An overview regarding the improvement of the nMPRA architecture

Nicoleta Cristina GAITAN

***Abstract*— The nMPRA (Multi Pipeline Register Architecture) architecture was developed to improve the real time response in applications developed on microcontroller. This article presents some improvements of the nMPRA architecture performances on response time to the occurrence of events and the implementation of a scheduling algorithm.**

***Index Terms*—nMPRA architecture, events, real time, microcontroller**

I. INTRODUCERE

SISTEMELE înglobate de timp real sunt utilizate la ora actuală pentru o multitudine de aplicații cum ar fi și acelea pentru control sau pentru procesarea de date. Sistemele de timp real (STR) sunt acele sisteme care trebuie să reacţioneze dinamic la schimbările de stare ale mediului, ale cărui evoluţii depind de comportamentul uman, de fenomene naturale sau artificiale, sau de procesele industriale din întreprinderi. Este binecunoscut faptul că îndeplinirea unor constrângeri de timp de tip deadline este o caracteristică specifică sistemelor de timp real. Pentru îndeplinirea acestui deziderat este de preferat utilizarea unui sistem de operare de timp real (RTOS). Utilizarea acestuia nu conduce implicit la faptul că sistemul își va respecta deadline-urile sale. Aceasta depinde și de modul în care sistemul per ansamblu este proiectat și structurat.

În prezent, pentru dezvoltarea sistemelor înglobate (embedded) sunt folosite procesoare de uz general [1]. Acestea pot avea un comportament non-deterministic, nefiind eficiente în dezvoltarea sistemelor de timp real. Datorită acestor probleme, dezvoltatori de sisteme embedded pot să supradimensioneze necesitățile de calcul și să folosească procesoare cu o putere de calcul mai mare decât este necesar pentru a garanta fără probleme timpul critic (deadline) pentru cazurile cele mai defavorabile (Worst Case Execution Time - WCET). De asemenea ele pot avea un consum foarte mare raportat la performanţele pe care aceste procesoare le oferă. Pe de altă parte, dezvoltarea explozivă a dispozitivelor FPGA [1][2] a permis dezvoltarea de controlere specializate care pot garanta dealine-urile la un consum mic de energie [3][4]. De asemenea, s-a permis dezvoltarea de sisteme pe chip (System on Chip - SOC) care să aibă implementate în hardware primitivele de sincronizare și de comunicație inter task specifice unui sistem de operare în timp real [5][6].

În orice sistem de operare de timp real problemele majore le reprezintă comutarea taskurilor, întreruperile, sincronizarea și comunicația între procese cât și planificarea taskurilor. Implementarea prin software a acestor mecanisme conduce la generarea unor întârzieri semnificative pentru unele aplicații. nMPRA este o arhitectură concepută pentru implementarea unor microcontrolere înglobate de timp real și care suportă execuţia concurenţială a *n* taskuri, permiţând comutarea foarte rapidă între acestea cu o întârziere de uzual 1 ciclu mașină și maxim 3 cicli maşină pentru instrucțiunile de lucru cu memoria [7][8]. Acest lucru se datorează faptului că fiecare task are propriul Program Counter, set de regiștrii pipeline și fișier de regiștrii generali. Arhitectura nMPRA [6] rezolvă la nivel hardware aceste probleme. Primele versiuni ale nMPRA au fost definite în [7] și [8]. Proiectul se bazează pe arhitectura MIPS modificată, acesta implică un PC, un set de 5 regiştrii pipeline și un fișier de regiștrii (register file) care conservă starea unui task la comutarea către alt task. Acest proiect înlocuiește politica de salvare pe stivă cu un algoritm de remapare care permite noului task să se execute chiar din ciclu mașină următor.

II. ARHITECTURA NHSE

nHSE (Hardware Scheduler Engine), ilustrat în Fig. 1 și prezentat în [6][9] este un automat cu stări finite (Finite State Machine - FSM) care are ca intrări evenimente de tip întrerupere, deadline, watchdog timer, mutex, evenimente de tip mesaj, eveniment de auto susținere a execuției, precum și semnale de validare ale planificatoarelor statice și dinamice cât și de inhibare în cazul execuției instrucțiunilor de tip load și store, și care generează semnalele de activare ale sCPUi.

Schema bloc, ilustrată în Fig. 1, conține registrul de memorare a ID-ului sCPU-ului activ împreună cu logica de sincronizare, planificatorul static, planificatorul dinamic și logica aferentă evenimentelor.

Planificatorul static este orientat pe priorități și este dezactivat la punerea sub tensiune. Planificatorul dinamic prevede un registru de prioritate. Pe baza acestui registru orice

Manuscript received December 2, 2014. This work was supported in part by the project "Sustainable performance in doctoral and post-doctoral research PERFORM–Contract no. POSDRU/159/1.5/S/138963", project co-funded from European Social Fund through Sectorial Operational Program Human Resources 2007-2013.

N. C. Gaitan is with the Computers, Electronics and Automation Department, Faculty of Electrical Engineering and Computers Science, Stefan cel Mare University of Suceava, 13 University street ROMANIA, (e-mail: cristinag@eed.usv.ro).

sCPUi poate avea o prioritate în gama 1 până la n-1, cu 1 prioritatea cea mai mare și n-1 prioritatea cea mai mică. Fiecare sCPU are un identificator fix care este un număr întreg de la 0 la n-1.

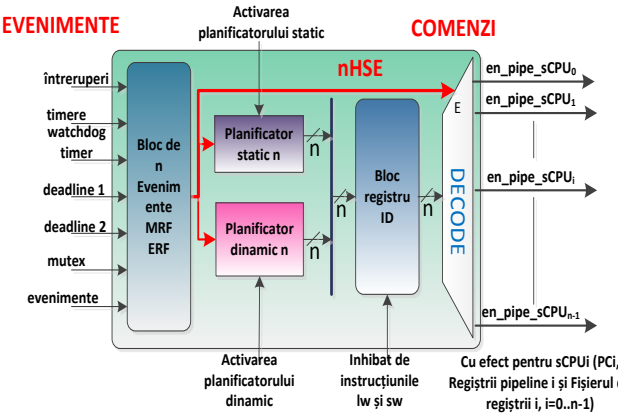

Fig. 1. Arhitectura nHSE

Identificatorul din registrul ID selectează care sCPU este activă dacă există un eveniment activ. Altfel, sistemul este în idle. El va ieși din această stare la apariția oricărui eveniment. Dacă taskul asociat sCPU-ului la care a apărut evenimentul achită evenimentul, acesta se va autosuspenda. În cazul în care dorește continuarea execuției, taskul trebuie să activeze evenimentul de autosusținere.

Fiecare sCPUi (fiecare task i) are un fișier de regiștrii organizat pe bancuri, care conține *b* seturi de regiștrii generali, așa cum se prezintă în Fig. 2. Aceste *b* bancuri de regiștrii au fost prevăzute pentru a accelera procesul de apelare și revenire din procedură. La un apel de procedură se va folosi bancul imediat superior iar la revenirea din procedură se va folosi bancul imediat inferior. Apelurile cuibărite (nested) și revenirile din procedură trebuie să fie pereche. Un banc de regiștrii conține 32 de regiștrii organizați ca 18 regiștrii de uz general, 4 regiștrii speciali și 10 regiștrii pentru parametrii.

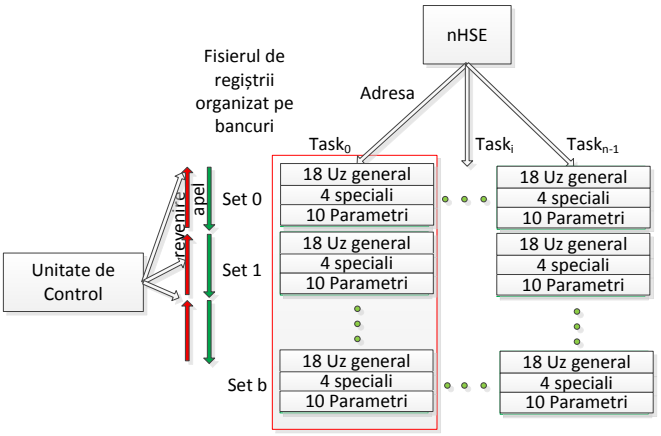

Fig. 2. Organizarea în bancuri a fișierului de regiștrii

Unitatea de control [8] este responsabilă de decodarea instrucțiunilor "apel" și instrucțiunile de "revenire" și de generarea semnalelor de control adecvate pentru fișierul de regiștrii. Ori de câte ori semnalul de control pentru instrucțiunea "apel" este activat, fișierul de regiștrii mapează cadrul următor care va fi utilizat pentru noua funcție și copie conținutul a 10 regiștrii parametri și PC-ul în regiștrii corespunzători noului context mapat. Acești regiștrii parametri sunt utilizați ca intrări-ieșiri, ceea ce înseamnă că o instrucțiune de "revenire" le pot folosi pentru a transfera date în contextul funcției de revenire. Schimbările de context a funcției pot fi utilizate numai de către unitatea de control prin activarea semnalelor de control adecvate și redirecționarea acestora în regiștrii pipeline. Remaparea contextelor unui task este realizată numai de către HSE (Hardware Scheduler Engine) în același timp cu remaparea întregului set de regiștrii

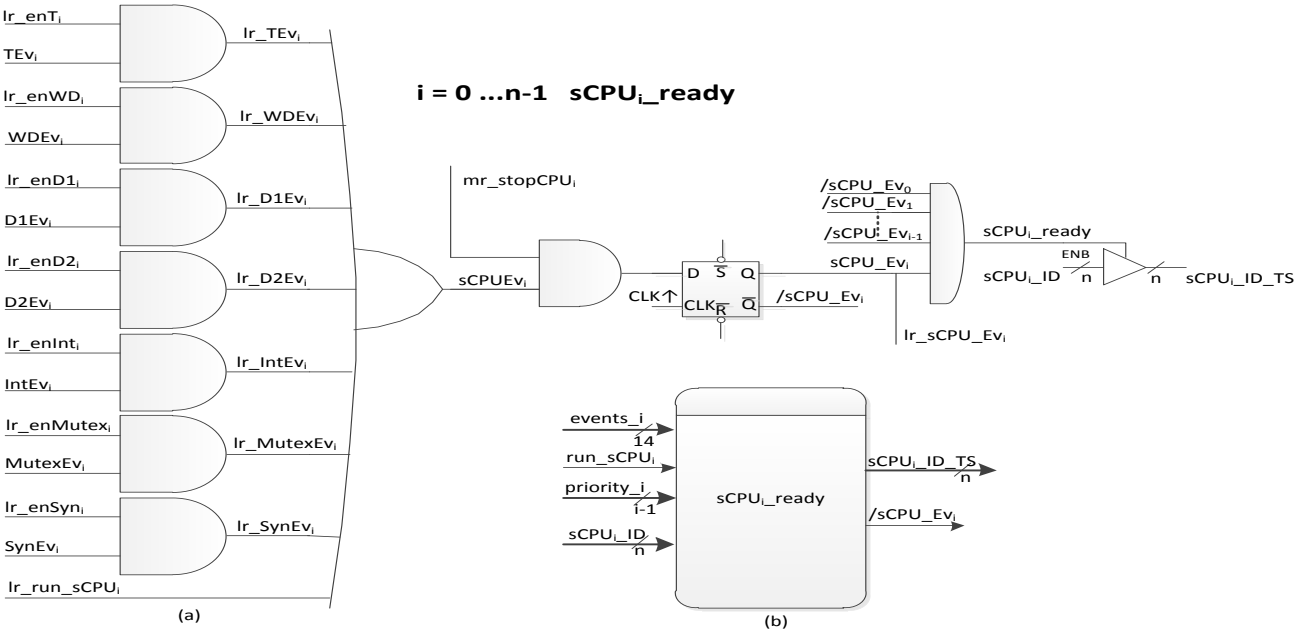

Fig. 3. Planificatorul hardware sCPUi (blocul nHSE) – (a) logica digitală pentru starea readly, (b) diagrama bloc [6][9]

pipeline pentru acest task.

Schema logică a evenimentelor la nivelul fiecărei sCPUi este prezentată în Fig. 3 [6][9]. Structura hardware a planificatorului la nivelul fiecărei sCPUi este ilustrată în Fig. 3a. Planificatorul se bazează pe apariția unor evenimente și așteptarea lor de către sCPUi. Evenimentele posibile pentru sCPUi sunt: întreruperea dată de timer (TEvi); întreruperea dată de watchdog timer (WDEvi); cele două întreruperi de semnalizare preventivă a expirării timpului pentru deadline (D1Evi și D2Evi); întreruperile atașate (IntEvi); mutex-urile folosite pentru accesul la resurse partajate (MutexEvi); evenimetele de sincronizare și comunicare între sCPUi (SynEvi) și lr_runs_CPUi care indică autosusținerea în execuție a sCPUi curentă.

Validarea evenimentelor așteptate se realizează cu instrucțiunea *wait*. Formatul instrucțiunii *wait* este *wait Rj* care blochează banda de asamblare în așteptarea unui eveniment care a fost selectat prin setarea lui în registrul Rj. Rj este automat transferat în registru task register. Instrucțiunea wait este foarte puternică deoarece permite sincronizarea execuției în același timp cu mai multe evenimente. Sub controlul software-ului, urmând o prioritate dată de sarcinile sCPUi (taskului i), aceste evenimente sunt tratate și achitate. Majoritatea RTOS moderne au implementate multe mecanisme de partajare a resurselor și pentru sincronizarea și comunicația între taskuri, dar au funcții care le apelează individual. De exemplu nu putem aștepta în același timp o întrerupere, eliberarea unui semafor și primirea unui mesaj.

III. O PRIVIRE DE ANSAMBLU ASUPRA ÎMBUNĂTĂȚIRII PERFORMANȚELOR NMPRA

În [6] se prezintă o arhitectură nouă denumită nMPRA, care conține o implementare originală a unei structuri hardware utilizată pentru planificarea statică și dinamică a taskurilor, gestiunea unitară a evenimentelor și întreruperilor, accesul la resursele partajate, generarea evenimentelor, și o metodă de asignare a întreruperilor la taskuri, care asigură o operare eficientă în contextul controlului de timp real. Scopul acestei noi arhitecturi este acela de a îmbunătății prin hardware performanțele sistemului de operare de timp real pentru microcontrolere privind comutarea între taskuri, timpul de răspuns la evenimente externe, comportarea la întreruperi, primitivele de sincronizare între procese (Inter Process Communication – IPC) cum ar fi evenimetele (events), mutex-urile (mutexes), mesajele (messages), etc.).

În [10] autorii extind schema de prioritizare globală prezentată în [11], cu scopul de a finaliza implementarea schemei de tratare a evenimentelor în hardware pentru arhitectura nMPRA. Se prezintă regiștrii capcană asociați cu categoriile de evenimente cât și structura registrului Program Counter (PCi). Schema de prioritizare prezentate în [10] permite tratarea evenimentelor în hardware și are avantajul de a reduce timpul pentru a detecta sursa evenimentului, și pentru a începe corespunzător rutina de servire a evenimentelor. Permite chiar și introducerea de noi evenimente în sistem prin adăugarea pur și simplu a câmpurilor necesare în registrul Task Register (TRi), registrul Event Status Task Register (ESTRi) și registrul Event Priority Register (EPRi), actualizarea schemei de prioritizare a unui eveniment global și introducerea unui nou registru capcană pentru fiecare categorie nouă de eveniment. Schema este simplă și poate fi aplicată la toate evenimentele.

În [12] s-a prezentat un algoritm de tipul dual priority care are rolul de a păstra un sistem funcțional chiar dacă acesta comută din starea normal într-o altă stare (diagnoză, service, test). Această trecere poate conduce la situații în care unele taskuri să fie întârziate nepermis de mult cu posibile consecințe grave. Algoritmul, prezentat în [12], asigură execuția fiecărui task chiar și pentru acele stări de funcționare diferite de starea normală, prin schimbarea priorității taskului care depășește cuanta de timp T și plasarea acestora în una dintre cozile iq sau tlq în funcție de lungimea timpului de execuție a taskului. După dispariția modificării Ci, algoritmul revine la funcționarea normală de tip TT (cu declanșare în timp). Utilizarea microcontrolerului nMPRA asigură timpi de comutare foarte buni, între 1 și trei cicli mașină mărind timpul din perioada T la dispoziția taskurilor. nMPRA, prin planificatorul dinamic asigură și o arbitrare automată a priorităților. Acest timp poate fi și mai mult îmbunătățit pe viitor prin implementarea în hardware a algoritmului dual priority. Analizând o bogată literatură de specialitate nu s-a întâlnit un algoritm asemănător.

În [9], s-a îmbunătățit arhitectura CPU (Central Processing Unit) ce a fost prezentată în [6] referitor la înteruperi. S-a propus o soluție inovativă pentru a prioritiza întreruperile atașate unui task. Față de soluția de testare în buclă a întreruperilor, soluția propusă oferă un timp de răspuns uniform pentru orice întrerupere dacă la momentul respectiv este singura întrerupere activă. În plus, soluția propusă în [9] asigură și prioritizarea fixă a întreruperilor.

IV. CONCLUZII

În final, putem concluziona că arhitectura nMPRA propusă în [6] este una foarte puternică pentru că: fără nici un RTOS software poate implementa aplicații de timp real doar cu instrucțiunile de la nivelul limbajului de asamblare; comutarea între taskuri este întârziată cu un ciclul mașină și maxim 3 cicli mașină în cazul execuției unor instrucțiuni de scriere în memoria globală; 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.

Arhitectura nMPRA (Multi Pipeline Register Architecture) a fost dezvoltată pentru a îmbunătății timpul de răspuns în timp real a aplicațiilor dezvoltate în jurul microcontrolerelor. În acest articol au fost prezentate câteva îmbunătățiri privind performanțele arhitecturii nMPRA în ceea ce privește timpul de răspuns la apariția evenimentelor cât și implementarea unui algoritm de planificare.

Pe viitor se pot găsi și dezvolta soluții de îmbunătățire a

timpului de răspuns la întreruperi în timp real pentru microcontrolerele ce se bazează pe arhitectura nMPRA (Multi Pipeline Register Architecture).

MULȚUMIRI

Această lucrare a beneficiat de suport financiar prin proiectul "Performanta sustenabila in cercetarea doctorala si post doctorala - PERFORM", Contract nr. POSDRU/159/1.5/S/138963", proiect cofinantat din Fondul Social European prin Programul Operational Sectorial Dezvoltarea Resurselor Umane 2007-2013.

BIBLIOGRAFIE

[1] Shawash, J.; Selviah, D.R.. Real-Time Nonlinear Parameter Estimation Using the Levenberg–Marquardt Algorithm on Field Programmable Gate Arrays. Industrial Electronics, IEEE Transactions on , vol.60, no.1, pp.170,176, Jan. 2013.

[2] Shahbazi, M.; Poure, P.; Saadate, S.; Zolghadri, M.R.. FPGA-Based Reconfigurable Control for Fault-Tolerant Back-to-Back Converter Without Redundancy. Industrial Electronics, IEEE Transactions on , vol.60, no.8, pp.3360,3371, Aug. 2013.

[3] Shahbazi, M.; Poure, P.; Saadate, S.; Zolghadri, M.R.. Fault-Tolerant Five-Leg Converter Topology With FPGA-Based Reconfigurable Control. Industrial Electronics, IEEE Transactions on , vol.60, no.6, pp.2284,2294, June 2013.

[4] Tran, T.; Ohishi, K.; Yokokura, Y.; Mitsantisuk, C. FPGA-based High-Performance Force Control System with Friction-Free and Noise-Free Force Observation. Industrial Electronics, IEEE Transactions on , vol.PP, no.99, pp.1,1, 0, 2013.

[5] Liu, I., Reineke, J., & Lee, E. A. (2010, November). A PRET architecture supporting concurrent programs with composable timing properties. In Signals, Systems and Computers (ASILOMAR), 2010 Conference Record of the Forty Fourth Asilomar Conference on (pp. 2111-2115). IEEE.

[6] GAITAN, Vasile Gheorghita; GAITAN, Nicoleta Cristina; UNGUREAN, Ioan. CPU Architecture based on a Hardware Scheduler and Independent Pipeline Registers. In: IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS. 2014. ISSN 1063-8210. DOI:10.1109/TVLSI.2014.2346542.

[7] E. Dodiu, V.G. Gaitan and 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, Croatia, 24 May 2012, ISSN: 1847-3946, ISBN 978-953-233-069-4.

[8] E. Dodiu and V.G. Gaitan. Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers – concept and theory of operation. 2012 IEEE EIT International Conference on Electro-Information Technology, Indianapolis, IN, USA, 6-8 May 2012, ISBN: 978-1-4673-0818-2, ISSN: 2154-0373.

[9] GAITAN, Nicoleta Cristina; GAITAN, Vasile Gheorghita; MOISUC, Elena-Eugenia Ciobanu. Improving interrupt handling in the nMPRA. In: Development and Application Systems (DAS), 2014 International Conference on. IEEE, 2014. pp. 11-15.

[10] E.E. (Ciobanu) Moisuc, Al. B. Larionescu, I. Ungurean. Hardware Event Handling in the Hardware Real-Time Operating Systems. 18th International Conference on System Theory, Control and Computing, October 17-19, 2014, Sinaia, Romania.

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

[12] GAITAN, Nicoleta Cristina; ANDRIES, Lucian. Using Dual Priority scheduling to improve the resource utilization in the nMPRA microcontrollers. In: Development and Application Systems (DAS), 2014 International Conference on. IEEE, 2014. pp. 73-78.