 10-1, durata maximă de execuție a oricărui ModX include și WCET al executivului TR al sistemului, HDIS (cu cele două componente ale sale – prefix și sufix).

Aceasta este o altă cerință importantă impusă de implementarea nucleului HARETICK asupra calculului timpilor WCET ai ModX-urilor.

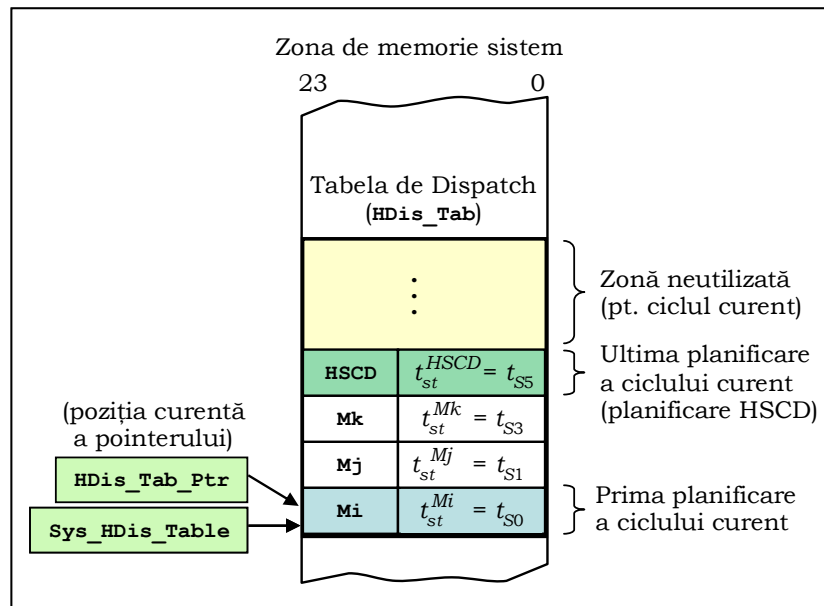


Figura 10-2. Conținutul Tabelei de Dispatch după execuția lui P-HSCD

- 2) În momentul activării întreruperilor RTC, task-ul SYSINIT este întrerupt și se lansează (se comută) contextul strict de execuție al HARETICK, prin apelarea componentei prefix a executivului, HDIS_PRE.

HDIS_PRE încarcă din Tabela de Dispatch (de la poziția curentă a pointerului) informația legată de execuția ModX-ului ce urmează a fi lansat în execuție la momentul curent. În scenariul discutat, este vorba despre M_i , planificat la momentul $t_{S_0} = t_{st}^{M_i} = 0$.

Înainte de lansarea în execuție a lui M_i , HDIS_PRE citește momentul de start al următoarei planificări (înregistrarea următoare din Tabela de Dispatch), în cazul nostru, $t_{S_1} = t_{st}^{M_j}$, și programează regiștrii de comparare ai timerelor RTC pentru generarea unei întreruperi la acel moment.

- 3) La terminarea execuției ModX-ului curent (M_i , care aici, are loc la instanța t_1 , anterioară momentului considerat la planificare), se va apela componenta sufix a executivului TR al HARETICK, HDIS_SUF.

HDIS_SUF avansează pointerul în Tabela de Dispatch (în acest punct, pointerul va indica spre ModX-ul M_j), și determină intervalul de timp rămas disponibil (din momentul curent) până la momentul planificării lui M_j .

- 4) Situația reprezentată de evenimentul (4) în Figura 10-1, este singurul caz în care executivul HDIS însuși este întrerupt de către RTC. Este vorba de starea specială a lui HDIS_SUF, denumită "RunIdle", și care reprezintă practic o buclă de așteptare a întreruperii, implementată de HDIS_SUF.

În faza de calculare a timpului disponibil (din momentul curent) până la momentul planificării lui M_j , HDIS_SUF se bazează pe o evaluare ce ține cont de o serie de parametri de eficiență a operării sistemului. Se apreciază astfel,

dacă timpii consumați de restaurarea și salvarea contextului (a stării procesor) pentru task-ul lejer întrerupt, sunt suficient de mici comparativ cu timpul rămas pentru continuarea execuției acestuia, pentru ca operația de comutare a contextelor de execuție să fie eficientă.

La momentul t_1 , această evaluare determină decizia intrării lui HDIS_SUF în starea "RunIdle", adică de a nu comuta contextul strict de execuție. Se va aștepta astfel, apariția întreruperii RTC, care declanșează execuția următorului ModX (M_j).

- 5) La momentul t_2 , se termină execuția lui HDIS_SUF corespunzătoare ModX-ului M_j . Intervalul de timp de la t_2 , la $t_{S_3} = t_{st}^{M_k}$ (momentul planificării lui M_k), este suficient de mare pentru a permite comutarea contextului strict în contextul lejer de execuție.

Principala operație realizată la activarea contextului SRT de către HDIS este restaurarea contextului de execuție al ultimului task lejer întrerupt în sistem. În cazul de față, este vorba de task-ul SYSINIT, care își va continua execuția cu inițializarea celorlalte task-uri lejere din sistem, după care va lansa planificatorul contextului SRT de operare al HARETICK. SSCD va lansa, la rândul lui, în execuție task-ul lejer L_i .

- 6) Contextul lejer de execuție (task-ul L_i) este întrerupt de către RTC, la momentul planificării ModX-ului M_k , $t_{S_3} = t_{st}^{M_k}$. Rutina de tratare a întreruperii apelează executivul TR (HDIS_PRE), care salvează prima dată contextul de execuție al task-ului lejer L_i .
- 7) Componenta sufix a lui HDIS este apelată la terminarea ModX-ului M_j , când se va efectua comutarea contextelor. HDIS_SUF restaurează contextul de execuție a task-ului lejer întrerupt ultima dată (L_i), iar acesta își va continua execuția.

- 8, 9) Întreruperea RTC pentru lansarea în execuție a următorului ModX (aici, HSCD) apare în timpul execuției planificatorului task-urilor lejere, SSCD. HDIS va proceda la salvarea contextului de execuție al SSCD, la fel ca și cu celelalte task-uri lejere din sistem.

- 10) Restaurarea contextului de execuție al lui SSCD are loc la momentul t_6 , după care acesta își continuă operarea, lansând în execuție un alt task lejer, L_j .

- 11) Cazul în care executivul HDIS constată că ModX-ul care urmează a fi lansat în execuție (conform planificării) este un ModX fantomă.

Înainte de lansarea în execuție a unui ModX, HDIS_PRE verifică valoarea contorului de execuție al ModX-ului ($M_n.ECT$), din tabela descriptorilor de procese (PDT). Dacă $M_n.ECT = 0$, HDIS_PRE lasă contorul nemodificat, și în locul lansării în execuție a lui M_n , va apela componenta sufix, HDIS_SUF.

10.2 Stările ModX-urilor

Operarea ModX-urilor în cadrul sistemului HARETICK poate fi descrisă printr-un set de cinci stări:

(1) Starea NOP ("No Operation").

Starea NOP descrie ModX-urile care nu sunt încărcate în cadrul nucleului HARETICK la un moment dat, ci se află în stadiu de dezvoltare și analiză în cadrul mediului integrat INVERTA, pe un calculator gazdă. Sistemul HARETICK nu are practic cunoștință despre aceste ModX-uri, ele neinfluențând în nici un fel operarea curentă a nucleului de pe platforma țintă.

Ca observație, remarcăm faptul că ModX-urile aparținând nucleului HARETICK (de exemplu, HSCD, HDIS, componentele stricte ale lui DATALINK, etc.) nu se vor găsi niciodată în starea NOP, ele fiind încărcate pe platforma țintă odată cu restul nucleului HARETICK în timpul operației de BOOT. Doar ModX-urile aplicațiilor ce vor fi încărcate în sistem la un moment dat, pot să se găsească în starea NOP, înaintea încărcării acestora.

(2) Starea RDY ("Ready").

Odată ce o aplicație oarecare a fost încărcată pe platforma țintă pe care rulează nucleul HARETICK, ModX-urile acesteia vor intra în starea RDY ("gata de planificare").

Un ModX în starea RDY este complet identificat și reprezentat în cadrul nucleului, având alocate majoritatea resurselor necesare execuției:

- identificare în cadrul aplicației de care aparține, prin tabela PDAGT;
- caracterizare completă prin parametrii săi din tabela PDT;
- alocarea zonei proprii de cod în memoria program a sistemului;
- alocarea zonei parametrilor de ieșire ("Data Out") în memoria de date;
- alocarea zonei cu variabilele globale ale aplicației și identificarea celor utilizate de ModX;
- identificarea variabilelor și parametrilor utilizați, prin intermediul tabelii de simboluri a aplicației.

(3) Starea "SCD" ("Scheduled").

ModX-urile din starea SCD aparțin planificării din cadrul ciclului curent, ca rezultat al aplicării algoritmilor de planificare strictă, non-preemptivă, implementați de HSCD, asupra setului de ModX-uri sistem și ale aplicației.

ModX-urile din starea SCD se regăsesc în cadrul Tabelei de Dispatch, rezultând că la un moment dat (într-un ciclu de planificare), există un număr bine determinat de ModX-uri în starea SCD.

(4) Starea "RUN" ("În execuție").

În starea RUN se poate găsi, la un moment dat, un singur ModX. Acestuia i se alocă restul resurselor sistem necesare execuției efective pe platforma țintă:

- zona "Heap" a sistemului, alocată exclusiv pentru parametrii de intrare și variabilele locale ale ModX-ului, pe toată durata execuției sale, și numai atunci;
- procesorul sistemului, care execută în regim non-preemptiv secvența de instrucțiuni a ModX-ului.

(5) Starea "GST" ("Ghost").

Starea GST este starea în care se află la un moment dat un ModX fantomă. ModX-ul fantomă este caracterizat prin faptul că, deși a fost planificat de către HSCD, în momentul lansării în execuție are contorul de execuții nul.

ModX-urile fantomă nu sunt lansate în execuție efectivă de către nucleu (adică de către executivul TR al nucleului, HDIS), deci nu li se alocă resursele suplimentare necesare execuției, ca în cazul stării RUN.

Trecerea unui ModX oarecare dintr-o stare în alta are loc în urma îndeplinirii unor condiții specifice, și are loc prin intermediul unui set distinct de operații. Figura 10-3 ilustrează cele cinci stări posibile, împreună cu configurația resurselor sistemului pentru fiecare stare în parte. Figura prezintă de asemenea, și modurile posibile de trecere a ModX-urilor dintr-o stare în alta, precum și task-urile sistem implicate în aceste operații.

Încărcarea unei aplicații oarecare în sistem este efectuată de către task-ul LOADER, prin intermediul componentei sistem DATALINK. LOADER-ul este responsabil de alocarea și completarea structurilor necesare reprezentării aplicației și a ModX-urilor acesteia în cadrul nucleului HARETICK: tabela PDAGT, tabela PDT, tabela de simboluri, zonele de cod și "Data Out" pentru fiecare ModX, zona variabilelor globale, etc.

Setul de operații ce corespunde trecerii ModX-urilor din starea NOP în RDY este denumit "load".

Descărcarea (eliminarea) unei aplicații din sistemul HARETICK este realizată tot de task-ul LOADER, eliberându-se resursele sistem utilizate de nucleu la reprezentarea și gestionarea aplicației. Setul de operații corespunzător trecerii unui ModX din starea RDY în starea NOP este denumit "unload".

Planificatorul contextului strict de execuție, HSCD, aplică algoritmul de planificare non-preemptivă special conceput pentru sistemul HARETICK, asupra setului de ModX-uri aflate în starea RDY. În cadrul unui ciclu de planificare, va fi selectat un număr bine definit de ModX-uri, cu care se construiește (completează) Tabela de Dispatch. Ca urmare, starea ModX-urilor planificate se schimbă, din RDY în SCD.

Ca observație, menționăm că un anumit ModX poate apărea în cadrul mai multor înregistrări din Tabela de Dispatch, fiind astfel planificat pentru execuție de mai multe ori în ciclul curent de planificare. De exemplu, un ModX cu perioadă mai mică are o frecvență mai mare de apariții în cadrul Tabelei de Dispatch, comparativ cu un ModX cu perioadă mai mare.

Setul de operații prin care un ModX trece din starea RDY în SCD este denumit "schedule". Cum se observă și din Figura 10-3, arhitectura nucleului HARETICK nu permite tranziția inversă a ModX-urilor, anume din SCD, direct în RDY. Cu alte

cuvinte, odată ce un ModX a fost planificat pentru ciclul curent de planificare, el va fi preluat în continuare de executivul TR al nucleului, HDIS, în vederea lansării în execuție la momentul specificat în planificare.

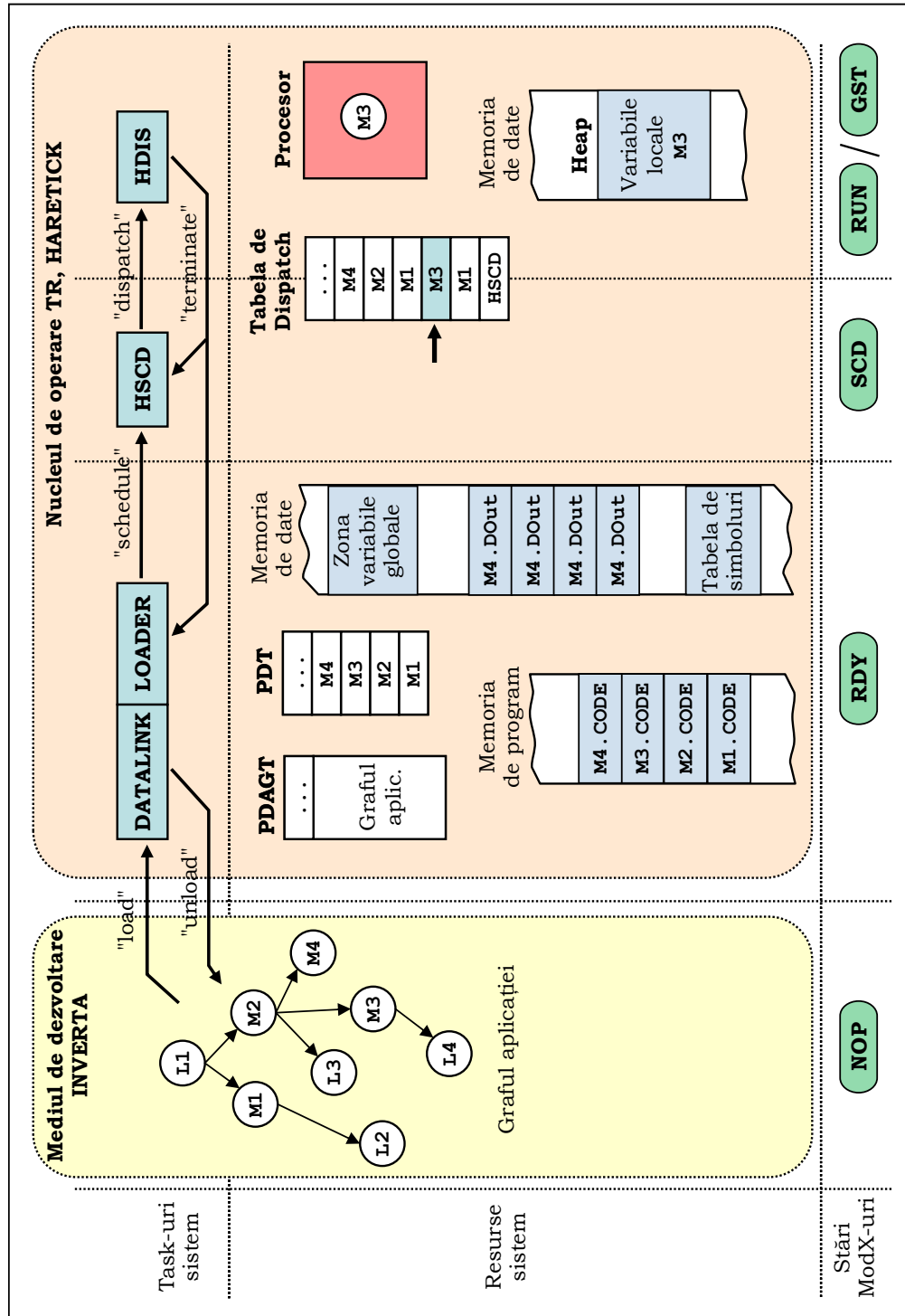


Figura 10-3. Stările ModX-urilor și operarea nucleului HARETICK

Tabelul 10-1. Sinteză a stărilor ModX-urilor și a tranzițiilor între stări

Tranziție stări	Denumire	Semnificație
NOP → RDY	"load"	Încărcarea unei aplicații dezvoltate și analizate în mediul integrat INVERTA de pe un calculator gazdă, pe platforma țintă ce rulează sub nucleul TR de operare HARETICK.
RDY → NOP	"unload"	Descărcarea (eliminarea) unei aplicații din sistemul HARETICK și eliberarea resurselor sistem alocate pentru reprezentarea și gestionarea ei.
RDY → SCD	"schedule"	Includerea unui subset de ModX-uri ale sistemului (aplicației) în ciclul curent de planificare, în vederea execuției pe sistemul HARETICK.
SCD → {RUN, GST}	"dispatch"	Preluarea de către executivul HDIS a unui ModX inclus în ciclul curent de planificare, în vederea execuției. În cazul în care contorul său de execuții este nenul, ModX-ul va fi executat efectiv (starea RUN), altfel, se va considera doar că a fost executat (starea GST).
{RUN, GST} → {RDY, SCD}	"terminate"	Terminarea execuției (efective – RUN, sau fictive – GST) a unui ModX, urmată de revenirea acestuia în starea "gata de planificare" (RDY) sau "planificat și gata de execuție" (SCD), în cazul în care ciclul curent de planificare prevede mai multe execuții ale ModX-ului.

ModX-urile planificate de către HSCD sunt preluate pe rând de către executivul HDIS, prin baleierea Tabelei de Dispatch, în vederea lansării în execuție pe baza mecanismului de întreruperi RTC. La un moment dat, ceasul timp-real al sistemului va genera o întrerupere, semnalând instanța de timp pentru care a fost planificată execuția ModX-ului curent referit în Tabela de Dispatch. Rutina de tratare a întreruperilor apelează executivul HDIS, care verifică valoarea contorului de execuții a ModX-ului curent. În cazul în care contorul de execuții este nul, ModX-ul devine un ModX fantomă, trecând astfel în starea GST. HDIS nu îl va lansa în execuție efectivă, ci va trece la următorul ModX din tabelă.

Prin urmare, ModX-urile din starea GST sunt considerate de sistem ca "fiind executate", fără a fi lansate efectiv în execuție (de aici și denumirea de "ModX fantomă").

În cazul în care contorul de execuții al ModX-ului curent ce urmează a fi executat este nenul, executivul HDIS îi alocă restul resurselor sistem necesare execuției acestuia (zona "Heap" pentru parametrii de intrare și variabilele locale, și procesorul sistemului). ModX-ul trece astfel în starea RUN, ocupând exclusiv procesorul sistemului pentru execuția în regim non-preemptiv a secvenței proprii de instrucțiuni.

Setul de operații implicate în tranziția ModX-urilor din starea SCD în RUN sau GST, este denumit "dispatch".

După execuția (efectivă sau "fantomă") a unui ModX aflat în starea RUN, respectiv GST, acesta poate reveni în starea RDY sau SCD. În primul caz, ModX-ul care tocmai a fost executat și-a terminat și planificările din ciclul curent de planificare, adică nu mai există o altă înregistrare corespunzătoare aceluiași ModX în Tabela de Dispatch. Cea de-a doua variantă, presupune contrariul celor de mai sus, adică ModX-ul respectiv se mai regăsește în cadrul tabelii, urmând a mai fi executat în același ciclu de planificare. El rămâne astfel, în starea de ModX planificat (SCD).

Setul operațiilor implicate de terminarea execuției ModX-urilor și trecerea lor din starea {RUN, GST} în starea {SCD, RDY}, este denumit "terminate".

Tabelul 10-1 oferă o sinteză a celor discutate până acum în legătură cu stările ModX-urilor sistemului HARETICK și tranzițiile posibile între stări.

10.3 Executivul TR al HARETICK: HDIS

10.3.1 Caracteristici generale

Executivul TR al HARETICK, HDIS (Hard real-time Dispatcher), este o componentă esențială a nucleului și are principala sarcină de a lansa în execuție ModX-urile sistem și ale aplicației aflate în starea SCD ("planificat"), și de a termina execuția acestora, în contextul strict de execuție al nucleului HARETICK. HDIS implementează astfel mecanismele necesare comutării celor două tipuri de contexte de execuție existente în sistem: contextul strict (contextul HRT – Hard Real-Time) în cadrul căruia se execută în regim non-preemptiv ModX-urile sistemului, și contextul lejer (contextul SRT – Soft Real-Time) ce corespunde execuției în regim preemptiv a task-urilor lejere.

Operarea HDIS se face într-o manieră complet predictibilă, bazată pe următoarele considerente:

- HDIS este apelat de către rutina de tratare a întreruperii de ceas timp-real (RTC), la momente de timp bine stabilite, conform planificării execuției task-urilor TR stricte în sistem;

- Durata maximă de execuție a HDIS ($WCET_{HDIS}$) este calculată cu atenție, luând în considerare toate situațiile posibile de operare ale nucleului. Pe orice ramură de program s-ar afla la un moment dat execuția codului HDIS, timpii maximi de execuție pot fi determinați cu precizie, fără a depinde de structuri sau mecanisme dinamice (cum ar fi de exemplu, numărul curent de ModX-uri în sistem, numărul parametrilor de intrare sau de ieșire al acestora, etc.);
- Întreaga arhitectură a HARETICK și a mecanismelor de reprezentare și gestionare a ModX-urilor în cadrul nucleului, este concepută și proiectată în așa fel încât să faciliteze operarea predictibilă a executivului și a întregului sistem. Sunt utilizate doar reprezentări statice și operații precis limitate în timp și spațiu. În orice moment, sunt cunoscute cu precizie valorile maxime ale dimensiunilor structurilor folosite și duratele maxime de execuție ale operațiilor.

Executivul HDIS are două componente principale: un prefix (HDIS_PRE) și un sufix (HDIS_SUF), cu roluri bine determinate. Astfel, HDIS_PRE inițiază execuția ModX-ului planificat a fi lansat la un moment dat, și este apelat de întreruperea RTC ce apare la acel moment în sistem. Odată activat, HDIS_PRE efectuează următoarele acțiuni:

- verifică dacă a fost întrerupt contextul lejer. În caz afirmativ, se efectuează salvarea contextului de execuție lejeră a task-ului întrerupt într-o zonă dedicată;
- identifică ModX-ul ce urmează a fi lansat în execuție la momentul curent, din Tabela de Dispatch;
- citește următoarea înregistrare din Tabela de Dispatch, corespunzând ModX-ului ce urmează ca planificare celui curent, și extrage momentul de start a execuției acestuia;
- programează timerele ceasului de timp-real (RTC) pentru a genera întrerupere la momentul de start a ModX-ului următor;
- decrementează contorul de execuții a ModX-ului curent (dacă acesta are valoare finită);
- trece ModX-ul curent din starea "planificat" ("SCD"), fie în starea "de execuție" ("RUN") – dacă valoarea contorului de execuție era nenulă înaintea decrementării, fie în starea "ModX fantomă" ("GST");
- dacă ModX-ul curent este în starea "RUN", îi cedează procesorul și îl apelează pentru execuție;
- dacă ModX-ul curent este în starea "GST", apelează direct componenta sufix a executivului, HDIS_SUF.

Pe de altă parte, componenta sufix a lui HDIS este apelată la terminarea execuției fiecărui ModX, chiar de către acesta. Odată activat, HDIS_SUF efectuează următoarele acțiuni:

- trece ModX-ul curent din starea "de execuție" (RUN) sau "ModX fantomă" (GST), înapoi în starea "planificat și gata de execuție" (SCD) sau în "gata de planificare" (RDY);
- avansează referința (pointerul) de citire al Tabelei de Dispatch către ModX-ul următor;

- verifică dacă există suficient timp pentru a comuta contextul strict cu cel lejer, permițând execuția task-urilor SRT ale sistemului și aplicației. Dacă nu este timp suficient, HDIS va iniția o buclă infinită ("RunIdle") în care va aștepta apariția întreruperii RTC ce marchează evenimentul lansării în execuție a ModX-ului următor. Dacă se decide comutarea controlului către contextul lejer, are loc restaurarea stării de execuție (a contextului) task-ului lejer care a fost întrerupt.

Pe lângă cele două componente principale, executivul HDIS mai utilizează o subrutină denumită "TiLT" ("Time Log Tool"), pentru generarea unor scurte rapoarte legate de evenimentele lansării în execuție și terminării ModX-urilor. Evenimentele sunt înscrise într-o tabelă dedicată, împreună cu valorile instanțelor de timp corespunzătoare.

Structura de date de bază a executivului HDIS este Tabela de Dispatch (vezi și Capitolul 9). Tabela de Dispatch (HDis_Tab) este o structură sistem cu un număr fix, bine stabilit de înregistrări ($\lambda + 1$), care specifică numărul maxim de planificări posibile pentru orice ciclu de planificare: λ planificări de ModX-uri normale (sistem sau ale aplicației) și o planificare de HSCD, care încheie ciclul curent și inițiază ciclul următor de planificare. HDIS accesează Tabela de Dispatch doar în citire, completarea înregistrărilor tabelii fiind efectuată de către planificatorul contextului strict de execuție al HARETICK, HSCD.

Fiecare înregistrare a Tabelei de Dispatch conține două câmpuri: (i) identificatorul ModX-ului, PID, și (ii) momentul de start al execuției ModX-ului, conform planificării realizate de HSCD în cadrul ciclului curent de planificare. Tabela este definită prin adresa de bază (Sys_HDis_Table) și de dimensiunea sa totală $((\lambda + 1) \cdot \langle \text{dimensiunea unei înregistrări} \rangle)$. Înregistrarea curentă în cadrul tabelii este referită printr-un pointer, HDis_Tab_Ptr.

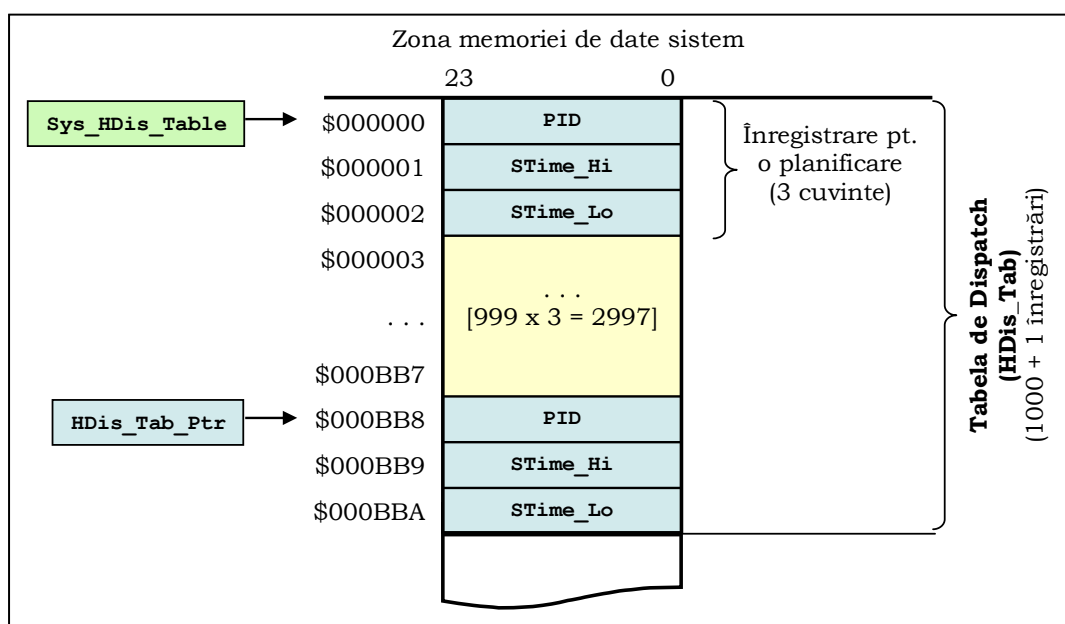


Figura 10-4. Implementarea curentă a Tabelei de Dispatch

Figura 10-4 ilustrează modul curent de implementare a Tabelei de Dispatch, pe arhitectura Motorola DSP56307. Fiecare înregistrare conține un cuvânt procesor pentru identificarea ModX-ului (PID) și două cuvinte (2·24 biți) pentru exprimarea timpului de start al execuției ModX-ului (*STime_Hi:STime_Lo*). Structura de timp a înregistrărilor tabelii este compatibilă cu arhitectura ceasului timp-real (RTC) care e compusă din două timere cascade (sau un timer și o variabilă sistem). În acest fel se ușurează operațiile de programare a regiștrilor de comparare ai RTC de către HDIS.

Dimensiunea totală a tabelii este de 1000 (înregistrări normale) + 1 (înregistrare specială), rezultând un număr de 3003 cuvinte de memorie. Prin urmare, un ciclu de planificare va cuprinde maxim 1001 execuții, din care una corespunde planificatorului HSCD.

10.3.2 Elemente de implementare și analiză a codului executivului

În continuare vom detalia pe rând funcționarea celor trei componente ale executivului HDIS: HDIS_PRE, HDIS_SUF și subrutina TiLT.

Separarea executivului în două componente, prefix și sufix, a fost realizată ținând cont de cele două ipostaze de apelare ale lui HDIS și de modul în care acesta încadrează execuția propriu-zisă a unui ModX. Componenta prefix este apelată de către întreruperea RTC și inițiază execuția ModX-urilor, pe când componenta sufix este apelată de către ModX-uri, la terminarea execuției acestora.

Ramurile de procesare ale executivului HDIS mai trebuie să trateze o situație importantă: execuția planificatorului contextului strict, HSCD. În cazurile în care ModX-ul curent ce urmează a fi lansat în execuție este identificat în Tabela de Dispatch ca fiind HSCD, executivul nu poate citi din tabelă timpul de start pentru ModX-ul următor, în vederea programării RTC. Acest lucru se datorează faptului că înregistrările din tabelă (ce identifică planificări ale ModX-uri normale) urmează a fi completate chiar de către planificator, în timpul execuției sale.

Soluția constă în faptul că, atunci când e vorba de lansarea în execuție a HSCD, programarea timerelor RTC nu se mai efectuează de către executivul prefix, ci de către componenta sufix, după terminarea execuției planificatorului ce are ca rezultat completarea tabelii cu planificările noului ciclu.

Figura 10-5, continuată cu Figura 10-6, prezintă organigrama de operare a componentei prefix a executivului. HDIS_PRE este lansată în execuție de către întreruperea RTC, anume de întreruperea de *Timer0*, în implementarea curentă a nucleului HARETICK pe arhitectura Motorola DSP56307. Fiind vorba de două timere cascade ca bază de timp în sistem, *Timer0* și *Timer1*, a apărut problema rezolvării unei întreruperi suplimentare – cea generată de *Timer1*. Acesta va genera întrerupere doar dacă între două planificări succesive de ModX-uri există un interval de timp ce depășește capacitatea curentă de numărare rămasă disponibilă în *Timer0* (registrul de numărare al *Timer0* se va "da peste cap").

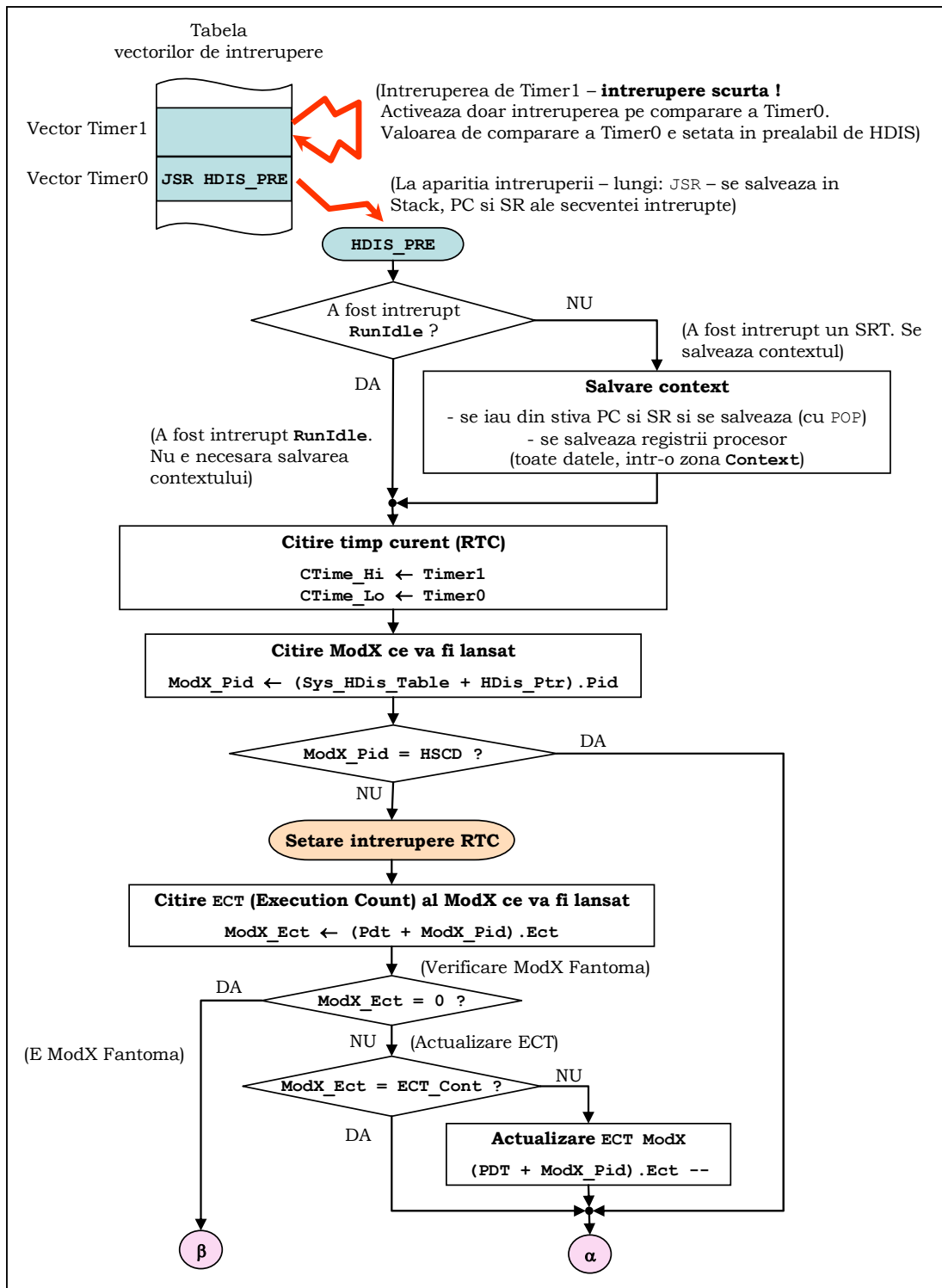


Figura 10-5. Organigrama de operare a componentei HDIS_PRE

Problema a fost rezolvată, în primul rând prin verificarea apariției acestor situații de depășire a capacității primului timer. Verificarea se face comparând cuvintele cele mai semnificative ale valorilor timpilor de start, pentru ModX-ul curent și cel următor, din Tabela de Dispatch. Prin această comparare se verifică

practic dacă valoarea din `Timer1` este aceeași pentru cei doi timpi, sau nu. Dacă este aceeași, înseamnă că timpul ce se va scurge de la execuția `ModX`-ului curent și până la lansarea următorului `ModX`, este suficient de mic pentru capacitatea de contorizare disponibilă în `Timer0`. Dacă se confirmă depășirea capacității curente a `Timer0`, se vor programa regiștrii de comparare ai ambelor timere, cu valorile corespunzătoare momentului de start al `ModX`-ului următor, și se va activa doar întreruperea pentru `Timer1`.

Rutina de tratare a întreruperii generate de `Timer1` are doar rolul de activare a întreruperii pentru `Timer0`. În arhitectura `DSP56307`, ea este denumită "întrerupere scurtă", pentru că nu implică salvarea stării procesor și a contorului de program în stivă.

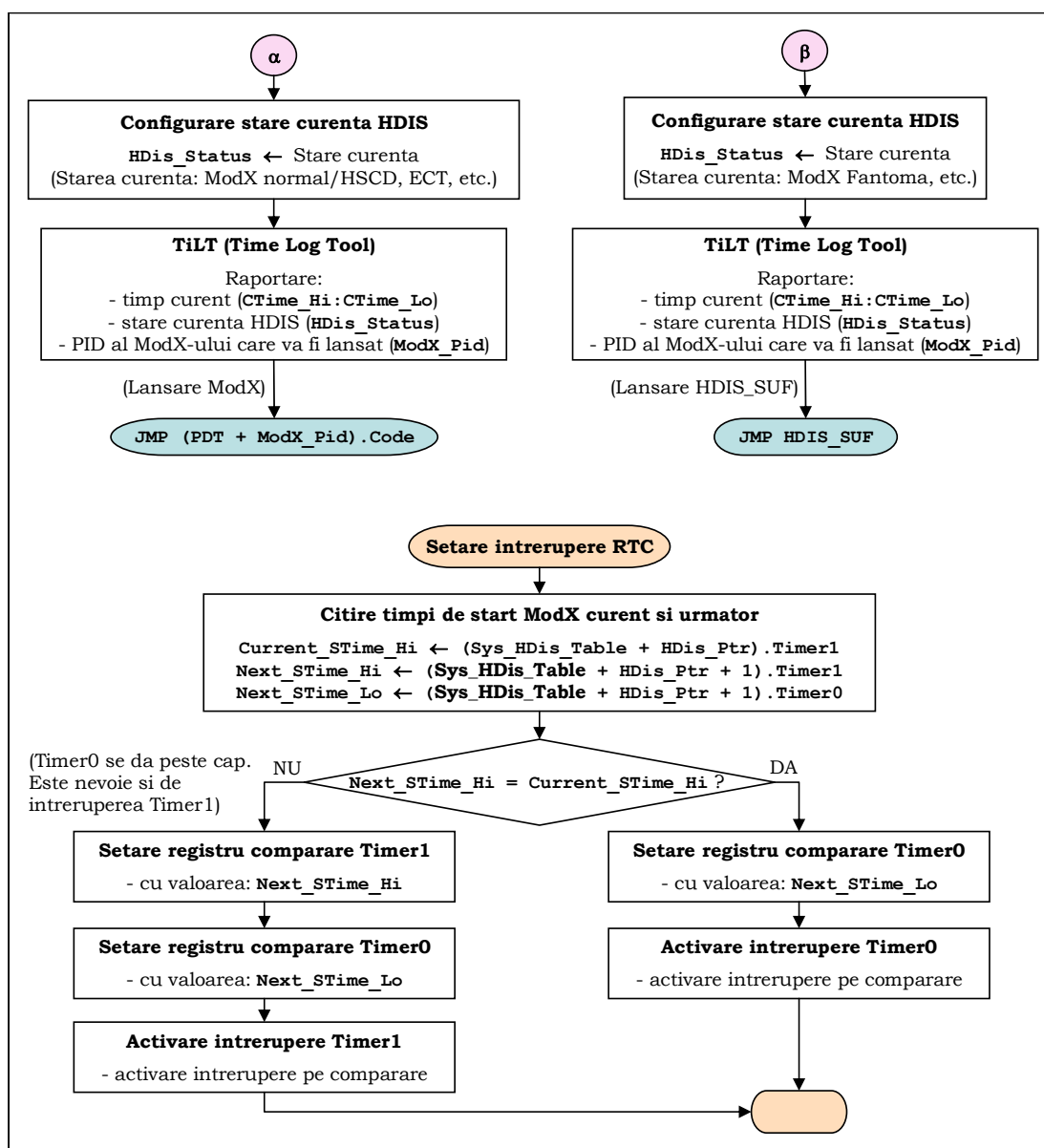


Figura 10-6. Organigrama HDIS_PRE (continuare)

După apelarea lui HDIS_PRE de către întreruperea generată de `Timer0`, se verifică prima dată dacă a fost întreruptă execuția unui task lejer, caz în care este necesară salvarea contextului său de execuție într-o zonă de memorie sistem dedicată, denumită "Context". Operația de verificare se bazează pe informația de stare pe care o menține HDIS de la o execuție la alta, prin intermediul unei variabile proprii (`HDis_Status`).

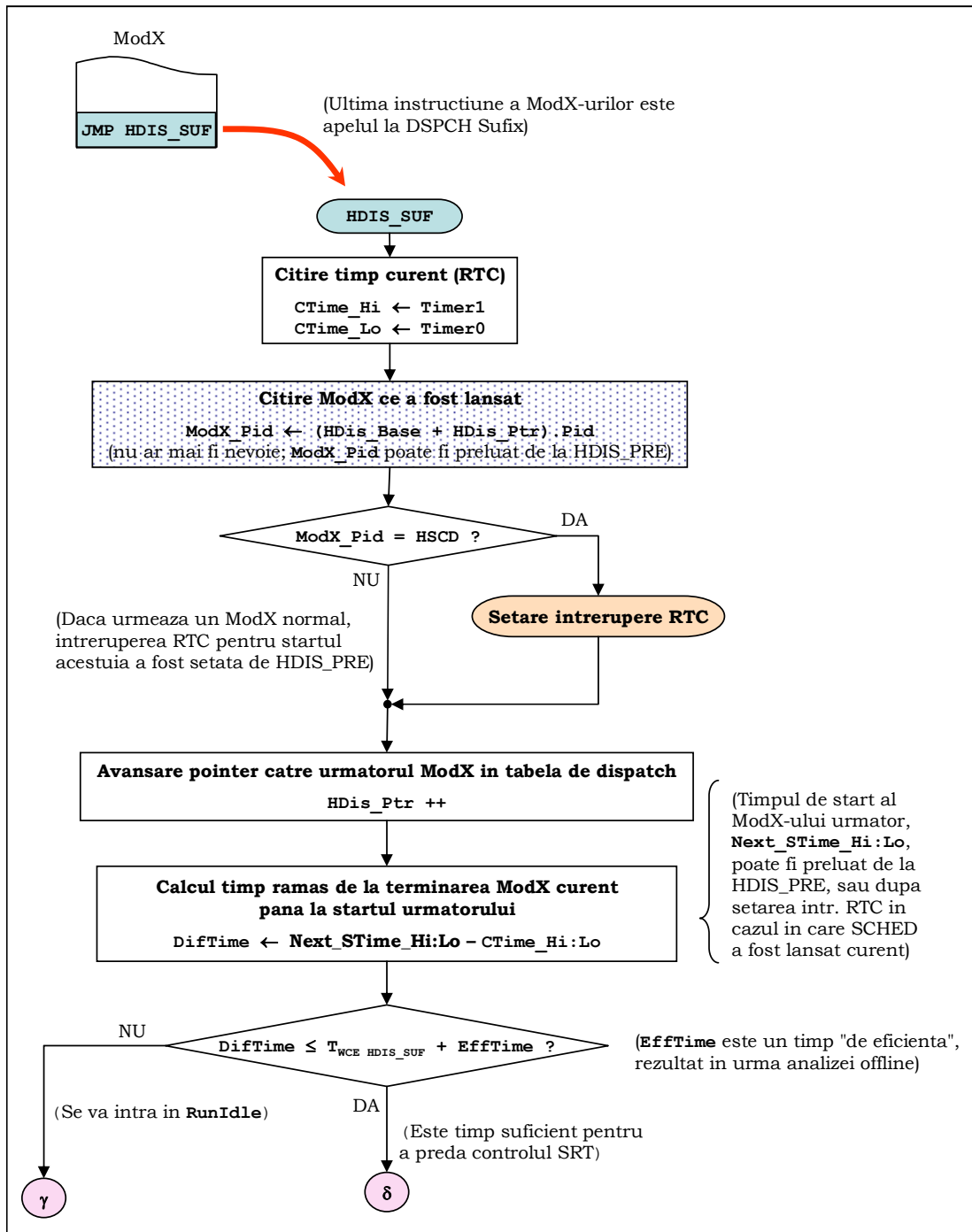


Figura 10-7. Organigrama de operare a componentei HDIS_SUF

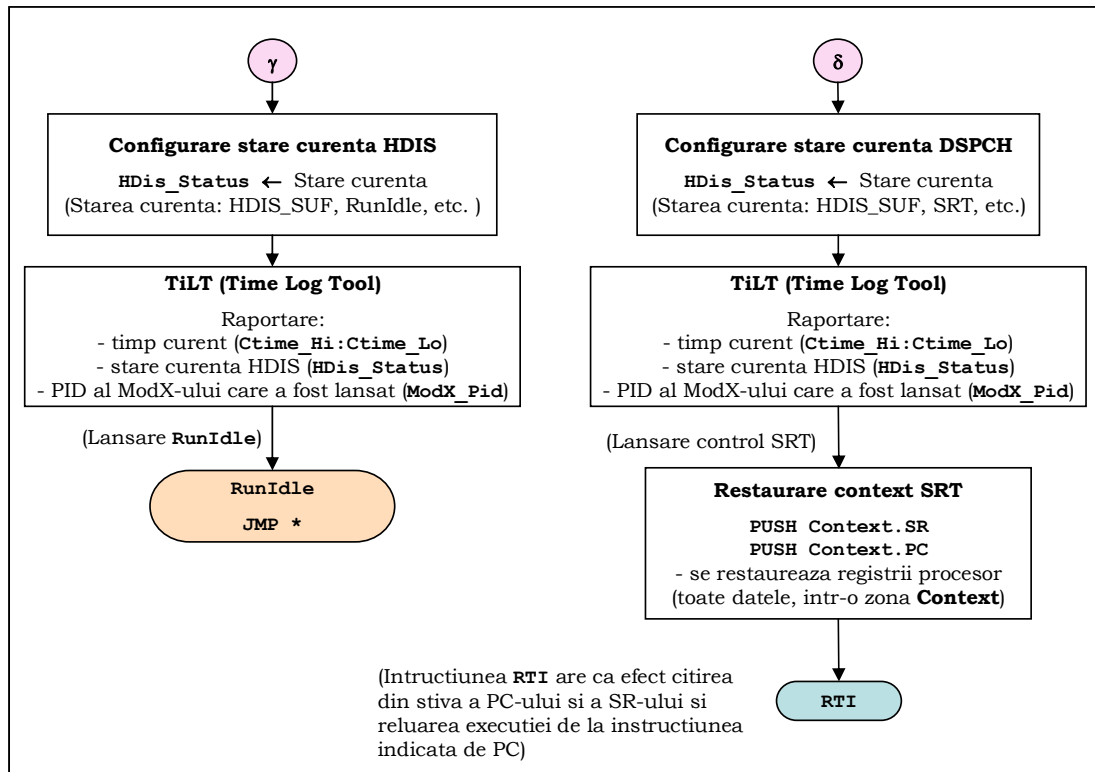


Figura 10-8. Organigrama HDIS_SUF (continuare)

Figura 10-7, continuată cu Figura 10-8, prezintă organigrama de operare a componentei sufix a executivului. HDIS_SUF este apelată de către ModX-ul care își încheie execuția în contextul strict al nucleului.

Se poate observa din organigramele celor două componente ale HDIS că, de fiecare dată când sunt activate, are loc citirea și înregistrarea momentului curent de timp (se citesc contoarele de timer ale ceasului timp-real). Informația de timp este utilizată în rapoartele pe care HDIS le generează prin intermediul subrutinei TiLT.

Subrutina TiLT gestionează o tabelă (un buffer circular), în care înscrie, pe lângă momentul curent al apariției unui eveniment, informații legate de tipul evenimentului, starea curentă a HDIS și contorul de execuții al ModX-ului care a fost procesat de HDIS.

10.4 Planificatorul HRT al HARETICK: HSCD

Planificatorul contextului strict de execuție al nucleului HARETICK este un modul esențial pentru operarea corectă a sistemului. Proprietățile și parametrii HSCD caracterizează și influențează în mod decisiv comportarea și operarea STR în ceea ce privește task-urile cu specificații temporale stricte.

Task-ul HSCD implementează algoritmi de planificare non-preemptivă introduși și studiați în Capitolele 5 și 6, asupra seturilor de task-uri TR stricte – ModX-urile (Capitolul 4). În cele ce urmează vom prezenta principiile de proiectare și implementare, vom descrie funcționarea și vom analiza principalii parametri ai planificatorului HRT al nucleului HARETICK. Așa cum se va vedea, task-ul HSCD implementează o serie de algoritmi și operații complexe, care se reflectă în durata sa de execuție – considerabil mai mare decât marea majoritate a celorlalte task-uri din sistem. În consecință, proiectarea și implementarea HSCD trebuie tratată cu o deosebită atenție, astfel încât performanța și eficiența de operare ale sistemului să fie cât mai puțin afectate de execuția acestui task, care este complet transparent pentru orice aplicație din sistem.

10.4.1 Principii de proiectare, implementare și operare ale HSCD

Obiectivul final în ceea ce privește planificarea task-urilor stricte din cadrul unui sistem (aplicații) timp-real îl reprezintă implementarea cât mai eficientă, pe platforma HARETICK, a algoritmilor care să rezolve planificarea non-preemptivă a unui set fezabil de ModX-uri cu dependențe de program, care să includă și task-uri cu execuție fixă în cadrul perioadei (FModX-uri). Evidențiem două aspecte generale legate de implementarea HSCD:

- (a) Eficiența: execuția task-ului de planificare trebuie să se desfășoare de fiecare dată într-un interval de timp cât mai scurt, pentru ca impactul asupra eficienței întregului sistem să fie cât mai mic;
- (b) Fezabilitatea: seturile de task-uri (ModX-uri) care vor fi planificate pe parcursul operării sistemului sunt în prealabil analizate offline din punct de vedere al fezabilității, cu ajutorul mediului de dezvoltare INVERTA (descriș în Capitolul 7), prin aplicarea algoritmilor de planificare similari cu cei pe care îi implementează și HSCD.

În stadiul curent al cercetării și dezvoltării, nucleul HARETICK dispune de o versiune preliminară, dar complet funcțională de HSCD, capabil să rezolve planificarea seturilor de ModX-uri independente, cu execuție fixă în cadrul perioadei (FModX-uri). *Însuși task-ul de planificare, HSCD, este la rândul său un FModX*, de unde rezultă o serie de proprietăți importante ale planificării pe platforma țintă:

- Ciclurile de planificare sunt periodice în timp;
- Numărul de planificări de task-uri (FModX-uri) variază de la un ciclu de planificare la altul;
- Tabela de Dispatch a nucleului conține un număr variabil de înregistrări de la un ciclu de planificare la altul, impunându-se totuși o limită superioară (definită de dimensiunea totală a tablei).

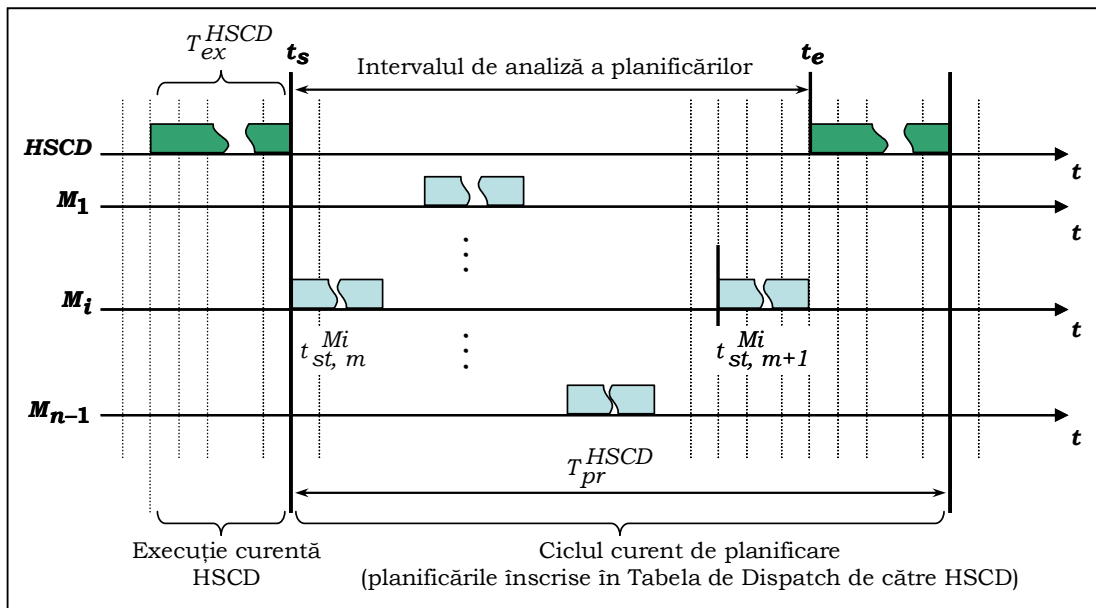


Figura 10-9. Structura unui ciclu de planificare cu HSCD

În Figura 10-9 este prezentat un ciclu de planificare (ciclul curent), pentru un set $M = \{M_1, M_2, \dots, M_{n-1}\}$ de FModX-uri, în condițiile operării planificatorului HSCD – definit la rândul lui ca FModX. Ciclul de planificare este definit în cadrul execuției curente a HSCD, fiind delimitat de instanțele temporale corespunzătoare terminării execuției curente a HSCD (t_s) și a celei următoare ale lui HSCD. Durata ciclurilor de planificare este egală cu perioada lui HSCD.

Cum fiecare ciclu se termină cu planificarea unei execuții HSCD, algoritmi implementați vor analiza și completa înregistrările Tabelei de Dispatch cu planificările FModX-urilor din setul M , pentru intervalul:

$$\left[t_s, t_e = t_s + T_{pr}^{HSCD} - T_{ex}^{HSCD} \right) \tag{10-1}$$

Algoritmul de planificare online, non-preemptivă, a setului $M \cup \{HSCD\}$ de FModX-uri se bazează pe următoarele elemente:

- Analiza fezabilității a fost realizată în prealabil (înaintea încărcării aplicației corespunzătoare pe platforma țintă), prin aplicarea algoritmului similar cu cel pe care îl va implementa HSCD (vezi Capitolul 5);
- În urma analizei de fezabilitate, pentru fiecare FModX din set, rezultă valoarea parametrului $T_{st}^{M_i}$ (intervalul de start al lui M_i în cadrul perioadei), care este de regulă direct proporțional cu perioada FModX-ului;
- Două execuții consecutive ale oricărui FModX sunt despărțite întotdeauna de intervalul de timp egal cu perioada acestuia, $T_{pr}^{M_i}$ (vezi și Figura 10-10);
- Prima execuție a fiecărui FModX se planifică la momentul $t_{st,1}^{M_i} = T_{st}^{M_i}$, egal cu intervalul de start al FModX-ului în cadrul perioadei sale.

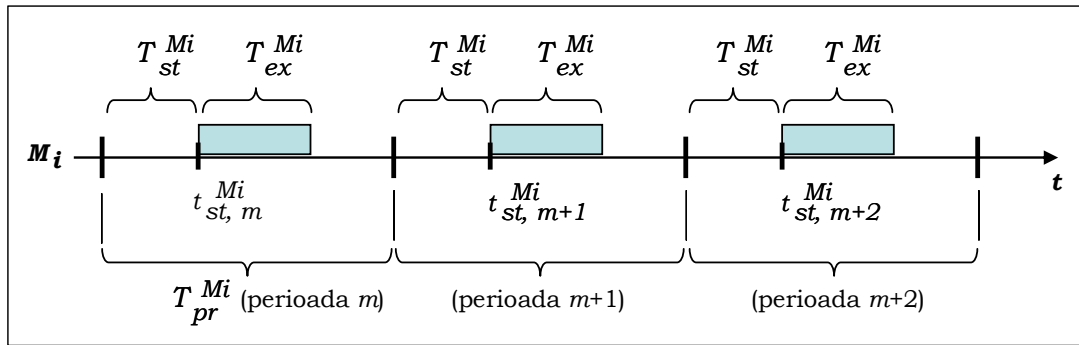


Figura 10-10. Execuția unui FModX pentru mai multe perioade

Având în vedere considerentele enumerate anterior, s-a ales un mod de operare al planificatorului online HSCD, bazat pe două faze: faza de *pre-planificare*, pentru completarea Tabelii de Dispatch cu planificările din primul ciclu, și faza de planificare normală, ce urmează pre-planificării, pe toată durata operării sistemului.

Algoritmul pe care îl implementează HSCD se bazează pe următorii parametri ai FModX-urilor din set: perioada ($T_{pr}^{M_i} \equiv \text{PER_Hi:Lo}$), durata de execuție (timpul maxim de execuție, WCET: $T_{ex}^{M_i} \equiv \text{WCE}$), intervalul de start ($T_{st}^{M_i} \equiv \text{DLY_Hi:Lo}$). Elementele din paranteze aparțin înregistrării corespunzătoare FModX-ului M_i din cadrul tabelii cu descriptorii de proces, PDT (vezi Figura 9-6 și Tabelul 9-1 din Secțiunea 9.4.1.)

Operarea de principiu a algoritmului de planificare online implementat de HSCD cuprinde următorii pași principali:

- 0) În faza de pre-planificare, se completează Tabela de Dispatch (HDis_Tab) a sistemului cu planificările FModX-urilor din setul dat, începând cu prima perioadă de execuție a fiecăruia. Primul ciclu se încheie atunci când s-a ajuns să se planifice o execuție HSCD.

Momentul inițial al planificării se consideră $t_{s0} = 0$, corespunzător activării contextului strict de execuție al HARETICK. Instanțele de start ale primelor execuții ale ModX-urilor planificate în primul ciclu sunt egale chiar cu intervalele de start corespunzătoare ($T_{st}^{M_i} \equiv \text{DLY_Hi:Lo}$).

După fiecare planificare reușită a unui FModX, se actualizează parametrul SCT_Hi:Lo al acestuia din tabela PDT cu momentul de timp corespunzător planificării. Astfel, SCT_Hi:Lo memorează ultimul moment al unei planificări reușite a FModX-ului respectiv.

În momentul planificării execuției HSCD care termină primul ciclu (t_e , vezi și Figura 10-9), parametrul SCT_Hi:Lo corespunzător din PDT va fi actualizat cu instanța de timp a execuției următoare celei planificate (deci cu instanța execuției HSCD care va termina ciclul următor):

$$\text{SCT_Hi:Lo}_{HSCD} \leftarrow t_e + \text{PER_Hi:Lo}_{HSCD} \quad (10-2)$$

- 1) Înainte de a începe calculul planificărilor ciclului curent, HSCD preia instanța de sfârșit a intervalului de analiză, t_e (conform (10-1)), din parametrul $SCT_Hi:Lo$ corespunzător din PDT, a cărui valoare a fost pregătită de execuția anterioară a HSCD:

$$t_e \leftarrow SCT_Hi:Lo_{HSCD} \quad (10-3)$$

- 2) În cadrul unei bucle, HSCD testează, pentru fiecare FModX din setul M , dacă poate fi planificat în intervalul de analiză, cu alte cuvinte, dacă pentru fiecare $M_i \in M$, $i = 1 \dots n-1$, se verifică relația:

$$t_{Si} = SCT_Hi:Lo_{Mi} + PER_Hi:Lo_{Mi} < t_e \quad (10-4)$$

În $SCT_Hi:Lo_{Mi}$ este memorată ultima instanță de planificare reușită a FModX-ului M_i .

- 2.1) Dacă (10-4) nu se verifică, înseamnă că FModX-ul M_i nu mai poate fi planificat în ciclul curent, și se trece la următorul FModX din set (M_{i+1}).

- 2.2) Dacă se verifică (10-4), parametrul $SCT_Hi:Lo_{Mi}$ din PDT este actualizat cu valoarea t_{Si} , și perechea $\{i, t_{Si}\}$ (adică PID-ul lui M_i și instanța de planificare) este inserată într-o listă de planificări ale ciclului curent, ordonată după timp.

- 3) În momentul în care s-a terminat baleierea tuturor FModX-urilor din M și s-a construit lista ordonată după timp a planificărilor acestora în cadrul ciclului curent, HSCD completează Tabela de Dispatch.

HDis_Tab este completată, pornind de la prima înregistrare a sa (de la baza tabelii), parcurgând lista în ordinea crescătoare a instanțelor de planificare. Ca rezultat, HDis_Tab va conține toate perechile de forma $\{PID_i, t_{Si}\}$ cu planificările FModX-urilor din M pentru ciclul curent.

- 4) În final, HSCD completează următoarea înregistrare liberă din HDis_Tab cu planificarea HSCD care încheie ciclul curent: $\{PID_{HSCD}, t_e\}$, și își actualizează valoarea parametrului $SCT_Hi:Lo_{HSCD}$ din PDT, conform (10-2).

10.4.2 Analiza codului HSCD

Așa cum s-a subliniat în paragrafele anterioare, implementarea algoritmului de planificare online trebuie făcută cu o deosebită atenție, avându-se în vedere în special minimizarea timpului de execuție al HSCD.

În forma actuală, HSCD a fost implementat pe platforma Motorola DSP56307, în întregime în limbaj de asamblare. Timpul maxim de execuție al HSCD depinde în mod direct de următoarele structuri de cod:

- (i) Secvența inițializărilor (include pasul 1 din algoritmul descris mai sus);
- (ii) Bucla de analiză a planificărilor pentru ciclul curent (include pasul 2). Secvența de analiză este practic formată din două bucle imbricate.

Bucula exterioră testează planificabilitatea în cadrul ciclului curent a fiecărui FModX, pe rând. În consecință, durata de execuție a buclei exterioare depinde de numărul total de FModX-uri din setul M (excluzând pe HSCD), pe care îl notăm cu n , și de numărul maxim de planificări admise pentru ciclurile de planificare, cu alte cuvinte, de dimensiunea utilă maximă a Tabelei de Dispatch a sistemului (totalul de înregistrări din HDis_Tab, excluzând planificarea HSCD de la finalul ciclului), notată aici cu λ . În consecință, timpul maxim de execuție a buclei exterioare este de ordinul:

$$WCET_{BuclaExt} = O(n + \lambda) \quad (10-5)$$

Astfel, în (10-5), λ reprezintă totalul încercărilor reușite de planificare în cadrul ciclului curent (trebuie să fie mai mic sau egal cu numărul maxim de planificări admise de HDis_Tab), iar n reprezintă totalul încercărilor nereușite de planificare (câte o încercare pentru fiecare FModX din M).

Bucula interioară se parcurge de fiecare dată când o încercare de planificare a reușit. Execuția buclei interioare constă din parcurgerea listei ordonate a planificărilor din ciclul curent și inserarea noii planificări în listă. Cum dimensiunea curentă a listei crește cu fiecare nouă planificare reușită, și cum λ este numărul maxim de planificări posibile într-un ciclu, durata maximă de execuție a buclei interioare este de ordinul:

$$WCET_{BuclaInt} = O(1 + 2 + \dots + \lambda) = O(\lambda(\lambda+1)/2) \quad (10-6)$$

- (iii) Bucula de parcurgere a listei ordonate pentru citirea planificărilor și completarea Tabelei de Dispatch. Secvența de cod include operațiile specificate la pasul 3 al algoritmului HSCD. Durata maximă de execuție este de ordinul:

$$WCET_{CitireListă} = O(\lambda) \quad (10-7)$$

- (iv) Secvența finală de cod a HSCD, ce include pasul 4 al algoritmului.

Figura 10-11 prezintă analiza duratei maxime de execuție a secvențelor de cod corespunzătoare buclelor imbricate de determinare a planificărilor pentru ciclul curent (ii). Timpii de execuție ai secvențelor de cod sunt calculați în cicluri de tact procesor (CLK). Pentru durata maximă de execuție a secvenței de determinare a planificărilor a rezultat valoarea:

$$WCET_{AnalizăPlanificări} = 15 \cdot (\lambda + n) + 2 \cdot n + 48 \cdot \lambda + 13 \cdot \lambda \cdot (\lambda + 1) \text{ [CLK]} \quad (10-8)$$

Analiza întregii secvențe de cod a planificatorului online HSCD este ilustrată în Figura 10-12. Durata maximă de execuție a codului HSCD devine:

$$WCET_{CodHSCD} = 56 + 71 \cdot \lambda + 15 \cdot (\lambda + n) + 2 \cdot n + 13 \cdot \lambda \cdot (\lambda + 1) \text{ [CLK]} \quad (10-9)$$

Observație

Datorită faptului că HSCD este, la rândul său, un FModX care se execută în contextul HRT al nucleului HARETICK, la durata sa de execuție se adaugă și cea a executivului HDIS (Secțiunea 9.6):

$$WCET_{HSCD} = WCET_{CodHSCD} + WCET_{HDIS}$$

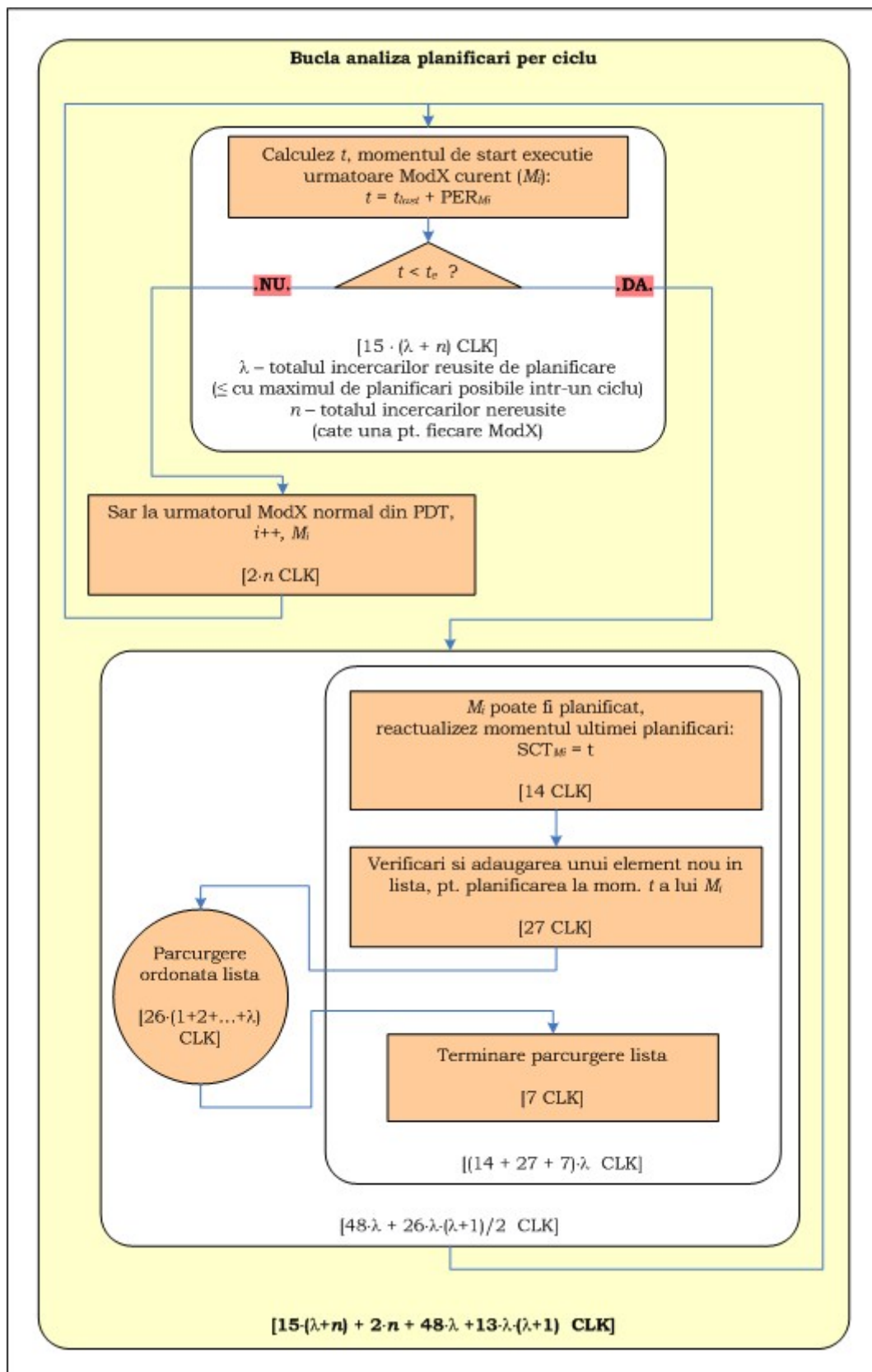


Figura 10-11. Secvența de cod HSCD pentru analiza planificărilor din ciclul curent

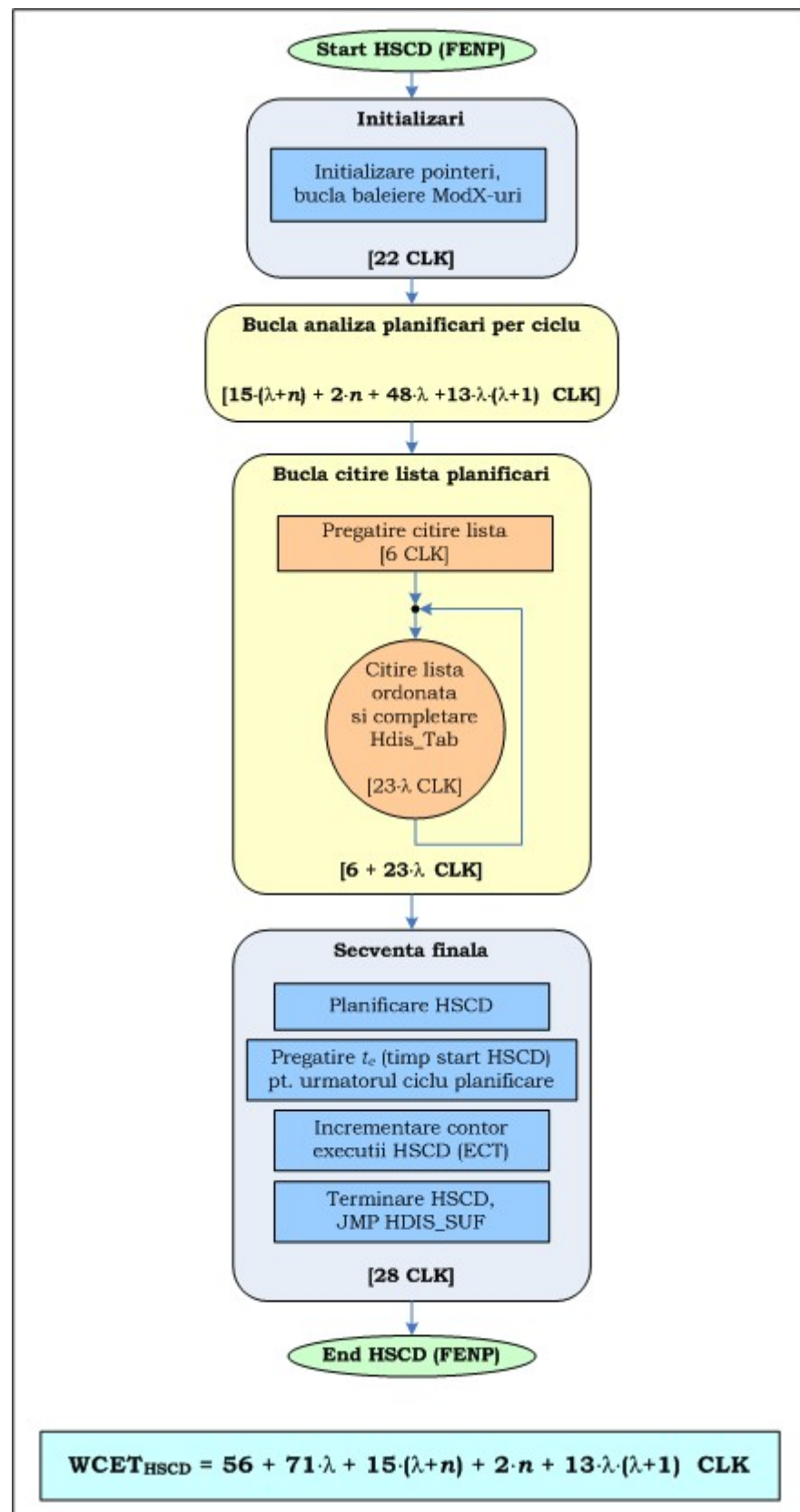


Figura 10-12. Structura întregii secvențe de cod a HSCD

10.5 Concluzii

Nucleul HARETICK este conceput pentru a oferi suport execuției pe platforme de control încorporat pentru aplicațiile timp-real, care, de regulă, conțin atât task-uri cu specificații temporale stricte (task-uri HRT) cât și task-uri cu specificații lejere (SRT). În HARETICK, cele două tipuri de task-uri se execută în cadrul a două contexte concurente:

- (1) Contextul strict de execuție (contextul HRT): oferă suportul necesar execuției task-urilor cu specificații temporale stricte, modelate cu ajutorul ModX-ului, garantând respectarea termenelor impuse la specificare.

ModX-urile sistem și ale aplicației sunt planificate cu ajutorul unui task sistem (ModX, la rândul său), HSCD, și se execută în regim non-preemptiv.

Contextul HRT are prioritatea cea mai mare în cadrul nucleului și este activat într-un mod complet predictibil, la instanțele de timp ce marchează începutul execuției ModX-urilor planificate. Activarea contextului HRT este realizată de către executivul nucleului, HDIS.

- (2) Contextul lejer de execuție (contextul SRT): este specific task-urilor cu specificații lejere de comportare temporală. Contextul SRT a fost propus și dezvoltat în cadrul nucleului HARETICK din considerente legate de creșterea eficienței și flexibilității întregului sistem.

Contextul SRT are precedență mai mică față de cel HRT, fiind întrerupt de către acesta prin mecanismul de ceas timp-real (RTC) al sistemului. Comutarea celor două contexte este efectuată de către HDIS, care, în aceste situații se ocupă și cu salvarea și restaurarea contextului SRT.

Planificarea task-urilor lejere este realizată într-o manieră clasică, de tip "time-sharing", pe bază de priorități, de către SSCD – planificatorul contextului SRT (un task SRT, la rândul său).

Datorită importanței task-urilor HRT în sistemele timp-real, preocupările noastre de cercetare-dezvoltare s-au concentrat asupra modului în care nucleul HARETICK tratează aceste task-uri în cadrul contextului strict de execuție. Pe parcursul operării, ModX-urile se pot găsi în 5 stări distincte:

- NOP ("No Operation") descrie starea ModX-urilor care nu au fost încărcate încă pe platforma țintă și nucleul nu are cunoștință de existența acestora;
- RDY ("Ready for Scheduling") este starea caracteristică ModX-urilor încărcate în sistemul HARETICK. Structurile de date necesare reprezentării ModX-urilor în cadrul nucleului au fost alocate și inițializate cu valorile corespunzătoare;
- SCD ("Scheduled") descrie starea ModX-urilor care sunt planificate în cadrul ciclului curent de planificare, fiind astfel prezente în Tabela de Dispatch a nucleului;
- RUN ("Running") reprezintă starea unui singur ModX, care la un moment dat este în execuție în cadrul sistemului țintă;

- GST ("Ghost") descrie starea ModX-ului planificat pentru execuție la momentul curent, dar care are contorul de execuții nul. Execuția efectivă a ModX-ului nu se mai petrece.

Rolul de a comuta cele două contexte de execuție ale HARETICK și de a salva și restaura informația contextului SRT la întreruperea acestuia, îi revine componentei de bază a nucleului – executivul HDIS. Executivul este compus din trei module: prefix (HDIS_PRE), sufix (HDIS_SUF) și TiLT ("Time Log Tool") și utilizează o structură sistem statică pentru lansarea în execuție a ModX-urilor planificate în cadrul unui ciclu: Tabela de Dispatch (HDis_Tab).

HDIS_PRE este apelat întotdeauna de către întreruperea ceasului timp-real (RTC) a sistemului, la instanțele de timp corespunzătoare planificării pentru execuție a ModX-ului curent din Tabela de Dispatch. Înaintea lansării în execuție a ModX-ului curent, HDIS_PRE setează regiștrii de comparare ai RTC pentru generarea întreruperii ce va iniția execuția ModX-ului următor din HDis_Tab. De asemenea, dacă a fost întrerupt contextul SRT, componenta prefix a executivului salvează starea și regiștrii procesor.

HDIS_SUF este apelat la terminarea execuției fiecărui ModX din sistem și are rolul de a decide comutarea contextelor (HRT către SRT). Efectuarea operației de comutare este precedată de restaurarea contextului task-ului SRT întrerupt de către HDIS_PRE.

Rutina TiLT are rolul de a înregistra principalele evenimente legate de execuția ModX-urilor într-o structură de tip buffer circular.

Planificarea execuției în contextul strict al HARETICK este efectuată de către HSCD, pe baza noțiunii de "ciclu de planificare", care este definit ca intervalul de timp dintre două execuții consecutive ale HSCD.

În prezent, a fost proiectată, implementată și testată pe platforma Motorola DSP56307 o versiune de HSCD bazată pe algoritmul de planificare non-preemptivă a ModX-urilor cu execuție fixă în cadrul perioadei (FModX-uri): algoritmul FENP (vezi Secțiunea 5.4). Proiectarea și implementarea planificatorului (scris în limbaj de asamblare pentru platforma țintă menționată) au fost realizate cu o deosebită atenție, vizându-se minimizarea duratei de execuție, astfel încât eficiența sistemului să fie cât mai puțin afectată.

11 Testarea și evaluarea performanțelor sistemului HARETICK

În stadiul curent de dezvoltare și implementare, nucleul HARETICK oferă suport complet funcțional pentru aplicațiile critice compuse din ModX-uri cu execuție fixă în cadrul perioadei (FModX-uri). Astfel, modulele sistem de bază ale contextului strict din cadrul nucleului au fost implementate și testate cu succes:

- structurile de date sistem de bază (tabela descriptorilor de proces PDT, Tabela de Dispatch HDis_Tab, etc.),
- ceasul timp-real (RTC), bazat pe un timer procesor pentru partea cea mai puțin semnificativă și o variabilă sistem pentru partea cea mai semnificativă,
- task-ul de inițializare SYSINIT,
- o versiune simplă de LOADER,
- executivul HDIS,
- diferite versiuni ale planificatorului HSCD.

Capitolul de față prezintă rezultatele aplicațiilor și testelor de evaluare a performanțelor componentelor nucleului și ale întregului sistem. Experimentele se bazează pe implementarea HARETICK pe platforma Motorola DSP56307 EVM [Motorola 99].

Platforma țintă a fost configurată pentru operarea la o frecvență a tactului intern de 32 MHz (1 CLK), iar ceasul timp-real al HARETICK aplică un factor de divizare de 1/4 frecvenței de tact procesor. Prin urmare, unitatea de timp sistem are valoarea (vezi și Subcapitolul 9.1):

$$\Delta t_{RTC} = 4 \cdot \Delta t_{CLK} = 125 \text{ ns (8 MHz)} \quad (11-1)$$

11.1 Testarea și analiza operării executivului HDIS

Executivul HDIS este modulul de bază al nucleului HARETICK, fiind responsabil de operarea corectă a contextului strict de execuție (cu implicații directe asupra mediului și/sau a operatorilor în cazul aplicațiilor critice). De asemenea, HDIS este responsabil de comutarea corectă a celor două contexte de execuție ce operează concurrent în cadrul nucleului: contextul HRT și SRT.

Implementarea și testarea executivului HDIS este unul din primii pași necesari pentru dezvoltarea nucleului HARETICK. În acest scop, pe platforma Motorola DSP56307 EVM, a fost conceput și implementat un cadru simplu de operare, în care o structură particulară a Tabelei de Dispatch (HDis_Tab) și o versiune simplă de planificare HRT sunt elementele cheie. În acest mediu de operare, a fost apoi dezvoltată o aplicație simplă, cu ajutorul căreia se pot efectua teste și măsurători practice ale execuției în contextul strict al nucleului.

Structura HDis_Tab a fost concepută ca un buffer circular cu dimensiune bine stabilită (1000 înregistrări pentru planificări de ModX-uri normale + 1 înregistrare pentru planificarea HSCD). Prima înregistrare din HDis_Tab este alocată totdeauna planificărilor HSCD. În Figura 11-1 este prezentată structura HDis_Tab și configurația sa pentru cazul primului ciclu de planificare a ModX-urilor aplicației de testare a executivului.

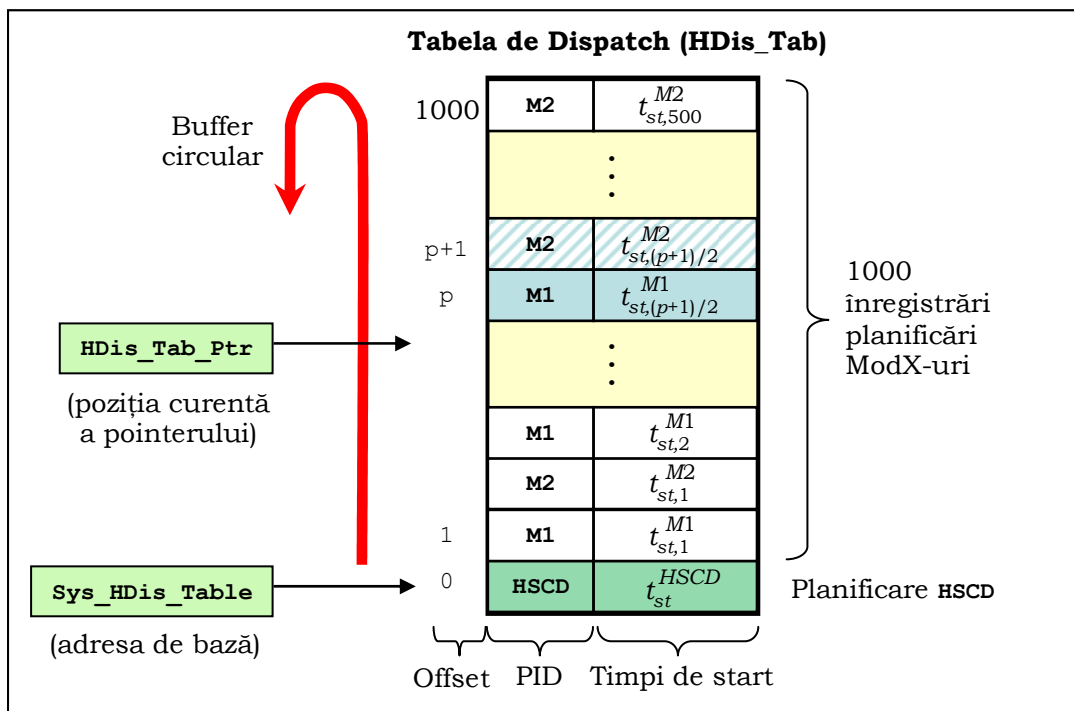


Figura 11-1. Structura HDis_Tab ca buffer circular, pentru planificarea simplă

Modulul de planificare HSCD (un ModX, rândul său), implementează un algoritm simplu, ce constă din completarea alternativă a înregistrărilor HDis_Tab cu planificările ModX-urilor aplicației (vezi Figura 11-1). Timpii de planificare ai ModX-urilor sunt calculați în așa fel încât momentul curent al planificării unui anume ModX apare la terminarea execuției ModX-ului planificat anterior. De exemplu, pentru prima planificare a lui M_2 din Figura 11-1 (înregistrarea a treia din HDis_Tab), momentul de start este calculat astfel:

$$t_{st,1}^{M_2} = t_{st,1}^{M_1} + t_{ex}^{M_1} \tag{11-2}$$

Când execuția în contextul strict al nucleului a ajuns la ultima planificare din HDis_Tab, pointerul HDis_Tab_Ptr va fi actualizat automat pentru a referi prima înregistrare a tabelii, care reprezintă planificarea HSCD ce definește un nou ciclu de planificare.

Principalii parametri de implementare a componentelor nucleului HARETICK pentru procedurile de testare a executivului HDIS, sunt sintetizați în Tabelul 11-1.

Tabelul 11-1. Parametrii de implementare a componentelor HARETICK

Denumire	Descriere	Dimensiune [24-bit words]	WCET [CLK]
PDT	Tabela descriptorilor de proces	1300	–
HDis_Tab	Tabela de Dispatch (1000 + 1 înregistrări)	3003	–
SRT_Context	Zonă de memorie pentru salvarea și restaurarea contextului SRT	39	–
TiLT_Tab	Buffer pentru rapoartele TiLT	5120	–
Sys_Heap	Zonă de tip "heap" pentru variabilele și parametrii ModX-ului în execuție	4096	–
RTC_OVR	Rutina de tratare a întreruperii de depășire a timerului RTC	26	37
TiLT	Subrutina "Time Log Tool"	33	43
HDIS_PRE	Componenta prefix a executivului	214	226
HDIS_SUF	Componenta sufix a executivului	177	137
HDIS	Executivul nucleului	450	486
HSCD	Planificatorul contextului HRT	75	36534
LOADER	Task-ul de încărcare a aplicațiilor în sistem și de gestionare a memoriei (task SRT)	40	–
SYSINIT	Task-ul de inițializare a sistemului (task SRT)	68	–

Durata maximă de execuție pentru HDIS ($WCET_{HDIS}$) rezultă din însumarea duratelor maxime ale rutinei de tratare a întreruperii de depășire (RTC_OVR), a componentei prefix (HDIS_PRE) ce apelează subrutina TiLT, și a componentei sufix (HDIS_SUF) ce apelează și ea subrutina TiLT:

$$WCET_{HDIS} = WCET_{RTC_OVR} + \left(WCET_{HDIS_PRE} + WCET_{TiLT} \right) + \left(WCET_{HDIS_SUF} + WCET_{TiLT} \right)$$

Observație

Deoarece la durata de execuție a oricărui ModX care operează în sistemul HARETICK se adaugă durata executivului, $WCET_{HDIS}$, acest parametru reprezintă o limită minimă. Cu alte cuvinte, pentru orice ModX din sistem nu se va putea considera o durată de execuție mai scurtă decât $WCET_{HDIS}$ (aici, 486 cicluri de CLK).

De asemenea, comportarea ModX-urilor fantomă este caracterizată de durata de execuție a HDIS.

Din Tabelul 11-1 se poate observa că planificatorul HSCD are durata de execuție cea mai mare. Aceasta se datorează numărului relativ mare de înregistrări ale HDis_Tab pe care HSCD trebuie să le completeze (să calculeze planificările): $1000 (\text{ModX-uri normale}) + 1$ (planificarea execuției următoare HSCD).

Aplicația dezvoltată cu scopul de a testa, măsura și evalua performanțele executivului HDIS, este compusă din trei task-uri: task-ul lejer de inițializare a aplicației, $() \text{AppInit} ()$, obligatoriu pentru toate aplicațiile ce rulează pe platforme HARETICK, și două ModX-uri, M_1 și M_2 . Principalul rol al celor două ModX-uri este comanda unui semnal de ieșire, care să poată fi măsurat cu ajutorul unui osciloscop.

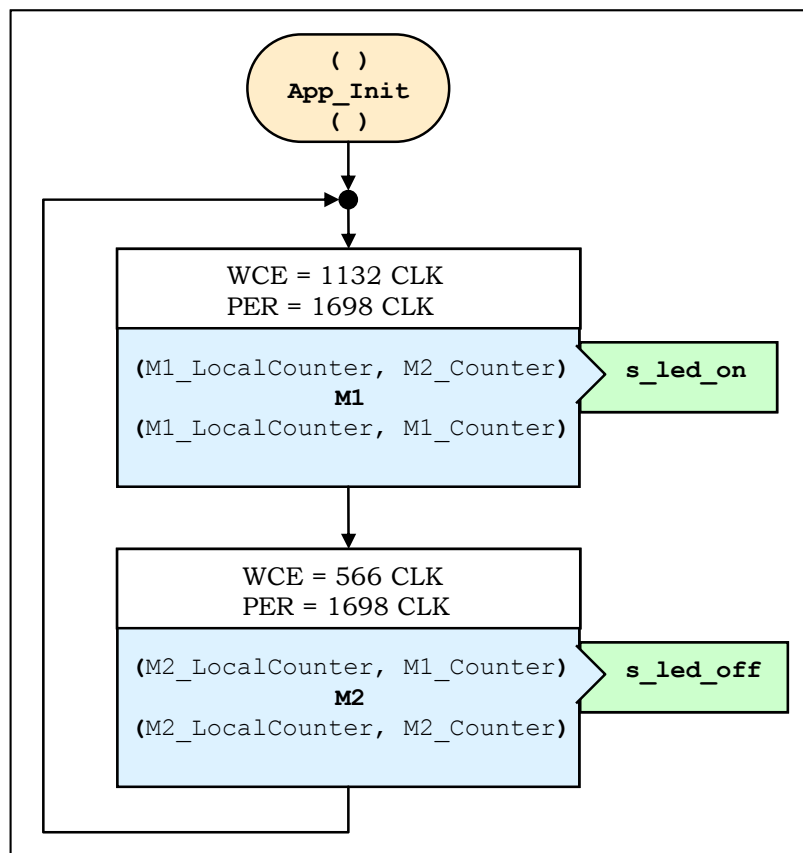


Figura 11-2. Aplicația de testare a executivului HARETICK

Figura 11-2 prezintă graficul aplicației de testare a executivului. Parametrii de intrare ale task-urilor apar în parantezele superioare iar parametrii de ieșire, în parantezele inferioare denumirii task-urilor.

Din punct de vedere al specificațiilor funcționale, ModX-ul M_1 setează un pin de ieșire al platformei țintă ($s_led_on \leftarrow "1"$), după care incrementează contorul local ($M1_LocalCounter \leftarrow M1_LocalCounter + 1$) și în final adună valoarea 2 la contorul pe care îl transmite lui M_2 ($M1_Counter \leftarrow M2_Counter + 2$). Pe de altă parte, ModX-ul M_2 resetează pinul de ieșire al sistemului ($s_led_off \leftarrow "0"$), își incrementează contorul local ($M2_LocalCounter \leftarrow M2_LocalCounter + 1$), și adună 2 la contorul pe care îl transmite lui M_1 ($M2_Counter \leftarrow M1_Counter + 2$).

Observație

Înainte de a se executa secvențele de cod corespunzătoare specificațiilor funcționale descrise anterior, fiecare ModX își va copia proprii parametri de intrare în variabilele locale corespondente (alocate în zona "Heap" a sistemului).

Se poate observa de exemplu că, în cazul lui M_1 , `M1_LocalCounter` este atât un parametru de intrare, cât și de ieșire, transmițând astfel o anumită stare (valoare) de la o execuție la alta a lui M_1 .

Terminarea ModX-urilor se face în mod obligatoriu cu apelarea componentei sufix a executivului, `HDIS_SUF`.

Task-ul de inițializare al aplicației rulează în context SRT și are ca principale roluri de a inițializa parametrii de ieșire ai ModX-urilor M_1 și M_2 (`M1_LocalCounter ← 0; M1_Counter ← 0; M2_LocalCounter ← 0; M2_Counter ← 0`), și de inițializare a portului de ieșire pentru pinul ce va fi comandat de M_1 și M_2 (`s_led_on` și `s_led_off`).

Tabelul 11-2. Parametrii de implementare ai celor două ModX-uri ale aplicației

Denumire	Descriere	Dimensiune [24-bit words]	WCET [CLK]
M1	Setează pinul de ieșire (<code>s_led_on</code>)	12	1132
M2	Resetează pinul de ieșire (<code>s_led_off</code>)	12	566

Așa cum rezultă din parametrii specificați pentru cele două ModX-uri ale aplicației (Tabelul 11-2 și Figura 11-2) și din tipul algoritmului de planificare pe care îl implementează HSCD, ModX-urile M_1 și M_2 vor avea o comportare *pseudo-periodică*, adică periodică pe durata fiecărui ciclu de planificare. Valoarea perioadei ModX-urilor rezultă ca suma duratelor individuale de execuție a acestora. Comportarea periodică este întreruptă pe parcursul execuției efective a planificatorului HSCD.

Ca urmare, la pinul de ieșire al platformei țintă se va putea măsura un semnal de tip rectangular, cu un factor de umplere de 2/3. Punând în corespondență duratele ce caracterizează comportarea aplicației, exprimate în cicli de tact procesor, cu valoarea unității de timp sistem, Δt_{RTC} (conform (11-1)), vor rezulta o serie de timpi (Tabelul 11-3) ce pot fi verificați practic, prin măsurarea cu ajutorul osciloscopului a semnalului rectangular de ieșire.

Măsurătorile au fost efectuate cu un osciloscop digital de tip HAMEG "Analog and Digital Scope, HM1507-3". Rezultatele obținute pentru măsurarea perioadei și a duratei de execuție a planificatorului HSCD sunt prezentate în Figura 11-3 și Figura 11-4. Se pot observa execuțiile periodice ale lui M_1 și M_2 în cadrul unui ciclu de planificare, definit de două execuții consecutive ale HSCD. Măsurarea duratelor de execuție ale lui M_1 și M_2 , și a perioadei acestora, este prezentată în cele trei componente din Figura 11-5.

Tabelul 11-3. Timpii caracteristici ai aplicației (valori rezultate din specificații)

Denumire	Descriere	Valoare [μ s]
$WCET_{HSCD}$	Durata de execuție HSCD	1141,750
PER_{HSCD}	Perioada HSCD	27766,750
$WCET_{M_1}$	Durata de execuție M_1	35,500
$WCET_{M_2}$	Durata de execuție M_2	17,750
$PER_{M_1} = PER_{M_2}$	Perioada M_1 și M_2	53,250

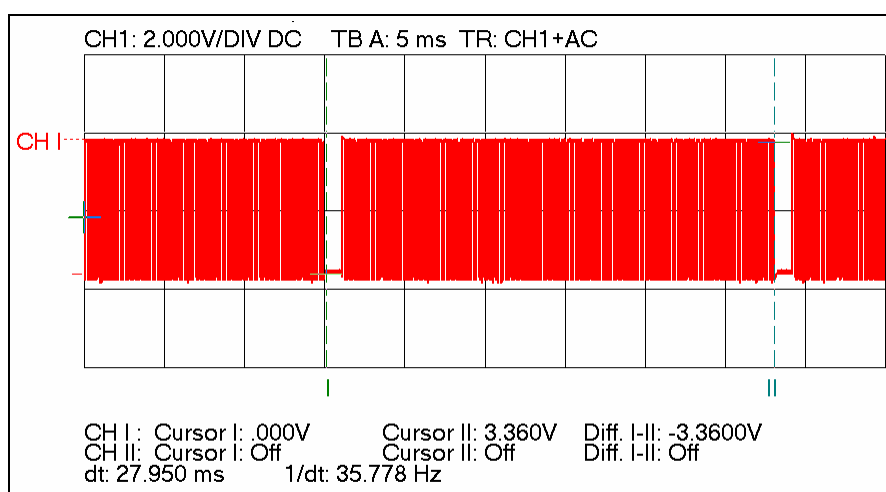


Figura 11-3. Măsurarea perioadei planificatorului HSCD

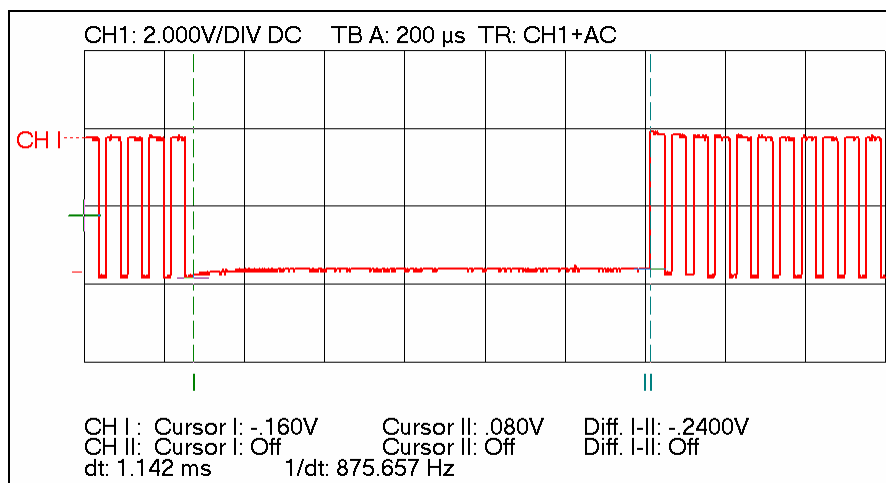


Figura 11-4. Măsurarea duratei de execuție a HSCD

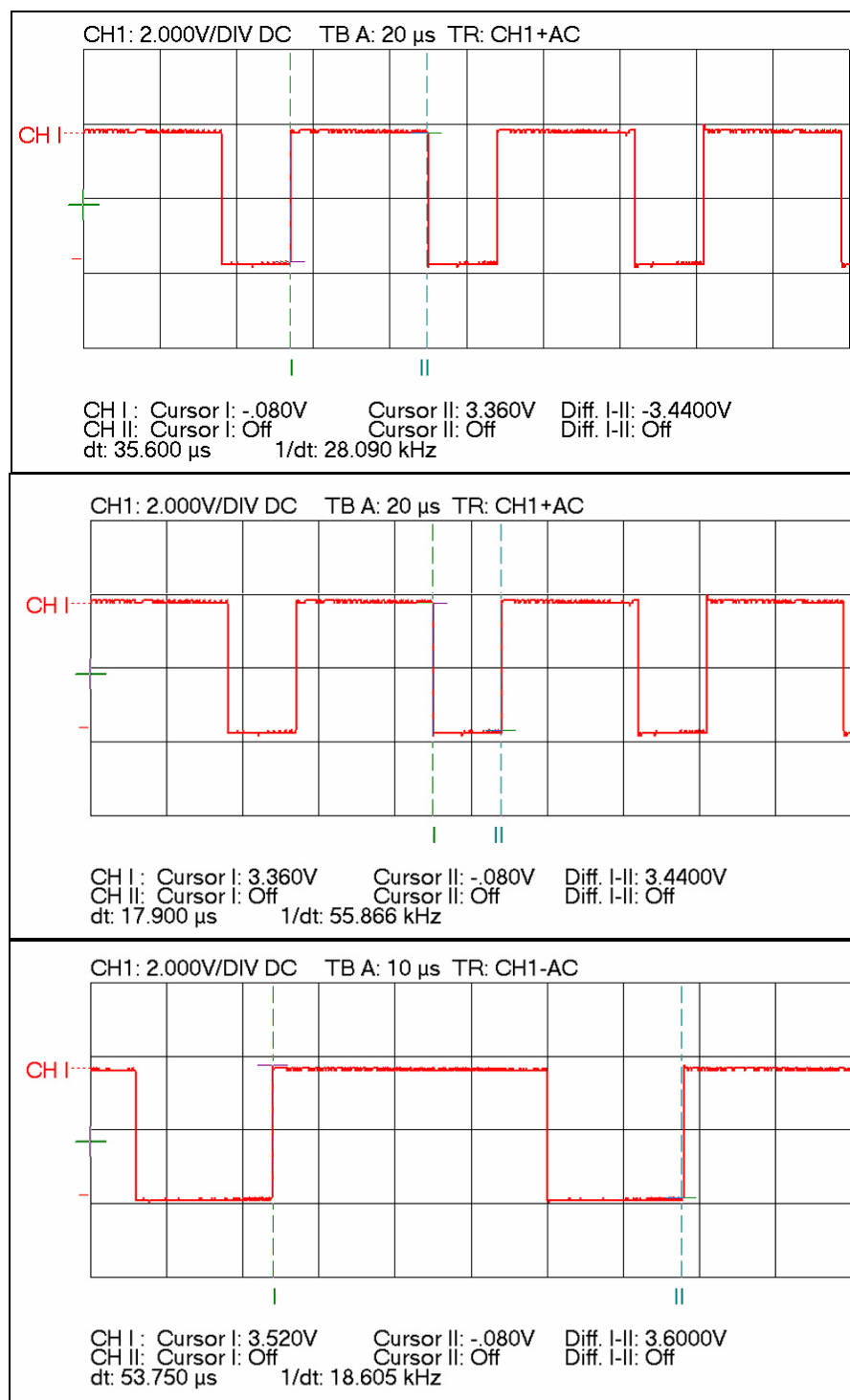


Figura 11-5. Măsurarea duratelor de execuție a lui M_1 (sus) și M_2 (mijloc), și a perioadei celor două ModX-uri (jos)

Rezultatele experimentelor și măsurătorilor practice confirmă operarea corectă a aplicației, conform specificațiilor (Tabelul 11-3), și prin urmare, executivul HDIS al nucleului conferă contextului strict de execuție al HARETICK caracteristicile dorite și specificate în faza de modelare și proiectare a sistemului.

11.2 Aplicații pentru evaluarea performanțelor sistemului

Pentru testarea și evaluarea performanțelor întregului sistem HARETICK, este necesară o configurație completă de execuție a aplicațiilor timp-real, ce include o versiune funcțională a planificatorului HSCD (spre deosebire de versiunea de planificare simplă utilizată în paragrafele anterioare).

Componentele nucleului HARETICK utilizate pentru aplicațiile următoare de evaluare a performanțelor sistemului nu sunt modificate față de cele descrise anterior (Tabelul 11-1), cu excepția planificatorului online HSCD.

HSCD implementează o variantă complet funcțională a algoritmului de planificare non-preemptivă a ModX-urilor cu execuție fixă în cadrul perioadei, FENP (Subcapitolul 10.4 și Capitolul 5). Cum parametrii temporali (reali) ai HSCD depind de cei ai aplicației încărcată la un moment dat în sistem, vor trebui respectate și verificate următoarele relații:

$$\begin{aligned} T_{ex}^{HSCD} &\equiv WCET_{HSCD} = f(n, \lambda) = \\ &= 56 + 71 \cdot \lambda + 15(\lambda + n) + 2 \cdot n + 13 \cdot \lambda(\lambda + 1) + WCET_{HDIS} [\text{CLK}] \end{aligned} \quad (11-3)$$

pentru durata de execuție a HSCD, care este direct dependentă de numărul total de FModX-uri ale aplicației (n) și de numărul maxim de planificări dintr-un ciclu (λ);

$$T_{ex}^{HSCD} \leq 2 \left(T_{pr}^{M_{pr \min}} - T_{ex}^{M_{pr \min}} \right) \quad (11-4)$$

care limitează durata de execuție a HSCD la dublul diferenței dintre perioada și durata de execuție a FModX-ului cu perioada cea mai mică din set (Capitolul 5);

$$T_{pr}^{HSCD} \Big|_{\min} = f(n, \lambda) = \left[\frac{\lambda - n}{2 \cdot \sum_{i=1}^n \frac{1}{T_{pr}^{M_i}}} \right] + T_{ex}^{HSCD} \quad (11-5)$$

pentru perioada minimă a HSCD, (la rândul ei o funcție de n și λ). Limita superioară a perioadei este dată de condiția de planificare online periodică (vezi Capitolul 5):

$$\sum_{i=1}^n \left[\frac{2 \left(T_{pr}^{HSCD} - T_{ex}^{HSCD} \right)}{T_{pr}^{M_i}} \right] \leq \lambda \quad (11-6)$$

În plus, pentru setul de FModX-uri va trebui verificată (în faza de analiză, premergătoare încărcării în sistem) condiția de necesitate a planificării FENP:

$$T_{ex}^{M_i} + T_{ex}^{M_j} \leq GCD \left\{ T_{pr}^{M_i}, T_{pr}^{M_j} \right\}, \quad \forall i, j = 1..n, i \neq j \quad (11-7)$$

adică, suma duratelor de execuție a oricăror două FModX-uri să nu depășească valoarea celui mai mare divizor comun al perioadelor acestora.

Reamintim faptul că pentru această variantă de planificator online, nucleul HARETICK utilizează faza de pre-planificare (în context SRT) pentru determinarea planificărilor primului ciclu.

11.2.1 Generarea unui semnal rectangular perfect periodic

Aplicația descrisă în subcapitolul anterior este utilizată într-o formă similară din punct de vedere funcțional (Figura 11-2) pentru generarea unui semnal rectangular, perfect periodic, măsurabil cu ajutorul osciloscopului la un pin de ieșire al platformei țintă Motorola DSP56307 EVM.

Din punct de vedere al comportării temporale însă, valorile parametrilor task-urilor M_1 și M_2 (care sunt FModX-uri de data aceasta), vor trebui specificate și analizate conform cu parametrii semnalului rectangular dorit și cu tipul planificării utilizate (FENP).

De asemenea, parametrii temporali ai planificatorului HSCD vor fi specificați în așa fel încât să rezulte un sistem fezabil și cât mai eficient posibil. Ca urmare, pentru HSCD vor trebui determinate o durată de execuție suficient de mică și o perioadă suficient de mare. Din (11-3) rezultă că T_{ex}^{HSCD} depinde de pătratul lui λ . Considerând $\lambda = 20$ (numărul maxim de planificări într-un ciclu), și având pe $n = 2$ (numărul total de FModX-uri ale aplicației), durata de execuție a HSCD va fi:

$$WCET_{HSCD} = 7297 + WCET_{HDIS} [\text{CLK}] = 7297 + 486 [\text{CLK}]$$

Exprimată în unități RTC (conform echivalenței dată de (11-1)), durata de execuție a HSCD va putea fi stabilită la valoarea:

$$T_{ex}^{HSCD} \equiv WCET_{HSCD} = 2000 [\text{RTC}] \quad (11-8)$$

Pentru specificarea perioadei HSCD este necesară determinarea parametrilor temporali ai FModX-urilor aplicației (M_1 și M_2), astfel încât să se respecte condiția (11-4). După implementarea codului pentru M_1 și M_2 , calculul timpului maxim (efectiv) de execuție al ambelor FModX-uri a generat valoarea:

$$WCET_{M1}|_{\text{efectiv}} = WCET_{M2}|_{\text{efectiv}} = 35 + WCET_{HDIS} = 35 + 486 [\text{CLK}]$$

Rezultă că durata maximă de execuție ce poate fi specificată pentru M_1 și M_2 nu poate fi mai mică decât 131 [RTC]. Astfel, pentru parametrii temporali ai FModX-urilor M_1 și M_2 s-au ales valorile:

$$\begin{cases} T_{pr}^{M1} = T_{pr}^{M2} = 3000 [\text{RTC}] \\ T_{ex}^{M1} = 400 [\text{RTC}] \\ T_{ex}^{M2} = 200 [\text{RTC}] \end{cases} \quad (11-9)$$

Conform (11-5), perioada minimă HSCD (durata unui ciclu de planificare) este:

$$T_{pr}^{HSCD} \Big|_{\min} = \left\lceil \frac{20 - 2}{2(1/3000 + 1/3000)} \right\rceil + 2000 = 15500 [\text{RTC}]$$

Punând $T_{pr}^{HSCD} = 18000 [RTC]$, se verifică și condiția de planificare online periodică (11-6), care fixează o valoare maximă pentru perioada HSCD.

Tabelul 11-4. Parametrii de implementare ai aplicației și HSCD

Denumire	Descriere	Dimensiune [24-bit words]	WCET [CLK]
HSCD_List	Listă ordonată pentru memorarea planificărilor normale (exceptând cea a HSCD) dintr-un ciclu	400	–
HSCD	Planificatorul online (algoritm FENP)	164	8000
App_Init	Task-ul (SRT) de inițializare a aplicației	17	–
M1	Setează pinul de ieșire (s_led_on)	32	1600
M2	Resetează pinul de ieșire (s_led_off)	32	800

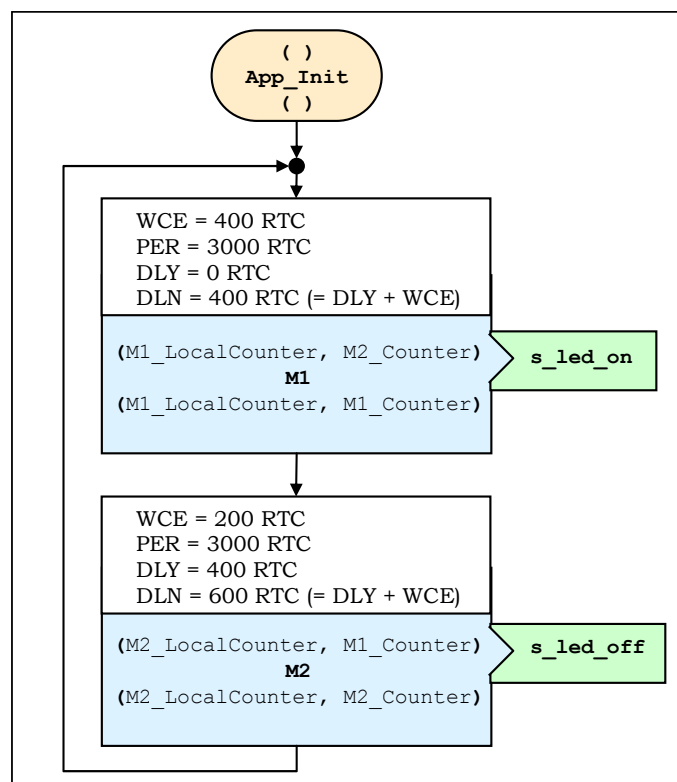


Figura 11-6. Aplicația de generare a unui semnal rectangular periodic

În acest punct, având toți parametrii temporali necesari, se poate trece la etapa de analiză a planificabilității setului extins de FModX-uri: $M^S \equiv \{M_1, M_2, HSCD\}$. În primul rând, condiția de necesitate a planificabilității, (11-7), este validată pentru toate perechile de FModX-uri din M^S . Figura 11-7 prezintă analiza planificabilității aplicației și diagrama de timp a semnalului periodic generat.

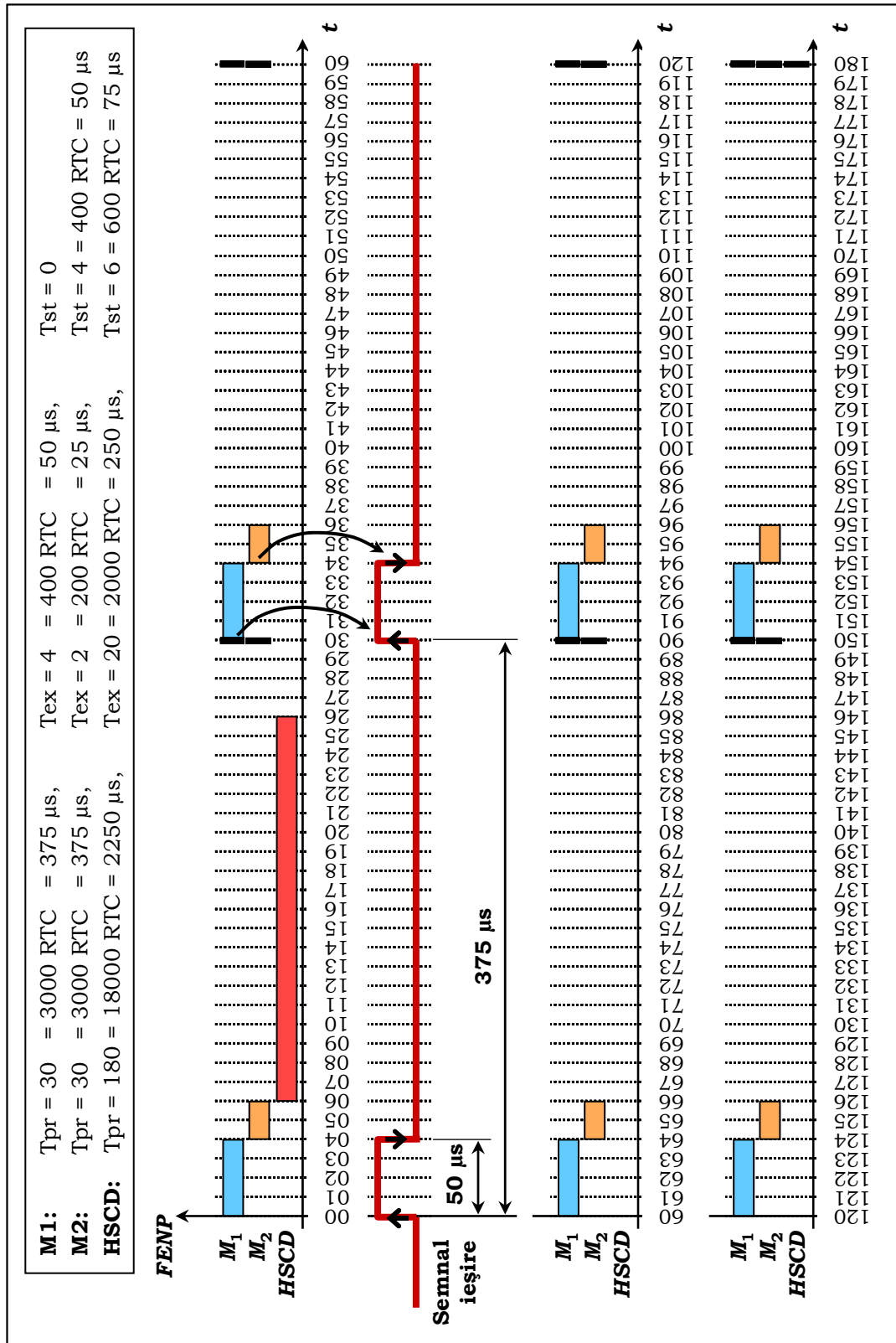


Figura 11-7. Analiza planificabilității aplicației

Un alt rezultat al analizei planificabilității îl reprezintă determinarea valorilor timpilor de start ai FModX-urilor în cadrul perioadelor acestora: $T_{st}^{M_1} = 0$ [RTC], $T_{st}^{M_2} = 400$ [RTC] și $T_{st}^{HSCD} = 600$ [RTC]. Graful aplicației, cu valorile finale ale parametrilor temporali pentru FModX-uri, este prezentat în Figura 11-6.

Utilizarea procesor a setului de FModX-uri ale aplicației, $M \equiv \{M_1, M_2\}$, are valoarea $U^M = 0,20$, iar utilizarea procesor a setului extins de FModX-uri are valoarea $U^{Ms} = 0,311$.

În Figura 11-7 se poate observa configurația planificărilor pentru primul ciclu, rezultate în urma execuției HSCD în regim de pre-planificare (P-HSCD): prima planificare a lui M_1 (la momentul 0), a lui M_2 (la momentul 4) și planificarea lui HSCD (la momentul 6). Așadar primul ciclu totalizează două planificări de FModX-uri ale aplicației și o planificare HSCD la final. Restul ciclurilor (rezultate ca urmare a execuțiilor normale HSCD, în cadrul contextului strict al nucleului) sunt compuse din câte 6 planificări ale M_1 , 6 planificări ale M_2 și o planificare HSCD la sfârșitul fiecărui ciclu. Prin urmare, există o discrepanță între configurația (și caracteristicile de timp) a ciclurilor de planificare considerate în faza de specificare a aplicației, față de situația reală din timpul operării sistemului. Comparația dintre cele două situații este prezentată în Tabelul 11-5.

Tabelul 11-5. Comparație între parametrii temporali specificați și cei efectivi, pentru FModX-urile sistem

Denumire	WCET specificat		WCET efectiv	
	[RTC]	[μ s]	[RTC]	[μ s]
P-HSCD	–	–	207	25,875
HSCD	2000	250	910	113,750
App_Init	–	–	–	–
M1	400	50	131	16,375
M2	200	25	131	16,375

Valorile efective ale WCET pentru P-HSCD și HSCD diferă de cea specificată pentru HSCD ($n = 2$, $\lambda = 20$), prin faptul că în cazul P-HSCD avem $\lambda_{efectiv} = 2$ (două planificări de FModX-uri ale aplicației) iar pentru HSCD, $\lambda_{efectiv} = 12$.

Rezultatele măsurărilor efectuate cu ajutorul osciloscopului digital tip HAMEG "Analog and Digital Scope, HM1507-3" sunt prezentate în Figura 11-8 la Figura 11-11. Semnalul periodic generat de aplicația descrisă în aceste paragrafe a fost comparat cu un semnal cu aceleași specificații, obținut cu ajutorul generatorului de funcții programabil tip HAMEG "Programmable Function Generator, HM8130". Rezultatele obținute atestă faptul că întregul sistem HARETICK, împreună cu aplicația dezvoltată și rulată pe acesta se comportă conform cerințelor impuse în faza de proiectare și specificare, operarea sistemului fiind caracterizată în același timp printr-o predictibilitate maximă.

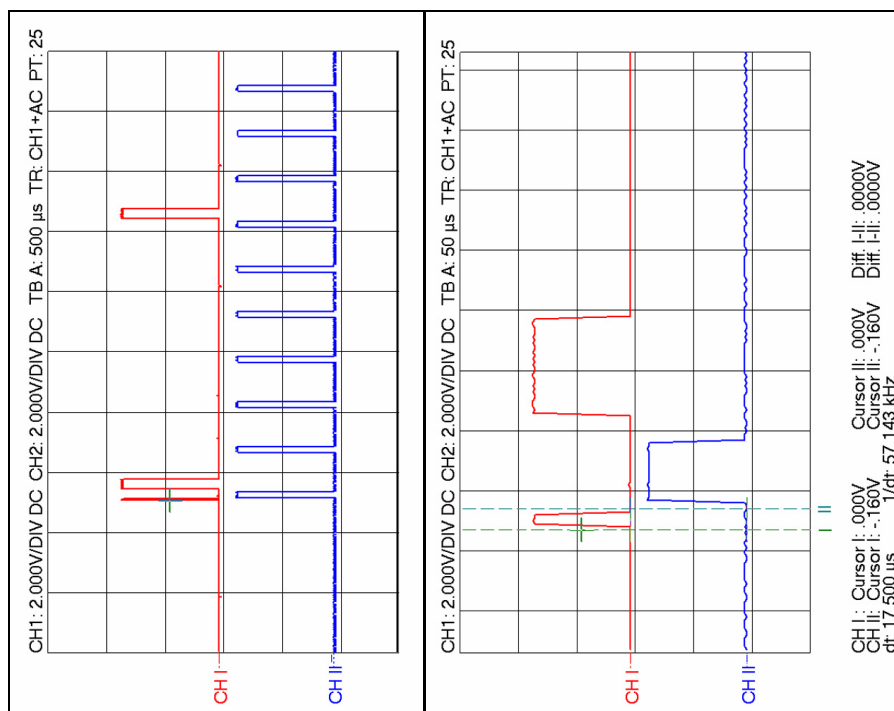


Figura 11-8. Momentul de start al operării aplicației (stânga), cu vedere de detaliu a execuției HSCD în regim de pre-planificare (dreapta). Canalul I (CH I) reprezintă execuția HSCD, iar canalul II (CH II) reprezintă semnalul periodic generat

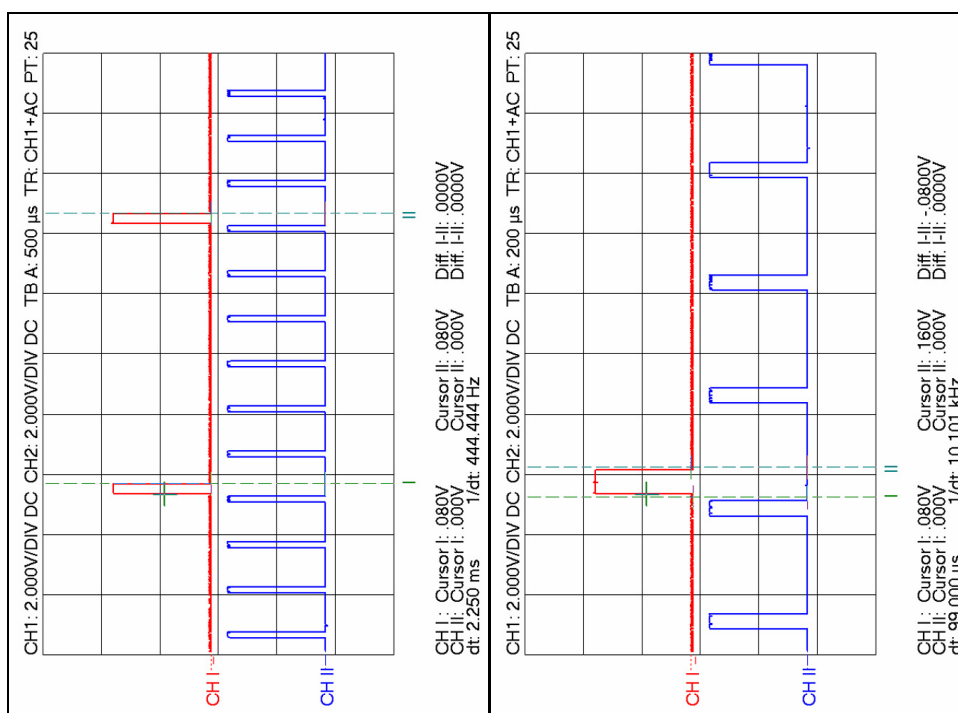


Figura 11-9. Perioada HSCD (durata unui ciclu de planificare) (stânga) și durata de execuție a HSCD (dreapta)

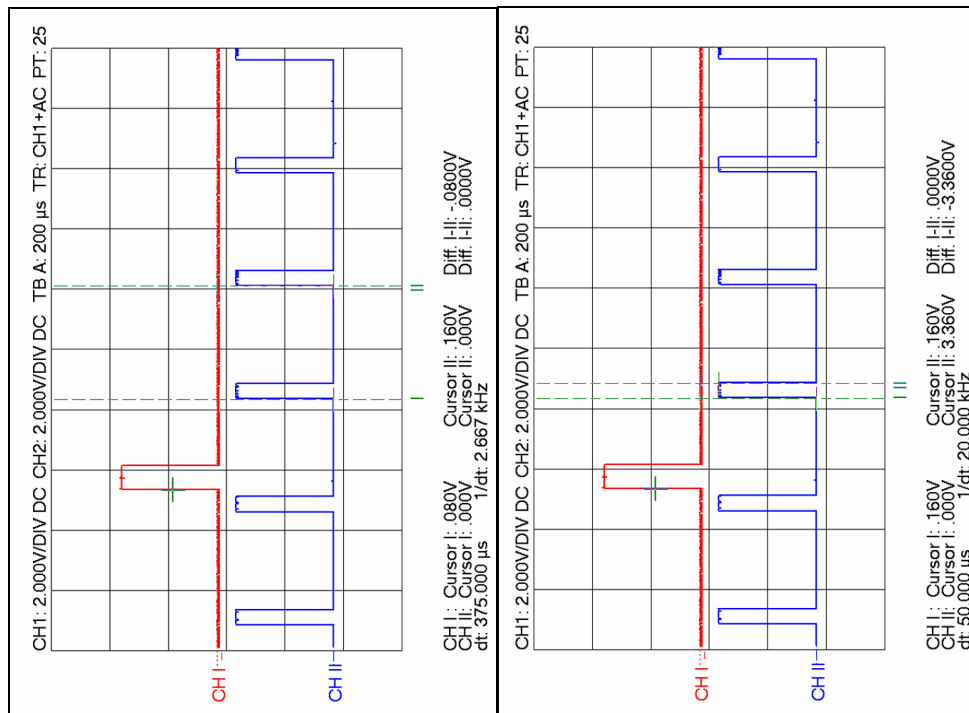


Figura 11-10. Perioada (stânga) și lățimea impulsului (dreapta) pentru semnalul rectangular generat

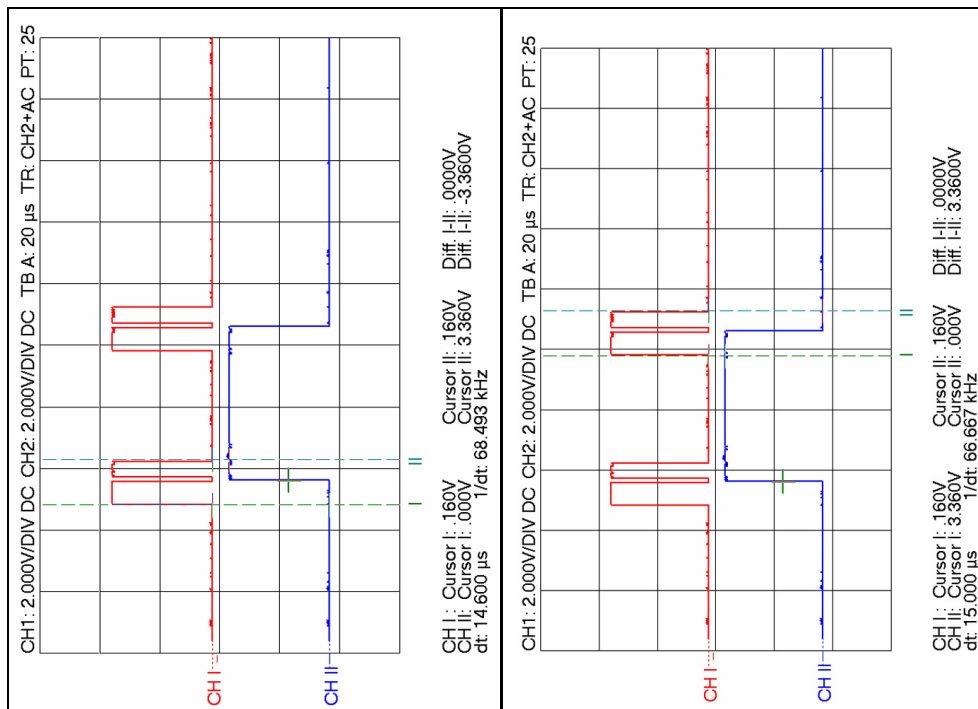


Figura 11-11. Duratele de execuție ale FModX-urilor M_1 (stânga) și M_2 (dreapta). Canalul I (CH I) vizualizează operarea executivului HDIS al nucleului, cu cele două componente ale sale: prefix (HDIS_PRE) și sufix (HDIS_SUF), care încadrează execuția FModX-urilor.

11.2.2 Evaluarea operării ModX-urilor fantomă

ModX-urile fantomă reprezintă modalitatea utilizată de nucleul HARETICK pentru a rezolva problemele legate de execuția discontinuă a task-urilor și de tratarea sau generarea semnalelor cu un număr de apariții finit, dat. Un ModX oarecare ce operează în cadrul sistemului devine ModX fantomă în momentul în care contorul său de execuții are valoarea 0 (Execution Count, $ECT = 0$). ModX-urile fantomă sunt planificate de către planificatorul online HSCD, dar nu sunt lansate în execuție de către executivul HDIS.

Pentru testarea și evaluarea comportării nucleului HARETICK în cazul ModX-urilor fantomă a fost dezvoltată și implementată o variantă adaptată a aplicației de generare a unui semnal periodic, discutată în paragrafele precedente. Aplicația, al cărui graf este prezentat în Figura 11-12, conține două FModX-uri (M_1 și M_2), în care M_1 , pe lângă rolul de setare a semnalului de ieșire, are și sarcina de a transforma pe M_2 în ModX fantomă și invers, în mod consecutiv. Pentru aceasta, M_1 dispune de un parametru circular (în același timp, parametru de intrare și de ieșire pentru acest ModX), $M1_Semafor$, căruia îi modifică alternativ valoarea, între "0" și "1". Dacă $M1_Semafor$ este găsit cu valoarea "0" la începerea unei instanțe de execuție a lui M_1 , acesta setează contorul de execuții al lui M_2 pe 0: $M2.ECT \leftarrow 0$, transformându-l în ModX fantomă. În celălalt caz, M_1 setează contorul lui M_2 pe "execuție continuă": $M2.ECT \leftarrow CONT$, transformându-l pe acesta din nou în FModX normal.

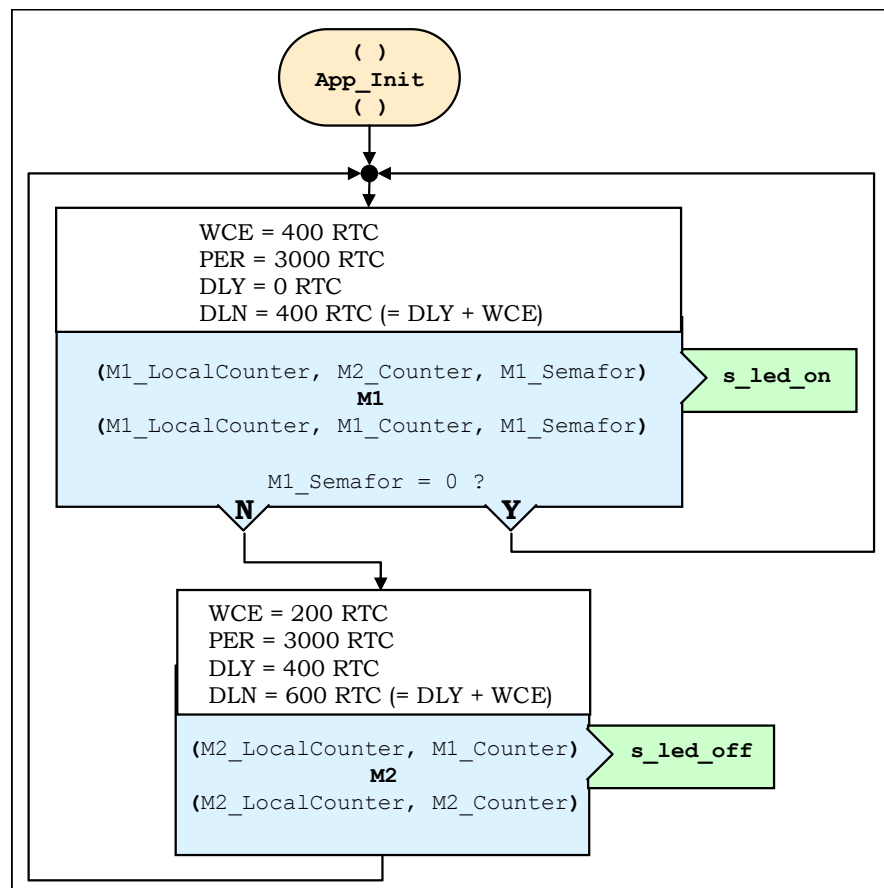


Figura 11-12. Graful aplicației de evaluare a ModX-urilor fantomă

Analiza aplicației este similară cu cea descrisă în Subcapitolul precedent (Figura 11-7), iar parametrii temporali ai FModX-urilor M_1 , M_2 și ai HSCD sunt, de asemenea, identici cu cei ai aplicației precedente. Semnalul generat la pinul de ieșire al platformei Motorola DSP56307 EVM este tot un semnal perfect periodic, având însă perioada de $750 \mu\text{s}$ și durata impulsului de $425 \mu\text{s}$. Aceste caracteristici ale semnalului de ieșire se datorează faptului că pinul corespunzător este resetat doar din două în două planificări ale lui M_2 (vezi Figura 11-7).

Câteva din măsurătorile efectuate cu ajutorul osciloscopului sunt prezentate în Figura 11-13 și Figura 11-14. Rezultatele măsurate confirmă și pentru această aplicație comportarea corectă a întregului sistem HARETICK, conform cu parametrii specificați la proiectare și cu valorile obținute în faza de analiză offline.

În cele două figuri se poate observa și execuția planificatorului online HSCD, prin apariția semnalului generat de executiv (canalul I – CH I, pentru partea stângă a figurilor) atunci când se termină un ciclu de planificare. Perioada HSCD este, ca și în cazul aplicației precedente, de $2250 \mu\text{s}$.

Figura 11-14 ilustrează în mod sugestiv comportarea sistemului și a executivului HDIS în momentul în care un ModX fantomă este planificat pentru execuție: componenta prefix (HDIS_PRE) apelează în mod direct componenta sufix (HDIS_SUF) atunci când determină că valoarea contorului de execuții al ModX-ului ce urmează a fi lansat în execuție este nulă.

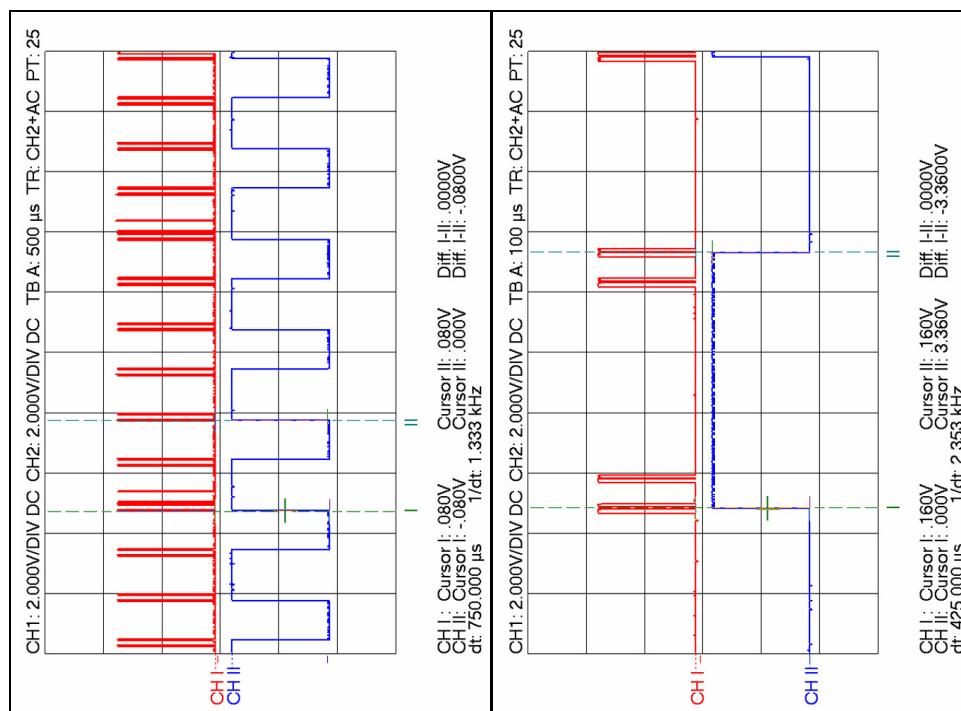


Figura 11-13. Perioada (stânga) și durata impulsului (dreapta) pentru semnalul periodic generat. Canalul II (CH II) vizualizează semnalul de ieșire, iar canalul I (CH I) vizualizează operarea executivului HDIS al nucleului, cu cele două componente ale sale: prefix (HDIS_PRE) și sufix (HDIS_SUF)

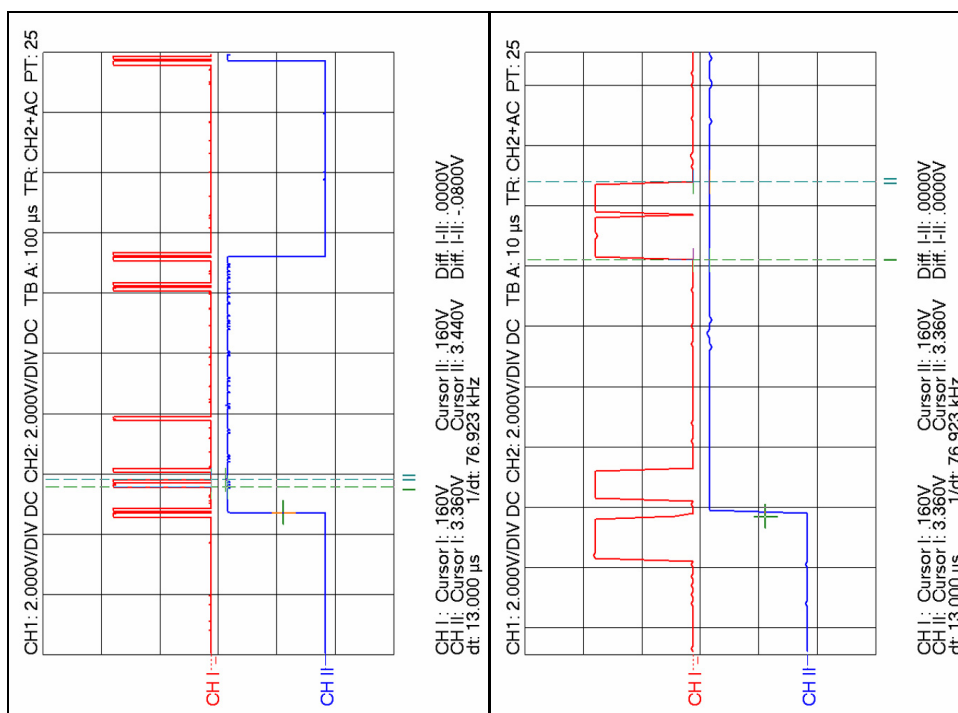


Figura 11-14. Operarea sistemului în cazul ModX-ului fantomă M_2 (stânga), cu vedere de detaliu (dreapta)

11.3 Concluzii

O variantă preliminară, dar complet funcțională a nucleului HARETICK a fost implementată și testată cu succes pe sistemul Motorola DSP56307 EVM. Evaluarea operării nucleului a fost realizată cu ajutorul unor aplicații special proiectate și dezvoltate. A fost vizată întâi operarea corectă a executivului TR al nucleului, HDIS, și apoi a întregului sistem.

Rezultatele măsurărilor realizate cu ajutorul osciloscopului digital și prin comparare cu semnale obținute de la generatoare digitale de funcții programabile, atestă atât corectitudinea comportării nucleului din punct de vedere computațional și temporal, cât și faptul că modelele de semnale și task-uri și algoritmi de planificare introduși și studiați teoretic în Secțiunea II, se verifică din punct de vedere practic.

Aplicațiile descrise în acest capitol demonstrează faptul că este posibilă obținerea de semnale de ieșire perfect periodice, atingându-se nivelul maxim de predictibilitate pentru întregul sistem – unul din obiectivele principale propuse în cadrul cercetării.

Întreg procesul de proiectare, dezvoltare, analiză, implementare și testare a aplicațiilor este relativ complicat și puțin flexibil. Astfel, de exemplu, modificarea perioadei (sau a duratei de impuls) pentru semnalele generate, mai ales în sensul micșorării valorilor, este foarte dificilă, în special pe parcursul operării aplicației. De asemenea, nu toate configurațiile posibile (dorite) pentru parametrii unui semnal se pot realiza pe platforma HARETICK. Motivul principal al acestui inconvenient

constă din faptul că planificatorul online HSCD este la rândul său un task executat în context non-preemptiv (ModX) și că durata sa de execuție are o valoare relativ mare în comparație cu cea a celorlaltor ModX-uri din sistem.

Rezultatele testelor efectuate relevă și faptul că duratele de execuție reale, măsurate cu ajutorul osciloscopului, pentru ModX-uri și chiar pentru executivul HDIS, diferă (sunt mai mici) comparativ cu valorile calculate în faza de analiză a sistemului. Acest lucru reprezintă o confirmare a faptului că în faza de analiză offline, duratele maxime de execuție (WCET) ale ModX-urilor se bazează pe evaluări pesimiste (calea cea mai lungă cale de execuție, numărul maxim de cicluri pentru bucle, etc.). Ca o consecință, deși sistemul HARETICK operează cu predictibilitate maximă, programatorul/inginerul de sistem nu se poate baza pe momentele de terminare a execuției ModX-urilor, ci doar pe cele de start. Astfel, de exemplu, pentru o aplicație de generare a unui semnal perfect periodic, sunt necesare două ModX-uri: unul pentru stabilirea frontului crescător al semnalului și celălalt pentru stabilirea frontului descrescător. Această abordare contrastează cu cea relativ clasică, ce utilizează un singur task care setează semnalul la începutul secvenței proprii de cod și apoi resetează semnalul înainte de terminarea task-ului.

SECȚIUNEA IV. CONCLUZII FINALE ȘI PERSPECTIVE

Secțiunea finală a tezei conține concluziile desprinse pe parcursul tratării subiectelor propuse în lucrare, un studiu comparativ cu alte sisteme timp-real din domeniu, rezumatul contribuțiilor și perspectivele de continuare a cercetării în această arie tematică.

12 Concluzii finale și perspective

12.1 Concluzii

În lucrarea de față se abordează *problematica sistemelor timp-real (TR) stricte destinate aplicațiilor critice de achiziție și prelucrare numerică a semnalelor (APNS) și de control digital încorporat*. Principalele idei pe care se bazează teza sunt:

- Utilizarea întreruperilor în sistemele TR stricte reprezintă o problemă din punctul de vedere al predictibilității, afectând capacitatea acestora de a garanta în orice condiții de operare respectarea termenelor impuse task-urilor TR stricte;
- Pentru a oferi predictibilitate maximă sistemelor TR stricte destinate aplicațiilor critice de achiziție și prelucrare numerică a semnalelor și de control digital încorporat, este necesară abordarea modelelor non-preemptive în ceea ce privește definirea semnalelor, a task-urilor și conceperea algoritmilor de planificare;
- Dezvoltarea viabilă a sistemelor și aplicațiilor TR stricte trebuie să aibă la bază, pentru toate fazele sale, metode și modele omogene, compatibile, în care timpul este o coordonată cheie;
- În procesul de dezvoltare a aplicațiilor TR stricte, faza de analiză offline, apriori execuției sale pe sistemul țintă, reprezintă o cerință obligatorie.

Obiectivele propuse în cadrul acestui program de cercetare sunt strâns legate de problemele curente ale domeniului STR pentru aplicații critice de APNS și de control digital încorporat, și se aliniază la ideile prezentate anterior. O discuție detaliată a modului în care au fost tratate și atinse aceste obiective este prezentată în continuare.

- (1) *Conceperea, dezvoltarea și demonstrarea unui set omogen de modele pentru semnale și task-uri TR, bazate pe un sistem corespunzător de reprezentare a timpului real.*

Un model al timpului real, cu rolul de element de bază pentru toate fazele dezvoltării STR stricte, a fost propus și analizat, plecând de la principiul de operare și sincronizare a sistemelor digitale: tactul procesor. Principalele caracteristici ale modelului temporal propus sunt:

- domeniul temporal are o structură liniară, discretă și limitată la stânga;
- domeniul temporal poate fi echivalat cu mulțimea numerelor naturale;
- unitatea de timp are o corespondență directă cu intervalul din domeniul temporal absolut stabilit de ceasul de timp-real (RTC) al sistemului;

- modelul temporal este prevăzut cu o metrică, rezultând posibilitatea exprimării de relații cantitative între instanțe sau intervale temporale;
- metrica pentru timp permite determinarea duratei intervalelor temporale;
- există o relație bine stabilită între domeniul temporal sistem și cel absolut.

Comportarea temporală a semnalelor are impact direct asupra comportării în timp a STR, și reciproc. Semnalele au fost studiate și clasificate, și, pe baza modelului temporal introdus, a fost definit și demonstrat un set minimal și complet de parametri care modelează comportarea temporală a semnalelor:

- perioada (sau durata minimă dintre aparițiile consecutive ale unui semnal, în cazul în care acesta nu e periodic);
- durata de activitate (intervalul de timp în care au loc aparițiile unui semnal);
- numărul de apariții;
- intervalul de răspuns al sistemului la apariția unui semnal.

Comportarea în timp a unui task TR generic a fost studiată și modelată cu ajutorul unui set de parametri și a relațiilor stabilite între aceștia. În continuare, au fost evidențiate avantajele și problemele ridicate de tratarea semnalelor de intrare cu ajutorul task-urilor periodice (utilizând tehnica de *polling*).

Sintetizând studiul semnalelor și al task-urilor pe baza modelului temporal considerat, a fost introdus și discutat modelul task-ului TR strict, ModX-ul:

- Task-urile periodice, executate în context non-preemptiv oferă predictibilitate maximă sistemelor TR.
- Abordarea modulară ușurează concepția, specificarea și proiectarea aplicațiilor.
- Sub aspect funcțional, este necesară impunerea de restricții la programarea ModX-urilor: evitarea recursivității, limitarea în timp și spațiu a buclilor de program, etc. Obiectivul major îl reprezintă facilitarea analizei temporale de program și determinarea timpului maxim de execuție a modulului (WCET).
- Comunicația între modulele aplicației se realizează prin intermediul parametrilor de intrare/ieșire și, într-o manieră limitată, prin variabilele globale. Datorită execuției ModX-urilor în context non-preemptiv, acestea implementează practic operații atomice, eliminându-se astfel necesitatea unor mecanisme suplimentare de sincronizare și control al acceselor concurente la resurse comune.
- Tratarea și generarea semnalelor este văzută ca un element important al operării sistemelor TR. Caracteristicile temporale ale ModX-urilor sunt în relații directe de legătură cu parametri similari ai semnalelor.
- Setul de parametri care definesc comportarea temporală a modelului de ModX propus, conține următoarele elemente: perioada, durata de execuție (considerată ca fiind timpul maxim de execuție a secvenței de cod a ModX-ului – WCET), intervalul limită (deadline), intervalul de întârziere (delay), contorul de execuție.

Două noțiuni relativ noi au fost introduse odată cu definirea modelului de ModX. "Contorul de execuție" al unui ModX este un parametru ce specifică și controlează numărul ciclurilor (perioadelor) sale de execuție. Valoarea contorului de execuție poate fi setată pe de o parte de către programatorul de sistem/aplicație (în

faza de specificare), iar pe de altă parte, de către alte ModX-uri în timpul operării aplicației. De asemenea, în cazul în care se specifică un număr finit de execuții pentru un anumit ModX, executivul sistemului TR propus va decrementa contorul ModX-ului de fiecare dată când îl lansează în execuție, până se ajunge la valoarea 0. Cea de-a doua noțiune introdusă este "ModX-ul fantomă", și reprezintă cazul ModX-urilor a căror contor de execuții ajunge la un moment dat la valoarea 0 în timpul operării. ModX-urile fantomă sunt planificate în continuare, dar nu vor fi lansate propriu-zis în execuție.

- (2) *Conceperea unei metodologii unitare de dezvoltare și implementare a sistemelor TR pentru aplicații critice de achiziție și prelucrare numerică de semnal. Elementul central al tuturor fazelor implicate (proiectare, specificare, verificare, analiză, implementare și testare) este sistemul de modele realizat în cadrul primului obiectiv.*

Sistemul OPEN-HARTS (Operating Environment for Hard Real-Time Systems) este conceput ca un mediu de dezvoltare și operare pentru sisteme și aplicații TR stricte, bazat pe setul de modele pentru timpul real, semnale și task-uri, dezvoltat în lucrare. OPEN-HARTS are două componente majore:

- Mediul integrat, vizual, INVERTA (Integrated Visual Environment for Real-Time Application Analysis and Development), destinat dezvoltării, specificării și verificării formale, programării, și analizei aplicațiilor TR stricte, în regim offline:
 - Exploatând principiul modularizării, aplicațiile vor fi proiectate într-o manieră vizuală, prin construirea grafului direcționat al aplicației, unde fiecare nod reprezintă un task TR lejer (fără restricții temporale) sau un task TR strict (cu specificații temporale ce trebuie respectate și garantate pe durata operării, în orice condiții) – ModX-ul;
 - Specificarea aplicației constă din stabilirea parametrilor fiecărui task: parametrii de intrare/ieșire și cei globali, setul semnalelor de intrare și de ieșire (împreună cu parametrii temporali corespunzători) și setul parametrilor temporali ai task-ului, în cazul ModX-urilor;
 - Programarea aplicației este realizată pentru fiecare task în parte (specificarea funcțională), utilizând un limbaj de programare (asamblare, C, etc.) modificat corespunzător (restricții de limitare a structurilor și secvențelor, în timp și spațiu, adnotări programator suplimentare, etc.);
 - Verificarea automată a specificațiilor aplicației, în special a specificațiilor temporale ale ModX-urilor;
 - Compilarea și analiza temporală a task-urilor, pentru stabilirea duratelor maxime de execuție (WCET);
 - Analiza planificabilității setului de ModX-uri ale aplicației, prin aplicarea condițiilor de planificabilitate și a algoritmilor de planificare non-preemptivă;
 - Pregătirea automată a aplicației pentru încărcarea în sistemul țintă (generarea structurilor de memorie, tabela cu identificatorii de proces, etc.), în cazul în care aplicația a fost analizată cu succes.

- HARETICK (Hard Real-Time Compact Kernel), un nucleu de operare timp-real, hibrid, multi-tasking și single-user, pentru sisteme de control încorporat. Nucleul HARETICK permite executarea aplicațiilor TR dezvoltate cu mediul INVERTA, oferind două contexte de execuție: contextul non-preemptiv este destinat execuției cu predictibilitate maximă a task-urilor TR stricte ale aplicației (ModX-urile), garantând respectarea termenelor specificate, iar contextul preemptiv este destinat execuției task-urilor TR lejere, fără a considera timpul ca o coordonată a operării acestora.

Încărcarea și executarea unei noi aplicații, dezvoltate în mediul INVERTA, pe o platformă țintă pe care rulează HARETICK se face prin înlocuirea celei care se execută în mod curent pe platforma țintă, existând un interval de timp (bine stabilit și controlat) în care are loc întreruperea efectivă a operării sistemului.

Conceptele introduse de către OPEN-HARTS reprezintă o soluție pentru problemele de predictibilitate a analizei planificabilității non-preemptive în cazul abordărilor online, și, prin definirea nucleului TR hibrid, se soluționează (cel puțin parțial) problema eficienței scăzute a execuției non-preemptive.

În prezent, mediul INVERTA se află încă în faza de dezvoltare. Un pas important îl reprezintă contribuția adusă de ing. Mircea Pățaș, prin realizarea utilitarului de analiză a codului pentru aplicații scrise în limbajul de asamblare al familiei Motorola DSP56300 [Motorola 96, Motorola 00]. Utilitarul este deja capabil să extragă structura generală a programului și să realizeze partajarea acestuia în blocuri de bază ("basic block partitioning") [Micea 04b], urmând să se finalizeze analiza temporală și calculul timpului maxim de execuție (WCET). Pe de altă parte, o versiune prototip, funcțională, a nucleului HARETICK a fost implementată și testată cu succes [Micea 04a, Micea 04c].

- (3) *Studiul detaliat al tehnicilor și metodelor ce permit operarea unui sistem TR strict cu garantarea respectării termenelor de timp impuse de aplicațiile critice și maximizarea predictibilității. Interesul a fost focalizat pe studiul algoritmilor de planificare non-preemptivă a seturilor de task-uri timp-real și a mecanismelor ce trebuie dezvoltate pentru rezolvarea elementelor asincrone din operarea sistemelor TR stricte.*

Componenta principală a dezvoltărilor teoretice din teza de față o constituie studiul detaliat al algoritmilor de planificare non-preemptivă a seturilor de ModX-uri (task-uri TR stricte, periodice), prezentat în Capitolul 5. Pentru început s-au discutat avantajele și dezavantajele utilizării algoritmilor de planificare non-preemptivă și s-a arătat faptul că introducerea etapei de analiză offline a planificabilității setului de task-uri, cu confirmarea (sau nu) a fezabilității acestuia, evită problemele de predictibilitate legate de timpii de factură non-polinomială necesari efectuării acestei analize în timpul operării efective a sistemului (analiza online).

Au fost considerate mai multe cazuri, pentru care s-au aplicat diverse restricții, plecând de la planificarea seturilor de ModX-uri simple, independente (modele apropiate de ideal) și avansându-se către situații mai complexe și mai realiste. S-a demonstrat faptul că pentru a verifica planificabilitatea unui set oarecare de ModX-uri, este suficientă analiza planificării pe un interval limitat de timp, egal cu cel mai mic multiplu comun al perioadelor ModX-urilor din set.

Pentru seturile de ModX-uri independente au fost introduși (adaptați) și studiați doi dintre cei mai eficienți algoritmi de planificare non-preemptivă: *MLFNP* (Minimum Laxity First Non-Preemptive) și *EDFNP* (Earliest Deadline First Non-Preemptive). În același context, a fost abordată problema evaluării fezabilității setului de task-uri utilizând condiții de necesitate și/sau suficiență. Am demonstrat că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniate la momentul inițial $t_0 = 0$), condiția a doua de necesitate enunțată în [Jeffay 91] nu este aplicabilă. O a treia condiție de necesitate a planificării, dictată de modelul de task TR utilizat (ModX-ul), a fost introdusă și demonstrată.

În continuare, au fost considerate cazurile STR stricte în care operează, pe lângă setul de ModX-uri ale aplicației, și un task special – planificatorul online, care utilizează o tabelă de dispatch de dimensiuni fixe, bine determinate. S-au analizat doi algoritmi de planificare online non-preemptivă: cu cicluri cu număr constant de planificări (*CEC-NPOS*) și cu cicluri periodice (*PC-NPOS*).

A fost abordată problema modelării și planificării seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora (pe baza noțiunii de FModX și a modelului matematic al acestuia, ce utilizează pe aritmetica modulară și pe funcția treaptă unitate). Planificarea non-preemptivă a seturilor compuse din două și mai mult de două FModX-uri independente, a fost studiată pe baza funcției de mapare și a proprietăților sale demonstrate.

În final s-au testat și evaluat performanțele algoritmilor de planificare non-preemptivă: evaluarea comparativă a algoritmilor *EDFNP* și *MLFNP*, evaluarea comparativă a algoritmilor de planificare online *CEC-NPOS* și *PC-NPOS*, și analiza performanțelor algoritmului *FENP*. Concluzia testelor este că *EDFNP* se comportă mult mai bine ca *MLFNP*, și *PC-NPOS* față de *CEC-NPOS*.

Față de obiectivele propuse și formulate în capitolul de introducere, pe parcursul cercetării au mai fost abordate și rezolvate următoarele aspecte, ce contribuie la demonstrarea practică a validității modelelor și tehnicilor propuse în lucrare:

- (4) *Implementarea la nivel de prototip funcțional a nucleului HARETICK pe o platformă Motorola DSP56307 EVM, ca un studiu de caz pentru modelele, metodele și algoritmi dezvoltati teoretic.*

Reprezentarea și gestionarea timpului și a task-urilor TR stricte în cadrul nucleului HARETICK a necesitat dezvoltarea unor structuri de date și mecanisme specifice:

- timpul absolut sistem, ceasul timp-real și timpul de planificare;
- tabela grafului direcționat, aciclic al aplicațiilor (PDAGT), tabela cu descriptorii de proces (PDT), tabela de dispatch (HDis_Tab), tabela de simboluri, zona de cod ($CODE_i$) și a parametrilor de ieșire ($DOut_i$) pentru fiecare ModX i , zona comună a ModX-urilor în execuție pentru variabilele interne (Heap), etc.

Pentru testarea și evaluarea nucleului HARETICK, au fost implementate versiunile de bază ale principalelor module: SYSINIT (inițializarea structurilor și componentelor nucleului), HDIS (executivul timp-real), LOADER (task-ul de încărcare a aplicațiilor și de gestionare a memoriei). De asemenea, au fost realizate

mai multe versiuni ale planificatorului contextului strict de execuție al nucleului, HSCD.

Executivul HDIS a fost conceput pentru a asigura execuția temporizată a ModX-urilor și rezolvarea comutării corecte a celor două contexte de execuție ale nucleului – contextul strict și cel lejer.

(5) *Testarea și demonstrarea operării corecte a HARETICK, și evaluarea performanțelor sale, cu ajutorul unor aplicații specifice.*

O primă aplicație a fost dezvoltată pentru testarea operării corecte a executivului HDIS. Algoritmii de planificare utilizat se bazează pe simpla programare de execuții consecutive a două ModX-uri în cadrul fiecărui ciclu de planificare.

Două aplicații pentru generarea de semnale de ieșire perfect periodice au fost apoi implementate și testate în conjuncție cu o versiune complet funcțională a planificatorului HSCD (bazată pe algoritmul *FENP*).

Rezultatele obținute cu ajutorul măsurătorilor practice cu echipamente specializate (osciloscop digital, generator de funcții programabile, etc.) ilustrează operarea corectă a nucleului HARETICK, în conformitate cu modelele și algoritmi teoretici dezvoltați pentru STR stricte.

Aplicațiile dezvoltate și testate pe platforma HARETICK demonstrează faptul că, prin posibilitatea obținerii de semnale de ieșire perfect periodice, întregul sistem poate atinge un nivel maxim de predictibilitate – unul din obiectivele principale propuse în cadrul cercetării.

12.2 Studiu comparativ cu alte sisteme timp-real

Deși este încă în stadiu de dezvoltare, asupra metodologiei OPEN-HARTS se pot efectua studii comparative la diferite niveluri cu alte sisteme timp-real existente. În cele ce urmează ne vom referi la avantajele și dezavantajele sistemului OPEN-HARTS în comparație cu metodologia "Giotto" [Henzinger 01, Henzinger 02, Kirsch 03, Henzinger 03a, Henzinger 03b, Henzinger 03c] și cu sistemul "Spring" [Stankovic 91a, Stankovic 91b, Stankovic 91c, Niehaus 93, Burleson 93, Stankovic 99], ambele descrise pe scurt în Capitolul 2.

Din punct de vedere al conceptelor implicate, fiecare din cele trei sisteme poate fi considerat la nivel de metodologie. Tabelul 12-1 sintetizează comparativ principalele concepte utilizate de Giotto, Spring și OPEN-HARTS. Se poate observa faptul că OPEN-HARTS, spre deosebire de celelalte, integrează modele bine definite pentru toate cele trei elemente pe care le considerăm esențiale în cadrul unui STR: timpul, semnalele și task-urile timp-real.

Modul de tratare a evenimentelor asincrone diferă între cele trei metodologii. Giotto utilizează întreruperile și mecanismul de drivere ce tratează la nivel primar evenimentele, în context non-preemptiv. Problemele de predictibilitate în situații de apariție concurentă a mai multor evenimente rămân, în opinia noastră, nerezolvate corespunzător. Spring implementează un sub-sistem separat, relativ autonom, de tratare a evenimentelor asincrone și a semnalelor de intrare-ieșire. Predictibilitatea sistemului este astfel asigurată, cu costul creșterii complexității (hardware și

software). OPEN-HARTS rezolvă problemele de predictibilitate în cazul evenimentelor asincrone prin metode de polling, ModX-urile implicate fiind specificate și analizate apriori. Dezavantajul constă din scăderea (uneori drastică) a eficienței sistemului.

Tabelul 12-1. Comparație la nivel de concepte între cele trei sisteme

Concepte implementate	Giotto (U. of California, Berkeley)	Spring (U. of Massachusetts)	OPEN-HARTS (DSPLabs, U. Politehnica, Timisoara)
Platforma țintă	embedded, mono- și multiprocesor	sisteme distribuite	sisteme APNS și embedded, mono-procesor
Aplicațiile țintă	capabilități timp-real stricte	TR distribuite, complexe	embedded, APNS, critice
Modelul temporal	implicit, ?	?	bine definit, integrat în toată metodologia
Modelul semnalelor	porturi (senzor, actuator, task) și drivere	semnale cu tratare lentă și rapidă, ?	bine definit, prin parametri temporali și relațiile între ei
Modelul task-urilor timp-real	modulare, periodice	periodice și aperiodice	modulare, periodice, cu set complet de parametri temporali, execuție NP
Tratarea evenimentelor asincrone	întreruperi, drivere NP	întreruperi, sistem I/O separat	polling, Teorema tratării semnalelor de intrare cu ModX-uri
Specificarea aplicațiilor	limbajul Giotto, ?, utilizarea "modurilor de execuție"	SDL (?)	mediul vizual INVERTA, parametri temporali și specificare funcțională
Programarea aplicațiilor	limbajul Giotto, C	Spring-C	ASM, C cu restricții și adnotări
Analiza și planificarea	offline, la compilare	online, "reflectivitate"	offline în INVERTA, compilare, extracție WCET, analiză fezabilitate
Tehnicile de planificare	selecție programator	algoritm combinat, "SSCoP"	non-preemptive pentru HRT și clasice pentru SRT

Planificarea task-urilor în metodologia Giotto este rezolvată de către compilator, ajutat, dacă e cazul, de programator prin adnotări corespunzătoare. Giotto oferă o flexibilitate mare în acest sens, prin faptul că nu se restricționează utilizarea unui anumit algoritm de planificare, atâta timp cât la compilare se poate genera o planificare fezabilă a aplicației. Task-urile în sistemul Spring sunt planificate în funcție de o serie de informații suplimentare, furnizate de programatorul aplicației sau extrase la compilare: diferiți parametri temporali (incluzând obligatoriu "termenul limită" al fiecărui task), importanța (prioritatea), necesar de resurse, cerințe de fiabilitate, etc. Algoritmii de planificare Spring rezolvă și garantează

execuția task-urilor TR critice în mod static, apriori operării în sistem, iar celelalte task-uri vor fi planificate dinamic. Metodologia OPEN-HARTS obligă programatorul să împartă task-urile aplicației în task-uri stricte (ModX-uri) și task-uri lejere încă de la faza de specificare și programare a aplicației. ModX-urile vor fi analizate din punct de vedere al planificării non-preemptive înaintea încărcării aplicației compilate în sistemul țintă. Task-urile lejere vor fi planificate în manieră clasică, dinamică, (planificare preemptivă, bazată pe priorități, în "time-sharing") în timpul operării aplicației.

O serie de concepte utilizate în sistemele studiate ne par extrem de interesante, urmând ca pe viitor să evaluăm în ce măsură pot fi adaptate și aplicate în OPEN-HARTS:

- "Modurile de execuție" din Giotto: descriu câte o anumită configurație bine definită de execuție a unui set de task-uri. Cu acest concept, planificarea întregului set de task-uri poate fi ușurată prin luarea în considerare doar a task-urilor corespunzătoare modului curent la un moment dat.
- "Reflectivitatea sistemului" Spring: reprezintă caracteristica sistemului (în principal a planificatorului Spring) de a lua în considerare cât mai multă informație utilă despre fiecare task pentru obținerea unei planificări eficiente și flexibile.
- "SSCoP" – Spring Scheduling Co-Processor: introducerea unui co-procesor dedicat planificării (și eventual, celorlalte rutine și primitive ale nucleului de operare), eliberează procesorul de aceste sarcini și task-uri complet transparente aplicației, crescând astfel viteza și performanțele întregului sistem.

Din punct de vedere al performanțelor, o comparație utilă și corectă între cele trei sisteme este dificil de realizat, din considerente ca:

- disponibilitatea tuturor sistemelor, în versiune complet operațională și cu toate uneltele hardware și software aferente;
- dezvoltarea, implementarea și evaluarea unei aceleași aplicații pe fiecare din sistemele testate, utilizând metrici identice.

Pe de altă parte, se poate considera că, în măsura în care fiecare sistem este capabil a executa, la parametrii specificați, setul de aplicații pentru care a fost conceput, scopurile sale sunt îndeplinite. Tabelul 12-2 prezintă câteva elemente comparative desprinse din studierea performanțelor sistemelor OPEN-HARTS și Giotto, rezultate în urma studierii documentației disponibile.

Tabelul 12-2. Elemente comparative legate de performanța sistemelor Giotto și OPEN-HARTS

Sistemul Giotto: E- și S- Machines	Sistemul OPEN-HARTS: Nucleul HARETICK
– Dimensiunea și durata de execuție a E-codului generat se modifică în funcție de aplicație (nr. de task-uri, nr. de semnale tratate, etc.).	+ Dimensiunea și durata de execuție a executivului HDIS au valori fixe, cunoscute apriori.

<ul style="list-style-type: none"> + Se pot specifica/genera diferite tehnici de planificare prin intermediul S-codului. – Existența driverelor, ce se execută non-preemptiv într-un context general preemptiv. Rezultă potențiale probleme de predictibilitate. 	<ul style="list-style-type: none"> – Tehnicile non-preemptive pe care se bazează întreaga concepție HARETICK au o eficiență și flexibilitate relativ scăzute – prețul plătit în schimbul asigurării unei predictibilități maxime. + Creșterea eficienței sistemului este soluționată prin introducerea contextului lejer de execuție. – Planificatorul online HSCD este văzut de sistem ca un ModX, utilizarea procesor implicată adăugându-se la cea a aplicației. + Sistemul este capabil a genera semnale de ieșire perfect periodice.
--	---

12.3 Rezumat al contribuțiilor

Lucrarea aduce o serie de contribuții în domeniul sistemelor TR stricte pentru aplicații critice. Un scurt rezumat al acestora este prezentat în continuare.

Capitolul 4:

- definirea unui model temporal ce permite specificarea comportamentului în timp al semnalelor și task-urilor unui STR strict (Capitolul 4);
- studiul semnalelor și modelarea comportamentului lor temporal prin introducerea unui set minimal și complet de parametri;
- studiul general al task-urilor TR și a interacțiunii acestora cu semnalele;
- abordarea comportamentului în timp al task-urilor TR periodice ce tratează semnale de intrare și demonstrarea relațiilor temporale ce derivă din această interacțiune (timpul de răspuns, etc.);
- evidențierea și exemplificarea faptului că metodele de tip *polling* de tratare a semnalelor de intrare garantează respectarea termenelor stricte, însă în detrimentul eficienței;
- definirea modelului de task TR strict – ModX-ul, și discutarea proprietăților și parametrilor acestuia relativ la aspectele considerate de importanță majoră în dezvoltarea și analiza sistemelor și aplicațiilor TR stricte;
- introducerea a două noțiuni noi, relativ la modelul ModX-urilor: contorul de execuție și ModX-urile fantomă.

Capitolul 5:

- evidențierea și exemplificarea avantajelor și dezavantajelor planificării și execuției non-preemptive a task-urilor TR;
- evidențierea necesității introducerii etapei de analiză offline a planificabilității setului de task-uri, cu confirmarea (sau nu) a fezabilității acestuia, pentru evitarea problemelor de predictibilitate legate de timpii de factură nepolinomială necesari efectuării acestei analize în timpul operării efective a sistemului (analiza online);
- studiul și exemplificarea algoritmilor de planificare non-preemptivă *MLFNP* și *EDFNP* pentru seturi de ModX-uri simple, independente, cu demonstrarea optimalității lui *EDFNP*;
- abordarea problemelor de evaluare a fezabilității setului de task-uri utilizând condiții de necesitate și/sau suficiență;
- demonstrarea faptului că pentru modelul de task-uri considerat (ModX-uri independente, cu perioadele aliniată la momentul inițial $t_0 = 0$), condiția a doua de necesitate enunțată în [Jeffay 91] nu este aplicabilă;
- introducerea și demonstrarea unei condiții suplimentare de necesitate a planificării non-preemptive, impusă de modelul setului de task-uri bazat pe ModX-uri (condiția a treia de necesitate);
- discutarea cazurilor, mai apropiate de realitate, ale sistemelor TR stricte ce execută, pe lângă setul de ModX-uri ale aplicației, și un task special – planificatorul online, ce utilizează o tabelă de dispatch de dimensiuni fixe, bine determinate, pentru planificarea ModX-urilor;
- ținând cont și de caracteristicile de comportare temporală a planificatorului online, au fost concepute și demonstrate condițiile de necesitate ale planificabilității sistemului, pentru cei doi algoritmi propuși: planificarea online în cicluri cu număr constant de execuții (*CEC-NPOS*) și planificarea online în cicluri periodice (*PC-NPOS*);
- prezentarea și verificarea prin simulări și exemple a algoritmilor *CEC-NPOS* și *PC-NPOS*;
- abordarea problemei modelării și planificării seturilor de task-uri ce necesită o execuție fixă în cadrul fiecărei perioade a acestora;
- introducerea noțiunii de ModX cu execuție fixă (FModX) și demonstrarea unui model matematic al acestuia, bazat pe aritmetica modulară și pe funcția treaptă unitate;
- studierea planificării non-preemptive a seturilor compuse din două și mai mult de două FModX-uri independente, pe baza funcției de mapare și a proprietăților demonstrate ale acesteia;
- introducerea și demonstrarea condiției de necesitate a planificabilității non-preemptive a seturilor de FModX-uri, ajungându-se la o formulă simplă, ce specifică faptul că dacă setul este fezabil, atunci suma duratelor de execuție a oricăror două FModX-uri nu depășește valoarea celui mai mare divizor comun al perioadelor lor;
- algoritmul de planificare a FModX-urilor, *FENP*, ce utilizează funcția de mapare a fost conceput și exemplificat.

Capitolul 6:

- studierea și evaluarea performanțelor algoritmilor de planificare non-preemptivă introduși și discutați în lucrare: evaluarea comparativă a algoritmilor EDFNP și MLFNP, evaluarea comparativă a algoritmilor de planificare online CEC-NPOS și PC-NPOS, și analiza performanțelor algoritmului FENP. Concluzia testelor este că *EDFNP* se comportă mult mai bine ca *MLFNP*, și *PC-NPOS* față de *CEC-NPOS*.

Capitolul 7:

- conceperea și prezentarea modelului unui sistem complet care unifică toate conceptele introduse și studiate în lucrare – OPEN-HARTS, cu două componente: INVERTA (un mediu integrat, vizual, destinat dezvoltării, specificării și verificării formale, programării și analizei aplicațiilor TR stricte) și HARETICK (un nucleu de operare TR hibrid, multi-tasking și single-user, pentru sisteme de control încorporat, conceput pentru a oferi predictibilitate maximă de execuție aplicațiilor TR critice);
- evidențierea faptului că prin conceptele introduse de către OPEN-HARTS se oferă o soluție pentru problemele de predictibilitate a analizei planificabilității non-preemptive în cazul abordărilor online, și, prin definirea nucleului TR hibrid, se soluționează (cel puțin parțial) problema eficienței scăzute a execuției non-preemptive.

Capitolele 8, 9, 10 și 11:

- dezvoltarea și implementarea unui prototip de nucleu de operare TR ce poate servi ca platformă pentru execuția aplicațiilor critice cu garantarea respectării termenelor impuse la specificare și cu maximizarea predictibilității de operare: nucleul HARETICK;
- implementarea și testarea executivului TR al HARETICK: HDIS, cu rol esențial în operarea task-urilor TR stricte ale sistemului și aplicațiilor;
- demonstrarea, cu ajutorul măsurătorilor practice, experimentale, a faptului că operarea executivului HDIS al nucleului corespunde specificațiilor de proiectare ale sistemului;
- dezvoltarea, implementarea și analiza unei versiuni funcționale de planificator online pentru contextul strict de execuție al HARETICK: HSCD;
- testarea practică și demonstrarea, cu ajutorul măsurătorilor experimentale, a faptului că operarea planificatorului HSCD, și, în conjuncție cu executivul HDIS, operarea întregului sistem HARETICK, corespunde întocmai specificațiilor de proiectare, și de asemenea, este în concordanță cu modelele teoretice propuse;
- implementarea și testarea cu succes a unor aplicații practice pentru nucleul HARETICK, capabile a genera semnale de ieșire perfect periodice.

12.4 Concluzii finale

Atât în natură, cât și în practica aplicațiilor de control digital, există fenomene și secvențe de acțiuni care se desfășoară fără a admite întreruperi. Pe de altă parte, operațiile care permit apariția întreruperilor în timpul execuției, pot fi descompuse la rândul lor, în sub-secvențe continue, neîntreruptibile.

Teza de față demonstrează faptul că utilizarea tehnicilor non-preemptive, în conjuncție cu un set omogen de modele pentru timp, evenimente și acțiuni, este perfect posibilă din punct de vedere teoretic și practic. Rezultatele obținute cu sisteme ce utilizează principii non-preemptive sunt similare din punct de vedere al corectitudinii operațiilor, cu cele ale sistemelor clasice. În schimb, atunci când e vorba de garantarea respectării termenelor de execuție și de predictibilitatea de operare, sistemele non-preemptive sunt singurele care se comportă corect în orice condiții.

Prin urmare, abordarea non-preemptivă permite realizarea aplicațiilor TR cu impact critic, astfel încât să fie respectate atât specificațiile funcționale, cât și cele temporale ale acestora. Acest lucru se obține însă cu prețul scăderii drastice a flexibilității și eficienței de operare a STR. Cele două probleme lasă deschise o multitudine de posibilități și perspective de cercetare-dezvoltare în domeniu.

Problema flexibilității a fost atinsă în materialul de față prin conceperea modelului OPEN-HARTS, care permite în același timp dezvoltarea/modificarea aplicațiilor TR și operarea (execuția) unei versiuni validate și fezabile a acestora.

Din punctul de vedere al eficienței STR, o soluție completă a fost prezentată în teză, prin descrierea implementării și testării unui nucleu de operare timp-real hibrid, HARETICK, ce conține două contexte complementare de execuție: contextul strict (HRT) și cel lejer (SRT). Contextul strict de execuție poate fi privit ca o "întrerupere prelungită" a celui lejer, conform unei planificări riguroase, analizată apriori din punctul de vedere al fezabilității (analiza "offline").

Un alt mod de a aborda problema este acela că, deasupra contextului lejer (reprezentând modul de operare tradițional, preemptiv, al sistemelor de calcul de tip "time-sharing") se poate implementa, ca o facilitate de execuție cu garantarea respectării termenelor și cu predictibilitate maximă, contextul strict, a cărui întrerupere (RTC) este cea mai prioritară în sistem. În plus, pe durata execuției contextului strict, celelalte întreruperi sunt suspendate.

12.5 Perspective de cercetare și dezvoltare

Problemele abordate în acest material, modelele, tehnicile și algoritmi discutați (în majoritate noi sau cu adaptări corespunzătoare) permit continuarea preocupărilor și deschid noi perspective în domeniul de mare interes al sistemelor și aplicațiilor TR stricte de achiziție și prelucrare numerică de semnal și de control digital încorporat.

La rândul nostru, dorim continuarea cercetărilor întreprinse în cadrul acestui program de doctorat, urmând două direcții distincte: (a) tehnici și unelte pentru dezvoltarea și analiza aplicațiilor TR, și (b) continuarea implementării nucleului HARETICK în vederea obținerii unei variante complet funcționale, care să poată

servi ca platformă de operare pentru cât mai multe aplicații practice. Cele două direcții de perspectivă avute în vedere pentru cercetarea în domeniu sunt detaliate în continuare.

(a) Tehnici și unelte pentru dezvoltarea și analiza aplicațiilor TR:

- Verificarea practică a seturilor de formule stabilite între parametrii temporali ai semnalelor și task-urilor aplicațiilor TR, pe baza modelului temporal propus. Aceste formule constituie fundamentul fazei de verificare și validare formală a aplicațiilor TR. Evaluarea necesității de formalizare a întregului sistem de modele, utilizând algebre și logici temporale.
- Planificarea seturilor de ModX-uri cu dependențe de program (cu alte cuvinte, planificarea unei aplicații TR complete).
- Integrarea tuturor cazurilor de planificare non-preemptivă și conceperea unui model unificat, capabil a fi utilizat în cât mai multe aplicații reale.
- Rafinarea specificațiilor, implementarea și testarea mediului integrat de dezvoltare a aplicațiilor TR (INVERTA): definirea unui limbaj adecvat de programare destinat descrierii funcționale a task-urilor aplicației (este vizat un subset al limbajului C, care să permită însă doar construcții limitate în ceea ce privește durata de execuție și dimensiunile structurilor de date), implementarea unui compilator dedicat, care să permită generarea de cod mașină sub formă de bibliotecă, separat, pentru fiecare task al aplicației.
- Continuarea dezvoltării utilitarului de analiză temporală a programelor și aplicațiilor TR, astfel încât să se poată extrage informații cu privire la durata maximă de execuție (WCET) a task-urilor. Integrarea acestuia în mediul INVERTA.
- Integrarea algoritmului unificat de planificare non-preemptivă a task-urilor TR stricte în mediul de dezvoltare a aplicațiilor TR, pentru a se implementa faza de analiză (offline) a planificabilității aplicației.

(b) Continuarea implementării nucleului HARETICK:

- Conceperea și implementarea unei structuri statice, convenabile, destinată reprezentării eficiente a grafului aplicațiilor ce vor fi încărcate în sistem (tabela PDAGT – Program Directed Acyclic Graph Table);
- Finalizarea implementării tuturor componentelor nucleului HARETICK;
- Înscrierea codului sistemului în memoria EPROM a platformei țintă și configurarea sistemului pentru a încărca secvența din EPROM la pornire sau după o operație de RESET;
- Finalizarea implementării protocolului și interfețelor de comunicație cu calculatorul gazdă (DATALINK), pentru a se putea citi din exterior rapoarte periodice ale execuției sistemului HARETICK;
- În momentul rezolvării cu succes a problemelor majore legate de implementarea curentă a HARETICK, în așa fel încât sistemul să fie funcțional pe platforma Motorola DSP56307, intenționăm portarea nucleului pe alte tipuri de platforme (de exemplu, sisteme cu microcontroller, etc.);

- Introducerea și implementarea conceptelor celor două contexte complementare de execuție din HARETICK în cadrul unor sisteme de operare TR existente.

Creșterea eficienței și flexibilității sistemelor TR prin utilizarea unui procesor dedicat planificărilor online ("scheduling coprocessor") este o altă abordare ce ni se pare valoroasă și interesantă pentru cercetările viitoare în domeniu.

Bibliografie

- [Acho 86] A. A. V. Acho, R. Sethi, J. D. Ullman, "Compilers: Principles, Techniques, Tools", Addison Wesley, Reading, Massachusetts, 1986.
- [Albanus 91] J. Albanus, "Coding Schemes Used with Data Converters", Application Note AN-175, Burr-Brown, Inc., 1991.
- [Allen 83] J. F. Allen, "Maintaining Knowledge about Time Intervals", Communications of the ACM, Vol. 26, No. 11, November 1983, pp. (832-843).
- [Allen 94] J. F. Allen, G. Ferguson, "Actions and Events in Interval Temporal Logic", Technical Report, TR-URCSD 521, University of Rochester, New York, July 1994.
- [Allombert 03] B. Allombert, A. Pacetti, "An Introduction to gp2c", 2003 [Online]. Available: <http://pari.math.u-bordeaux.fr/pub/pari/>.
- [Audsley 91] N. C. Audsley, A. Burns, M. F. Richardson, A. J. Wellings, "Hard Real-Time Scheduling: The Deadline-Monotonic Approach", in Proceedings of the 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, USA, May 1991.
- [Barbacci 86] M. R. Barbacci, J. M. Wing, "Specifying Functional and Timing Behavior for Real-time Applications", Technical Report ESD-TR-86-208, CMU/SEI-86-TR-4, Software Engineering Institute, Carnegie Mellon University, 1986.
- [Baya 99] A. Baya, L. E. Anton, V. Ancusa, M. V. Micea, "Real Time Data Acquisition System for a Compound Straight Pipe", in Scientific and Technical Bulletin of the "Politehnica" University of Timisoara, Mechanics, Tom (44) 58, 1999, pp. (41-50).
- [Bellini 00] P. Bellini, R. Mattolini, P. Nesi, "Temporal Logics for Real-time System Specification", ACM Computing Surveys, Vol 32, No. 1, March 2000.
- [Berry 00] G. Berry, "The Esterel v5 Language Primer: Version v5_91", Centre de Mathematiques Appliquees, Ecole des Mines and INRIA, July 2000.
- [Bucci 95] G. Bucci, M. Campanai, P. Nesi, "Tools for Specifying Real-time Systems", Real-time Systems, 8 (2/3), March/May 1995, pp. (117-172).

- [Burleson 93] W. Burleson, J. Ko, D. Niehaus, K. Ramamritham, J. A. Stankovic, G. Wallace, C. Weems, "The Spring Scheduling Co-Processor: A Scheduling Accelerator", in Proceedings of the 1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors, ICCD'93, October 1993, pp. (140-144).
- [Burns 90] A. Burns, A. Wellings, "Real-Time Systems and Their Programming Languages", Addison-Wesley, 1990.
- [Burns 91] A. Burns, M. Nicholson, N. Zhang, "Program Execution Time Analysis (SPIRITS Project Deliverable)", Department of Computer Science, University of York, July 1991.
- [Burr-Brown 95a] Burr-Brown, Inc., "Data Conversion Products", Burr-Brown IC Databook, 1995.
- [Burr-Brown 95b] Burr-Brown, Inc., "ADS774: Microprocessor-Compatible Sampling CMOS Analog-to-Digital Converter", Product Data Sheet PDS-1109F, 1995.
- [Carter 04] E. F. Carter Jr., "Generating Gaussian Random Numbers", Taygeta Scientific, Incorporated, June 2004 [Online]. Available: <http://www.taygeta.com/random/gaussian.html>.
- [Chapman 94] R. Chapman, "Program Timing Analysis", Technical Report, Dependable Computing Systems Centre, University of York, 1994.
- [Chen 98] D. Chen, A. Mok, S. Baruah, "On Modeling Real-time Task Systems", Lecture Notes in Computer Science, No. 1494, Springer-Verlag, October 1998, pp. (153-169).
- [Cheng 88] S. Cheng, J. A. Stankovic, "Scheduling Algorithms for Hard Real-Time Systems: A Brief Survey", in Hard Real Time Systems, J. A. Stankovic and K. Ramamritham (Eds.), IEEE Computer Society Press, 1988, pp. (150-173).
- [Chouinard 02] J.-Y. Chouinard, "Notes on the Modular Arithmetics and Galois Fields", Course Notes, Design of Secure Computer Systems CSI4138/CEG4394, School of Information Technology and Engineering, University of Ottawa, Canada, September 2002.
- [Chung 95] T. M. Chung, H. G. Dietz, "Language Constructs and Transformation for Hard Real-Time Systems", in ACM SIGPLAN Notices, Vol. 30, No. 11, November 1995, pp. (41-49).
- [Colnarić 93] M. Colnarić, W. A. Halang, "Architectural Support For Predictability in Hard Real Time Systems", in Control Engineering Practice, No. 1, (1), February 1993, pp. (51-57).
- [Cretu 99] V. Cretu, M. V. Micea, I. Guzun, V. Guzun, "Aquarius-DSP, from its Design to Applications Integration", in Transactions on

- Automatic Control and Computer Science, Vol. 44 (58), No. 1, 2, Periodica Politehnica, Timisoara, Romania, 1999, pp. (5-16).
- [Cretu 02] V. Cretu, L. E. Anton, M. V. Micea, A. Baya, R. Resiga, D. Chiciudean, "Applications for Experimental Study of Hydraulic Phenomena in Hydrodynamic Circuits", in Transactions on Automatic Control and Computer Science, Vol. 47 (61), Periodica Politehnica, Timisoara, Romania, 2002, pp. (65-70).
- [Cretu 03] V. Cretu, T. Jurca, M. V. Micea, I. Sora, "Instrumentation and Measurement in Romania: Technical Developments at 'Politehnica' University of Timisoara", in IEEE Instrumentation & Measurement Magazine, Vol. 6, No. 3, September 2003, pp. (41-47).
- [Ferrante 87] J. Ferrante, K. J. Ottenstein, J. D. Warren, "The Program Dependence Graph and Its Use in Optimisation", ACM Transactions on Programming Languages and Systems, No. 9 (3), July 1987, pp. (319-349).
- [Fohler 01] G. Fohler, T. Lennvall, G. Buttazzo, "Improved Handling of Soft Aperiodic Tasks in Offline Scheduled Real-Time Systems Using Total Bandwidth Server", in Proceedings of the 8th IEEE International Conference on Emerging Technologies & Factory Automation, Nice, France, October 2001.
- [Gai 02] P. Gai, L. Abeni, G. Buttazzo, "Multiprocessor DSP Scheduling in System-on-a-chip Architectures", in Proceedings of the 14th Euromicro Conference on Real-Time Systems (ECRTS'02), Vienna, Austria, June 2002, pp. (231-240).
- [George 96] L. George, N. Rivierre, M. Spuri, "Preemptive and Non-Preemptive Real-Time Uni-Processor Scheduling", Rapport de recherche, Nr. 2966, Institut National de Recherche en Informatique et en Automatique, INRIA, Rocquencourt, France, September 1996.
- [Gerber 97] R. Gerber, S. Hong, "Slicing Real-time Programs for Enhanced Schedulability", ACM Transactions on Programming Languages and Systems, Vol. 19, No. 3, May 1997, pp. (525-555).
- [Ghosh 94] K. Ghosh, B. Mukherjee, K. Schwan, "A Survey of Real-Time Operating Systems", Technical Report GIT-CC-93/18, College of Computing, Georgia Institute of Technology, Atlanta, Georgia, February 1994.
- [Coffman 73] E. G. Coffman Jr., P. J. Denning, "Operating Systems Theory", Prentice-Hall Series in Automatic Computation, Prentice-Hall, New Jersey, USA, 1973.
- [Gupta 96] R. Gupta, M. Spezialetti, "A Compact Task Graph Representation for Real-time Scheduling", Real Time Systems, 10, 1-32 (1996), Kluwer Academic Publishers, 1996.

- [Halpern 91] J. Y. Halpern, Y. Soham, "A Propositional Modal Logic of Time Intervals", *Journal of the ACM*, Vol. 38, No. 4, October 1991, pp. (935-962).
- [Henzinger 01] T. A. Henzinger, B. Horowitz, C. M. Kirsch, "Embedded Control Systems Development with Giotto", in *Proceedings of the International Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES)*, ACM Press, 2001, pp. (64-72).
- [Henzinger 02] T. A. Henzinger, C. M. Kirsch, "The Embedded Machine: Predictable, Portable Real-Time Code", in *Proceedings of the International Conference on Programming Language Design and Implementation (PLDI)*, ACM Press, 2002, pp. (315-326).
- [Henzinger 03a] T. A. Henzinger, B. Horowitz, C. M. Kirsch, "Giotto: A Time-triggered Language for Embedded Programming", in *Proceedings of the IEEE*, No. 91, 2003, pp. (84-99).
- [Henzinger 03b] T. A. Henzinger, C. M. Kirsch, M. A.A. Sanvido, W. Pree, "From Control Models to Real-Time Code Using Giotto", in *IEEE Control Systems Magazine*, No. 23 (1), 2003, pp. (50-64).
- [Henzinger 03c] T. A. Henzinger, C. M. Kirsch, S. Matic, "Schedule Carrying Code", in *Proceedings of the Third International Conference on Embedded Software (EMSOFT)*, *Lecture Notes in Computer Science 2855*, Springer-Verlag, 2003, pp. (241-256).
- [Howell 95] R. R. Howell, M. K. Venkatrao, "On Non-Preemptive Scheduling of Recurring Tasks Using Inserted Idle Times", in *Information and Computation Journal*, Vol. 117, No. 1, February, 1995.
- [Ifeachor 93] E. C. Ifeachor, B. W. Jervis, "Digital Signal Processing: A Practical Approach", Addison-Wesley, 1993.
- [Jahanian 88] F. Jahanian, A. K. Mok, D. A. Stuart, "Formal Specification of Real-time Systems", UTCS Technical Report, UTCS-TR-88-25, Department of Computer Sciences, University of Texas at Austin, 1988.
- [Jeffay 91] K. Jeffay, D. Stanat, C. Martel, "On Non-Preemptive Scheduling of Periodic and Sporadic Tasks", in *Proceedings of the Twelfth IEEE Real-Time Systems Symposium*, San Antonio, Texas, December 1991, IEEE Computer Society Press, pp. (129-139).
- [Kang 98] S.-I. Kang, H.-K. Lee, "Analysis and Solution of Non-preemptive Policies for Scheduling Readers and Writers", in *ACM Operating Systems Review*, No. 32 (3), July, 1998, pp. (30-50).

- [Kim 80] K. H. Kim, M. Naghibzadeh, "Prevention of Task Overruns in Real-Time Non-Preemptive Multiprogramming Systems", ACM, 1980, pp. (267-276).
- [Kirsch 03] C. M. Kirsch, T. A. Henzinger, M. A. A. Sanvido, "A Programmable Microkernel for Real-Time Systems", Technical Report UCB/CSD-3-1250, University of California, Berkeley, USA, June 2003.
- [Klingerman 86] E. Klingerman, A. D. Stoyenko, "Real-time Euclid: A Language for Reliable Real-time Systems", IEEE Transactions on Software Engineering, SE-12 (9), September 1986.
- [Knuth 98] D. E. Knuth, "The Art of Computer Programming", Vol. 2, "Seminumerical Algorithms", Addison-Wesley, Reading Massachusetts, 1998.
- [Lehoczky 87] J.P. Lehoczky, L. Sha, J.K. Strosnider, "Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", in Proceedings of the 8th Real-Time Systems Symposium, San Jose, California, 1987, pp. (261-270).
- [Lehoczky 89] J. P. Lehoczky, L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior", in Proceedings of the 10th Real-Time Systems Symposium, IEEE Computer Society Press, December 1989, pp. (166-171).
- [Lerma 03] M. A. Lerma, "Modular Arithmetic", Department of Mathematics, Northwestern University, USA, March 2003 [Online]. Available: http://www.math.northwestern.edu/~mlerma/problem_solving/results/modular_arith.pdf.
- [Letia 00] T. S. Letia, "Sisteme de timp-real" ("Real-Time Systems"), "Editura Albastra" Publishing House, Cluj-Napoca, Romania, 2000.
- [Liu 73] C. L. Liu, J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", in Journal of the ACM, Vol. 20, No 1., January 1973, pp. (46-61).
- [Lynn 92] P. A. Lynn, W. Fuerst, "Introductory Digital Signal Processing with Computer Applications", John Wiley & Sons, 1992.
- [Marlowe 90] T. J. Marlowe, B. G. Ryder, "Properties of Data Flow Frameworks", Acta Informatica, 28, pp. (121-163), 1990.
- [Mattolini 01] R. Mattolini, P. Nesi, "An Interval Logic for Real-Time System Specification", in IEEE Transactions on Software Engineering, Vol. 27, No. 3, March 2001, pp. (208-227).
- [Mattolini 96] R. Mattolini, P. Nesi, "Using TILCO for Specifying Real-Time Systems", in Proceedings of the Second IEEE International Conference on Engineering of Complex Computer Systems

- (Montreal, Canada), IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. (18-25).
- [Melliari-Smith 87] P. M. Melliari-Smith, "Extending Interval Logic to Real Time Systems", in Proceedings of the Conference on Temporal Logic Specification (UK), Ed. B. Banieqbal, H. Barringer, A. Pnuelli, Springer-Verlag, New York, 1987, pp. (224-242).
- [Mercer 92] C. W. Mercer, "An Introduction to Real-Time Operating Systems: Scheduling Theory", Draft, School of Computer Science, Carnegie Mellon University, Pittsburg, Pennsylvania, November 1992.
- [Micea 00a] M. V. Micea, "Interfacing DAQ Systems to Digital Signal Processors", Transactions on Automatic Control and Computer Science, Special Issue Dedicated to the Fourth International Conference on Technical Informatics, CONTI'2000, October 12-13, Vol. 45 (59), No. 4, "Politehnica" University of Timisoara, 2000, pp. (23-28).
- [Micea 00b] M. V. Micea, V. Cretu, D. Chiciudean, "Interfacing a Data Acquisition System to the DSP56303", Application Note AN2087/D Rev. 0, 12/2000, Motorola, Inc., USA, 2000.
- [Micea 00c] M. V. Micea, "Sisteme de achizitie numerica a datelor: Indrumator de laborator", ("Signal Acquisition and Conditioning Systems: Laboratory Workshops"), Comanda 270/2000, Centrul de multiplicare al Universitatii POLITEHNICA Timisoara, 2000.
- [Micea 00d] M. V. Micea, V. Cretu, D. Chiciudean, "Communication Protocols Implementation on DSP-based Systems", in Proceedings of the International Symposium on Systems Theory, SINTES 10, 10-th Edition, Automation, Computers, Electronics, May 25-26, University of Craiova, 2000, pp. (C 205-208).
- [Micea 00e] M. V. Micea, D. Chiciudean, L. Muntean, "ECP Standard Parallel Interface for DSP56300 Devices", Application Note AN2085/D Rev. 0, 11/2000, Motorola, Inc., USA, 2000.
- [Micea 00f] M. V. Micea, V. Cretu, D. Chiciudean, "Interfacing a Data Acquisition System to the DSP56303", Application Note AN2087/D Rev. 0, 12/2000, Motorola, Inc., USA, 2000.
- [Micea 01a] M. V. Micea, A. Trifu, M. Trifu, "Integrating the DSP563xx in Distributed Computing Environments", Application Note AN2088/D Rev. 0, 3/2001, Motorola, Incorporated, USA, 2001.
- [Micea 01b] M. V. Micea, L. Muntean, D. Brosteanu, "Simple Real-time Sonar with the DSP56824", Application Note AN2086/D Rev. 0, 6/2001, Motorola, Incorporated, USA, 2001.

- [Micea 01c] M. V. Micea, M. Stratulat, D. Ardelean, D. Aioanei, "Implementing Professional Audio Effects with DSPs", in Transactions on Automatic Control and Computer Science, Vol. 46 (60), Periodica Politehnica, Timisoara, Romania, 2001, pp. (55-60).
- [Micea 04a] M. V. Micea, V. Cretu, "Non-Preemptive Execution Support for Critical and Hard Real-Time Applications on Embedded Platforms", in Proceedings of the 2004 International Symposium on Signals, Systems and Electronics, ISSSE'04, Linz, Austria, August 2004, in print.
- [Micea 04b] M. V. Micea, V. Cretu, L. M. Patcas, "Program Modeling and Analysis of Real-Time and Embedded Applications", in the Scientific Bulletin of "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science, Vol. 49 (63), No. 3, Timisoara, Romania, May 2004, pp. (207-212).
- [Micea 04c] M. V. Micea, "HARETICK: A Real-Time Compact Kernel for Critical Applications on Embedded Platforms", in Proceedings of the 7th International Conference on Development and Application Systems, DAS2004, Editura Universitatii Suceava, Suceava, Romania, May 2004, pp. (16-23).
- [Mok 83] A. K. Mok, "Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment", PhD. Thesis, MIT Technical Report MIT/LCS/TR-297, Laboratory of Computer Science, Massachusetts Institute of Technology, Massachusetts, May 1983.
- [Mok 84] A. K. Mok, "The Design of Real-Time Programming Systems Based on Process Models", in IEEE Proceedings of the 5th Real Time Systems Symposium, Austin, Texas, December 1984, pp. (5-17).
- [Motorola 00] Motorola, Inc., "DSP56300: 24-Bit Digital Signal Processor: Family Manual", Revision 3, DSP56300FM/AD, Semiconductor Products Sector, DSP Division, Austin, USA, November 2000.
- [Motorola 95] Motorola, Inc., "DSP56000: 24-Bit Digital Signal Processor: Family Manual", DSP56KFAMUM/AD, Semiconductor Products Sector, DSP Division, Austin, USA, 1995.
- [Motorola 96] Motorola, Inc., "Motorola DSP: Assembler Reference Manual", Version 6.0, Semiconductor Products Sector, DSP Division, Austin, USA, 1996.
- [Motorola 98] Motorola, Inc., "DSP56307: 24-Bit Digital Signal Processor: User's Manual", DSP56307UM/D, Revision 0, 08/10/98, Semiconductor Products Sector, DSP Division, Austin, USA, August 1998.

- [Motorola 99] Motorola, Inc., "DSP56307EVM User's Manual", Revision 1.8, 3/1999, DSP56307EVMUM/D, Semiconductor Products Sector, DSP Division, Austin, USA, March 1999.
- [Muchnick 97] S. S. Muchnick, "Advanced Compiler Design and Implementation", Academic Press, Morgan Kaufmann Publishers, 1997.
- [NatInst 99] National Instruments Corp., "Measurement and Automation Catalogue", 1999.
- [Niehaus 91] D. Niehaus, "Program Representation and Translation for Predictable Real-time Systems", Department of Computer and Information Science, University of Massachusetts, 1991.
- [Niehaus 93] D. Niehaus, K. Ramamritham, J. A. Stankovic, G. Wallace, C. Weems, W. Burleson, J. Ko, "The Spring Scheduling Co-Processor: Design, Use and Performance", in Real-Time Systems Symposium, December 1993.
- [Ning 02] P. Ning, "Basic Number Theory (Topic 2.3)", Course Notes, Information Systems Security CSC 495N/574, Department of Computer Science, North Carolina State University, USA, 2002.
- [Oppenheim 96] A. V. Oppenheim, R. W. Schaffer, "Digital Signal Processing", Prentice-Hall, 1996.
- [Ostroff 92] J. S. Ostroff, "Formal Methods for the Specification and Design of Real-time Safety Critical Systems", Journal of Systems and Software, Vol. 18, No. 1, 1992, pp. (33-60).
- [Panzieri 93a] F. Panzieri, L. Donatiello, L. Poretti, "Scheduling Real Time Tasks: A Performance Study", Technical Report UBLCS-93-10, May 1993, Laboratory for Computer Science, University of Bologna, Italy.
- [Panzieri 93b] F. Panzieri, R. Davoli, "Real Time Systems: A Tutorial", Technical Report UBLCS-93-22, Laboratory of Computer Science, University of Bologna, Italy, October 1993.
- [Parashkevov 94] A. Parashkevov, "Distributed Real-time Computer Systems", Technical Report CRTS-94-01, Department of Computer Science, University of Adelaide, Australia, 1994.
- [Park 91] C. Y. Park, A. C. Shaw, "Experiments with a Program Timing Tool Based on Source-level Timing Schema", IEEE Computer, May 1991, pp. (48-57).
- [Pentek 96] Pentek, Inc., "Digital Signal Processing and Data Acquisition", Product Catalog, 1996.
- [Podgurski 89] A. Podgurski, L. A. Clarke, "The Implications of Program Dependences for Software Testing, Debugging and Maintenance", ACM Software Engineering Notes, No. 14 (8), 1989, pp. (168-178).

- [Proakis 96] J. G. Proakis, D. G. Manolakis, "Digital Signal Processing: Principles, Algorithms and Applications", 3rd Edition, Prentice-Hall, 1996.
- [Puschner 89] P. P. Puschner, C. Koza, "Calculating the Maximum Execution Time of Real-time Programs", *Journal of Real-time Systems*, 1 (2), September 1989, pp. (159-176).
- [Puschner 91] P. P. Puschner, A. V. Schedl, "Computing Maximum Task Execution Times – A Graph-based Approach", Kluwer Academic Publishers, 1991.
- [Puschner 00] P. P. Puschner, A. Burns, "A Review of Worst-Case Execution Time Analysis", Guest Editorial, in *Real-Time Systems*, No. 18 (2/3), May 2000, pp. (115-127).
- [QNX 99a] QNX Software Systems, Ltd., "QNX Neutrino v2 Realtime Operating System: Building Embedded Systems", 3rd Edition, Ontario, Canada, May 1999.
- [QNX 99b] QNX Software Systems, Ltd., "QNX Neutrino v2 Realtime Operating System: Development Tools", 2nd Edition, Ontario, Canada, June 1999.
- [Radulescu 02] A. Radulescu, A. J. C. van Gemund, "Low-Cost Task Scheduling for Distributed-Memory Machines", in *IEEE Transactions on Parallel and Distributed Systems*, Vol 13, No. 6, June 2002, pp. (648-658).
- [Ramamritham 94] K. Ramamritham, J. A. Stankovic, "Scheduling Algorithms and Operating Systems Support for Real-Time Systems", in *Proceedings of the IEEE*, Vol. 82, No. 1, January 1994, pp. (55-67).
- [Ranvidran 02] B. Ranvidran, "Engineering Dynamic Real-Time Distributed Systems: Architecture, System Description Language and Middleware", in *IEEE Transactions on Software Engineering*, Vol. 28, No. 1, January 2002, pp. (30-57).
- [Robu 02] N. Robu, "Concurrent Programming. Real-Time Oriented Support Mechanisms", ("Programare concurenta. Mecanisme suport orientate timp real"), Editura "Politehnica" Timisoara, Romania, 2002.
- [Shamus 03] Shamus Software Ltd., "MIRACL Users Manual", Version 4.8, Dublin, Ireland, August 2003.
- [Shaw 89] A. C. Shaw, "Reasoning about Time in Higher Level Language Software", *IEEE Transactions on Software Engineering*, 15 (7), July 1989, pp. (875-889).
- [Shoup 02] V. Shoup, "A Tour of NTL", 2002 [Online]. Available: <http://www.shoup.net>.

- [Sprunt 89a] B. Sprunt, J.P. Lehoczky, L. Sha, "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System", Technical Report CMU/SEI-890TR-11, April 1989.
- [Sprunt 89b] B. Sprunt, L. Sha, J.P. Lehoczky, "Aperiodic Task Scheduling for Hard-Real-Time Systems", in *Real-Time Systems*, Vol. 1, No. 1, June 1989, pp. (27-60).
- [Stallings 98] W. Stallings, "Operating Principles: Internals and Design Principles", 3rd Edition, Prentice-Hall, New Jersey, USA, 1998.
- [Stankovic 88] J. A. Stankovic, "Misconceptions about Real-Time Computing", *IEEE Computer*, Vol. 21, No. 10, October 1988, pp. (10-19).
- [Stankovic 90] J. A. Stankovic, "The Spring Architecture", in *Proceedings of the IEEE Workshop on Real Time, Euromicro'90*, June 1990, pp. (104-113).
- [Stankovic 91a] J. A. Stankovic, "On the Reflective Nature of the Spring Kernel", invited paper, in *Proceedings of Process Control Systems '91*, February 1991.
- [Stankovic 91b] J. A. Stankovic, K. Ramamritham, "The Spring Kernel: A New Paradigm for Hard Real-Time Operating Systems", in *IEEE Software*, No. 8 (3), May 1991, pp. (62-72).
- [Stankovic 91c] J. A. Stankovic, D. Niehaus, K. Ramamritham, "SpringNet: A Scalable Architecture for High Performance, Predictable, and Distributed Real-Time Computing", *Univ. of Massachusetts, TR 91-74*, October 1991.
- [Stankovic 92a] J. A. Stankovic, "Distributed Real-Time Computing: The Next Generation", Invited keynote paper, Special issue of "Journal of the Society of Instrumentation and Control Engineers of Japan", Vol. 31, No. 7, pp. (726-736), 1992.
- [Stankovic 92b] J. A. Stankovic, "Real-Time Computing", Invited paper, *BYTE*, pp. (155-160), August 1992.
- [Stankovic 99] J. A. Stankovic, K. Ramamritham, D. Niehaus, M. Humphrey, G. Wallace, "The Spring System: Integrated Support for Complex Real-Time Systems", in the special issue of the *Real-Time Systems Journal*, Vol. 16, No. 2/3, 1999.
- [Stewart 01] D. B. Stewart, "Twenty-five Most Common Mistakes with Real-time Software Development", in *2001 Embedded Systems Conference, Class 270*, San Francisco, April 2001.
- [Stoyenko 91] A. D. Stoyenko, C. Hamacher, R. C. Holt, "Analyzing Hard Real-time Programs for Guaranteed Schedulability", *IEEE Transactions on Software Engineering*, 17 (8), August 1991, pp. (737-750).

- [Stoyenko 95] A. D. Stoyenko, T. J. Marlowe, M. F. Younis, "A Language for Complex Real-time Systems", in *The Computer Journal*, Vol. 38, No. 4, 1995, pp. (319-338).
- [Tanenbaum 97] A. S. Tanenbaum, A. S. Woodhull, "Operating Systems: Design and Implementation", 2nd Edition, Prentice-Hall, New Jersey, USA, 1997.
- [TexasInst 92] Texas Instruments, Inc., "TMS320C31 Embedded Control Technical Brief", SPRU083, August 1992.
- [TexasInst 97] Texas Instruments, Inc., "TMS320 DSP Development Support Reference Guide", SPRU011E, January 1997.
- [TexasInst 00a] Texas Instruments, Inc., "DAC707/708/709: Microprocessor-Compatible 16-Bit Digital-to-Analog Converters", Burr-Brown Product Data Sheet PDS-557H (SBAS145), 2000.
- [TexasInst 00b] Texas Instruments, Inc., "DAC80/DAC80P: Monolithic 12-bit Digital-to-Analog Converters", Burr-Brown Product Data Sheet PDS-643F (SBAS148), 2000.
- [TexasInst 00c] Texas Instruments, Inc., "MPC506A/507A: Single-Ended 16-Channel / Differential 8-Channel CMOS Analog Multiplexers", Burr-Brown Product Data Sheet PDS-774E (SBFS018), 2000.
- [Toma 96] L. Toma, "Sisteme de achiziție și prelucrare numerică a semnalelor" ("Digital Signal Acquisition and Processing Systems"), Editura de Vest, Timisoara, 1996.
- [Ulrich 98] R. Ulrich, "Gaussian Distribution: FAQ", 1998 [Online]. Available: <http://www.pitt.edu/~wpilib/stats99.html>.
- [Young 82] S. Young, "Real Time Languages: Design and Development", Ellis Horwood Publishers, 1982.

Lista lucrărilor publicate

- [1] M. V. Micea, V. Cretu, "Non-Preemptive Execution Support for Critical and Hard Real-Time Applications on Embedded Platforms", in Proceedings of the 2004 International Symposium on Signals, Systems and Electronics, ISSSE'04, Linz, Austria, August 2004, [CD-ROM version].
- [2] M. V. Micea, V. Cretu, L. M. Patcas, "Program Modeling and Analysis of Real-Time and Embedded Applications", in the Scientific Bulletin of "Politehnica" University of Timisoara, Transactions on Automatic Control and Computer Science, Vol. 49 (63), No. 3, Timisoara, Romania, May 2004, pp. (207-212).
- [3] M. V. Micea, "HARETICK: A Real-Time Compact Kernel for Critical Applications on Embedded Platforms", in Proceedings of the 7th International Conference on Development and Application Systems, DAS2004, Editura Universitatii Suceava, Suceava, Romania, May 2004, pp. (16-23).
- [4] V. Cretu, T. Jurca, M. V. Micea, I. Sora, "Instrumentation and Measurement in Romania: Technical Developments at 'Politehnica' University of Timisoara", in IEEE Instrumentation & Measurement Magazine, Vol. 6, No. 3, September 2003, pp. (41-47).
- [5] M. V. Micea, "Introducere in Prelucrarea Numerica a Semnalelor: Suport de curs", ("Introduction to Digital Signal Processing: Course Notebook"), Comanda 120/2003, Centrul de multiplicare al Universitatii POLITEHNICA Timisoara, 2003.
- [6] V. Cretu, L. E. Anton, M. V. Micea, A. Baya, R. Resiga, D. Chiciudean, "Applications for Experimental Study of Hydraulic Phenomena in Hydrodynamic Circuits", in Transactions on Automatic Control and Computer Science, Vol. 47 (61), Periodica Politehnica, Timisoara, Romania, 2002, pp. (65-70).
- [7] M. V. Micea, L. Muntean, D. Brosteanu, "Embedded Techniques for Autonomous Robot Orientation", in Proceedings of the 6-th International Conference on Development and Application Systems, DAS2002, Suceava, Romania, May 2002, pp. (22-27).
- [8] M. V. Micea, M. Stratulat, D. Ardelean, D. Aioanei, "Implementing Professional Audio Effects with DSPs", in Transactions on Automatic Control and Computer Science, Vol. 46 (60), Periodica Politehnica, Timisoara, Romania, 2001, pp. (55-60).
- [9] M. V. Micea, L. Muntean, D. Brosteanu, "Simple Real-time Sonar with the DSP56824", Application Note AN2086/D Rev. 0, 6/2001, Motorola, Incorporated, USA, 2001.

-
- [10] M. V. Micea, A. Trifu, M. Trifu, "Integrating the DSP563xx in Distributed Computing Environments", Application Note AN2088/D Rev. 0, 3/2001, Motorola, Incorporated, USA, 2001.
- [11] M. V. Micea, V. Cretu, D. Chiciudean, "Interfacing a Data Acquisition System to the DSP56303", Application Note AN2087/D Rev. 0, 12/2000, Motorola, Incorporated, USA, 2000.
- [12] M. V. Micea, "Interfacing DAQ Systems to Digital Signal Processors", in Transactions on Automatic Control and Computer Science, Special Issue Dedicated to the Fourth International Conference on Technical Informatics, CONTI'2000, Vol. 45 (59), No. 4, Timisoara, Romania, October 2000, pp. (23-28).
- [13] M. V. Micea, L. Muntean, D. Brosteanu, D. Chiciudean, "Simple Real-time SONAR Implemented with DSP-based Boards", in Proceedings of the 5-th International Conference on Development and Application Systems, DAS2000, Suceava, Romania, May 2000, pp. (67-72).
- [14] M. V. Micea, D. Chiciudean, L. Muntean, "ECP Standard Parallel Interface for DSP56300 Devices", Application Note AN2085/D Rev. 0, 11/2000, Motorola, Incorporated, USA, 2000.
- [15] M. V. Micea, A. Trifu, M. Trifu, "Distributed Digital Signal Processing Using DSP-based Boards and the Khoros Package on LINUX Platforms", in Proceedings of the 5-th International Conference on Development and Application Systems, DAS 2000, Suceava, Romania, May 2000, pp. (61-66).
- [16] M. V. Micea, V. Cretu, D. Chiciudean, "Communication Protocols Implementation on DSP-based Systems", in Proceedings of the International Symposium on Systems Theory, SINTES 10, 10th Edition, Automation, Computers, Electronics, Craiova, Romania, May 2000, pp. (C 205-208).
- [17] M. V. Micea, "Sisteme de achizitie numerica a datelor: Indrumator de laborator", ("Signal Acquisition and Conditioning Systems: Laboratory Workshops"), Comanda 270/2000, Centrul de multiplicare al Universitatii POLITEHNICA Timisoara, 2000.
- [18] V. Cretu, M. V. Micea, I. Guzun, V. Guzun, "Aquarius-DSP, from its Design to Applications Integration", in Transactions on Automatic Control and Computer Science, Vol. 44 (58), No. 1, 2, Periodica Politehnica, Timisoara, Romania, 1999, pp. (5-16).
- [19] A. Baya, L. E. Anton, V. Ancusa, M. V. Micea, "Real Time Data Acquisition System for a Compound Straight Pipe", in Scientific and Technical Bulletin of the "Politehnica" University of Timisoara, Mechanics, Tom (44) 58, 1999, pp. (41-50).
- [20] M. V. Micea, I. Guzun, V. Guzun, "Performant Multifunction I/O Board for the IBM PC AT", in Proceedings of the International Conference on Microelectronics and Computer Science, Vol. 1, Technical University of Chisinau, Rep. of Moldova, 1997, pp. (167-171).
- [21] M. V. Micea, "Real-Time Data Acquisition and Digital Signal Processing Systems: Present and Prospects", PhD Report No. 1, Department of Computer Science & Engineering, "Politehnica" University of Timisoara, Romania, 2002.

- [22] M. V. Micea, "On Design and Implementation of Hard Real-Time Systems for Critical Applications", ("Consideratii referitoare la proiectarea si implementarea sistemelor timp-real stricte pentru aplicatii critice"), PhD Report No. 2, Department of Computer Science & Engineering, "Politehnica" University of Timisoara, Romania, 2003.
- [23] M. V. Micea, "Design and Implementation of Hard Real-Time Systems for Critical Applications: the HARETICK system", ("Proiectarea si implementarea sistemelor timp-real stricte pentru aplicatii critice: sistemul HARETICK"), PhD Report No. 3, Department of Computer Science & Engineering, "Politehnica" University of Timisoara, Romania, 2003.