Universitatea Ştefan cel Mare Suceava
Facultatea de Inginerie Electrică şi Ştiinţa Calculatoarelor

CONTRIBUŢII ASUPRA PROIECTĂRII OPTIMIZATE A ARHITECTURII CPU BAZATĂ PE PROGRAMARE ÎN TIMP REAL ŞI REGIŞTRI PARALELI (PIPELINE)

- rezumat -

Coordonator ştiinţific,
prof. univ. dr. ing. Vasile-Gheorghiţă GĂITAN

Doctorand,
ing. MOISUC (CIOBANU) Elena-Eugenia

Suceava
- 2016 -

Investeşte în oameni!
FONDUL SOCIAL EUROPEAN
Proiect cofinanțat din Fondul Social EUROPEAN prin
Programul Operaţional Sectorial Dezvoltarea Resurselor
Umane 2007 – 2013

Această lucrare a beneficiat de suport financiar prin proiectul

Performanță sustenabilă în cercetarea doctorală și post doctorală - PERFORM

Contract nr. POSDRU 159/1.5/S/138963

Axa prioritară 1 - Educaţia şi formarea profesională în sprijinul creşterii economice şi dezvoltării societăţii bazate pe cunoaştere

Domeniul major de intervenţie 1.5 - „Programe doctorale şi postdoctorale în sprijinul cercetării”

În memoria soțului meu, Dănuț!

CUPRINS

ABREVIERI

ADC - Analog-to-Digital Converter
ADDI - Addition Immediate
ADEOS - Adaptive Domain Environment for Operating Systems
AHB - Advanced High Performance Bus
ALM - Adaptive Logic Modules
ALU - Arithmetic and Logical Unit
AMBA - Advanced Microcontroller Bus Architecture
APB - Advanced Peripheral Bus
API - Application Programming Interface
ARM - Advanced Risc Machines
ASIC - Application Specific Integrated Circuit
BAI - Blocul de Atașare a Întreruperilor
BCET - Best Case Execution Time
BCI - Blocul de Centralizare a Întreruperilor
BEQ - Branch if Equal
CAD - Computer Aided Design
CAM - Content Addressable Memory
CAN - Controller Area Network
CAP - Codificatorul de Adrese Prioritar
CF - Compact Flash
CISC - Complex Instruction Set Computer
CJA - Conditional Jump Address
CLB - Configurable Logic Block
CP - Coprocessor
CPLD - Complex Programmable Logic Device
CPSR - Current Program Status Register
CPU - Central Processor Unit
CS - Comutare de context
CSA - Context Save Area
DA - Data Access
DFF - D-Flip Flop
DIP - Dual Inline Package
DMA - Direct Memory Access
DRAM - Dynamic Random Access Memory
EAB - Embedded Array Blocks
EDDL – Electronic Device Description Language
EDF - Earliest Deadline First
EES - Early Engineering Sample
EmS - Embedded Systems
EX - Execute
EXMEM - Execute / Memory
FDI - Field Device Integration
FDT - Field Device Tool
FIFO - First In First Out
FIQ - Fast Interrupt Request
FPGA - Field Programabble Gate Array
GPIO - General Purpose Input Output
GPR - General Purpose Register
HART - Highway Addressable Remote Transducer
HDL - Hardware Description Language
HDU - Hazard Detection Unit
HSE -Hardware Scheduler Engine
ID - Instruction Decode
IDEX - Instruction Decode / Execute
IF - Instruction Fetch
IFID - Instruction Fetch Instruction Decode
ILP - Instruction Level Parallelism
IO - Input Output
IR - Instruction Registers
IRQ - Interrupt Request
ISA - Instruction Set Architecture
ISR - Interrupt Service Routine
IST - Interrupt Service Task
J - Jump
JTAG - Joint Test and Access Group

LAB - Logic Array Block
LB - Load Byte
LE - Logic Elements
LUT - Look-Up Table
MEM - MEMory
MEMWB - Memory/Write Back
MIPS - Microprocessor without Interlocked Pipeline Stages
MMU - Memory Management Unit
MPRA - Multi Pipeline Register Architecture
MPSoC - Multi-processor System on Chip
MR - Mask Register
MUX - Multiplexor
NOP - No Operation
OLE - Object Linking and Embedding
OPC - OLE Process Control
OSEK - Open Systems and their Interfaces for the Electronics in Motor Vehicles
PC - Program Counter
PIA - Programmable Interconnect Array
PID - Proportional Integral Derivative Controller
PLC - Programmable Logic Controller
PLD - Programmable Logic Device
PLL - Phase Locked Loop
PWM - Pulse Width Modulation
QoS - Quality of Service
RAM - Random Access Memory
RISC - Reduced Instruction Set Computer
RMA - Rate Monothonic Algorithm
ROM - Read Only Memory
RR – Round Robin
RTAI - Real Time Application Interface
RTL - Register Transfer Level
RTOS - Real Time Operating System
RTS - Real-Time System
RTU - Real-Time Unit
SB - Store Byte
SCADA - Supervisory Control and Data Acquisition
SDRAM - Synchronous Dynamic Random Access Memory
SMT - Simultaneous Multithreading
SNTP - Simple Network Time Protocol
SO - Sistem de Operare
SoC - System on Chip
SP - Stack Pointer
SPI - Synchronous Peripheral Interface
SRAM - Static Random Access Memory
SRR - Service Request Register
STR - Sistem de timp real
TAS - Test and set
TCB - Task Control Block
TLB - Translation Lookaside Buffer
TTB - Translation Table Base
UA - Unified Architecture
UART - Universal Asinchronous Receiver
UCJA - Unconditional Jump Address
URI - Universal Resource Identificator
USB - Universal Serial Bus Transmitter
VHDL - Very high speed integrated circuit Hardware Description Language
VIC - Vectored Interrupt Controller
WB - Write Back
WCET - Worst Case Execution Time
WCRT - Worst Case Response Time
XML - Extensible Markup Language
µC - Microcontroller
µP - Microprocessor

INTRODUCERE

În domeniul sistemelor integrate sistemul de operare în timp-real (STRO) este adesea folosit pentru structurarea codul de aplicaţii şi asigurarea respectării deadline-urilor (termenelelor limită). În mod obişnuit, sistemele de timp-real sunt implementate în software, dar tot mai multe sisteme integrate sunt dezvoltate pe o platformă FPGA, aşa că este firesc să se investigheze modul în care un FPGA poate fi utilizat pentru execuţia paralelă a diferitelor task-uri.

Dezavantajele majore ale standardelor software bazate pe STRO sunt determinate de supracontrol, jitter-e şi adesea de o cerinţă mare de memorie. Concluzia, care reiese din literatura de specialitate a ultimelor două decenii şi care este preluată în acest studiu, este că cele mai importante dezavantaje software, în cazul Sistemelor de Timp-Real, pot fi eliminate prin implementarea întregului kernel (nucleu) în hardware. Problemele planificatoarelor hardware, ca viteza lentă de comunicare şi inflexibilitatea, au fost depăşite prin implementarea pe FPGA-uri, a căror evoluţie a determinat ca implementarea hardware să devină, în plus faţă de eliminarea dezavantajelor software, ieftină, flexibilă şi cu viteză relativă de execuţie mare.

Arhitectura propusă în acest studiu, *nMPRA*, constă dintr-o structură hardware originală utilizată pentru planificarea task-ului, static și dinamic și oferă managementul unitar al evenimentelor.

Scopul:

- a îmbunatăți prin hardware performanţele RTOS pentru microcontrolere,
- a comuta mai rapid între task-uri,
- a îmbunătăți timpul de răspuns la evenimente externe,
- a îmbunătăți comportamentul întreruperilor, care sunt tratate ca evenimente în acest caz
- a oferi mai multe tipuri de bază de comunicare inter-task-uri (mesaje, mutex-uri etc.).

Obiectivul studiului de faţă este de a analiza şi soluţiona implementarea hardware a managementului evenimentelor, întreruperilor, resurselor unui Sistem de Timp-Real, pentru a putea introduce apoi un planificator hardware care să elimine nedeterminismul, supracontrolul şi pentru a atinge cel mai scurt posibil timp de aşteptare (latenţă) la comutarea de context.

Teza este structurată astfel: *Introducere*, patru *Capitole*, *Bibliografie* şi *Anexe*.

Capitolul I, în prima parte, face o introducere în STR şi în planificarea în timp-real; în partea a doua este prezentat istoricul şi apoi evoluţia sistemelor de timp-real, începând cu prezentarea sistemelor încorporate, apoi a sistemelor de management al task-urilor, evoluţia implementării hardware pe FPGA şi analiza planificatoarelor software vs. Hardware; în partea a treia se prezintă stadiul actual privind proiectarea hardware/software a planificatoarelor (schedulers) în timp-real pentru sisteme integrate (embedded) şi anume un planificator dedicat, un accelerator de planificare, un exemplu de planificator hardware şi un planificator hardware integrat; în încheierea primului capitol sunt prezentate concluziile respective.

Problema cea mai importantă a unui STR o constituie planificarea resurselor: procesor, memorie şi reţelele de comunicaţie în cazul sistemelor distribuite. Astăzi, tot mai multe sisteme complexe se bazează, parţial sau în mod absolut, pe controlul procesorului.

Evoluţia neobişnuită a dispozitivelor FPGA a influenţat puternic atât metodologia de proiectare, cât şi cerinţele impuse uneltelor de dezvoltare.

Ca orice sistem tehnic, un sistem încorporat de timp real trebuie să fie fiabil şi sigur. Domeniile de utilizare pot implica însăşi siguranţa umană, impunând condiţii de timp stricte. De aceea, dezvoltarea de mecanisme de timp real şi planificarea proceselor cu condiţii stricte de timp este o provocare. În acest context, sistemele de operare trebuie să pună la dispoziţie mecanisme de timp real şi de planificare pentru task-urile stricte astfel încât să garanteze satisfacerea condiţiilor de timp.

Planificatorul de timp-real este o unitate de program care controlează lansarea în execuţie, întreruperea temporară şi terminarea execuţiei unor module-program pe baza unui algoritm prestabilit, cu scopul de a satisface restricţiile de timp impuse.

Planificatoarele hardware au rolul de a degreva procesorul de activitatea de planificare a task-urilor preluând ele această sarcină.

Arhitectura MPRA utilizează un planificator hardware care este parte constituentă a procesorului, iar controlul acestuia se face prin instrucţiuni dedicate care sunt transmise prin banda de asamblare.

Capitolul II face o scurtă prezentare a arhitecturii MIPS32 pipeline, a cărei structură organizaţională va fi utilizată pentru design-ul *nMPRA*, arhitectură de microcontroller propusă şi analizată în acest studiu. MIPS (*Micropocessor whithout Interlocked Pipeline Stages*) este o arhitectură de microprocesoare RISC (*Reduced Instruction Set Computer*) dezvoltată de *MIPS Technologies*.

Consumul scăzut şi caracteristicile de încălzire ale sistemelor embedded ce reprezintă implementări MIPS32, disponibilitatea utilitarelor pentru dezvoltare în domeniul embedded precum şi faptul că MIPS32 a devenit o

arhitectură cunoscută, toate îi asigură procesorului de tip MIPS un rol important în industria embedded.

Procesorul MIPS32 pipeline este, în esență, un procesor într-un singur ciclu, având câteva caracteristici avansate adăugate, patru registre pipeline, o unitate de hazard și o unitate de forward. O altă diferență majoră este poziția branch-ului și a jump-ului logic.

Avantajul de a avea aceste patru registre pipeline este acela că mai mult de o instrucțiune poate fi prezentă în același timp. Acest lucru permite următoarei instrucțiuni să intre în prima etapă de îndată ce instrucțiunea curentă o părăsește. Ca diferență, la cele 5 cicluri de așteptare pentru pentru a finaliza o instrucțiune, acum până la 5 instrucțiuni pot fi procesate în același timp.

Utilizând structura organizațională a arhitecturii MIPS32, este propusă arhitectura cu regiștrii pipeline multiplicați, *MPRA*.

Multi Pipeline Register Architecture (*MPRA*) a fost modificată și transformată în *n*-task *MPRA* (*nMPRA*). Ideea de bază a fost de a replica registrele pipeline și de a crea mai multe instanțe ale procesorului, numite *semi CPU*, pentru a avea un semi-procesor pentru fiecare task *i* ($sCPU_i$).

Capitolul III cuprinde studiul autorului asupra arhitecturii propuse de echipa de cercetare, *nMPRA*; în prima parte a acestui capitol se prezintă rezultatele obținute de autor, susținute și publicate în cadrul conferințelor internaționale și concluziile respective; partea a doua cuprinde analiza și sinteza implementării *4MPRA* și *8MPRA* pe FPGA-ul Cyclone V, Quartus II de la ALTERA și concluziile care rezultă.

MPRA oferă un mecanism dinamic de gestionare a întreruperilor. Este propus și un mecanism hardware pentru gestionarea evenimentelor în scopul de a îmbunătăți soluția software, care nu este eficientă deoarece generează întârzieri. Metoda este implementată în arhitectura *n*-task-uri MPRA (*Multi Pipeline Register Architecture*), care este o arhitectură de microcontroller cu capacități de real-time implementată în hardware, care permite comutarea între task-uri în timp de un ciclu-procesor și un timp de răspuns la evenimente de până la 1,5 cicluri-procesor.

Arhitectura *Multi Pipeline Register* (*MPRA*) include planificatorul hardware. În procesorul *MPRA*, registrul *Program Counter* (*PC*), regiștrii pipeline și registrele generale sunt resurse multiplicate, iar memoria necesară pentru punerea în aplicare a acestor registre este direct proporțională cu numărul de task-uri din sistem. În scopul de a asigura predictibilitatea STR, *MPRA* pune în aplicare un sistem de planificare rigidă, care se bazează pe codarea adresei de prioritate în *Hardware Scheduler Engine* (*HSE*). Cu toate acestea, lasă suficientă libertate utilizatorului de a pune în aplicare algoritmi doriți de planificare, ca *Earliest Deadline First* (*EDF*) sau *Rate Monotone*

Algorithm (*RMA*), printr-o ordonare corectă a task-urilor în funcție de frecvența de execuție sau deadline-uri.

Capitolul IV cuprinde concluziile finale şi prezentarea, pe scurt, a contribuţiei autorului la studiul propus.

CAPITOLUL I

1. SISTEME DE TIMP REAL

În prezent, necesitatea optimizării modalității în care este folosit timpul poate fi observată cu uşurinţă în toate domeniile în care roboţii, de exemplu, au reuşit să degreveze activităţile umane. Apariţia procesoarelor şi a sistemelor de control bazate pe acestea, precum şi a algoritmilor de planificare software au permis eficientizarea organizării timpului în sistemele digitale cu rezoluţii de ordinul *ms* sau chiar mai mici. Ca şi în viaţa de zi cu zi, partajarea timpului rămâne astfel singura tehnică de execuţie pseudo-paralelă a task-urilor într-un sistem embedded de timp real (STR).

Astfel, în **Introducere** se prezintă sistemele de timp real, în general, cu definiţie, particularităţi, cerințe, domenii de utilizare, despre task-uri şi planificarea în timp real.

În **partea a 2-a** s-a considerat necesar un **scurt istoric** deoarece toate cercetările actuale se bazează pe cele dinainte, nu s-a făcut nici o „revoluţie", încă, în domeniu. De altfel, domeniul este atât de „bogat", de întins, de abordat din toate punctele de vedere, încât este foarte greu de relatat, în câteva pagini, conceptele şi apoi rezultatele cercetărilor din lume. De aceea se vor prezenta aici doar câteva idei, continuate cu evoluţia până la zi a STR. Tot aici s-a considerat necesară o scurtă privire şi asupra evoluţiei FPGA deoarece multe dintre structurile hardware, mai mult sau mai puţin actuale, folosesc FPGA.

În **partea a 3-a** se prezintă câteva exemple actuale (2012, 2013) de planificatoare hardware şi acceleratoare pentru planificator hardware.

Concluziile încheie acest subcapitol, arătând importanţa şi avantajul palnificării în hardware, care, printre altele, elimină supracontrolul datorat sistemului de operare, îmbunătăţind astfel limita de planificare a setului de task-uri şi performanţele sistemului per ansamblu.

1.1. Introducere

Un **STR** (***S**istem de **T**imp **R**eal*) este *acel sistem ce oferă un răspuns la cererile adresate înainte de un termen limită denumit deadline şi care reacţionează dinamic la schimbările mediului înconjurător*.

Problema predictibilităţii este *una din cele mai importante proprietăţi ale unui sistem de operare*, apoi viteza de reacţie, apoi planificabilitatea bazată pe priorităţi.

1.2. Planificarea în timp-real

Planificatoarele, care trebuie să asigure funcţionarea sistemelor de control al instalaţiilor sau aparatelor ce ar putea pune în pericol procesele industriale sau viaţa oamenilor (automotive, aerospace), trebuie să aibă o construcţie foarte fiabilă, conformă cu standardele în domeniu.

Planificatorul de timp-real este *o unitate de program care controlează lansarea în execuţie, întreruperea temporară şi terminarea execuţiei unor module-program pe baza unui algoritm prestabilit, cu scopul de a satisface restricţiile de timp impuse*.

Planificatoarele de timp real trebuie proiectate conform unor principii teoretice foarte solide pentru a asigura corectitudinea execuţiei programelor şi implicit funcţionarea corectă a aplicaţiilor controlate.

2. ISTORICUL ŞI EVOLUŢIA STR

2.1. Sisteme încorporate

Sistemele încorporate (**EmS** – ***Em****bedded* ***S****ystems*) sunt sisteme la care calculatorul/microprocesorul sunt doar simple **componente.** Principalul scop al utilizării microprocesorului este să **simplifice construcţia sistemului** şi să ofere **flexibilitate** în proiectare şi construcţie.

Un EmS este un ***sistem pe bază de microprocesor*** construit pentru a ***controla o funcţie sau un domeniu de funcţii*** particulare şi care ***nu este proiectat pentru a fi programat*** de către utilizatorul final. Sigura interacţiune cu utilizatorul se face în scopul realizării funcţiilor impuse sistemului - aplicaţiei.

Un sistem încorporat foloseşte o combinaţie de hardware şi software („o maşină computaţională") pentru a rezolva o ***funcţie specifică*** lucrând într-un ***mediu reactiv şi care impune constrângeri de timp***.

2.1.2. Sisteme încorporate de timp real

Sistemele încorporate (embedded) cunosc în ultima perioadă o dezvoltare exponenţială, justificată prin faptul că oferă o soluţie fiabilă, eficientă din punct de vedere economic şi tehnic pentru aplicaţii diverse. Aceste sisteme sunt adesea dezvoltate folosind resurse hardware limitate şi luând în considerare restricţii de timp care asigură comportamentul funcţional dorit. În cazul sistemelor încorporate de timp real, timpul poate fi tratat, de asemenea, ca o resursă partajată, respectarea restricţiilor de timp real devenind un obiectiv esenţial în proiectare, implementare şi testare. În acest context, cheia stă în

asigurarea unei secvenţe adecvate de execuţie a proceselor (task-uri şi rutine de tratare a întreruperilor).

Cazul cel mai dificil de tratat este cel al *sistemelor de timp real critice* – pentru care restricţiile de timp asociate sunt critice, însemnând că nu se acceptă încălcarea lor. În aceste situaţii, rezultatele produse după termenul limită predefinit sunt considerate incorecte şi/sau inutile. Sistemele de timp real critice impun reguli stricte de dezvoltare referitoare la arhitectura hardware, sistemul de operare folosit şi aplicaţia software.

2.1.3. Planificarea şi alocarea în sisteme distribuite de timp real

O altă direcţie importantă de studiu vizează analiza **alocării** şi **planificării** în sisteme distribuite de timp real. Sunt cunoscute soluţii bazate pe alocarea fermă a unui task pe un procesor (alocare partiţionată), precum şi soluţii care permit migrarea task-urilor între resursele active. Politicile de alocare şi planificare sunt aplicate la nivel global sau local. În primul caz, lista proceselor ce trebuie executate este gestionată la nivelul întregului sistem distribuit (alocare globală). Aceasta permite ca alocarea să folosească mai eficient resursele de procesare disponibile, dar aduce dezavantajul unei încărcări suplimentare a sistemului prin transmiterea mesajelor suplimentare necesare. Cercetările au abordat iniţial *planificarea partiţionată* pentru planificatoare cu priorităţi statice [23], [24], dar şi dinamice [25], [26] [26], rigiditatea abordării partiţionate determinând ca eforturile ulterioare să fie dedicate dezvoltării abordării *globale* [27], [28], [29], [30]. Avantajele dar şi dezavantajele celor două abordări au condus la apariţia de studii comparative [31], [32], [33] şi la dezvoltarea de planificări de tip *hibrid* [32], [34], [35] ce permit ca în cadrul unei alocări partiţionate să poată exista migrări ale task-urilor de la un procesor la altul.

2.2. Sisteme de management al task-urilor

Într-o aplicaţie de tip real de timp real se execută, de regulă, două tipuri de procese - task-uri şi rutine de tratare a întreruperilor (***I**nterrupt **S**ervice **R**outine*/**ISR**). Alocarea se referă la repartizarea proceselor pe resursele active (unităţi de procesare) disponibile, iar *planificarea* desemnează *stabilirea secvenţei de execuţie pe fiecare resursă activă în parte*. Planificarea task-urilor este în responsabilitatea **planificatorului integrat** în SOTR, iar ISR-urile sunt deservite prin sistemul de întreruperi şi au priorităţi (gestionate prin **controller-**ul **de întreruperi**) uzual mai mari decât priorităţile task-urilor. Trebuie remarcat faptul că, deşi planificatorul nu gestionează execuţia ISR-urilor, orice execuţie a unui ISR solicită ocuparea unei resurse active şi afectează secvenţa de execuţie a task-urilor.

2.2.1. Planificatoare statice

Primele alternative de planificare în sistemele de timp real au fost de tip static. În acest caz planificarea este configurată „offline” şi memorată pentru utilizare „online”. Soluţia oferă **simplitate, predictibilitate şi stabilitate numai pentru un context de lucru aprioric cunoscut**.

2.2.2. Planificatoare dinamice

Primele studii privind **planificarea bazată pe alocarea de priorităţi dinamice** au fost efectuate de Liu şi Layland, pentru algoritmul EDF. Aceasta se bazează pe atribuirea dinamică a priorităţilor în funcţie de termenele limită ale task-urilor. Task-ul cu termenul limită absolut cel mai apropiat primeşte prioritatea cea mai mare. Atribuirea priorităţilor este realizată la fiecare activare a planificatorului care investighează termenele limită ale tuturor task-urilor ce pot fi executate. O altă alternativă de planificare cu priorităţi dinamice este oferită de algoritmul **LLF** (***L**east **L**axity **F**irst*) sau **MLF** (***M**inimum **L**axity **F**irst*), prezentat de Mok. MLF atribuie dinamic nivelul de prioritate cel mai mare task-ului ce are cea mai scurtă rezervă până la termenul său limită [39], [12].

2.2.2.1. Tratarea task-urilor sporadice şi aperiodice în planificatoare dinamice

Task-urile sporadice şi aperiodice sunt tratate în cadrul planificatoarelor dinamice în funcţie de modul de alocare a priorităţilor task-urilor. În cazul planificării bazate pe priorităţi statice, cea mai simplă abordare constă în a aloca task-urilor neperiodice niveluri de prioritate mai mici decât nivelurile de prioritate alocate task-urilor periodice. Acest mod de tratare a task-urilor neperiodice introduce un dezavantaj major, deoarece un astfel de task, deşi neperiodic, poate realiza nişte procesări importante sau poate avea restricţii de timp severe, critice.

2.3. Planificarea şi alocarea în sisteme multiprocesor

Un volum important al cercetărilor este canalizat către analiza planificării task-urilor periodice şi sporadice în sistemele multiprocesor şi distribuite. Odată ce există mai multe resurse active disponibile, planificarea trebuie strâns corelată cu alocarea task-urilor pe procesoare. Unele abordări vizează extinderea algoritmilor de planificare dedicaţi sistemelor uniprocesor, dar există şi abordări specifice.

2.3.1. Metodologia de proiectare a unui sistem de control implementat în hardware pe FPGA

Dispozitivele FPGA permit proiectarea de arhitecturi *hardware* specializate, cu avantajul flexibilităţii mediului programabil în care se

realizează implementarea [49]. Acest lucru oferă un grad de libertate în plus în proiectarea de sisteme de control numeric faţă de folosirea microprocesoarelor, deoarece arhitectura *hardware* a sistemului de control nu este impusă *a priori*.

În majoritatea abordărilor din literatură există o ruptură între proiectarea sistemului de control şi validarea sa prin simulare şi implementarea pe FPGA şi verificarea experimentală; doar uneori se propune o abordare „integrată" [58] pentru simulare şi proiectare/implementare.

2.4. Planificatoare Software/Planificatoare Hardware

Planificatoarele folosesc un algoritm implementat software ce determină care este modul de lansare în execuţie a unui set de task-uri. Un **task** este *o entitate software ce foloseşte partajat procesorul, memoria şi celelalte resurse ale sistemului, dar care oferă o funcţionalitate de sine stătătoare bine conturată.*

În cazul planificatoarelor software, supracontrolul datorat execuţiei planificatorului şi timpul necesar comutării contextelor încarcă schema de planificare şi micşorează timpul util care ar fi trebuit alocat execuţiei task-urilor.

Planificatoarele hardware au rolul de a degreva procesorul de activitatea de planificare a task-urilor preluând ele această sarcină.

În arhitectura MPRA (***M**ulti **P**ipeline **R**egister **A**rchitecture*) abordarea este diferită. S-a considerat absolut necesar ca într-un sistem multitasking să existe cel puţin un mecanism de comunicaţie inter-task. Realizarea unui astfel de procesor fără comunicaţie inter-task ar fi o soluţie nerealistă care ar putea respecta doar matematica sfârşitului anilor '80 referitoare la STR după principiile formulate de Lehockzy, însă nu ar fi nicidecum o soluţie viabilă cu aplicabilitate practică.

Avantajul major al **MPRA** în acest caz, este acela că, deşi a fost proiectat pentru modul de lucru *single thread*, benzile de asamblare ale task-urilor nu sunt afectate de operaţia de remapare şi implicit nici performanţa procesorului nu este degradată. O altă caracteristică a modelului ARPA-MT este aceea că utilizează un coprocesor extern pentru rularea algoritmului de planificare. MPRA utilizează în schimb un planificator intern care nu induce supracontrol datorat sincronizărilor interprocesor şi nici nu necesită timpi adiţionali pentru arbitrarea magistralelor de interconectare.

3. STADIUL ACTUAL PRIVIND PROIECTAREA HARDWARE/SOFTWARE A PLANIFICATOARELOR (SCHEDULERS) ÎN TIMP REAL PENTRU SISTEME INTEGRATE (EMBEDDED) MULTIPROCESOR

Arhitectura MP-SoPC bazată pe FPGA a fost folosită mai mult pentru a face un alt pas spre îmbunătăţirea performanţelor de planificare (scheduling) pe platforme hardware re-programabile, adică prin dezvoltarea unui accelerator hardware al planificatorului nucleului IP. Multe lucrări în acest domeniu s-au ocupat de proiectarea şi validarea experimentală a unui planificator accelerator hardware care să pună în aplicare politicile de planificare în întregime în hardware, eliberarând RTOS de problema planificării.

3.1. Planificator dedicat

În continuare este detaliat un studiu de actualitate [88], specific: Ranjit Adiga descrie modul în care compania sa a procedat în nevoia de implementare completă a RTOS utilizând un instrument de modelare de sistem hardware/software pentru a construi un **planificator dedicat**. De asemenea, a promovat o versiune executabilă web a proiectului, unde utilizatorii pot vizualiza configurarea software a proceselor şi definirea platformei hardware.

În concluzia acestui studiu putem afirma că selecţia corectă a unui planificator RTOS depinde de aplicaţie şi activităţile task-urilor care urmează să fie planificate. Există mulţi algoritmi de planificare din care se poate alege. Dar în această analiză s-a constatat că, având mai multe sarcini mai mici fără preempţiune, se asigură cea mai bună performanţă de ansamblu. Acest lucru a permis, de asemenea, să se ofere o prioritate mai mare pentru anumite sarcini şi să permită preempţiunea pentru task-uri extrem de rare.

3.2. Accelerator de planificare

În 2012, Primiano Tucci prezintă un **scheduling accelerator** implementat cu **VHDL** (***V**HSIC **H**ardware **D**escription **L**anguage*) în acord cu metodologia Altera SoPC [63].

Ca o concluzie, în ceea ce priveşte design-ul hardware, disponibilitatea unei arhitecturi de referinţă care defineşte în mod clar metodologiile şi modelele de interacţiune între componente a dezvăluit strategia câştigătoare pentru dezvoltarea rapidă a unui sistem clar şi bine proiectat.

3.3. Exemplu de planificator hardware

În sistemele embedded de timp real cu execuţie preemptivă apar deseori probleme legate de planificarea timpului. În formula de calcul a utilizării procesorului intervin parametri precum supracontrolul datorat comutării contextelor task-urilor şi a întreruperilor sau supracontrolul planificatorului.

Toţi aceşti parametri pot contribui la apariţia unui efect de jitter care poate influenţa negativ determinismul sistemului. Două posibile soluţii de diminuare a jitter-ului aplicabile în planificatoarele software constau în implementarea unor algoritmi de planificare orientaţi spre satisfacerea timpilor limită (ex. EDF) sau în alegerea unor scheme de priorităţi convenabile pentru întreruperi. Aceste metode nu rezolvă însă problema supracontrolului planificatorului [91]. O soluţie care oferă răspuns la această problemă constă în utilizarea unor planificatoare hardware ce permit degrevarea procesorului de secvenţa de planificare, aceasta fiind realizată în hardware. Planificatoarele hardware reduc considerabil aceşti timpi, dar nu total. Soluţiile propuse de alţi autori care susţin reducerea totală a acestor timpi nu sunt soluţii realiste cu aplicabilitate practică, deoarece nu implementează mecanisme de sincronizare sau comunicaţie inter-task.

3.4. Planificator hardware integrat

Arhitectura hardware propusă [94] în următoarea prezentare este o soluţie realistă de **planificator hardware** care permite remaparea contextelor într-un singur ciclu de ceas şi expune un jitter la doar 3 cicluri de ceas, dat de necesitatea de implementare a mecanismelor de comunicaţie şi sincronizare inter-task atât de necesare în aplicaţiile practice. Arhitectura permite îmbunătăţirea semnificativă a limitei de planificare şi a determinismului sistemului prin utilizarea unei scheme de planificare statică şi a unui spaţiu unificat de priorităţi pentru task-uri şi întreruperi.

Găitan propune o arhitectură [95] hardware de **planificator integrat** în structura unei CPU speciale cu multiplicare de resurse, aceste resurse fiind regiştrii pipeline şi setul de regiştri ai CPU. Sunt propuse câteva arhitecturi hardware originale pentru planificarea statică şi dinamică a taskurilor, gestiunea unitară a evenimentelor, accesul la resursele partajate ale arhitecturii, generarea de evenimente şi transferul de mesaje între taskuri care asigură sincronizarea şi comunicaţia între acestea, precum şi o metodă de asignare a întreruperilor la taskuri care permite controlul comportării acestora în cadrul organizat al planificării taskurilor. Arhitectura permite timpi de comutare între task-uri de un 1 ciclu procesor (maxim trei pentru instrucţinile speciale de scriere în memoria globală) şi un timp de răspuns la evenimente de maxim 1,5 cicli procesor.

Arhitectura MPRA utilizează un planificator hardware care este parte constituentă a procesorului, iar controlul acestuia se face prin instrucţiuni dedicate care sunt transmise prin banda de asamblare. Procedura de salvare a contextelor în arhitecturile de procesor clasice presupune salvarea contextelor în memoria internă de lucru sau pe stivă, iar acestea pot determina efect de jitter şi implicit pot afecta stabilitatea STR critice. Pentru a elimina acest inconvenient, MPRA realizează comutarea contextelor pe principiul remapării

resurselor multiplexate. În procesorul MPRA registrul intern PC, regiştrii pipeline şi fişierul de regiştri sunt resurse multiplexate, iar dimensiunea de memorie consumată pentru implementarea acestor regiştri este direct proporţională cu numărul de task-uri ale aplicaţiei software.

Pentru a asigura predictibilitatea STR, MPRA implementează o schemă de planificare rigidă care are la bază codificatorul de adrese prioritar din componenţa **HSE** (***H**ardware **S**cheduler **E**ngine*). Acesta lasă însă suficientă libertate utilizatorului pentru a implementa algoritmul de planificare dorit (RMA sau EDF) printr-o ordonare corespunzătoare a task-urilor în funcţie de frecvenţa de execuţie sau deadline. Blocul de timer-e din componenţa HSE este o resursă valoroasă care oferă posibilitatea proiectantului sistemului să implementeze un mecanism de tip round-robin cu ignorarea schemei de prioritizare pentru asigurarea unui sistem predictibil.

Fişierul de regiştri MPRA are o implementare specială care, pe lângă faptul că realizează remaparea contextelor sub controlul direct al HSE, permite şi izolarea contextelor procedurilor astfel încât la apelul funcţiilor utilizatorul nu mai trebuie să salveze contextele.

Arhitectura descrisă foloseşte doar structura organizatorică a arhitecturii MIPS. Arhitectura înglobează funcţionalitatea de planificator într-un bloc funcţional al procesorului şi oferă posibilitatea comutării contextelor task-urilor în doar jumătate de ciclu de ceas eliminând dezavantajele specifice planificatoarelor software.

Componenta responsabilă de operaţia de remapare este planificatorul hardware. Procesorul nu poate efectua această operaţie. Din punctul de vedere al performanţei, diferenţele sunt net superioare faţă de arhitecturile clasice care salvează regiştrii pe stivă. În acel caz, timpul necesar schimbării contextelor este direct proporţional cu mărimea şi numărul de regiştri care trebuie salvaţi pe stivă. Pentru că fiecare operaţie de salvare sau restaurare a contextului se materializează în cicluri de acces la memorie, toti aceşti timpi cumulaţi vor determina un supracontrol adiţional care reduce timpul util al procesorului. Arhitectura propusă este net superioară pentru că realizează această operaţie prin remapare. Operaţia de remapare durează doar ½ ciclu de ceas, indiferent de numărul şi dimensiunea regiştrilor de lucru.

3.5. Concluzii

Rolul specific al STR este acela de a asigura controlul predictibil şi determinist al unui proces. STR sunt acele sisteme care oferă un răspuns corect într-un interval de timp prestabilit. Rapiditatea nu este o caracteristică specifică a STR, aceasta fiind mai mult un termen abstract. Viteza de răspuns la evenimente este însă un concept propriu STR diferit de cel menţionat anterior. *Execuţia schemelor de planificare în hardware elimină supracontrolul datorat sistemului de operare îmbunătăţind astfel limita de planificare a setului de*

task-uri şi performanţele sistemului per ansamblu. Deoarece arhitectura procesorului cu planificator hardware integrat este una bazată pe multiplexare de resurse în care consumul de memorie pentru implementare variază proporţional cu numărul de task-uri utilizat, este important de precizat că această arhitectură este destinată aplicaţiilor embedded industriale şi automotive în care numărul de task-uri este cuprins în intervalul [8, 32]. De cele mai multe ori un număr de task-uri care variază în jurul valorii 16 este mai mult decât suficient pentru majoritatea aplicaţiilor de acest tip.

CAPITOLUL II

1. ARHITECTURA MIPS32

În **Introducere** s-a prezentat **arhitectura MIPS32** în general, pornind de la procesoarele **RISC**.

Apoi s-a considerat necesar un **scurt istoric** al procesoarelor MIPS.

Urmează prezentarea procesorului **MIPS32 pipeline**, cu o scurtă incursiune în etapele pipeline.

În final se face o comparaţie **MIPS** *versus* **ARM**.

Concluziile încheie acest capitol, arătând importanţa în sistemele de timp-real a arhitecturii MIPS32, folosită ca structură organizaţională pentru MPRA.

1.1. Introducere

MIPS Technologies şi-a creat o bază puternică în electrocasnice şi media playere portabile. RISC (*Reduced Instruction Set Computer*) este o filosofie de design care a devenit populară în ultimii ani, dominând industria extrem de competitivă de calculatoare. Procesorul RISC operează cu foarte puține tipuri de date și face operații simple. El suportă câteva moduri de adresare şi se bazează în special pe registre. Cele mai multe dintre instrucțiuni operează asupra datelor prezente în registrele interne. MIPS este un procesor RISC.

1.2. Scurt istoric MIPS

MIPS (*Micropocessor whithout Interlocked Pipeline Stages*) este o arhitectură de microprocesoare **RISC** (*Reduced Instruction Set Computer*) dezvoltată de *MIPS Technologies*. [123]

În 1981 la Universitatea Stanford, o echipă condusă de John L. Hennessy a început să lucreze la ceea ce va deveni primul procesor MIPS. Ideea ce stătea la baza proiectului era creşterea performanţelor microprocesorului cu ajutorul unei benzi de asamblare cu cât mai multe nivele,

o tehnică deja cunoscută însă dificil de implementat. Până la sfârşitul anilor ’90 MIPS a devenit un lider în industria procesoarelor pentru sisteme embedded. Din profiturile de astăzi ale companiei, jumătate provin din licenţierea de design în timp ce o mare parte din cealaltă jumătate reprezintă muncă de design sub contract pentru alţi producători.

În concluzie, procesoarele MIPS au fost şi sunt un succes comercial, fiind folosite astăzi în multe aplicaţii destinate consumului sau industriei. Nuclee MIPS se pot găsi în routere, modem-uri pentru cablu, imprimante laser, roboţi, console de jocuri video şi PDA-uri.

1.3. MIPS32 pipeline

Consumul scăzut şi caracteristicile de încălzire ale sistemelor embedded ce reprezintă implementări MIPS, disponibilitatea utilitarelor pentru dezvoltare în domeniul embedded precum şi faptul că MIPS a devenit o arhitectură cunoscută, toate îi asigură procesorului de tip MIPS un rol important în industria embedded.

Procesorul MIPS pipeline este, în esenţă, un procesor într-un singur ciclu, având câteva caracteristici avansate adăugate, patru registre pipeline, o unitate de hazard şi o unitate de forward. O altă diferenţă majoră este poziţia branch-ului şi a jump-ului logic. Avantajul de a avea aceste patru registre pipeline este acela că mai mult de o instrucţiune poate fi prezentă în acelaşi timp. Acest lucru permite următoarei instrucţiuni să intre în prima etapă de îndată ce instrucţiunea curentă o părăseşte. Ca diferenţă, la cele 5 cicluri de aşteptare pentru pentru a finaliza o instrucţiune, acum până la 5 instrucţiuni pot fi procesate în acelaşi timp.

1.4. Conceptul benzii de asamblare (pipeline)

Registrele pipeline sau registrele de stare sunt cele mai importante componente ale întregului procesor. Fără ele ar fi un procesor într-un singur ciclu, ceea ce ar duce la aplicaţii lente şi performanţă scăzută.

Aceste patru registre pipeline permit procesorului să lucreze pe 5 instrucţiuni la un moment dat (într-un singur ciclu). Creşterea vitezei şi a performanţei sunt direct legate de numărul de etape pipeline.

Microprocesorul serial:

- execută *n* instrucţiuni utilizând *s* etape în *ns* cicluri
- total timp de execuţie = *ns*

Microprocesorul pipeline:

- execută *n* instrucţiuni utilizând *s* etape
- total timp de execuţie = *s+(n-1)*

1.5. Datapath

Figura 1 arată datapath-ul pentru un procesor MIPS pipeline. Regiştrii pipeline sunt între stadiile de pipeline. Aceste etape sunt denumite astfel încât să arate conexiunea de la un stadiu la următorul. Se cunoaşte că fiecare operaţiune trebuie să fie completă într-un ciclu de ceas. Registrele pipeline sunt necesare deoarece fiecare operaţiune ce trece de la o etapă la următoarea are nevoie să fie stocată temporar în registrul pipeline corespunzător.

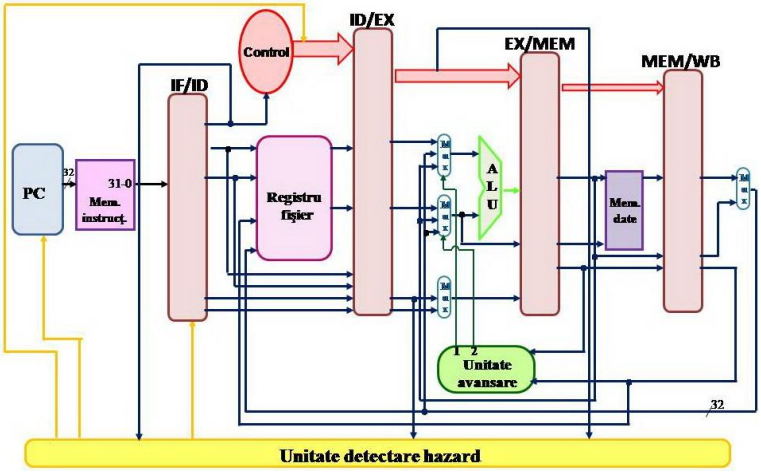


Figura 1 Datapath pipeline cu unitate de control, unitate de forward şi unitate de detecţie a hazardului

Să aruncăm o privire asupra operațiunilor care au loc la fiecare etapă pipeline.

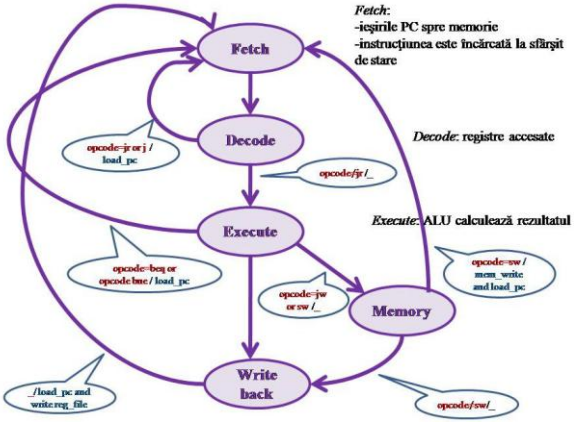


Figura 2 MIPS secvenţial

Pentru o înţelegere aprofundată şi practică a procesorului MIPS pipeline, s-au implementat pe *Quartus II* – Cyclone V mai multe elemente şi instanţe având ca sursă [122].

1.6. MIPS *versus* ARM

Fundamentele tehnologiei RISC, împreună cu design-ul extensibil hardware şi software în arhitectura MIPS, se combină pentru a oferi o mai mare performanţă, consum scăzut de putere, soluție mai compactă, comparativ cu soluțiile similare de la ARM. Patrimoniul tehnologiei MIPS constă în proiectarea stațiilor de lucru de performanță înaltă și a serverelor, în timp ce ARM a început dezvoltarea nucleelor de bază pentru sisteme mobile low-end.

1.7. Concluzii

Procesorul MIPS utilizează un set redus de instrucțiuni, prin urmare rezultă un minim efort uman. Pipeline-ul îmbunătățește rata de transfer a instrucţiunii. Există intenţia de a spori viteza procesorului şi a simplifica hardware-ul, din motive de costuri. Dacă se implementează acest procesor folosind tehnologia FPGA, este posibil să se upgradeze sistemul cu noi caracteristici cerute de utilizatori.

Arhitectura MIPS32 este la fel de actuală, cea mai amplă analiză şi care a rămas de bază fiind făcută de cel care a şi introdus pe piaţă acest procesor, în 1983, Hennessy.

O nouă generație de aplicații embedded este în curs de dezvoltare, condusă de capacitatea de performanță tot mai mare a arhitecturii CPU pe 32 de biți. Tehnologia MIPS de înaltă performanță și lider de performanță/putere-eficientă este perfect adaptată pentru a conduce această nouă generație de produse.

CAPITOLUL III

1. STUDIU ASUPRA UNEI ARHITECTURI DE MICROCONTROLER ÎN SISTEME HARDWARE DE TIMP-REAL

1.1. Introducere

Acest studiu propune un mecanism hardware pentru gestionarea evenimentelor enumerate mai sus, în scopul de a îmbunătăți soluția software, care nu este eficientă deoarece generează întârzieri. Metoda este implementată în arhitectura *n*-task-uri *MPRA* (*Multi Pipeline Register Architecture*), care este

o arhitectură de microcontroller cu capacități de real-time implementată în hardware, care permite comutarea între task-uri în timp de un ciclu-procesor și un timp de răspuns la evenimente de până la 1,5 cicluri-procesor.

Acest Capitol prezintă câteva rezultate ale cercetărilor efectuate pentru doctorat, prezentate şi publicate la conferinţele internaţionale. În prima parte, se propune un sistem de întrerupere pentru un planificator în timp real implementat în hardware, care elimină supratask-ul generată de sistemul de operare și operațiile de comutare de context.

Cercetarea prezentată în această lucrare se bazează pe o arhitectură funcțională de procesor cu regiştri multipipeline (MPRA), prezentată în [100] [122], care prevede un timp foarte redus pentru operaţiunile de comutare de context ca urmare a arhitecturii multipipeline. Arhitectura *Multi Pipeline Register* (*MPRA*) include planificatorul hardware. Salvarea de context a task-ului în arhitectura clasică de procesor implică salvarea registrelor procesorului în memorie sau pe stivă și se poate produce jitter, astfel timpul de răspuns al STR critice poate fi afectat. În procesorul *MPRA*, registrul *Program Counter* (*PC*), regiştrii pipeline și registrele generale sunt resurse multiplicate, iar memoria necesară pentru punerea în aplicare a acestor registre este direct proporțională cu numărul de task-uri din sistem. În scopul de a asigura predictibilitatea RTS, *MPRA* pune în aplicare un sistem de planificare rigidă, care se bazează pe codarea adresei de prioritate în *Hardware Scheduler Engine* (*HSE*).

Scopul acestui studiu este de a extinde în continuare soluția prezentată în [112] prin îmbunătățirea performanțelor și predictibilitatea sistemului de întreruperi și de tratare a evenimentelor. Noutatea prezentată în această lucrare se referă la implementarea hardware a mecanismului de selectare a evenimentelor folosind registre-capcană, la arhitectura registrului *PC* și introducerea instrucțiunii *ret_esr* (*return from the event service routine*), pentru a permite gestionarea automată a evenimentelor. Soluția pentru sistemul de întreruperi prezentată în această lucrare are un grad ridicat de flexibilitate și, spre deosebire de soluțiile software, furnizează același timp de răspuns pentru toate întreruperile și evenimentele. Soluția este aplicabilă pentru microcontrollere.

Acest subcapitol este structurat după cum urmează: **secțiunea 2** prezintă ***nMPRA***, **secțiunea 3** prezintă arhitectura ***nHSE*** și **tratarea întreruperilor**, în **secţiunea 4** este prezentată **tratarea hardware a evenimentelor, prioritizarea globală a evenimentelor** pe categorii, folosind un sistem de prioritizare hardware, **secțiunea 5** prezintă **analiza evenimentelor *syn* şi *mutex***, în **secţiunea 6** este prezentată **arhitectura PC_i** modificată, iar **secţiunea 7** conţine **concluzii**.

1.2. nMPRA

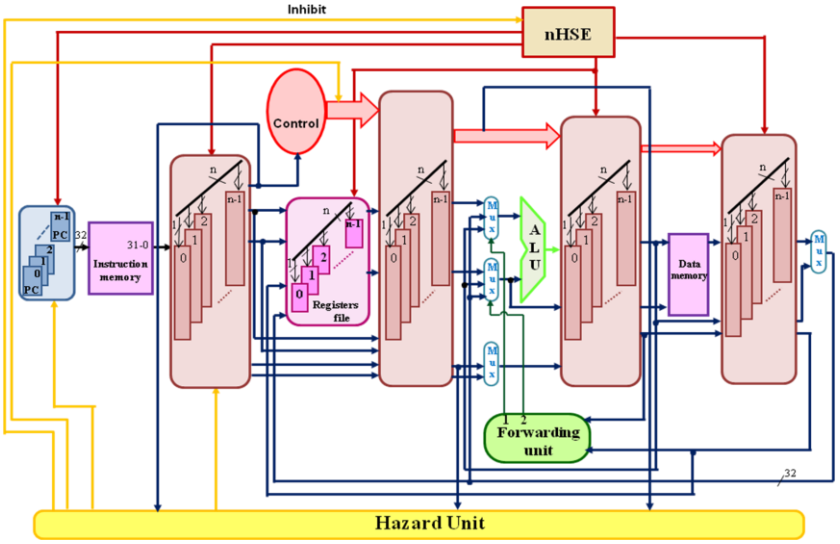


Figura 1 *nMPRA* (*sursa*: [114])

Arhitectura *nMPRA* este prezentată în Figura 3. *MPRA* a fost transformată în *nMPRA*, adică a fost multiplicată de *n* ori; pentru fiecare task avem câte un set de regiştrii pipeline (IFID, IDEX, EXMEM, MEMWB), un Program Counter (PC) şi un Banked Register File. Datorită faptului că fiecare task are propriul set de registre pipeline şi propriul set de registre generale comutarea de context se poate face într-un ciclu procesor, iar răspunsul la un eveniment extern poate fi întârziat maximum 1,5 cicli procesor. Din aceste motive arhitectura este foarte rapidă. Celelalte resurse sunt folosite în comun de către toate taskurile. O instanță a acestui procesor o vom numi „semi CPU" ($sCPU_i$) pentru task-ul *i* (*i=0, ..., n-1*) şi va executa un singur task ($task_i$).

Toate $sCPU_i$ sunt identice cu excepția $sCPU_0$ care va fi singura activă după reset, singura care va executa instrucțiuni supervizor și singura care va avea acces la *registrele de monitorizare* ale *nMPRA*. $sCPU_0$ are întotdeauna prioritatea 0 și este cea mai prioritară *sCPU*.

Planificatorul în *nMPRA* monitorizează constant toate evenimentele care se adresează la $sCPU_i$. Lista posibilelor evenimente așteptate constă în: evenimente timer (TEv_i), evenimente watchdog timer ($WDEv_i$), două evenimente deadline ($D1Ev_i$ și $D2Ev_i$, primul fiind echivalent cu o alarmă și al doilea fiind echivalent cu o avarie), întreruperi atașate task-ului *i* ($IntEv_i$), mutex-uri utilizate pentru tratarea resurselor partajate ($MutexEv_i$), evenimente de sincronizare și de comunicare între task-uri ($SynEv_i$) și semnalul de execuție

auto-susţinere pentru $sCPU_i$ curent ($lr\_runs\_CPU_i$). Ori de câte ori un eveniment este tratat şi sursa sa este ştearsă, actualul $sCPU_i$ poate pierde controlul asupra procesorului. Evenimentele enumerate mai sus pot fi validate cu ajutorul următoarelor semnale: lr_enT_i, lr_enWD_i, lr_enD1_i, lr_enD2_i, lr_enInt_i, $lr\_enMutex_i$ şi lr_enSyn_i, care sunt grupate într-un registru special, numit Task Register (TR_i). Singura excepţie este $lr\_run\_sCPU_i$. Semnalele rezultate lr_TEv_i, lr_WDEv_i, lr_D1Ev_i, lr_D2Ev_i, lr_IntEv_i, $lr\_MutexEv_i$, lr_SynEv_i şi $lr\_run\_sCPU_i$ sunt grupate într-un registru numit Event Status Task Register ($ESTR_i$), care poate fi accesat pentru a vedea ce evenimente aşteptate de task-ul *i* au apărut.

1.3. Arhitectura *nHSE* şi tratarea întreruperilor

În arhitectura *MPRA*, planificatorul hardware *HSE* [112] este inclus în procesor şi, prin urmare, nu necesită timp suplimentar pentru arbitrarea magistralelor, nici întârzieri ale rezultatelor din cauza transferului de date între planificator şi procesor şi poate fi controlat direct de instrucţiunile transmise pipeline. Pentru că avem mai multe registre pipeline, se poate provoca o izolare a contextelor hardware. *nHSE* (Figura 4) are la intrare evenimente (întreruperi, deadline, timer-e watchdog, timer-e, mutex-uri, evenimente mesaj, ca şi semnale de validare pentru planificatoarele statice şi dinamice şi inhibarea executării instrucţiunilor de încărcare şi de memorare) utilizate pentru a genera semnalele de activare a $sCPU_i$ [113].

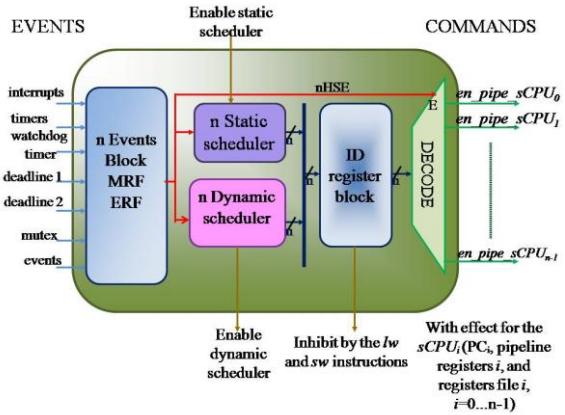


Figura 4 Arhitectura *nHSE* (*sursa*: [113])

Arhitectura *nHSE* este ilustrată în Figura 4. În soluţia propusă, întreruperile sunt evenimente care pot fi atribuite nucleului de timp real sau task-urilor. Nucleul trebuie să fie un monitor în timp real. După ce a intrat în monitor,

nu mai poate fi întrerupt (întreruperile sunt dezactivate). Ca urmare, funcțiile de monitorizare trebuie să fie scurte, fără interacțiuni și pauze. Întreruperile atașate unui task pot întrerupe numai task-uri cu prioritate strict mai mică, care sunt în stare de rulare.

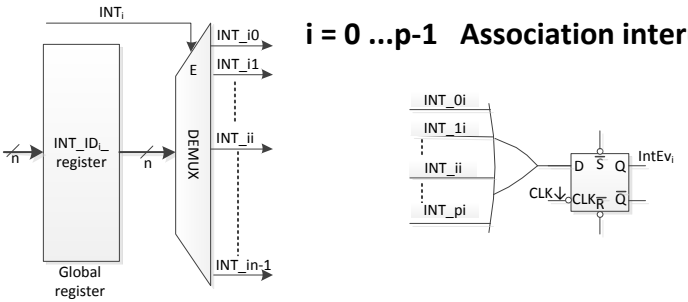


Figura 5 Asocierea întreruperilor la *sCPU$_i$* (task *i*) (*sursa*: [100])

Să presupunem că există *p* întreruperi în sistem. Pentru fiecare întrerupere a sistemului, există un registru global, numit *INT_ID$_i$_register*, cu *n* biți utili care memorează ID-ul task-ul asociat întreruperii. Întreruperea activată *INT$_i$* validează demultiplexorul *DEMUX* care, la rândul său, va activa unul dintre semnalele *INT_i$_0$* ... *INT_i$_{n-1}$*. Poarta *SAU* poate colecta toate întreruperile din sistem. Toate întreruperile pot fi atașate la *sCPU$_i$* dacă toți *p* regiștrii *INT_ID$_i$_register* (*i = 0, ..., p-1*) sunt setați cu valoarea *i*. De asemenea, nici o întrerupere nu poate fi atașată la *sCPU$_i$* dacă niciunul dintre cei *p* regiștrii *INT_ID$_i$_register* (*i = 0, ..., p-1*) nu este scris cu valoarea *i*. Bistabilul *D* sincronizează apariția aleatorie a evenimentelor, cum ar fi întreruperea *INT$_i$* produce evenimentul *IntEv$_i$* (Figura 5) și este înregistrată pe frontul descrescător al ceasului de sistem.

Presupunem că dispozitivele care generează întreruperi au un bit pentru a semnala starea de întrerupere, un bit de validare a întreruperii și un bit de ștergere a întreruperii. Analizând schema de întreruperi, am observat următoarea problemă: ce se întâmplă dacă toate întreruperile sunt atașate la același *sCPU$_i$* și au loc simultan? În această parte vom prezenta două soluții la această problemă:

1. Soluția software este prezentată în Figura 6. Este simplă (nu are nevoie de module hardware suplimentare) și versatilă deoarece prioritățile întreruperilor pot fi schimbate cu ușurință. Unul dintre dezavantaje este întârzierile introduse de blocurile de test și de rutinele de tratare a întreruperilor în cazul în care mai multe întreruperi sunt atașate aceluiași *sCPU$_i$* și au loc simultan. În plus, întârzierea generată de blocurile de test depinde de poziția întreruperii în blocul de test.

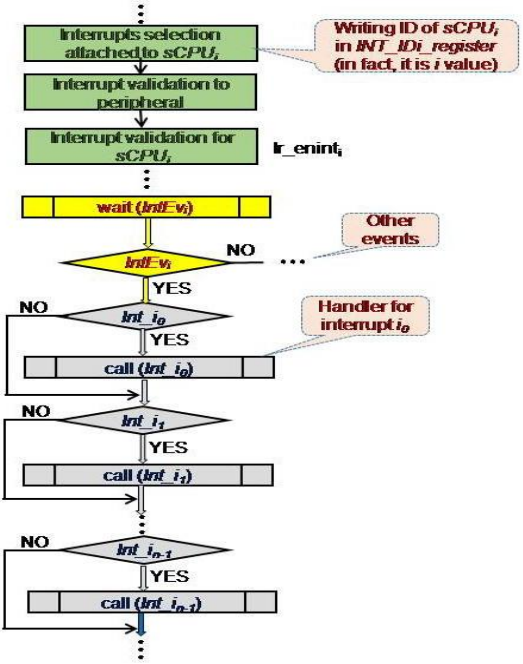


Figura 6 Soluția software (*sursa*: [113])

2. O altă soluție implică un bloc hardware suplimentar așa cum se arată în Figura 7.

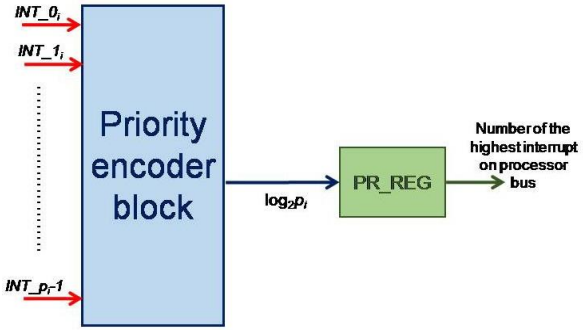


Figura 7 Blocul hardware adiţional (*sursa*: [113])

La apariția uneia sau a mai multor întreruperi, blocul codificator de prioritate va genera un număr corespunzător întreruperii de cea mai înaltă prioritate. Acest număr se înmulțește cu 4 pentru a calcula deplasamentul într-un

tabel cu celule-capcană pentru întreruperi (Figura 8). Se citeşte de acolo adresa handler-ului pentru întreruperi şi i se transferă acestuia controlul. De data aceasta, pentru fiecare întrerupere, întârzierea dată de blocul de decizie va fi aceeaşi. Soluţia este mai rapidă (Figura 9), dar necesită un bloc hardware suplimentar [113], a cărui complexitate este dată de numărul total de întreruperi din procesor; ca exemplu, se prezintă în Figura 10 tabela de adevăr, ecuaţiile de funcţionare şi codul VHDL pentru un codificator de prioritate pentru $p_i=4$ (a) şi pentru $p_i=8$ (b) (INT_0_i este cea mai mare întrerupere).

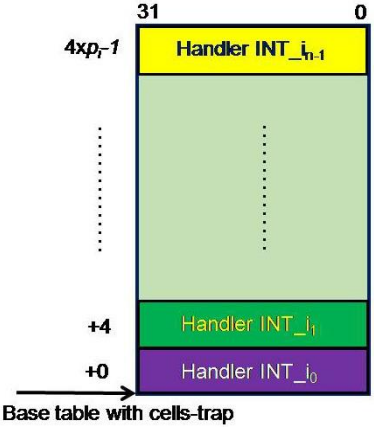


Figura 8 Tabel cu celule capcană (*sursa*: [113])

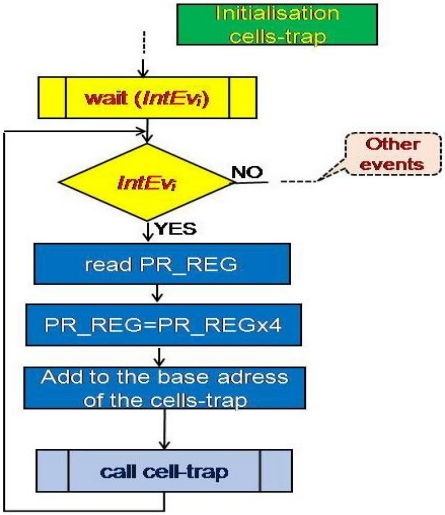
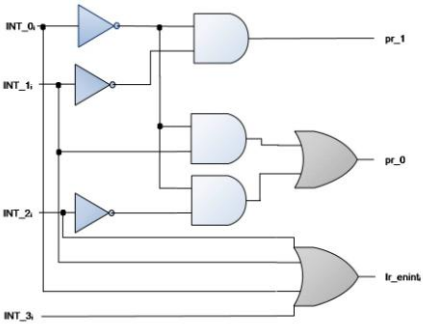


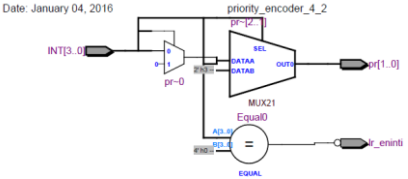
Figura 9 Soluţia hardware (*sursa*: [113])

INT_0_i	INT_1_i	INT_2_i	INT_3_i	pr_1	pr_0	lr_eninti_i
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

$$pr\_1 = INT\_0_i + INT\_1_i$$
$$pr\_0 = INT\_0_i + INT\_2_i \cdot \overline{INT\_1_i}$$
$$lr\_enint_i = INT\_0_i + INT\_1_i + INT\_2_i + INT\_3_i$$




```
1   LIBRARY IEEE;
2   USE IEEE.STD_logic_1164.ALL;
3   Entity priority_encoder_4_2 is
4   port (INT:IN std_logic_vector (3 downto 0);
5   pr:       OUT std_logic_vector (1 downto 0);
6       lr_eninti: OUT std_logic);
7   END priority_encoder_4_2;
8
9   ARCHITECTURE arch OF priority_encoder_4_2 IS BEGIN
10  pr <= "11" when INT(0)= '1' else
11  "10" when INT(1)= '1' else
12  "01" when INT(2)= '1' else
13    "00";
14  lr_eninti <= '0' when INT = "0000" else '1';
15  END arch;
```



(a)

INT_0_i	INT_1_i	INT_2_i	INT_3_i	INT_4_i	INT_5_i	INT_6_i	INT_7_i	pr_2	pr_1	pr_0	lr_eninti_i
0	0	0	0	0	0	0	0	x	x	x	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	x	0	0	1	1
0	0	0	0	0	1	x	x	0	1	0	1
0	0	0	0	1	x	x	x	0	1	1	1
0	0	0	1	x	x	x	x	1	0	0	1
0	0	1	x	x	x	x	x	1	0	1	1
0	1	x	x	x	x	x	x	1	1	0	1
1	x	x	x	x	x	x	x	1	1	1	1

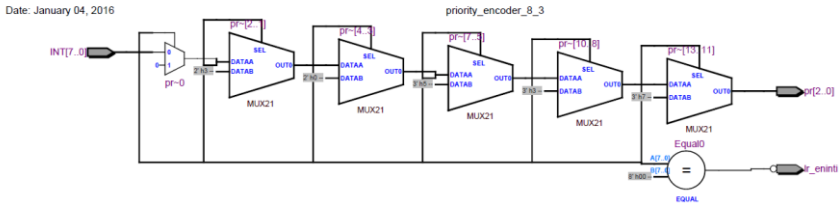
```
1   LIBRARY IEEE;
2   USE IEEE.STD_logic_1164.ALL;
3   Entity priority_encoder_8_3 is
4   port (INT:IN std_logic_vector (7 downto 0);
5   pr:       OUT std_logic_vector (2 downto 0);
6       lr_eninti: OUT std_logic);
7   END priority_encoder_8_3;
8
9   ARCHITECTURE arch OF priority_encoder_8_3 IS BEGIN
10  pr <= "111" when INT(0)= '1' else
11  "110" when INT(1)= '1' else
12  "101" when INT(2)= '1' else
13  "100" when INT(3)= '1' else
14  "011" when INT(4)= '1' else
15  "010" when INT(5)= '1' else
16  "001" when INT(6)= '1' else
17     "000";
18  lr_eninti <= '0' when INT = "00000000" else '1';
19  END arch;
```

$$lr\_enint_i = INT\_0_i + INT\_1_i + INT\_2_i + INT\_3_i + INT\_4_i + INT\_5_i + INT\_6_i + INT\_7_i$$

$$pr\_0 = \overline{INT\_0_i} \cdot \overline{INT\_1_i} \cdot \overline{INT\_2_i} \cdot \overline{INT\_3_i} \cdot \overline{INT\_4_i} \cdot \overline{INT\_5_i} \cdot \overline{INT\_6_i} \cdot INT\_7_i +$$
$$+ \overline{INT\_0_i} \cdot \overline{INT\_1_i} \cdot \overline{INT\_2_i} \cdot \overline{INT\_3_i} \cdot INT\_4_i + \overline{INT\_0_i} \cdot \overline{INT\_1_i} \cdot INT\_2_i + INT\_0_i$$

$$pr\_1 = \overline{INT\_0_i} \cdot \overline{INT\_1_i} \cdot \overline{INT\_2_i} \cdot \overline{INT\_3_i} \cdot \overline{INT\_4_i} \cdot INT\_5_i$$
$$+ \overline{INT\_0_i} \cdot \overline{INT\_1_i} \cdot \overline{INT\_2_i} \cdot \overline{INT\_3_i} \cdot INT\_4_i + \overline{INT\_0_i} \cdot INT\_1_i + INT\_0_i$$

$$pr\_2 = \overline{INT\_0_i} \cdot \overline{INT\_1_i} \cdot \overline{INT\_2_i} \cdot INT\_3_i$$
$$+ \overline{INT\_0_i} \cdot \overline{INT\_1_i} \cdot INT\_2_i + \overline{INT\_0_i} \cdot INT\_1_i + INT\_0_i$$



(b)

Figura 10 Codificator de prioritate (a) $p_i = 4$, (b) $p_i = 8$

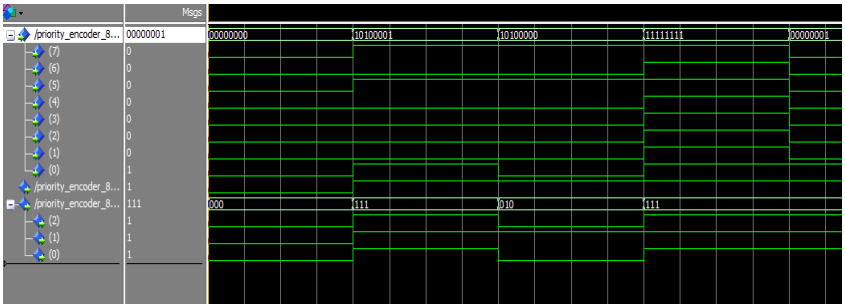


Figura 11 Forme de undă pe ModelSim cu funcţionarea codificatorului de prioritate 8:3

În arhitectura *MPRA* utilizatorul are opţiunea de a selecta modul în care întreruperile sunt tratate: externe task-urilor (situaţie în care întreruperile sunt văzute ca şi task-uri) sau ataşate acestora.

1.4. Tratarea hardware a evenimentelor

În caz de activare simultană a mai multor evenimente asociate cu un task ($sCPU_i$), trebuie să existe o modalitate de a selecta ordinea în care evenimentele sunt tratate. Pentru aceasta, fiecare $sCPU_i$ are ataşat un *Event Priority Register* (EPR_i), care conţine nivelul de prioritate al fiecărui tip de eveniment asociat cu acel $sCPU_i$. Prioritatea este diferită pentru fiecare tip de evenimente, de la 0 la 7 (există 8 tipuri de evenimente). După resetare, la pornire, *nMPRA* activează $sCPU_0$, care va executa tot software-ul de iniţializare şi secvenţele de pornire pentru toate celelalte $sCPU_i$ ($i = 1 \ldots n-1$). $sCPU_0$ stabileşte nivelul de prioritate pentru fiecare tip de eveniment asociat cu fiecare $sCPU_i$ conform cerinţelor utilizatorului.

Figura 12 prezintă o schemă de prioritizare care selectează tipul de eveniment activ curent cu cea mai mare prioritate, în scopul de a fi servit [114]. În funcţie de cerinţele utilizatorului, nivelul de prioritate al fiecărui tip de eveniment poate fi static şi stabilit la începutul aplicaţiei (offline) sau poate fi

modificat dinamic în timpul executării aplicației (online). Dacă tipul selectat de eveniment conține mai multe evenimente active, trebuie să facem o altă selecție a evenimentului care va fi tratat în primul rând, în funcție de tipul evenimentului. Având în vedere că în *nMPRA* avem opt tipuri de evenimente, vom avea nevoie de opt scheme de decodare și de selectare a evenimentelor (Figura 12a). Prioritățile categoriilor de evenimente sunt grupate în registrul EPR_i (*Event Priority Register*) ce poate stoca prioritățile următoarelor tipuri de evenimente [110]: Pri_TEv_i, Pri_WDEv_i, Pri_D1Ev_i, Pri_D2Ev_i, Pri_IntEv_i, $Pri\_MutexEv_i$, Pri_SynEv_i, și Pri_RunEv_i. Semnalul de activare a fiecărui tip de eveniment activează un decodor care generează prioritatea tipului de eveniment conform nivelul de prioritate memorat în EPR_i. Ieșirea câmpului de prioritate este utilizată și pentru selectarea intrării active a multiplexoarelor *MUX*, care colectează rezultatul schemei de prioritizare prezentată în Figura 12b. Porțile *SAU* permit fiecărui tip de evenimente de a-și selecta prioritatea, porțile *ȘI* validează o anumită prioritate, iar bistabilul *D* are rolul de a realiza sincronizarea cu ceasul sistemului.

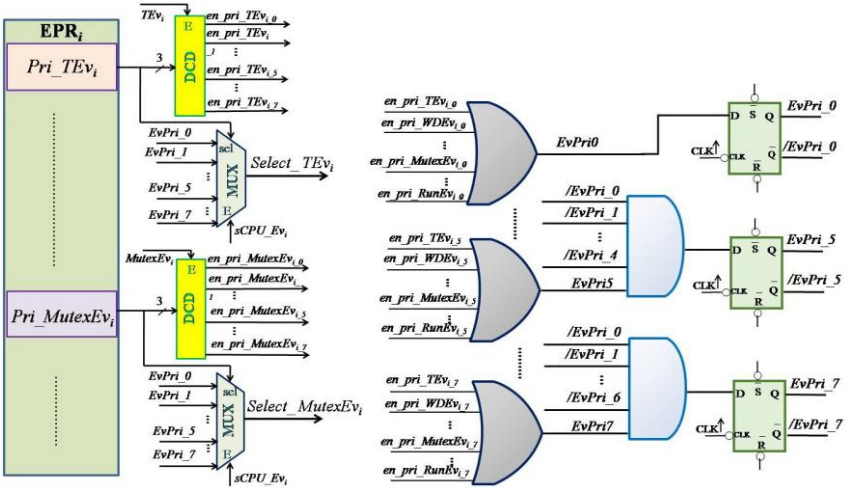


Figura 12 Schema globală de prioritizare a evenimentelor (*sursa*: [114])

RTL-ul pentru schema parțială de prioritizare globală a evenimentelor este prezentat în Anexa 5.

Atunci când mai multe evenimente atribuite unui task ($sCPU_i$) sunt activate simultan, trebuie să găsim o metodă de a selecta ordinea de tratare (Figura 13, 14). După ce am selectat această ordine se obține adresa rutinei de serviciu pentru eveniment asociată cu evenimentul selectat (rutina de serviciu pentru eveniment este executată de task-ul $sCPU_i$).

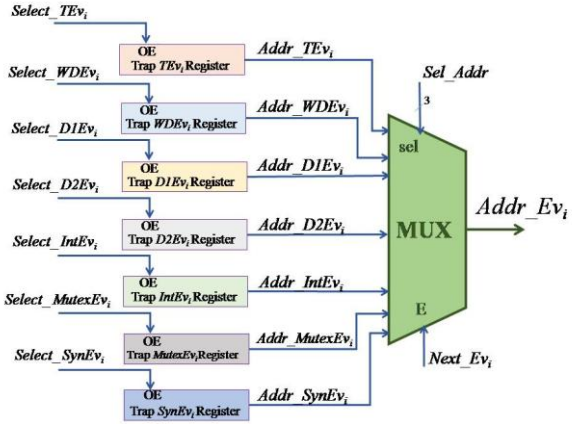


Figura 13 Selecţia adresei evenimentului de cea mai înaltă prioritate (*sursa*: [114]

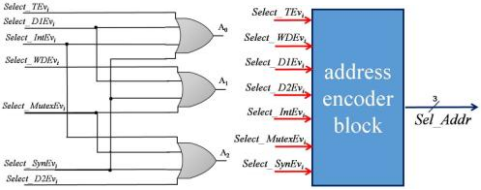
Select_TEv_i	Select_WDEv_i	Select_D1Ev_i	Select_D2Ev_i	Select_IntEv_i	Select_MutexEv_i	Select_SynEv_i	A_2	A_1	A_0	
1	0	0	0	0	0	0	0	0	1	1
0	1	0	0	0	0	0	0	1	0	2
0	0	1	0	0	0	0	0	1	1	3
0	0	0	1	0	0	0	1	0	0	4
0	0	0	0	1	0	0	1	0	1	5
0	0	0	0	0	1	0	1	1	0	6
0	0	0	0	0	0	1	1	1	1	7

$A_0 = Select\_TEv_i + Select\_D1Ev_i + Select\_IntEv_i + Select\_SynEv_i$

$A_1 = Select\_WDEv_i + Select\_D1Ev_i + Select\_MutexEv_i + Select\_SynEv_i$

$A_2 = Select\_D2Ev_i + Select\_IntEv_i + Select\_MutexEv_i + Select\_SynEv_i$

```
1   LIBRARY IEEE;
2   USE IEEE.STD_logic_1164.ALL;
3   Entity encoder_8_3 is
4   port (Sel:IN std_logic_vector (7 downto 0);
5   A:       OUT std_logic_vector (2 downto 0));
6      END encoder_8_3;
7
8   ARCHITECTURE arch OF encoder_8_3 IS BEGIN
9   A <= "000" when Sel(0)= '1' else
10  "001" when Sel(1)= '1' else
11  "010" when Sel(2)= '1' else
12  "011" when Sel(3)= '1' else
13  "100" when Sel(4)= '1' else
14  "101" when Sel(5)= '1' else
15  "110" when Sel(6)= '1' else
16     "111";
17  END arch;
```



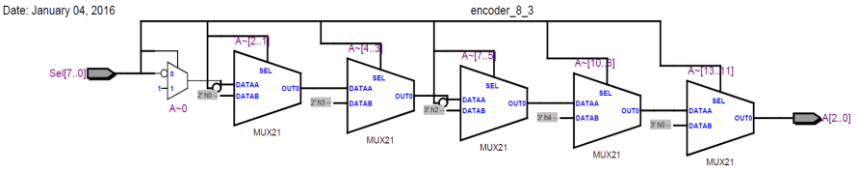


Figura 14 Blocul codificator de selecţie a adresei semnalelor

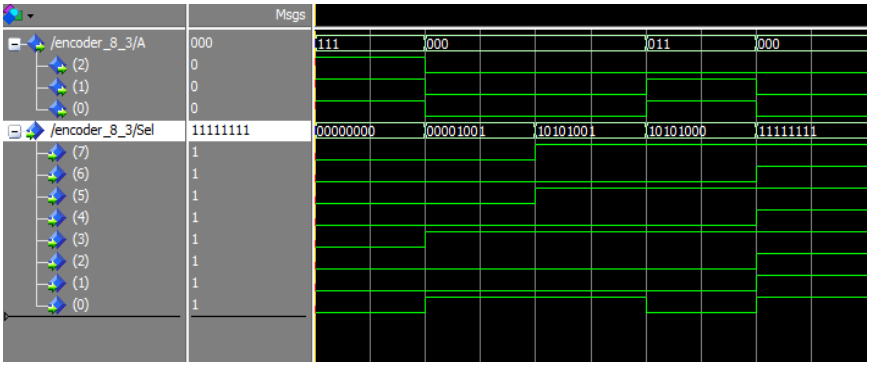


Figura 15 Forme de undă cu funcţionarea codificatorului 8:3

Toate evenimentele care sunt singurele din categoria lor, cum ar fi cele de timp (TEv_i, $WDEv_i$, $D1Ev_i$ and $D2Ev_i$) și cele care sunt tratate la nivel global prin intermediul software-ului, au asociat un registru-capcană, care indică spre rutina lor de service a evenimentului. Adresele de rutină de serviciu a evenimentelor sunt încărcate pentru fiecare task într-un registru-vector de eveniment (Figura 16) la pornire, prin $sCPU_0$, după resetare.

address of the *Timer* event servicing routine	Trap TEv_i Register
address of the *WatchDog* event servicing routine	Trap $WDEv_i$ Register
address of the *Deadline1* event servicing routine	Trap $D1Ev_i$ Register
address of the *Deadline2* event servicing routine	Trap $D2Ev_i$ Register
base address of the trap-cells table for interrupts	Trap $IntEv_i$ Register
address of the *Mutex* events servicing routine	Trap $MutexEv_i$ Register
address of the *Syn* events servicing routine	Trap $SynEv_i$ Register

Figura 16 Registru-vector de eveniment (*sursa*: [114])

În cazul în care unul dintre aceste evenimente a avut loc, task-ul cu care acesta este asociat devine activ și are cea mai mare prioritate printre evenimentele active, registrul Program Counter corespunzător $sCPU_i$ (PC_i) este încărcat automat cu conținutul registrului-pointer, determinând executarea

rutinei asociate cu evenimentul. Adresa de întoarcere, care a fost valoarea din registrul PC_i înainte de încărcarea adresei de rutină, se salvează automat într-un registru de rezervă în interiorul PC_i. După terminarea execuţiei rutinei de serviciu a evenimentului, dacă există alte evenimente active pentru task-ul *i*, adresa de rutină pentru evenimentul cu cea mai mare prioritate printre cele rămase va fi încărcată automat în PC_i. După servirea tuturor evenimentelor active, PC_i este încărcat cu adresa de revenire din registrul „backup”.

Procedura descrisă mai sus este valabilă pentru fiecare eveniment din sistem. În secțiunea următoare vom analiza cazul evenimentelor *syn* și *mutex*, care pot fi mai mult de unul din categoria lor.

1.5. Analiza evenimentelor *SYN* şi *MUTEX*

Evenimentele de sincronizare și de comunicare între $sCPU_i$ ($SynEv_i$) nu pot fi tratate individual, din cauza metodei alese pentru punerea lor în aplicare, care utilizează un set de registre globale cu acces rapid.

Registrul *ERF* (Event Register File) este format din *s* registrer de $2n+k+1$ biți, memorând evenimentul însuși pe cel mai semnificativ bit. Următorii *n* biți reţin ID-ul task-ului sursă care a activat evenimentul, următoarii *n* biți reţin ID-ul task-ului destinație pentru căruia i se adresează evenimentul, iar ultimii *k* biți sunt utilizaţi ca un domeniu mesaj care poate fi utilizat pentru orice scop.

În *nMPRA*, *mutexes* sunt implementate cu ajutorul unui set de registre globale cu acces rapid (se poate face în faza de execuție - EX) [100]. Registrul *MRF* (*Mutex Register File*) este compus din *m* registre de lungime $n + 1$ biți care conțin bit-ul *mutex* în poziția cea mai înaltă și ID-ul task-ului proprietar, care deține *mutex* în cea mai mică prioritate a celor *n* biți.

1.6. Arhitectura *Program Counter*

Atunci când are loc un eveniment, task-ul asociat devine gata pentru execuție și, dacă are o prioritate mai mare decât task-ul în starea de execuție, atunci registrul *Program Counter* corespunzător $sCPU_i$ (PC_i) (vezi Figura 17) este setat cu conținutul registrului-capcană (adresa rutinei asociate pentru evenimentul produs). Pentru a putea implementa redirectarea automată către rutinele de tratare a evenimentelor, PC_i trebuie modificat astfel încât să salveze intern adresa de revenire dintr-o rutină de tratare a unui eveniment [114].

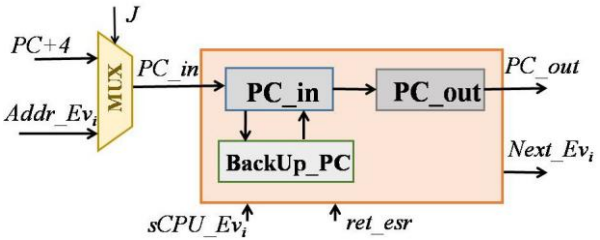


Figura 17 Arhitectura *Program Counter* (*sursa*: [114])

În interiorul PC_i vom avea un registru, numit *BackUp_PC*, în care se salvează adresa curentă din PC_i în momentul apariției unui eveniment ce trebuie tratat, semnalizat de semnalul $sCPU\_Ev_i$. În PC_i se va încărca automat, folosind registrul capcană corespunzător, adresa rutinei de tratare a evenimentului activ cel mai prioritar. Revenirea din rutina de tratare a evenimentului este semnalizată prin execuţia instrucţiunii *retesr* (*return from the event service routine*), care detemină activarea semnalului *ret_esr* ce semnalizează PC_i să încarce adresa de revenire din registrul *BackUp_PC*, continuând astfel execuția normală a programului. Salvarea adresei de revenire în registrul *BackUp_PC* determină dezactivarea semnalului $NextEv_i$, indicând astfel că se tratează un eveniment şi niciun alt eveniment nu mai poate fi tratat până când nu se termină tratarea evenimentului curent. După revenirea din rutina de tratare a evenimentului curent, semnalul $NextEv_i$ va fi reactivat pentru a semnala că se poate trece la tratarea evenimentului următor [114]. În Figura 17 este prezentată structura PC_i.

1.7. Concluzii

Prin cercetările prezentate în această lucrare am îmbunătățit arhitectura CPU prezentată în [100] printr-o soluție inovatoare pentru prioritizarea întreruperilor atașate la aceeași task. Am propus o soluție inovativă de atașare a întreruperilor la task-uri (*sCPU*). În acest context, întreruperea împrumută prioritatea şi comportamentul task-ului. Astfel, comportamentul întreruperilor este mult mai predictibil în contextul unei aplicații de timp real (un task nu poate fi întrerupt decât de întreruperile atașate unui task mai prioritar). Schema propusă are unele caracteristici puternice şi interesante, cum ar fi: nu trebuie să existe un controler specializat pentru întreruperi; întreruperile moştenesc prioritatea task-ului (*sCPU*); un task poate ataşa niciuna, una, mai multe sau chiar toate cele *p* întreruperi din sistem; întreruperea ataşată unui task poate întrerupe un task mai puţin prioritar dar nu poate întrerupe execuţia taskului căruia îi este ataşată, sau un task mai prioritar; întreruperea poate fi ataşată unui singur task; întreruperea poate fi un task; toate întreruperile pot fi ataşate unui singur task; întreruperea nu resetează banda de asamblare a altor $sCPU_i$; nu implică salvări şi restaurări de context; întreruperile pot fi nested; priorităţile

întreruperilor pot fi dinamice (prin reatașarea la un task sau prin schimbarea priorității taskului la care este atașată). Spre deosebire de soluția software, prin hardware, orice întrerupere are același timp de răspuns. Mai mult decât atât, soluția propusă poate oferi priorități statice pentru întreruperi. Putem spune că soluția prezentată conține un management unitar al întreruperilor și o soluție hardware pentru a atașa întreruperile la task-urile sistemelor de timp-real implementate în hardware.

Schema de prioritizare globală [113] termină implementarea schemei de tratare hardware a evenimentelor [114] pentru arhitectura *nMPRA*. Se prezintă registrele-capcană asociate categoriilor de evenimente și structura registrului *Program Counter* (PC_i) [114]. Pentru sincronizarea și comunicația între taskuri s-a prevăzut un set de evenimente, implementat sub forma unui ERF, de asemenea ca resursă globală pentru toate $sCPU_i$. Accesul pentru setarea unui eveniment este fără adresă (nu necesită căutări în tabele pentru a depista un loc liber). Hardware-ul depistează automat prima adresă liberă și setează evenimentul și informația atașată acestuia, sau sesizează și semnalizează că nu există nici un eveniment liber. Task-ul destinație nu trebuie decât să citească din ERF pentru a afla dacă un eveniment i se adresează sau nu. Achitarea evenimentului se face automat. Schema de prioritizare prezentată permite tratarea hardware a evenimentelor. Avantajul este reducerea timpului de detectare a sursei evenimentului și de începere a rutinei corespunzătoare de service a evenimentelor. Chiar se permite introducerea de noi evenimente în sistem prin simpla adăugare a câmpurilor necesare în *Task Register* (TR_i), *Event Status Task Register* ($ESTR_i$) și *Event Priority Register* (EPR_i), actualizarea schemei de prioritizare globală a evenimentelor și introducerea unui nou registru-capcană pentru fiecare categorie nouă de eveniment. Schema este simplă și poate fi aplicată tuturor evenimentelor [114].

2. IMPLEMENTAREA ETAJELOR PIPELINE PE FPGA

2.1. Introducere

Acest capitol conține o descriere a Cyclone V de la Altera, apoi se prezintă implementarea etajelor pipeline a microcontrolerelor din studiu, 4MPRA și 8MPRA, analiza și consumul de resurse, schemele logice furnizate de Quartus II pe Cyclone V - 5CGTFD9E5F35C7 și câteva rezultate ale simulării.

2.2. *nMPRA* implementată pe FPGA – QuartusII, Cyclone V

2.2.1. Cyclone V de la Altera

Cyclone V oferă cea mai mare flexibilitate pentru implementarea protocoalelor la cel mai mic consum de energie posibil [117].

Figura 18 Cyclone V - 5CGTFD9E5F35C7 de la ALTERA

2.2.2. Implementarea nMPRA pe Cyclone V

nMPRA constă dintr-o structură hardware originală utilizată pentru planificarea task-ului, static și dinamic și oferă managementul unitar al evenimentelor. Scopul a fost de a îmbunătăți prin hardware performanțele RTOS pentru microcontrolere, pentru a comuta mai rapid între task-uri, pentru a îmbunătăți timpul de răspuns la evenimente externe, pentru a îmbunătăți comportamentul întreruperilor, care sunt tratate ca evenimente în acest caz și pentru a oferi mai multe tipuri de bază de comunicare inter-task-uri (mesaje, mutex-uri etc.).

Arhitectura *nMPRA* este prezentată în Figura 19. MPRA a fost transformată în *nMPRA*, adică a fost multiplicată de *n* ori; pentru fiecare task avem câte un set de regiștrii pipeline (IFID, IDEX, EXMEM, MEMWB), câte un Program Counter (PC) și câte un Banked Register File. Datorită faptului că fiecare task are propriul set de registre pipeline și propriul set de registre generale comutarea de context se poate face într-un ciclu procesor, iar răspunsul la un eveniment extern poate fi întârziat maximum 1,5 cicli procesor. Din aceste motive arhitectura este foarte rapidă. Celelalte resurse sunt folosite în comun de către toate taskurile. O instanță a acestui procesor o vom numi „semi CPU” ($sCPU_i$) pentru task-ul *i* (*i=0, ..., n-1*) și va executa un singur task ($task_i$).

Toate $sCPU_i$ sunt identice cu excepția $sCPU_0$ care va fi singura activă după reset, singura care va executa instrucțiuni supervizor și singura care va avea acces la *registrele de monitorizare* ale *nMPRA*. $sCPU_0$ are întotdeauna prioritatea 0 și este cea mai prioritară *sCPU* [98].

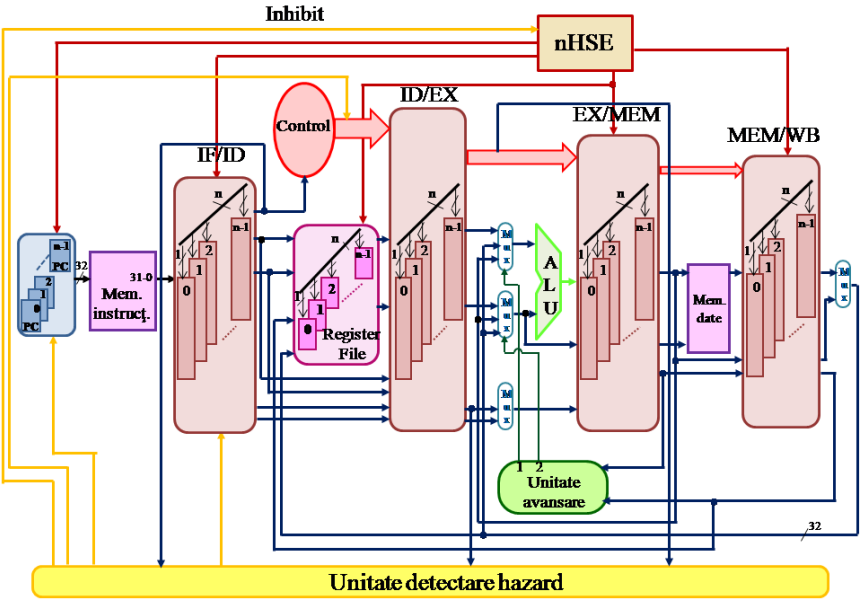


Figura 19 ***nMPRA*** (*sursa*: [115])

Arhitectura propusă constituie o metodă eficientă de optimizare a procesorului, având numeroase avantaje: reacția la un eveniment nu depășește 1,5 cicli-procesor dacă evenimentul apărut este atașat unui task mai prioritar decât taskul curent; nu resetează banda de asamblare, nu necesită salvare/restaurare de context, accelerează execuția prin apeluri de subrutine cu copierea automată a parametrilor și comutarea setului de regiștrii; stivă locală de mare viteză; instrucțiuni puternice pentru partajarea resurselor, sincronizarea și comunicația între taskuri.

2.3. Etajele și regiștrii pipeline

MPRA utilizează o linie de asamblare cu 5 etaje pentru îmbunătățirea performanțelor de calcul. Regiștrii pipeline pe care arhitectura MPRA îi folosește sunt: **IFID, IDEX, EXMEM, MEMWB**. La acestea patru se adaugă și **PC** care nu este considerat un registru de pipeline dar este gestionat de către HSE în aceeași manieră ca și regiștrii pipeline. Scopul unui registru pipeline este de a captura date de la o etapă pipeline și de a le furniza următoarei etape pipeline. Acest lucru creează cel puțin un ciclu de ceas întârziere, dar reduce lungimea pentru calea combinatorie a semnalelor, ceea ce permite viteze mai mari de ceas.

2.4. Analiza şi sinteza pe FPGA a arhitecturilor MPRA, 4MPRA şi 8MPRA

Pentru implementare şi analiză s-a utilizat codul verilog pentru procesorul *MIPS32r1 latest* [122].

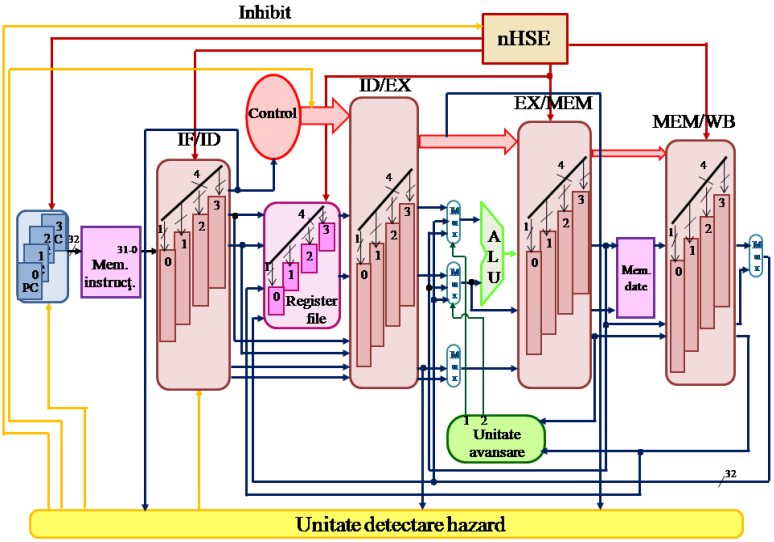


Figura 20 *4MPRA*

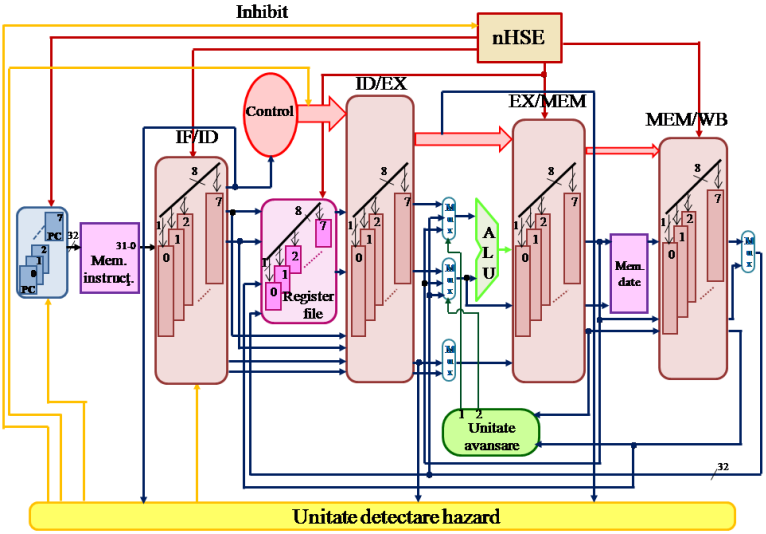


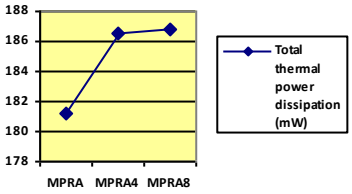
Figura 21 *8MPRA*

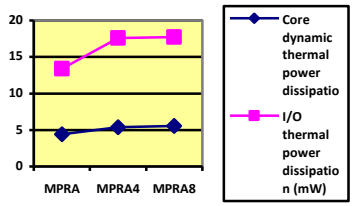
RTL-urile obţinute sunt exemplificate în Anexa 1, respectiv 2 şi 3.

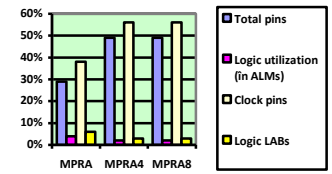
Codurile Verilog pentru regiştrii pipeline aferenţi arhitecturilor 4MPRA şi 8MPRA sunt prezentate în Anexa 4.

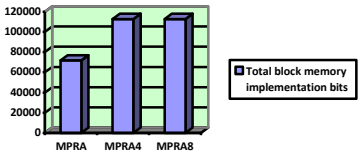
Tabel 1 Analiză implementare MPRA, MPRA4, MPRA8

Cyclone V 5CGTFD9E5F35C7	**MPRA**	**MPRA4**	**MPRA8**
Total pins	181 (29%)	303 (49%)	303 (49%)
Total registers design implementation	5161	5243	5250
Logic utilization (în ALMs)	4818 (4%)	2580 (2%)	2562 (2%)
Clock pins	6 (38%)	9 (56%)	9 (56%)
Global clocks	4 (25%)	4 (25%)	4 (25%)
Total block memory implementation bits	71680	112640	112640
M10K blocks	7	11	11
Maximum fan-out	2089	3330	3330
Highest non-global fan-out	1794	1273	1276
Total fan-out	41290	22236	22288
Logic LABs	638 (6%)	335 (3%)	334 (3%)
Combinational ALUT usage for logic	4353	735	736
Combinational ALUT usage for route-throughs	2240	3117	3136
Delay added I/O (ns)	193,3	197,3	181,3
Total thermal power dissipation (mW)	181,26	186,48	186,74
Core dynamic thermal power dissipation (mW)	4,42	5,38	5,55
I/O thermal power dissipation (mW)	13,40	17,61	17,71









2.5. Concluzii

Aplicațiile actuale necesită procesoare de mare viteză pentru transmiterea cantităților mari de date, în timp real. În comparație cu alternativele software, aplicaţiile hardware oferă algoritmi extrem de siguri şi soluții rapide pentru înaltă performanță.

Comparația detaliată a soluțiilor hardware vs. software pentru implementarea arhitecturii de procesor MPRA este prezentată în Tabelul 2. Pe baza comparației, soluția hardware este o alegere mai bună în cele mai multe cazuri, deoarece are înaltă performanță. Principalul avantaj al FPGA în alternative hardware este că FPGA-urile au densitate scăzută şi consum redus de zonă. [124]

Integrarea logică, dimensiunea și densitatea sunt principalele dezavantaje în ASIC.

Tabel 2 Hardware vs. software pentru implementarea procesorului

Parametrii	**Software**	**Hardware**	
		FPGA	**ASIC**
Performanţă	Scăzută	Medie-mare	Foarte mare
Consum putere	Depinde	Foarte mare	Scăzută
Integrare logică	Scăzută	Scăzută	Mare
Cost utilitare	Scăzut	Scăzut	Scăzut
Complexitate test	Foarte scăzut	Foarte scăzut	Mare
Densitate	Mare	Foarte scăzută	Mare
Efortul pentru design	Scăzut-mediu	Scăzut-mediu	Mare
Consum de timp	Scurt	Scurt	Mare
Mărime	Mică-medie	Mică	Mare
Memorie	Bună	Bună	Bună
Flexibilitate	Mare	Mare	-
Timp pe piaţă	Scurt	Scurt	Mare
Timp configurare	-	Mare	-

✓ Un mare avantaj al *nMPRA* este **implementarea hardware**, care elimină dezavantajele implementării HW/SW.

✓ O implementare a arhitecturii MPRA ce funcționează la 50 MHz este capabilă să realizeze o **comutare a contextelor într-un interval de timp cuprins între 20 și 60ns (1-3 cicli-procesor)**. Răspunsul la un eveniment extern este cuprins în același interval.

✓ **Viteza foarte mare de comutare de context a task-urilor este întâlnită doar la *nMPRA***.

Viteza de comutare a task-urilor la arhitecturi real-time:

- ***nMPRA***: 1-3 cicli-procesor
- ***hthreat***: 525-990 cicli-procesor [91]
- ***ARPA-MT***: 72 cicli-procesor [70]
- ***Kuacharoen***: 125 cicli-procesor [62]

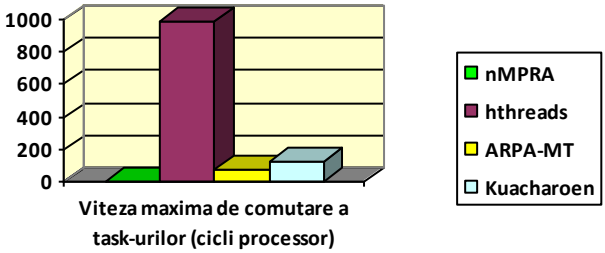


Figura 22 Viteza de comutare a task-urilor la arhitecturi STRO

Mai facem o comparaţie care se bazează pe criterii de replicare de resurse și implementarea hardware sau hardware-software.

✓ Deși poate părea o soluție simplă, *nMPRA* este singura unitate care **asigură replicarea regiştrilor pipeline cu efecte funcționale asupra vitezei de comutare a task-ului**. Doar arhitectura Arpa-MT [70] realizează replicarea IF și ID pentru fiecare task. Cu toate acestea, aceasta intră în competiție pentru ultimele trei etape pipeline – EXE, MEM și WB. Deși soluția de replicare a fișierului care conține registrele este o soluție hardware scumpă, este, de asemenea, un nou factor de convergenţă spre a ajunge la o viteză de comutare între 1 și 3 cicluri mașină, realitate care poate fi întâlnit doar la *nMPRA*.

✓ În același timp, *nMPRA* are o **implementare hardware simplă**, spre deosebire de celelalte implementări care sunt o combinație de hardware-software, ceea ce conduce prin deducție la un interval de timp mai lung de comutare [100].

Tabel 3 Procesoare pentru STRO (*sursa*: [100])

Facilităţi	***nMPRA***	***hthreads*** [91]	***ARPA-MT*** [70]	***Kuacharoen*** [62]
Replicare regiştrii generali	da	nu	nu	nu
Replicare regiştrii pipeline	da	nu	IF şi ID	nu
Viteză comutare task	1-3 cicli-procesor	525-900 cicli-procesor (300	72 cicli-procesor (24	125 cicli-procesor

		MHz)	MHz)	
Coprocesoare	nu	da	da	Da, cuplat pe bus extern
Implementare	HW	HW/SW	HW/SW	HW/SW

O altă comparație între aceleași arhitecturi pentru STRO se bazează pe caracteristicile planificatorului. Soluțiile comparate se referă la planificatoarele statice și dinamice implementate hardware, cu excepția *hthreads*.

✓ *nMPRA* este singura care **are planificatorul implementat la nivel de procesor** și posibilitatea de a aduna informații de planificare direct de la hardware, fără a afecta o posibilă comunicare *bus*-ului așa cum este cazul cu alte soluții luate în considerare. Toate soluțiile propun algoritmi de planificare statică.

✓ Deoarece **se trece de la un task la altul cu o întârziere de 1-3 cicluri de mașină**, putem admite că ***nMPRA* este cea mai rapidă**.

În ceea ce privește punerea în aplicare a algoritmilor dinamici de planificare, ARPA-MT [70] și Kuacharoen *et al.* [62] au soluții mai avansate, implementate hardware. *nMPRA* va implementa hardware planificatorul dinamic în perioada următoare.

Altă comparație se referă la sistemul de întreruperi și este prezentată în Tabelul 4. Conceptul de întrerupere de fire se găsește în [92] și [62]. În ceea ce privește ARPA-MT [70], nu există nici o referire explicită la managementul întreruperilor. În [92], controlerul de întreruperi este descris pe scurt, indicând faptul că acesta controlează opt niveluri de întreruperi. Fiecare întrerupere pot fi asociate cu un task, delegat să gestioneze nivelurile asociate de întreruperi. Fiecare întrerupere poate fi configurată ca o întrerupere rapidă (tratarea întreruperii suspendă activitatea task-ului curent) sau ca o întrerupere lentă (tratarea task-ului este inserată la capătul liniei de prioritate). Spre deosebire de alte implementări, sistemul de întreruperi al *nMPRA* este complet alocat, astfel încât o întrerupere poate fi atașată doar la un singur task, în timp ce un task poate avea atașate mai multe întreruperi (chiar toate).

✓ Un avantaj major al nMPRA este că **întreruperile sunt tratate într-un mod unitar, la fel ca și celelalte evenimente** din sistem.

Tabel 4 Controler pentru întreruperi (*sursa*: [100])

Facilităţi	***nMPRA***	***hthreads*** [91]	***ARPA-MT*** [70]	***Kuacharoen*** [62]
Controler?	distribuit	specializat	specializat	specializat
Întreruperi ca threads?	da	nu	-	nu

Întreruperi ataşate task-ului?	da	da	-	da
Întreruperile iau prioritatea task-ului?	da	da, dar un task poate avea ataşată o singură întrerupere	-	da
Întreruperile afectează pipeline-ul?	nu	da	-	da
Întreruperile cer salvare de context software?	nu	da	-	da

Ultima comparație între aceleași arhitecturi de procesor pentru STRO se referă la implementarea sincronizării și comunicării între task-uri.

✓ *nMPRA* este singura arhitectură, printre cele deja prezentate, care **implementează în hardware sincronizarea şi comunicarea** task-urilor. În cazul în care are loc un eveniment și dacă acesta este asociat cu un task de o prioritate mai mare decât task-ul curent, atunci comutatarea de task apare cu o întârziere de 1-3 cicluri. *nMPRA* poate sincroniza simultan cu toate evenimentele acceptate, folosind doar o singură instrucţiune *wait*.

✓ La o frecvenţă de 50MHz, replicarea registrelor pipeline la ***n=8*** **cere 0,59 kB de RAM**, **memoria necesară registrelor generale este de 256 B**, deci **memoria totală cerută de replicarea resurselor *n=8* este de 0,84 kB**, ceea ce este mai mult decât acceptabil ţinând seama că sunt arhitecturi de microcontroler care folosesc pentru uzul general 2,6 MB RAM, fără a mai lua în considerare memoria necesară la implementarea procesorului şi a altor componente hardware integrate pe chip.

✓ Deşi *nMPRA* este o arhitectură cu resurse partajate, totuşi **costul este mult mai eficient** decât al altor arhitecturi comerciale.

CAPITOLUL 4

1. CONCLUZII FINALE

Pentru a ajunge la obiectivul studiului, s-a analizat mai întâi stadiul actual privind implementarea sistemelor de operare utilizând structuri hardware. S-a efectuat un studiu amplu asupra teoriei sistemelor de timp real în scopul găsirii direcțiilor noi de dezvoltare și de îmbunătățire a performanțelor STR.

În **Capitolul I** s-a făcut o sinteză privind istoricul şi apoi evoluţia sistemelor de timp-real, începând cu prezentarea sistemelor încorporate, apoi a sistemelor de management al task-urilor, implementarea hardware pe FPGA şi analiza planificatoarelor software vs. hardware.

Rolul specific al STR este acela de a asigura controlul predictibil şi determinist al unui proces. STR sunt acele sisteme care oferă un răspuns corect într-un interval de timp prestabilit. **Rapiditatea nu** este o caracteristică specifică a STR, aceasta fiind mai mult un termen abstract. *Viteza de răspuns la evenimente* este însă un concept propriu STR diferit de cel menţionat anterior. STR joacă un rol important în societatea zilelor noastre deoarece majoritatea sistemelor utilizate pentru a uşura munca omului sunt controlate de microprocesoare.

Planificatoarele care stau la baza funcţionării STR trebuie să ruleze pe resurse hardware limitate în care necesarul de memorie şi procesor sunt semnificativ mai mici decât sistemele de tip desktop. În aceste condiţii STR trebuie să fie robuste şi să asigure o funcţionare corectă a echipamentelor în care ele sunt instalate, chiar şi în acele momente în care cerinţele adresate sistemului ating puncte de maxim. Robusteţea şi toleranţa la erori sunt două trăsături caracteristice STR. *Operaţia de planificare şi cea de comutare a contextelor poate influenţa semnificativ limita de planificare pentru sistemele în care frecvenţa de comutare a task-urilor este mare.* Acesta este motivul pentru care în paralel cu perfecţionarea algoritmilor de planificare software s-a urmărit *implementarea acestora în hardware*, tocmai *pentru a degreva procesorul de activitatea de planificare şi pentru a diminua supracontrolul specific sistemului de operare*. Una dintre cerinţele fundamentale ale STR hard este necesitatea de a asigura determinismul timpului real pentru task-urile critice. Rata de execuţie a task-urilor, supracontrolul sistemului de operare şi timpii de comutare a contextelor task-urilor sunt doar câţiva dintre parametrii care pot determina apariţia jitter-ului şi ratarea deadline-urilor în STR cu planificatoare soft. În sistemele embedded cu planificatoare software eliminarea totală a jitter-ului nu este posibilă, dar există o serie de mecanisme prin care acesta poate fi diminuat.

În a doua parte a Capitolului I se prezintă stadiul actual privind proiectarea hardware/software a planificatoarelor (schedulers) în timp-real pentru sisteme integrate (embedded) şi anume un planificator dedicat, un accelerator de planificare, un exemplu de planificator hardware şi un planificator hardware integrat. În ultima arhitectură prezentată, planificatorul hardware este inclus în procesor şi poate fi controlat direct cu instrucţiuni transmise prin pipeline, iar secvenţa de comutare a contextelor se realizează printr-un mecanism de remapare care este foarte rapid. Execuţia schemelor de planificare în hardware elimină supracontrolul datorat sistemului de operare îmbunătăţind astfel limita de planificare a setului de task-uri şi performanţele

sistemului per ansamblu. De cele mai multe ori un număr de task-uri care variază în jurul valorii 16 este mai mult decât suficient pentru majoritatea aplicaţiilor de acest tip.

În **Capitolul II** se face o scurtă prezentare a arhitecturii MIPS32 pipeline, a cărei structură organizaţională a fost utilizată pentru design-ul *nMPRA*, arhitectură de microcontroller propusă şi analizată în acest studiu. *Procesorul MIPS utilizează un set redus de instrucţiuni, prin urmare rezultă un minim efort uman. Pipeline-ul îmbunătăţeşte rata de transfer a instrucţiunii.* Intenţia este de a spori viteza procesorului şi a simplifica hardware-ul, din motive de costuri. Dacă se implementează acest procesor folosind tehnologia FPGA, este posibil să se upgradeze sistemul cu noi caracteristici cerute de utilizatori. S-au studiat şi articolele ştiinţifice publicate în ultimii ani şi concluzia este că arhitectura MIPS32 este la fel de actuală, cea mai amplă analiză şi care a rămas de bază fiind făcută de cel care a şi introdus pe piaţă acest procesor, în 1983, Hennessy.

Studiul autorului asupra arhitecturii propuse, *nMPRA*, este prezentat în **Capitolul III**, care cuprinde rezultatele obţinute de autor, susţinute şi publicate în cadrul conferinţelor internaţionale; partea a doua cuprinde analiza şi sinteza implementării *4MPRA* şi *8MPRA* pe FPGA-ul Cyclone V, Quartus II de la ALTERA şi concluziile care rezultă.

Arhitectura propusă în această teză, ***MPRA*, înlocuieşte metodele de salvare a stivei cu un algoritm de remapare ce permite execuţia unui nou task începând cu următorul ciclu-procesor, conţine o implementare originală bazată pe o structură hardware utilizată pentru planificarea statică şi dinamică a task-urilor, permite un management unitar al evenimentelor şi întreruperilor, defineşte o metodă de ataşare a întreruperilor la task-uri, asigurând cerinţele sistemelor de timp-real**.

Performanţele arhitecturii descrisă în teză se referă la:

- **viteza de comutare a task-ului**
- **comportarea întreruperilor**
- **timpul de răspuns la evenimente externe**
- **timpul de execuţie a comunicaţiilor pentru sincronizare interprocese.**

Deşi este o arhitectură cu multiplexare de resurse, cantitatea de memorie necesară pentru implementarea procesorului variază între 10 şi 35 kB în funcţie de numărul de task-uri şi de adâncimea de imbricare pentru apelurile funcţiilor, rezonabil dacă ţinem cont că în microcontrolerele actuale doar cantitatea de memorie RAM pentru uz general poate varia între 256kB şi 2,6MB. Din punct de vedere al puterii, concluzia este că *8MPRA* este o arhitectură de

microcontroller eficientă, consumul este redus şi nu supradimensionează platformele.

Ca o concluzie finală, *nMPRA* este **foarte puternică** deoarece:

- comutarea între task-uri se face în mod uzual într-un singur ciclu-procesor până la maxim trei cicli-procesor când CPU lucrează cu memoria globală;
- fără nici un RTOS software poate implementa aplicații de timp real doar cu instrucțiunile de la nivelul limbajului de asamblare;
- reacţia sistemului la evenimentele externe nu depăşeşte 1,5 cicli-procesor;
- remaparea contextelor task-urilor nu afectează în nici un fel performanța benzii de asamblare;
- arhitectura utilizează un fișier de regiștri special proiectat, care permite transferul automat al parametrilor între apelurile de funcții, degrevând astfel procesorul de operația de salvare și restaurare a regiștrilor;
- fiind o arhitectură RISC pe 32 biți, procesorul asigură un spațiu de adresă de 2^{32} cuvinte ceea ce îl face utilizabil în majoritatea aplicațiilor largi consumatoare de memorie;
- oferă un grad înalt de siguranță în utilizare datorită izolării totale a contextelor task-urilor, eliminând astfel posibilitatea de corupere a datelor;
- instrucțiuni puternice pentru partajarea resurselor, sincronizarea și comunicația între taskuri;
- asigură determinismul datorită mecanismului de priorități care garantează acest lucru.

2. CONTRIBUŢIILE AUTORULUI

❖ Autorul propune în această teză o soluție inovatoare pentru prioritizarea întreruperilor atașate la aceeași task. În acest context, întreruperea împrumută prioritatea şi comportamentul task-ului. Astfel, comportamentul întreruperilor este mult mai predictibil în contextul unei aplicaţii de timp real (un task nu poate fi întrerupt decât de întreruperile ataşate unui task mai prioritar).

Spre deosebire de soluția software, prin hardware orice întrerupere are același timp de răspuns. Mai mult decât atât, soluția propusă poate oferi priorități statice pentru întreruperi. Putem spune că soluția prezentată conține un management unitar al întreruperilor și o soluție hardware pentru a atașa întreruperile la task-urile sistemelor de timp-real implementate în hardware.

Această schemă are unele caracteristici puternice și interesante:

- nu trebuie să existe un controler specializat pentru întreruperi;
- ***întreruperile moștenesc prioritatea taskului (sCPU$_i$);***
- un task poate atașa niciuna, una, mai multe, sau chiar toate cele *p* întreruperi din sistem;
- pentru întreruperile atașate aceluiași task prioritatea o stabilește programatorul;
- întreruperea atașată unui task poate întrerupe un task mai puțin prioritar dar nu poate întrerupe execuția taskului căruia îi este atașată, sau un task mai prioritar;
- întrerupere poate fi atașată unui singur task.
- întreruperea poate fi un task;
- toate întreruperile pot fi atașate unui singur task;
- întreruperea nu resetează banda de asamblare a altor *sCPU$_i$*;
- nu implică salvări și restaurări de context.
- întreruperile pot fi nested.
- prioritățile întreruperilor pot fi dinamice (prin reatașarea la un task sau prin schimbarea priorității taskului la care este atașată).

Soluția hardware pentru gestionarea întreruperilor implică un bloc hardware adițional. *Blocul codor de prioritate* generează identificatorul întreruperii de cea mai înaltă prioritate.

❖ S-a implementat în VHDL codificatorul de prioritate pentru 4 şi pentru 8 task-uri şi s-a făcut simularea cu ModelSim.

N.C. Gaitan, V.G. Gaitan, **E.E. (Ciobanu) Moisuc** - *Improving Interrupt Handling in the nMPRA*, International Conference on Development and Application Systems, 10.1109/DAAS.2014.6842419, IEEE (BDI)/ISI Thomson Web of Science, http://www.dasconference.ro/dvd2014/, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6828590, ISBN: 978-1-4799-5094-2, 2014.

❖ Schema de prioritizare a evenimentelor propusă în această lucrare tratează hardware evenimentele, oferă soluția optimă pentru tratarea evenimentelor multiple, este simplă şi se poate aplica tuturor tipurilor de evenimente.

E.E. (Ciobanu) Moisuc, Al. B. Larionescu, V. G. Gaitan - *Hardware Event Treating in nMPRA*, 12th International Conference on Development and Application Systems, Suceava, Romania, May 15-17, 2014, International Conference on Development and Application

Systems, 10.1109/DAAS.2014.6842429, IEEE (BDI)/ISI Thomson Web of Science, http://www.dasconference.ro/dvd2014/ http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6828590, ISBN: 978-1-4799-5094-2/14.

Noutatea prezentată în studiu se referă la:

- o nouă implementare hardware a mecanismului de selecţie a evenimentului utilizând registrul-capcană;
- arhitectura registrului PC_i;
- introducerea instrucţiunii *ret_esr* (*return from the event service routine*) pentru a permite tratarea automată a evenimentelor.

Avantajul tratării hardware a evenimentelor este *reducerea timpului de detectare a sursei evenimentului şi de începere a rutinei corespunzătoare de servire a evenimentelor. Chiar se permite introducerea de noi evenimente în sistem* prin:

- simpla adăugare a câmpurilor necesare în *Task Register* (TR_i), *Event Status Task Register* ($ESTR_i$) şi *Event Priority Register* (EPR_i),
- actualizarea schemei de prioritizare globală a evenimentelor
- introducerea unui nou registru-capcană pentru fiecare categorie nouă de eveniment.

❖ S-a implementat în VHDL blocul codor de selecţie a adreselor şi s-a făcut simularea cu ModelSim.

❖ Schema de prioritizare propusă a fost implementată parţial în Verilog, compilată, analizată şi simulată pe Cyclone V de la Altera.

❖ Se prezintă registrele-capcană asociate categoriilor de evenimente şi o structură modificată a registrului *Program Counter* (PC_i).

❖ Pentru sincronizarea şi comunicaţia între taskuri s-a prevăzut un set de evenimente, implementat sub forma unui *ERF* (**E**vent **R**egister **F**ile), de asemenea ca resursă globală pentru toate $sCPU_i$. Accesul pentru setarea unui eveniment este fără adresă (nu necesită căutări în tabele pentru a depista un loc liber). Hardware-ul depistează automat prima adresă liberă şi setează evenimentul şi informaţia ataşată acestuia sau sesizează şi semnalizează că nu există nici un eveniment liber. Task-ul destinaţie nu trebuie decât să citească din *ERF* pentru a afla dacă un eveniment i se adresează sau nu. Achitarea evenimentului se face automat. Totul se întâmplă într-un singur ciclu-maşină. Din nou sunt eliminate căutările în diferite tabele de la nivelul software.

E.E. (Ciobanu) Moisuc, Al.B. Larionescu, I. Ungurean - *Hardware Event Handling in the Hardware Real-Time Operating Systems*, The 18th International Conference on System Theory, Control and Computing, 10.1109/ICSTCC.2014.6982390, IEEE (BDI), http://www.ace.tuiasi.ro/icstcc2014/index.html, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6961601, ISBN 978-1-4799-4602-0, 2014.

❖ S-a implementat şi analizat pe Cyclone V arhitectura MPRA pentru 4 și 8 task-uri, care include:

- implementarea în Verilog a elementelor de bază ale procesorului: unitate de control, memorie de date și instrucțiuni, unități de detecție a hazardului, unități de redirecționare, ALU, multiplexoare;
- implementarea în Verilog a structurilor cu resurse multiplexate specifice procesorului MPRA: registrul PC, regiștrii pipeline, fișierul de regiștri.

❖ S-a efectuat compilarea, analiza şi simularea arhitecturilor MPRA, 4MPRA şi 8MPRA, simularea şi testarea individuală şi în sistem a structurilor cu multiplexare de resurse: registrul PC, regiştrii pipeline şi fişierul de regiştri. S-a efectuat o analiză a puterii consumate de fiecare în parte şi apoi o comparaţie pentru a demonstra că este îndeplinită cerinţa sistemelor de timp-real de consum cât mai mic de putere, pentru a evita supradimensionarea platformelor.

Studii viitoare:

- În etapele următoare de cercetare, consider necesar a se studia mai atent, din mai multe puncte de vedere, mai ales viteză şi suprafaţă, mediul cel mai eficient de implementare pentru arhitectura propusă, Xilinx sau Altera.
- Este important a se analiza arhitecturile din punctul de vedere al frecvenţei de lucru.
- Un alt studiu de viitor pe care l-am dedus din cercetările prezente ar trebui să cuprindă o comparaţie între procesorul cu 5 stadii pipeline, cel cu un număr mai mare de stadii pipeline şi cel cu multiplicare de resurse, dar cu 5 stadii pipeline (*nMPRA*).
- În etapa imediat următoare este necesară finalizarea implementării şi simulării planificatorului pe FPGA.
- O idee desprinsă din cercetările făcute ar fi analiza eficienţei unui planificator care să fie implementat ca un co-procesor 0.

MULŢUMIRI

Mulţumesc coordonatorului de doctorat, prof. univ. dr. ing. Vasile Găitan, pentru că a avut încredere în mine, în ciuda piedicilor generate de vârsta mea şi de distanţa mare în timp (1983) de la terminarea Facultăţii de Automatizări şi Calculatoare – IP Iaşi.

Mulţumesc îndrumătorilor şi conducerii Şcolii doctorale că mi-au oferit ocazia de a-mi împlini munca de o viaţă, că mi-au dat şansa de a o mai lua o dată de la capăt; pentru mine, acest titlu academic, obţinut cu mult efort, cu multă muncă, are o mare însemnătate, îmi aduce o mândrie şi o bucurie pe care doar la vârsta mea o poţi înţelege.

Mulţumesc soţului meu, în memoria căruia am elaborat această teză; el m-a înscris la doctorat pentru a-mi demonstra că pot şi merit mai mult în această viaţă. A fost foarte greu pentru mine, mai ales că m-a părăsit în ultimul an de doctorat, dar m-am străduit să-i împlinesc ultima dorinţă şi să obţin acest titlu.

Mai am mult de studiat, cei trei ani au trecut prea repede, a fost ca un al doilea job pentru mine, domeniul este vast şi evoluţia este extrem de rapidă, abia am înţeles ce am de făcut, ce aşteptări sunt. Dacă facultatea va avea nevoie de ceea ce am învăţat prin acest doctorat, voi fi bucuroasă să continui munca alături de o echipă pe care o respect şi să împlinesc încă o dorinţă a soţului meu, aceea de a împărtăşi din experienţa mea în domeniu, de a preda şi de a-mi continua studiul alături de studenţii acestei prestigioase facultăţi.

Mulţumesc!

BIBLIOGRAFIE

[1] Colnaric, M., Verber, D. şi Halang, W.A. – *Distributed Embedded Control Systems*. Springer-Verlag London Limited, 2008.

[2] Liu, C.L. şi Layland, J.W. – *Scheduling algorithms for multiprogramming in a hard real-time environment*, Journal of the ACM, 20(1),46-61, 1973.

[3] Bini, E., Buttazzo, G. C., şi Buttazzo, G. M. – *A Hyperbolic Bound for the Rate Monotonic Algorithm*. *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, Delft, The Netherlands; Rate monotonic analysis: the hyperbolic bound. *IEEE Transactions on Computers*, 52(7), 933-942, 2001.

[4] Bini, E., şi Buttazzo, G. C. – *The Space of Rate Monotonic Schedulability*. *Proceedings of the 23th IEEE Real-Time Systems Symposium*, Austin, TX, USA, 2002.

[5] Bini, E., şi Buttazzo, G. C. – *Schedulability Analysis of Periodic Fixed Priority Systems*. *IEEE Transactions on Computers*, 53 (11), 1462-1473, 2004.

[6] Bini, E., şi Di Natale, M. – *Optimal Task Rate Selection in Fixed Priority Systems*. *Proceedings of the 26th IEEE Real-Time Systems Symposium*, Miami, FL, USA, 2005.

[7] Bini, E., Di Natale, M., şi Buttazzo, G. C. – *Sensitivity Analysis for Fixed-Priority Real-Time Systems*. *Real-Time Systems*, 39 (1-3), 5-30, 2007.

[8] Buttazzo, G.C. – *Rate monotonic vs. EDF: Judgment Day*. *Real-Time Systems*, 29(1), 5-26, 2005.

[9] Buttazzo, G. – *Research Trends in Real-Time Computing for Embedded Systems*. *ACM SIGBED Review*, Vol. 3, No. 3, 2006.

[10] Holenderski, M., Cools, W., Bril, R.J., şi Lukkien, J.J. – *Extending an Open-source Real-time Operating System with Hierarchical Scheduling*, Technical Report, Eindhoven University of Technology, CS-10-10, 2010.

[11] Bini, E., şi Buttazzo, G.C. – *Measuring the Performance of Schedulability Tests*. *Real-Time Systems*, v.30 n.1-2, p.129-154, 2005.

[12] Sha, L., Abdelzaher, T., Arzen, K., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., şi Mok, A. K. – *Real Time Scheduling Theory: A Historical Perspective. Real-Time Systems* 28, 2-3, 101-155, 2004.

[13] Albers, K. şi Slomka, F. – *An event stream driven approximation for the analysis of real-time systems*. In *Proceedings of the Euromicro Conference on Real-Time Systems*, 187–195, 2004.

[14] Fisher, N., şi Baruah, S. – *A polynomial-time approximation scheme for feasibility analysis in static-priority systems with bounded relative deadlines. Proceedings of the 13th International Conference on Real-Time Systems*, Paris, France, 2005.

[15] Fisher, N., şi Baruah, S. – *A polynomial-time approximation scheme for feasibility analysis in static-priority systems with arbitrary relative deadlines*. *Proceeding of the 17th Euromicro Conference on Real-Time Systems*, Palma de Mallorca, Spain, 2005.

[16] Bini, E., Huyen, T., Nguyen, C., Richard, P., şi Baruah, S.K. – *A Response-Time Bound in Fixed-Priority Scheduling with Arbitrary Deadlines*. IEEE Transactions on Computers, Volume 58 Issue 2, 279-286, 2009.

[17] Abeni, L., şi Buttazzo, G. – *Resource Reservation in Dynamic Real-Time Systems*. *Real-Time Systems*, Vol. 27, No. 2, 123-167, 2004.

[18] Buttazzo, G., şi Bini, E. – *Optimal Dimensioning of a Constant Bandwidth Server*. *Proceedings of the 27th IEEE International Real-Time Systems Symposium*, 169-177. Rio de Janeiro, Brasil, 2006.

[19] AbouTrab, M.S., Brockway, M., Counsell, S., şi Hierons, R.M. – *Testing real-Time Embedded Systems Using Timed Automata Based Aproaches*. *Journal of Systems and Software*, 86 (5), 129-1223, 2013.

[20] Waszniowski, L., şi Hanzalek, Z. – *Formal verification of multitasking applications based on timed automata model*. *Real-Time Systems*, vol. 38, no. 1, 39–65, 2008.

[21] Klobedanz, K., Kuznik, C., Thuy, A., şi Muller, W. – *Timing modeling and analysis for autosar-based software development - a case study*. DATE. IEEE, 642–645, 2010.

[22] Zhu, L., Liu, P., Wang, Z., Shi, J., şi Zhu, H. – *A Timing Verification Framework for AUTOSAR OS Component Development based on Real-Time Maude*. *Proc of 7th International Symposium on Theoretical Aspects of Software*, Birmingham, UK, 2013.

[23] Andersson, B., Baruah, S., şi Jonsson, J. (2001). *Static-priority scheduling on multiprocessors*. *Proceedings of the IEEE International Real-Time Systems Symposium*, London, UK. IEEE Computer Society Press, 193-202.

[24] Phillips, C.A., Stein, C., Torng, E., şi Wein, J. (2002). Optimal Time-Critical Scheduling via Resource Augmentation. In *Proceedings of Algorithmica*, 163-200.

[25] Lopez, J.M., Díaz, J.L., şi García, D.F. (2004). *Minimum and Maximum Utilization Bounds for Multiprocessor Rate Monotonic Scheduling*. In *Proceedings of IEEE Trans. Parallel Distrib. Syst.*, 642-653.

[26] Burns, A., Davis, R. I., Wang, P., & Zhang, F. (2012). *Partitioned EDF Scheduling for Multiprocessors using a C=D Scheme*. *Real-Time Systems*, 48(1), 3-33.

[27] Baruah, S., şi Fisher, N. (2006). The Feasibility Analysis of Multiprocessor Real-Time Systems. *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, Dresden, Germany. IEEE Computer Society Press.

[28] Cucu, L., şi Goossens, J. (2006). *Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors*. *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*, 397–405.

[29] Fisher, N., şi Baruah, S. (2007). *The Global Feasibility and Schedulability of General Task Models on Multiprocessor Platforms*. *Proceedings of the EuroMicro Conference on Real-Time Systems*, Pisa, Italy. IEEE Computer Society Press.

[30] Baruah, S. (2007). *Techniques for Multiprocessor Global Schedulability Analysis.* 28th IEEE International Real-Time Systems Symposium, 119-128.

[31] Carpenter, J., Funk, S., Holman, P., Srinivasan, A., Anderson, J., şi Baruah, S. (2004). *A Categorization of Real-time Multiprocessor Scheduling Problems and Algorithms*. In *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Joseph Y-T Leung (ed). Chapman Hall/ CRC Press.

[32] Davis, R.I., şi Burns, A. (2009). *Robust priority assignment for messages on Controller Area Network (CAN)*. *Real-Time Systems*, Volume 41, Issue 2, pages 152-180.

[33] Bastoni, A., Brandenburg, B., şi Anderson, J. (2010). *An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor Real-Time Schedulers*. *Proceedings of the 31st IEEE Real-Time Systems Symposium*, pp. 14-24.

[34] Nair, G.T.R., şi Persya, C.A. (2011). *Critical Task Re-assignment under Hybrid Scheduling Approach in Multiprocessor Real-Time Systems*. 23rd IASTED International Conference on Parallel and Distributed Computing Systems, Dallas.

[35] Safaei, A.A., Haghjoo, M.S., şi Abdi, F. (2011). *PFGN*: *A Hybrid Multiprocessor Real-Time Scheduling Algorithm for Data Stream Management Systems*. Communications in Computer and Information Science.Vol. 167, 180-192.

[36] J. Stankovic, K. Ramamritham, *The spring kernel: a new paradigm for real-time systems*, IEEE Software, vol. 8, pp. 62 –72, May 1991.

[37] Hansson, H., Nolin, M. şi Nolte., T. (2006). *Real-Time in Embedded Systems*. In Zurawski R. (ed.), *Embedded Systems Handbook*, 2-1, 2-35, CRC Press.

[38] Buttazzo, G.C. (2008). *Real-Time Scheduling and Resource Management*. In Lee, I., Leung, J.Y-T., and Son, S.H. (ed.), *Handbook of Real-Time and Embedded Systems*, 2-1, 2-16. Chapman & Hall.

[39] Mok, A. K. (1983). *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment*. Ph.D. Thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, MA.

[40] Sprunt, B., Sha, L., şi Lehoczky, J. (1989). *Aperiodic Task Scheduling for Hard Real-Time System*. *Journal of Real-Time Systems*, 1, 27-60.

[41] Collette, S., Cucu, L., şi Goossens, J. (2007). *Algorithm and complexity for the global scheduling of sporadic tasks on multiprocessors with work-limited parallelism*. *15th International Conference on Real-Time and Network systems (RTNS'07)*, Nancy.

[42] Fisher, N., Baruah, S., şi Baker., T. P. (2006). *The Partitioned Scheduling of Sporadic Tasks according to Static Priorities*. *Proceedings of the 18th Euromicro Conference on Real-Time Systems*, Dresden, Germany. IEEE Computer Society Press.

[43] Phillips, C.A., Stein, C., Torng, E., şi Wein, J. (2002). *Optimal Time-Critical Scheduling via Resource Augmentation*. In *Proceedings of Algorithmica*, 163-200.

[44] Andersson, B., Baruah, S., şi Jonsson, J. (2001). *Static-priority scheduling on multiprocessors*. *Proceedings of the IEEE International Real-Time Systems Symposium*, London, UK. IEEE Computer Society Press, 193-202.

[45] Bastoni, A., Brandenburg, B., şi Anderson, J. (2010). *An Empirical Comparison of Global, Partitioned, and Clustered Multiprocessor Real-Time Schedulers*. *Proceedings of the 31st IEEE Real-Time Systems Symposium*, 14-24.

[46] Li, Q., şi Yao, C. (2003). *Real-Time Concepts for Embedded Systems*. CMP Books, USA.

[47] *http://www.embeddedlinux.org.cn/RTConforEmbSys/*, 2013.

[48] Marau, R., Leite, P., Velasco, M., Marti, P., Almeida, L., Pedreiras, P., şi Fuertes, J.M. (2008). *Performing Flexible Control on Low-Cost Microcontrollers Using a Minimal Real-Time Kernel*. *IEEE Transactions on Industrial Informatics*, 4(2), 125-133.

[49] Alecsa, B., *Sisteme încorporate cu FPGA pentru controlul proceselor rapide*, Teză de doctorat, Universitatea Tehnică „Gheorghe Asachi" din Iaşi, 2011.

[50] Monmasson, E., Cirstea M.N., *FPGA design methodology for industrial control systems – a review*, *IEEE Transactions on Industrial Electronics*, vol. 54, no. 4, august 2007.
[51] Andraka, R., *A survey of CORDIC algorithms for FPGA based computers*, *Proceedings of ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, *FPGA '98*, Monterey, Caligornia, SUA, februarie 1998.
[52] White, S.A., *Applications of distributed arithmetic to digital signal processing: a tutorial review*, *IEEE ASSP Magazine*, vol. 6, no. 3, iulie 1989.
[53] Fang, Z., Carletta J.E., Veillette R.J., *A methodology for FPGA-based control implementation*, *IEEE Transactions on Control Systems Technology*, vol. 13, no. 6, noiembrie 2005.
[54] Hilaire, T., Menard D., Sentieys O., *Bit accurate roundoff noise analysis of fixed point linear controllers*, *IEEE International Symposium on Computer-Aided Control System Design*, *CACSD 2008*, San Antonio, Texas, SUA, septembrie 2008.
[55] Woods, R., McAllister J., Lightbody G., Yi Y., *FPGA-based implementation of signal processing systems*, John Wiley and Sons, 2008.
[56] Idkhajine, L., Monmasson E., Maalouf A., "Extended Kalman filter for AC drive sensorless speed controller - FPGA-based solution or DSP-based solution", *Proceedings of the 2010 IEEE International Symposium on Industrial Electronics, ISIE 2010*, Bari, Italia, iulie 2010.
[57] Monmasson, E., Idkhajine L., Cirstea M.N., Bahri I., Tisan A., Naouar M.W., *FPGAs in industrial control applications*, *IEEE Transactions on Industrial Informatics*, vol. 7, no. 2, mai 2011.
[58] Carbone, S., Delli Colli V., Di Stefano R., Figalli G., Marignetti F., *Design and implementation of high performance FPGA control for permanent magnet synchronous motor*, *Proceedings of the 35th Annual Conference of the IEEE Industrial Electronics Society, IECON 2009*, Porto, Portugalia, noiembrie 2009.
[59] Ioan, A.D., *Contribuţii la implementarea structurilor hardware cu circuite numerice programabile CPLD şi FPGA*, teză de doctorat, Universitatea Tehnică „Gheorghe Asachi" din Iaşi, 2010.
[60] Kaeslin, H., *Digital integrated circuit design: from VLSI architectures to CMOS fabrication*, Cambridge: Cambridge University Press, 2008.
[61] Munden R., *ASIC and FPGA verification: a guide to component modeling*, Morgan Kaufman Publishers, Elsevier, 2005.

[62] Kuacharoen, P., Shalan, M., and Mooney III, V.J., *A Configurable Hardware Scheduler for Real-Time Systems*, Proc. Engineering of Reconfigurable Systems and Algorithms, 2003, 95-101.

[63] Tucci, P.: *Hardware/Software Design of Dynamic Real-Time Schedulers for Embedded Multiprocessor Systems*, Online: https://sourceforge.net/p/xrt/, 2012.

[64] Bloom, G., Parmer, G., Narahari, B., and Simha, R., *Real-Time Scheduling with Hardware Data Structures*, IEEE Real-Time Systems Symposium, 2010, RTSS 2010, Dec. 2010.

[65] M. Vetromille, L. Ost, C.A.M. Marcon, C. Reif, and F. Hessel, *RTOS Scheduler Implementation in Hardware and Software for Real Time Applications*, Seventeenth IEEE International Workshop on Rapid System Prototyping, 2006, pp. 163-168.

[66] J. Tarrillo, L. Bolzani, F. Vargas, *A Hardware-Scheduler for Fault Detection in RTOS-Based Embedded Systems*, 12th Euromicro Conference on Digital System Design/Architectures, Methods and Tools, 2009.

[67] T. Nakano, A. Utama, M. Itabashi, A. Shiomi, and M. Imai, *Hardware Implementation of a Real-Time Operating System*, Proceedings of the 12th TRON Project International Symposium, 1995, 34-42.

[68] L. Lindh, *Fastchart-a fast time deterministic CPU and hardware based real-time-kernel*, Real Time Systems, 1991. Proceedings., Euromicro '91 Workshop on, pp.36-40, 12-14 Jun. 1991.

[69] M. Song, S. H. Hong, and Y. Chung, *Reducing the overhead of real-time operating system through reconfigurable hardware*, 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, 2007. DSD 2007, 311-316, Aug. 2007.

[70] A.S.R.Oliveira, L.Almeida, and A.B.Ferrari, *The arpa-mt embedded smt processor and its RTOS hardware accelerator*, Industrial Electronics, IEEE Transactions on, vol. PP, no. 99, 1-1, 2009.

[71] N. Gupta, S.K. Mandal, J. Malave, A. Mandal, R.N. Mahapatra, *A Hardware Scheduler for Real Time Multiprocessor System on Chip*, 23rd International Conference on VLSI Design, 2010, 264-269.

[72] L. Lindh, *FASTHARD - A Fast Time Deterministic HARDware Based Real-time Kernel*, Proceedings of the Fourth Euromicro Workshop on Real-Time Systems, Jun. 1992, 21-25.

[73] J. Lee, I. Mooney, V.J., A. Daleby, K. Ingstrom, T. Klevin, and L. Lindh, *A comparison of the rtu hardware RTOS with a hardware/software RTOS*, Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific Date of Conference: 683 - 688, Jan. 2003.

[74] S. Nordstrom, L. Lindh, L. Johansson, and T. Skoglund, *Application specific real-time microkernel in hardware*, Real Time Conference, 14th IEEE-NPSS, 10 Jun. 2005.

[75] T. Nakano, A. Utama, M. Itabashi, A. Shiomi, and M. Imai, *Hardware Implementation of a Real-Time Operating System*, Proceedings of the 12th TRON Project International Symposium, 1995, 34-42.

[76] V.J. Mooney III, and D.M. Blough, *A hardware-software real-time operating system framework for SoCs*, IEEE Design & Test of Computers, 2002, 44-51.

[77] *http://hartos.dk/publications/msc-thesis/hartos.pdf*, 2012

[78] S. Chandra, F. Regazzoni, and M. Lajolo, *Hardware/Software partitioning of operating systems: a behavioral synthesis approach*, Proc. of ACM GLSVLSI, 2006, 324-329.

[79] N. Silva, A. Oliveira, Santos, R. Almeida, L, *The OReK real-time micro kernel for FPGA-based systems-on-chip*, Embedded Systems for Real-Time Multimedia, ESTImedia 2008. IEEE/ACM/IFIP Workshop on, 75-80.

[80] N. Maruyama, T. Ishihara, and H. Yasuura, *An RTOS in hardware for energy efficient software-based tcp/ip processing*, Application Specific Processors (SASP), 2010 IEEE 8th Symposium on, 58 –63, Jun. 2010.

[81] J. Hildebrandt and D. Timmermann, *An FPGA based scheduling coprocessor for dynamic priority scheduling in hard real-time systems*, in Field-Programmable Logic and Applications: The Roadmap to Reconfigurable Computing (R. Hartenstein and H. Grünbacher, eds.), vol. 1896 of Lecture Notes in Computer Science, 777–780, Springer Berlin /Heidelberg, 2000.

[82] *** *TriCore 1, 32 bit Unified Processor Core*, Volume 1, Core Architecture V 1.3 & V 1.3.1, Jan. 2008

[83] *** *Intel i960 Jx Microprocessor Developer's Manual*, Dec. 1997

[84] Lee, E. A., *What's ahead for embedded software?*, Computer 33, Nr. 9, S. 18-26, 2000.

[85] S. Ben Othman, A.K. Ben Salem & S. Ben Saoud, *MPSoC design of RT control applications based on FPGA SoftCore processors. In: Proc. 15th IEEE Int. Conf. Electronics, Circuits and Systems ICECS, 2008, S.* 404-409

[86] T. Dorta, J. Jimenez, J.L. Martin, U. Bidarte & A. Astarloa, *Overview of FPGA-Based Multiprocessor Systems*, Proc. Int. Conf. Reconfigurable Computing and FPGAs ReConFig, 2009, S. 273-278

[87] N. Maruyama, T. Ishihara & H. Yasuura, *An RTOS in hardware for energy efficient software-based TCP/IP processing*, Application Specific Processors (SASP), 2010 IEEE 8th Symposium on., 2010, S. 58-63.

[88] Ranjit Adiga, *Selecting the right RTOS scheduling algorithms using system modelling*, CMR Design Automation - August 26, 2013.

[89] R. Bhagwan, B. Lin, *Fast and scalable priority queue architecture for high-speed network switches*, INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, 2000, 538–547 vol.2.

[90] A. Ioannou, M. Katevenis, *Pipelined heap (priority queue) management for advanced scheduling in high-speed networks*, Networking, IEEE/ACM Transactions on (2007) 450–461.

[91] W. Peck, E. Anderson, J. Agron, J. Stevens, F. Baijot, and D. Andrews, *hthreads: A Computational Model for Reconfigurable Devices*, in Proceedings of the 16th International Conference on Field Programmable Logic and Applications (FPL), vol. 1. IEEE, August 2006, pp. 885–888.

[92] Chetan Kumar N Ga, Sudhanshu Vyasa, Jonathan A. Shidalb, Ron K. Cytronb, Christopher D. Gillb, Joseph Zambrenoa, Phillip H. Jonesa, *Improving System Predictability and Performance via Hardware Accelerated Data Structures*, Conference Papers, Proceedings of Dynamic Data Driven Application Systems (DDDAS), 2013.

[93] S.-W. Moon, K. Shin, J. Rexford, *Scalable hardware priority queue architectures for high-speed packet switches*, in: Real-Time Technology and Applications Symposium, 1997. Proceedings., Third IEEE, 1997, 203–212.

[94] E. Dodiu, *Planificator de timp real implementat hardware pentru sisteme embedded bazate pe FPGA*, Teză de doctorat, Universitatea „Ștefan cel Mare” Suceava, Facultatea de Inginerie Electrică și Știința Calculatoarelor, 2013.

[95] V.G. Gaitan, A. Graur, E. Dodiu, *Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – architecture description*, IEEE 35’th Jubilee International Convention on Information and Communication Technology, Electronics and Microelectronics, Croația, mai 2012.

[96] V. Gaitan, *The Highly Functional Distributed System*, 4th International Symposium on Automatic Control and Computer Science, Vol. 1, Iași 1993, 277-282.

[97] E. Dodiu, V.G. Gaitan, A. Graur, *Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – architecture description*, IEEE 35’th Jubilee International Convention on Information and Communication Technology, Electronics and Microelectronics, Croația, Mai 2012.

[98] E. Dodiu, V.G. Gaitan, *Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – concept and*

theory of operation, IEEE International Conference on Electro/Information Technology EIT 2012, Indianapolis, US, Mai 2012.

[99] J. Hennessy, D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, USA, 2007.

[100] V.G. Gaitan, N.C. Gaitan, I. Ungurean, *CPU Architecture based on a Hardware Scheduler and Independent Pipeline Registers*, IEEE Transactions on VLSI System, vol. 23, no. 9, september 2015.

[101] L.E. Leyva-del-Foyo, and P. Mejia-Alvarez, *Custom Interrupt Management for Real-Time and Embedded System Kernels*, in Proceedings of the 10th IEEE ECOOP Workshop on Exception Handling in Object Oriented Systems Development, 2005.

[102] Shi-Hai Zhu, *Hardware Implementation Based on FPGA of Interrupt Management in a Real-time Operating System*, Information Technology Journal, 2013.

[103] B.C. Alecsa, *FPGA implementation of a matrix structure for integer division*, Proceedings of the 3rd International Symposium on Electrical and Electronics Engineering, Galati, Romania, 2010.

[104] I. Liu, J. Reineke, E.A. Lee, *A PRET architecture supporting concurrent programs with composable timing properties*, in Signals, Systems and Computers, 2010, Conference Record of the Fortz Fourth Asilomar Conference on (pp. 2111/2115), IEEE.

[105] M. Shahbazi, P. Poure, S. Saadate, M.R. Zolghadri, *Fault-Tolerant Five-Leg Converter Topology with FPGA-Based Reconfigurable Control*, IEEE Transactions on, vol. 60, no. 6, June 2013.

[106] L. Cheng-Min, *Nested interrupt analysis of low cost and high performance embedded systems using GSPN framework*, IEICE Trans. Inform. Syst., E93-D: 2509-2519, 2010.

[107] Tan, S.L.; Bao Anh, T.N., *Real-time operating system (RTOS) for small (16-bit) microcontroller*, Consumer Electronics, ISCE '09, IEEE 13th International Symposium on, pp.1007-1011, 25-28 May 2009.

[108] Bao Anh, T.N.; Tan, S.L., *Survey and performance evaluation of real-time operating systems (RTOS) for small microcontrollers*, Micro, IEEE Computer Society, ISSN: 0272-1732, 21 August 2009.

[109] G. Butazzo, *Hard Real-Time Computing Systems*, Pavia-Italz: Spinger, 2005.

[110] N.C. Gaitan, *Real-time Acquisition of the Distributed Data by using an Intelligent System*, Electronics and Electrical Engineering, Kaunas: Technologija, No. 8, Issue 104, Octomber 2010, ISSN 1392-1215.

[111] M.V. Micea, C.S. Certejan, V. Stangaciu, R. Cioarga, V. Cretu, and E. Petriu, *Inter-Task Communication and Synchronization in the Hard*

Real-Time Compact, ROSE 2008 – IEEE International Workshop on Robotic and Sensors Environments, Ottawa – Canada, 2008.

[112] G. Butazzo, P. Gai, *Efficient EDF Implementation for Small Embedded Systems*, OSPERT 2006- Workshop on Operating Systems Platforms for Embedded Real-Time applications, Dresden- Germany, 2006.

[113] N.C. Gaitan, V.G. Gaitan, **E.E. (Ciobanu) Moisuc**, *Improving Interrupt Handling in the nMPRA*, 12th International Conference on Development and Application Systems, Suceava, Romania, May 15-17, 2014, ISBN: 978-1-4799-5094-2/14.

[114] **E.E. (Ciobanu) Moisuc**, Al. B. Larionescu, V. G. Gaitan, *Hardware Event Treating in nMPRA*, 12th International Conference on Development and Application Systems, Suceava, Romania, May 15-17, 2014, ISBN: 978-1-4799-5094-2/14.

[115] **E.E. (Ciobanu) Moisuc**, Al. B. Larionescu, I. Ungurean, *Hardware Event Handling in the Hardware Real-Time*, 18th International Conference on System Theory, Control and Computing to be held in Sinaia, Romania, October 17-19, 2014, ISBN: 978-1-4799-4602-0 ©2014 IEEE.

[116] *http://www.xilinx.com/fpga/*

[117] *https://www.altera.com/products/fpga/cyclone-series/cyclone-v/*

[118] *http://www.drdobbs.com/architecture-and-design/onegiant-leap-the-apollo-guidance-compu/184404139*

[119] *http://en.wikipedia.org/wiki/Comparison_of_embedded_computer_systems_on_board_the_Mars_rovers*

[120] *http://www.ecs.umass.edu/ece/ece232/*

[121] *http://cobweb.ecn.purdue.edu/~ece437l/materials*

[122] *http://opencores.org/project.mips_enhanced*, 2015.

[123] *https://imgtec.com/mips/architectures*

[124] K. Singh, S. Parmar, *Design of High Performance MIPS Cryptography Processor Based on T-DES Algorithm*, International Journal of Engineering Research & Technology (IJERT) Vol. 1 Issue 3, May - 2012 ISSN: 2278-0181

LUCRĂRILE PUBLICATE ALE AUTORULUI:

1. **E.E. (Ciobanu) Moisuc**, Al.B. Larionescu, V.G. Gaitan - *Hardware Event Treating in nMPRA*, International Conference on Development and Application Systems, IEEE (BDI)/ISI Thomson Web of Science, 10.1109/DAAS.2014.6842429, http://www.dasconference.ro/dvd2014/, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6828590, ISBN 978-1-4799-5092-8, 2014.

2. N.C. Gaitan, V.G. Gaitan, **E.E. (Ciobanu) Moisuc** - *Improving Interrupt Handling in the nMPRA*, International Conference on Development and Application Systems, 10.1109/DAAS.2014.6842419, IEEE (BDI)/ISI Thomson Web of Science, http://www.dasconference.ro/dvd2014/, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6828590, ISBN 978-1-4799-5092-8, 2014.

3. **E.E. (Ciobanu) Moisuc**, Al.B. Larionescu, I. Ungurean - *Hardware Event Handling in the Hardware Real-Time Operating Systems*, The 18th International Conference on System Theory, Control and Computing, 10.1109/ICSTCC.2014.6982390, IEEE (BDI), http://www.ace.tuiasi.ro/icstcc2014/index.html, http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6961601, ISBN 978-1-4799-4602-0, 2014.

4. L. Andries, V.G. Gaitan, **E.E. (Ciobanu) Moisuc** - *Programming paradigm of a Microcontroller with Hardware Scheduler Engine and Independent Pipeline Registers – A software approach*, The 19th International Conference on System Theory, Control and Computing, 10.1109/ICSTCC.2015.7321376, IEEE (BDI), http://www.aie.ugal.ro/icstcc2015, http://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=7321255, ISBN 978-1-4799-8481-7, 2015.

5. **E.E. (Ciobanu) Moisuc**, N.C. Gaitan - *Study on a microcontroller architecture in hardware real-time operating system*, ANNALS OF "DUNAREA DE JOS" UNIVERSITY OF GALATI MATHEMATICS, PHYSICS, THEORETICAL MECHANICS FASCICLE II, YEAR VII (XXXVIII) 2015, No. 1, B+ (BDI), http://www.phys.ugal.ro/Annals_Fascicle_2/Year2015/Vol1.htm, ISSN 2067-2071, 2015.

6. **Moisuc (Ciobanu) Elena-Eugenia**, *Study on Optimized Design of CPU Architecture Based Real-Time Scheduling and Pipeline Registers*, Research and Innovation Exhibition „UGAL INVENT"- The first event supporting innovation promoted by "Dunarea de Jos" University of Galati, proceedings, http://www.invent.ugal.ro/.

7. **Elena-Eugenia (CIOBANU) MOISUC**, *Implementarea, testarea şi evaluarea unui UCP cu sistem de operare de timp real înglobat*, „Dunarea de Jos" University of Galati, Galați, România, 5-7 Noiembrie 2015, PERFORM, proceedings.