طراحی CPU بر اساس زمان بندی سخت افزاری و ثبات های خط لوله مستقل

چکیده: سوئیچ کردن(تعویض) وظایف، هماهنگ‌سازی و ارتباطات بین فرایندها، برای هر سیستم عامل بی درنگ، مسائل مهمی می‌باشند. اجرای نرم‌افزارهای خاص ، ممکن است منجر به تاخیرهای قابل‌توجهی شود که می‌توانند الزامات بی ضرب الاجل(deadline) را برای برخی از برنامه‌های کاربردی تحت تاثیر قرار دهند. ین مقاله، یک معماری زمان‌بندی سخت‌افزار یکپارچه در ساختار CPU را ارائه می‌دهد که فن‌آوری ایجاد نقشه‌ی جدید منبع را برای ثبات‌های خط لوله pipeline و برای ثبات‌های در حال کار CPU استفاده می‌کند. ما یک اجرای اصلی از ساختار سخت‌افزاری استفاده شده برای زمان‌بندی دینامیکی (پویا) و استاتیک وظیفه، را برای مدیریت واحد، برای دسترسی به منابع با اشتراک گذاشته شده‌ی معماری ، ایجاد رویداد، و برای روش استفاده شده برای اختصاص دادن وقفه‌هایی برای وظایفی که یک عملیات کارآمد را در قشر یا زمینه‌ی کنترل بی درنگ بیمه می‌کند ارائه می‌کنیم. یک دستورالعمل همگذار برای هماهنگ‌سازی وظیفه‌ی همزمان با چندین منبع رویداد استفاده می‌شود. این معماری، زمان سوئیچ کردن وظیفه از یک چرخه‌ی ساعت (با بدترین حالت سناریوی 3 چرخه‌ی ساعت بری دستوالعمل‌های خاص ستفاده شده برای دسترسی‌های حافظه‌ی خارجی) و یک زمان پاسخ از تنها 1.5 چرخه‌ی ساعت برای رویدادها را میسر می‌سازد. برخی مکانیسم‌ها برای بهبود سرعت اجرای برنامه در نظر گرفته می‌شوند.

عبارات شاخص: زمان‌بندی سخت‌افزاری، میکروپردازنده‌ها و میکروکامپیوترها، سیستم بی درنگ و سیستم‌های جاسازی شده

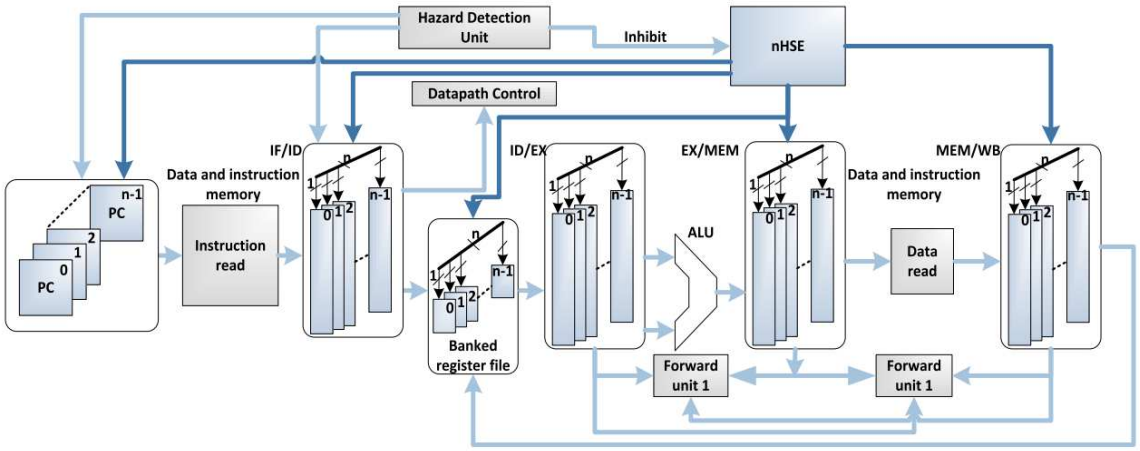
1. مقدمه

استفاده از سیستم عامل‌های بی درنگ اختیاری و تجاری کنونی(RTOSs) برای سیستم‌های جاسازی شده، از نظر ما دو مسئله عمده را ایجاد می‌کند. در حالی که یکی به دستگذار (یک فایلی یا تسهیلاتی که وقفه را بعهده دارد) وقفه اشاره دارد، دومی به این حقیقت اشاره می‌کند که یک وظیفه نمی‌توان بطور همزمان با رویدادهای استفاده شده برای

هماهنگ‌سازی، اشتراک‌گذاری منابع، ارتباطات هماهنگ شود. چنین رویدادی، سیگنال‌ها، تیرهای راهنما (semaphores)، پردازنده‌ی کلمات متنی (انحصار متقابلes)، پیام‌ها، پرچم‌ها و سایرین خواهند شد.

این مسائل در RTOSs شناسایی شدند که در میکروکنترل‌کننده‌ها بدون واحد مدیریت حافظه‌ی مجازی و مخزن حافظه اجرا می‌شوند. مثال‌هایی که از RTOSs پیروی می‌کنند عبارتند از:

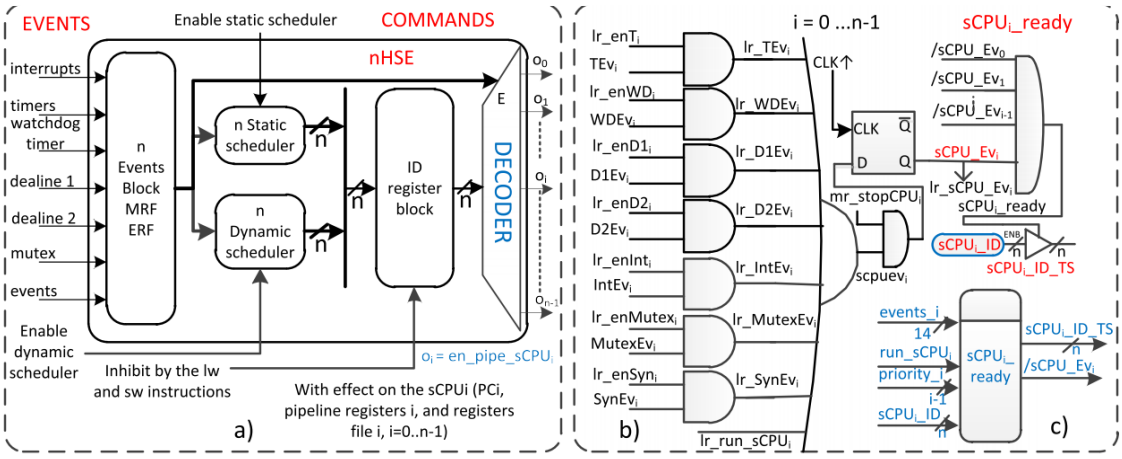
µITRON, µTKernel, µC/OS-II, EmbOS, FreeRTOS, SharcOS, XMK OS, eCOS, Erika, Hartik, KeilOS



شکل 1. معماری nMPRA مرحله‌ی رمز گشایی دستورالعمل واکشی دستورالعمل-PC, IFID . مرحله‌ی رمز گشایی دستورالعمل ID/EX، مرحله‌ی حافظه-اجرا EX/MEM، مرحله‌ی نوشتن برگشتی-حافظه MEM/WB

اولین مسئله، بخصوص با استفاده از جریان‌های عادی سرویس وقفه‌ی تولید شده، حرکت نامنظم اتفاقی(jitter) می‌باشد. به دلیل اینکه، آن برای محاسبه مشکل است، یک جزء مهمی از سیستم‌های بی درنگ می‌باشد. این ممکن است منجر به فقدان بی ضرب الاجل (deadline) شود. دومین مسئله، گسترش یافتن زمان اجرای کار می‌باشد. این بسط، با استفاده از فراخوان‌های پی‌درپی توابع واسط برنامه‌نویسی (API) برنامه‌ی RTOS برای تشخیص رخداد یکی از رویدادهای بالا ایجاد می‌شود. موضوع مهم دیگر، صرف زمان توسط RTOS برای سوئیچ کردن قشر یا قشری وظیفه می‌باشد (سوئیچ قشری، یک عملیات انجام شده توسط زمان‌بندی RTOS می‌باشد که نیاز به زمان زیادی دارد). بعلاوه فراخوان‌های تابع API ، مصرف کننده‌ی زمان می‌شوند، بخصوص اگر پردازنده نیاز به انتقال از حالت کاربر به حالت ناظر یا بالعکس داشته باشد. پردازنده‌های همه منظوره‌های کنونی برای سیستم‌های جاسازی شده استفاده می‌شوند

اما آنها می‌توانند مشکلاتی را به دلیل عملکرد غیرمحتمل و صرف انرژی ناکارآمد ایجاد کنند. به منظور پرهیز از چنین مشکلاتی، فن‌آوری‌های طراحی محتاطانه‌ایی ممکن است اتخاذ شود. این فن‌آوری‌ها می‌توننند پلت‌فرم (سطوح) بسیار بزرگی را ایجاد کنند که قادر به رفتار مناسب تحت بدترین شرایط می‌باشند. بعنوان یک نتیجه، استفاده از این پردازنده‌ها، کاربردپذیری را محدود کرده است و آنها برای سیستم‌های جاسازی شده با ویژگی‌های زمان وقعی سخت و الزامات یا تقاضاهای مصرف انرژی پایین نامناسب هستند. از طرف دیگر، اخیراً، دستگاه‌های آرایه‌ی (FPGA) قابل برنامه‌ریزی میدان با ارزش‌های کارامدتر و با ظرفیت معادل در مدخل‌های منطقی (بیش از میلیون) گسترده می‌شوند. به این دلیل ما یک پشتیبان سخت‌افزاری را بر اساس سیستم‌های FPGA پیشنهاد می‌دهیم.



شکل 2. معماری nHSE. (a). زمان‌بند سخت‌افزاری سطح sCPUi(b) منطق دیجیتالی برای حالت آمده (b) نمودار بلوکی

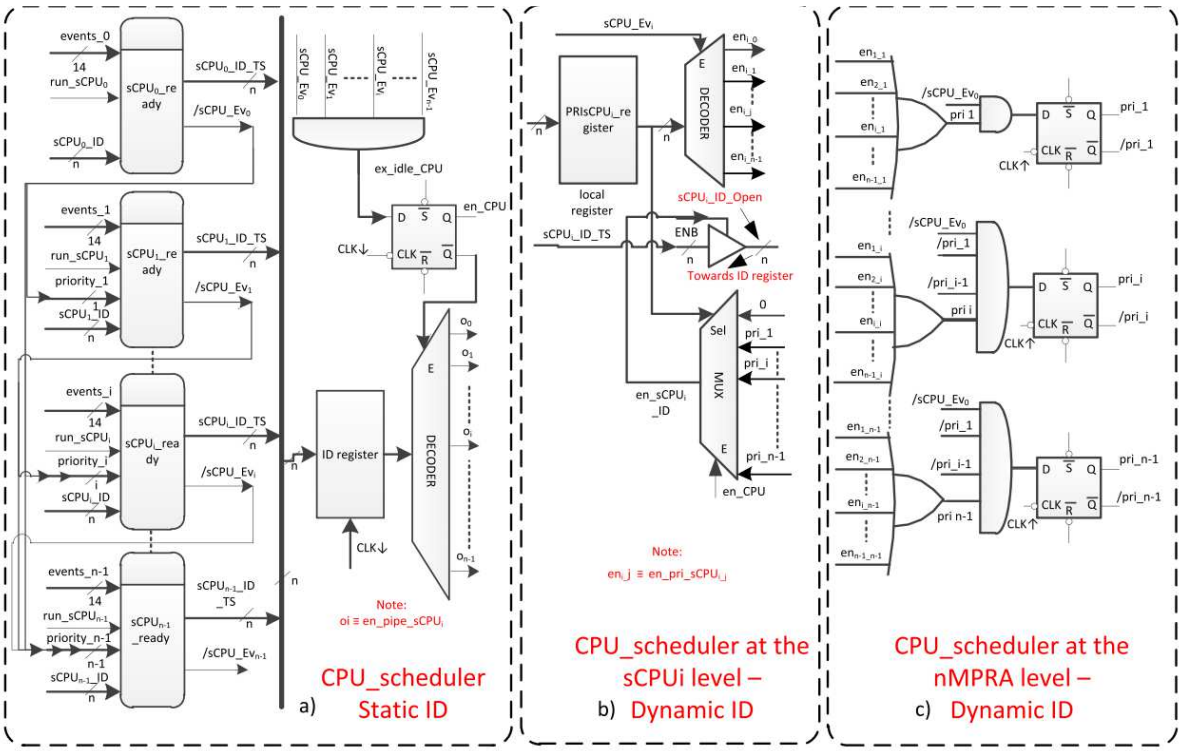
ما یک معماری زمانبندی سنتی را ارائه می‌کنیم که یک طراحی سخت‌افزاری با تکرار منابع می‌باشد (شمارنده‌ی برنامه (PC)، ثبات‌ی خط لوله (pipeline)، و ثبات‌ی همه منظوره‌ی CPU) چنانکه در مراجع 6 و 7 تعریف شده‌اند. معماری ما بر اساس پردازنده‌ی میکرو بدون معماری مراحل خط لوله‌ی در هم قفل شده (MIPS) می‌باشد که بطور خاص برای پشتیبانی عملیات زمان‌بندی سخت‌افزاری بعنوان بخشی از CPU آن اتخاذ شد. آن بک مجموعه از 4 ثبات‌ی خط لوله را برای هر وظیفه که برای نگهداری دستوالعمل‌های در حال اجرای CPU استفاده شدند بکار می‌رود. فایل ثبات برای هر وظیفه یا کار تکرار می‌شود. این امر موجب سوئیچ کردن سریع قشر یا زمینه، به سادگی با استفاده از ایجاد نقشه‌ی

مجدد از قشر یا زمینه‌ی فعال برای اجرا شدن می‌شود. این معماری، که در مرجع 7 معماری ثباتی چند خط لوله‌ایی (multipipeline) نامیده شد (MPRA)، روش های ذخیره‌ی پشته (stack saving) را با الگوریتم ایجاد نقشه‌ی جدید(remapping) جایگزین می‌کند این الگوریتم قادر به اجرای شروع وظیفه‌ی جدید با چرخه‌ی ساعت بعدی می‌باشد.

معماری جدید به صورت زیر مشخص می‌شود:

آن شامل یک پیاده‌سازی اصلی ساختار سخت‌افزاری استفاده شده برای زمان‌بندی دینامیک (پویا) و استاتیک وظایف می‌باشد، آن قادر به مدیریت واحد رویدادها و وقفه‌ها می‌باشد، آن دسترسی به منابع به اشتراک گذاشته را فراهم می‌کند و هچمین روش استفاده شده برای اتصال وقفه‌ها به وظایف فراهم می‌کند بنابراین یک عملیات کارامد را در زمینه‌ی الزمات یا تقاضاهای بی درنگ ایجاد می‌کند.

هدف از طراحی جدید، بهبود عملکردهای میکروکنترل‌کننده‌های RTOSs می‌باشد. علکردها مربوط به موارد زیر هستند:



شکل 3. زمان‌بند سخت‌افزار کلی (a)زمان‌بند استاتیک (b) حمایت برای زمان‌بند دینامیک (پویا)

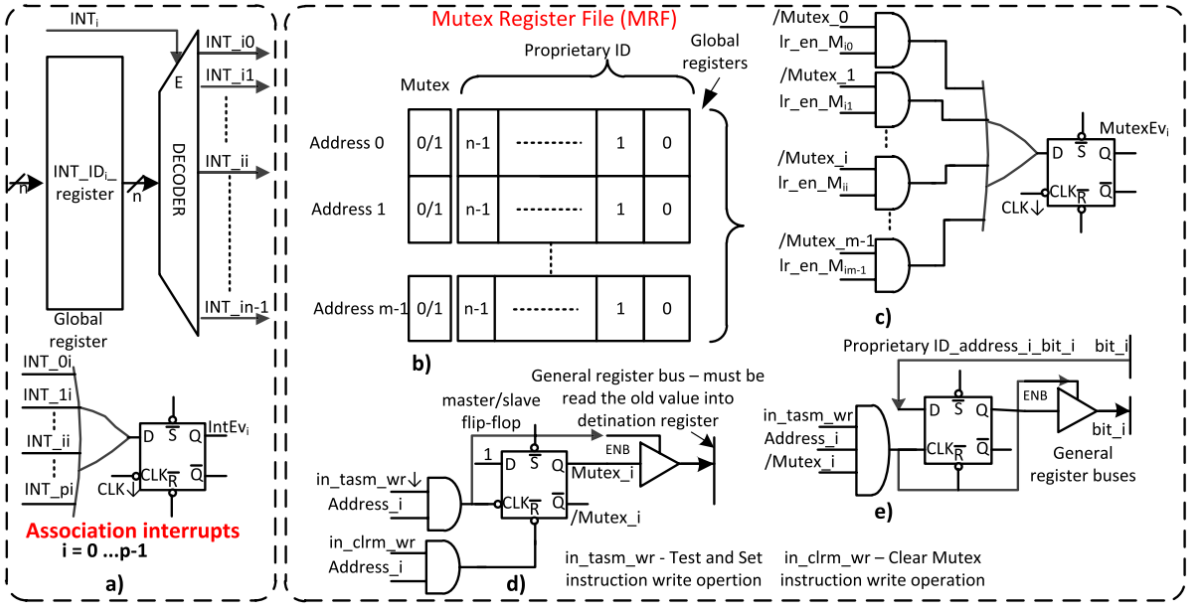
زمان سوئیچ کردن وظایف، زمان پاسخ به رویدادهای خارجی، رفتار وقفه‌ها، و زمان اجرای هماهنگ‌سازی ارتباطات داخل فرایندی (IPC) عناصر اولیه‌ی یک برنامه (رویدادها، پیام‌ها، انحصار متقابلes(پردازنده‌ی متنی) و غیره)

این مقاله بصورت زیر سازماندهی می‌شود:

معماری nMPRA، در بخش II ارائه می‌شود و معماری nHSE و از جمله تمام تسهیلات RTOSs در بخش III ارائه می‌شود. بخش IV یک سری آزمایشات را در طوا پیاده‌سازی معماری یا طرح پیشنهادی ارائه می‌دهد. بخش V شامل کار مرتبطو مقایسه با معماری nMPRA می‌باشد.در نهایت نتایج در بخش VI آورده می‌شود.

II معماری nMPRA

ما به معماری پیشرفته بعنوان nMPRA (بری n وظیفه) اشاره خواهیم کرد در حالی که زمان‌بندی تعبیه شده، موتور زمان‌بندی سخت‌افزاری برای n وظیفه نامیده خواهد شد(nHSE). طراحی nMPRA در شکل 1 نشان داده شده است. قبل از شروع توصیف طرح، ما نمادهای زیر را تعریف خواهیم کرد: یک مثالی از CPU، CPU نیمه (sCPUi برای وظیفه‌ی i)نامیده خواهد شد. چنین مثال سخت‌افزاری، شامل ثبات PC خود، ثبات‌های خط لوله، فایل ثبات و



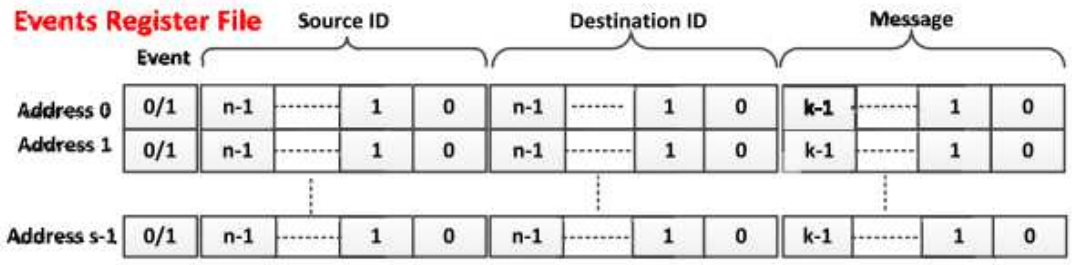
شکل 4. (a) پیاده‌سازی وقفه‌ها (b)-(e) پیاده‌سازی mutexes

ثبات‌های کنترل خود می‌باشد. آن همچنین دستوالعمل‌های وظیفه‌ی i (i = 0,..., n − 1) ر اجرا می‌کند. تمام بخش‌های sCPUi، سایر اجزای خط لوله‌ی پردازندهبه جزء sCPU0 ، که تنها واحد فعال بعد از تنظیم مجدد می‌باشد یکسان است. تنها sCPU0، ثبات‌های پیکربندی و نظارت nMPRA را میسر می‌سازد (مفید برای نظارت منبع و زمان‌بندی).

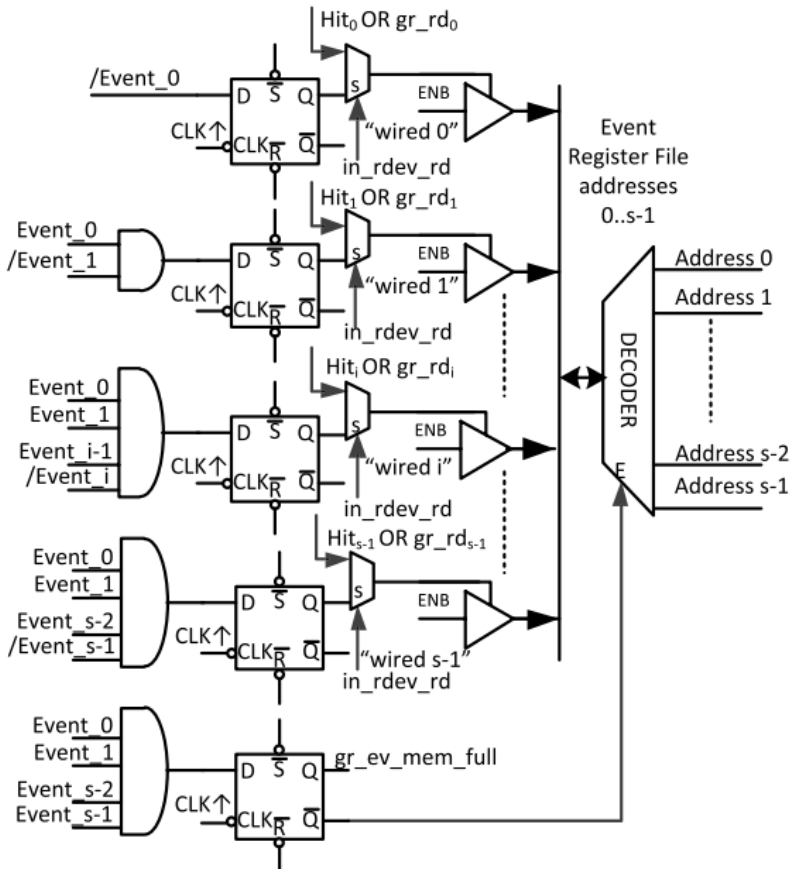
nMPRA جدید با هر دوی زمان‌بندی دینامیک و استاتیک ارائه می‌شود. زمان‌بندی ستاتیک با الویت‌های استاتیک، انحصاری می‌باشد.بعنوان یک تازگی، زمان‌بندی می‌تواند عملیات سوئیچ کردن را در رخداد یکی از رویدادهای زیر سریع انجام دهد (این حالت خاص طراحی nMPRA می‌باشد):

زمان، وقفه، تایمر نگهبان، انحصار متقابل، ضرب الاجل(deadline)، وظیفه‌ی داخلی intertask، ارتباطات و فعال-سازی اجرا. انتخاب این رویدادها می‌تواند با استفاده از یک دستورالعمل همگذاری ساده بدست بیاید. هر یک از این رویدادها می‌تواند از طریق ثبات کنترل، فعال یا غیرفعال شوند.این ویژگی مشابه، برای زمان‌بندی دینامیک یا پویا بکار می‌رود.

با توجه به زمان‌بندی دینامیک، برای هر sCPUi (با استثنای sCPU0) که تغییر الویت وظیفه را تحت کنترل یک الگوریتم زمان‌بندی دینامیک(پویا) میسر می‌کند، یک ثبات کنترل وجود دارد. تمام رویددهای متصل به وظیفه، الویت جدیدی را به ارث می‌برند. برای وقفه‌های سخت‌افزاری، در اینجا هیچ کنترل‌کننده‌ی خاصی وجود ندارد اما طرح، یک طرح توزیع شده‌ایی دارد که اتصال وقفه‌ها را به هر وظیفه میسر می‌سازد. رخداد این وقفه،

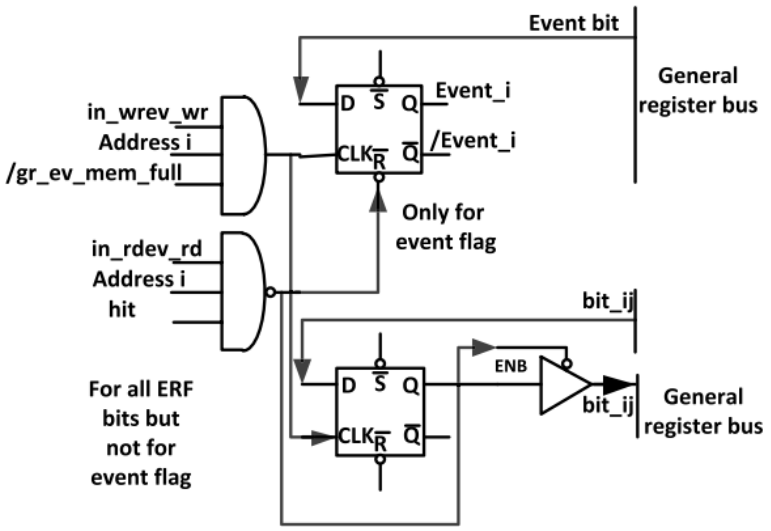


شکل 5. فایل ER

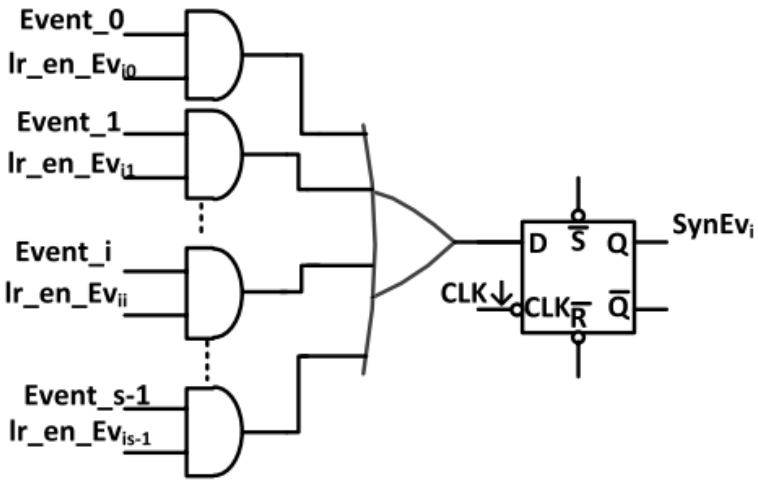


شکل 6. سخت‌افزار بری تولید خودکار رویداد اختیاری بعدی

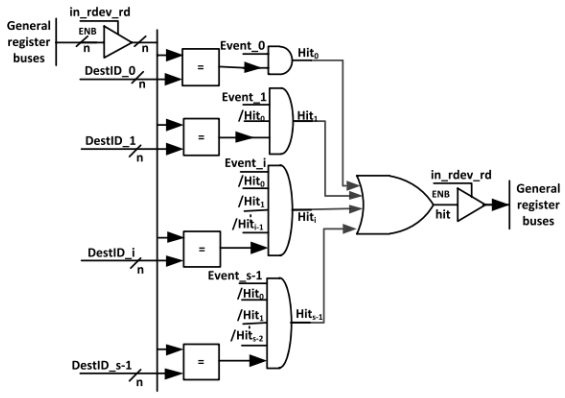
قشرها یا زمینه‌های وظایف را تحت تاثیر قرار نمی‌دهد (آن، ثبات‌های خط لوله را برای وقفه‌ها از خط مونتاژ خط لوله ثبات نمی‌کند) بعلاوه، در طول ثبات، یک وقفه ممکن است دوباره به وظیفه‌ی دیگر با استفاده از عملیات WRITE تک برای یک ثبات ،توزیع شود. nMPRA برای پاسخ به محدوده‌ی وسیعی از رویدادهای بی درنگ مانند رخداد بی درنگ، و دو رخداد ضرب الاجل (ولی معادل با یک هشدار و دومی معادل با یک خطا است) طراحی شد. بعنوان یک ویژگی جدید معماری، ما می‌توانیم رخدادهای نوع ضرب‌الاجل را نشان دهیم که می‌توانند



شکل 7. خواندن و نوشتن از EPR



شکل 8. فعال/غیر فعال شدت رویداد sCPUi



شکل 9. خواندن محتوای Eri بر اساس اصل CAM

به الگوریتم زمان‌بندی بری رسیدن به ضرب‌الاجل‌ها کمک کنند. به منظور دسترسی کنترل به منابع به اشتراک گذاشته شده، nMPRA، انحصارهای متقابل را در سخت‌افزار اجرا می‌کند. انحصارهای متقابل، در یک فایل ثبات انحصار متقابل (MRF) گروه‌بندی می‌شوند این فایل از یک سری از ثبات‌هایی که شامل بیت انحصار متقابل در بیت معنی‌دار ثبات و شناسه‌ی منحصر به فرد (ID) وظیفه‌ی اختصاصی ذخیره شده در بیت‌های پایین‌ترین الویت هستند تشکیل می‌شود. مزایای این طرح این است که یک دستورالعمل همگذار تک، برای بدست آوردن یک انحصار متقابل، سریعتر از یک فراخوان تابع API داخل RTOS اجرا شده در نرم‌افزار استفاده می‌شود.

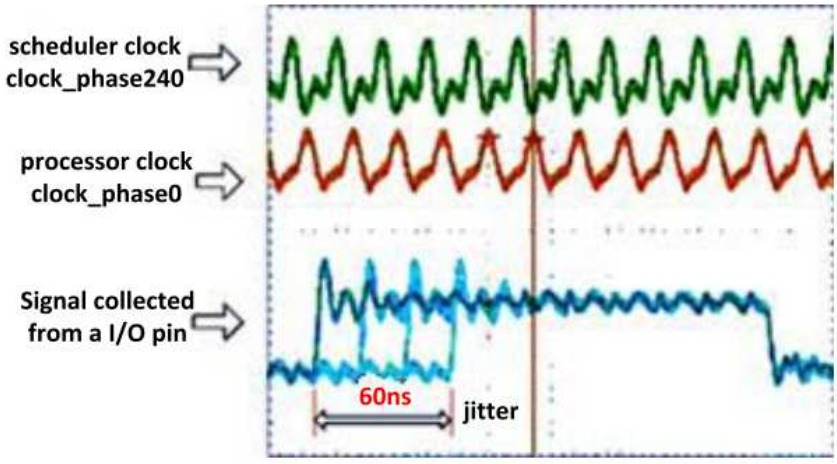
با توجه به ICP، یک واحد شامل ثبات‌های رخداد (ERs) ارائه می‌شود. این ثبات‌ها، وضعیت رخدادها را در بلاترین موقعیت بیت ذخیره می‌کنند.بیت‌های بعدی ID وظیفه‌ایی را که رویداد را فعال می‌کند نگه می‌دارند، بیت‌های بعدی ID وظیفه‌ی مقصد را بری اینکه رخداد تعیین شود نگه می‌دارند و در نهایت بیت‌های آخری بعنوان یک زمینه‌ی پیام استفاده می‌شوند که برنامه‌ریز می‌تواند برای هر هدفی بکار گرفته شود. اگر رخداد اختیار شده، متعلق به وظیفه‌ایی با بالاترین الویت باشد، یک عمل تعویض زمینه در چرخه‌ی ساعت بعدی انجام می‌شود. باز هم مزایای طرح، استفاده از دستوالعمل همگذار تک برای فعالسازی یک رویداد می‌باشد که سریعتر از فراخوان تابع API داخل RTOS اجرا شده در یک نرم‌افزار می‌باشد. در نتیجه اگر الویت بالا باشد، روند تعویض وظیفه‌ی جدید بسیار سریع می‌شود (1-3 چرخه‌ی پردازنده). این رخداد تمام زمان مورد نیاز را برای جستجوی رخداد اختیاری حذف می‌کند. اگر رخداد i توسط یک رخداد آگاهی داده شود، پیدا کردن منبع این رویداد مهم می‌شود. اما جستجوی فایل ER (ERF) برای یافتن منبع، ممکن است یک مقدار غیر قبولی از زمان اتخاذ کند. برای جلوگیری

Number of tasks n	*Memory required for pipeline registers*	*Memory required for general registers*	*Total memory for replicated resources*
8	0.59kB	256B	0.84kB
16	1.18kB	512B	1.68kB
32	2.36kB	1024B	3.36kB

جدول 1. لزامات حافظه برای منابع تکرار شده‌ی پردازنده‌ی nMPRA

Architecture / Producer / Model	*Working RAM memory*
ARM Cortex M4/ ST Microelectronics/ STM32F427IG	256kB
Tricore / Infineon / Aurix 29X	780kB
SuperH / Renesas / R5S726A1P216FP	1366kB
SuperH / Renesas / R5S72681W266FP	2624kB

جدول 2. حافظه‌ی کار برای برخی از میکروکنترل‌کننده‌ها از طبقه‌ایی با عملکرد بالا



شکل 10 نامنظمی حرکات اتفاقی وظیفه‌ایی با بالاترین الویت برجسته شده در ارتباط با رخداد رویداد و اغاز اجرای وظیفه‌ایی با بالاترین الویت (منظور سیگنال افت کننده است)

از این مشکل، هر زمانی که دستورالعمل READ جرا شود، جستجو در یک چرخه‌ی پردازنده بر اساس اصل حافظه قابل آدرس دهی با محتوا (CAM) انجام می‌شود.

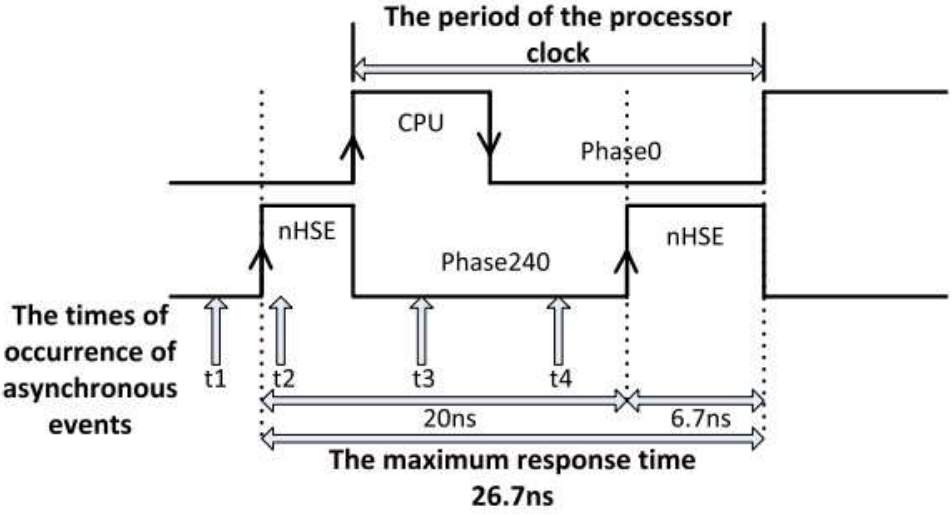
همانطور که در شکل 1 می‌توان دید برای هر CPU، یک مجموعه‌ی مستقل ثبات‌های خط لوله (IF/ID, ID/EX, EX/MEM, MEM/WB) ، یک PC، یک فایل ثبات و یک مجموعه‌ایی از ثبات‌های خاص (که در nHSE یافت می‌شوند) وجود دارد. با توجه به این حقیقت که هر وظیفه‌ایی، مجموعه‌ایی از ثبات‌های خط لوله و ثبات‌های کار خود را برای هدف کلی دارد، تعویض زمینه‌ی وظیفه می‌تواد در یک چرخه‌ی ساعت تک انجام شود و پاسخ به یک رویداد خارجی می‌تواند بعد از 1.5 چرخه بدست آورده شود. به این دلیل ما می‌توانیم استدلال کنیم که معماری بسیار سریع است چنانکه آن، وظایف را قادر به تعویض با یک حداقل تاخیر از 1 یا 1.5 چرخه‌ی پردازنده و یک حداکثر تاخیر از 3 چرخه‌ی پردازنده(برای دستورالعمل‌های حافظه‌ی در حال کار) سازد. منابع دیگر توسط تمام وظایف به اشتراک گذاشته می‌شوند. شمارنده‌ی تنبل و ترتیبی (run) بری هر sCPUi در داخل sCPU0 اجرا می‌شوند. این حالت ، ارزیابی دقیق و دوره‌ایی ضرب‌الاجل را میسر می‌سازد. این انتخاب، احتمال اجرا در نرم‌فزار الگوریتم‌های زمان‌بندی دینامیک یا پویا در sCPU0، معمولاً اختصاص داده شده به بالاترین الویت را در نظر می‌گیرد.

در ارائه‌ی nMPRA، ما موارد زیر را تعریف می‌کنیم: ثبات‌های کنترل با سطح دسترسی وظیفه‌ی خاص cr برای sCPUi، ثبت‌های محلی (lr_) (که بخشی از فضای خصوصی هر sCPUi می‌باشند)، ثبات‌های کلی (gr_)(که بخشی ز فضای آدرس کلیnMPRA هستند و توسط تمام sCPU قابل دسترسی می‌باشند) و ثبات‌های نظارتی (mr_) (که می‌توانند تنها توسط sCPU0 قابل دسترسی شوند).

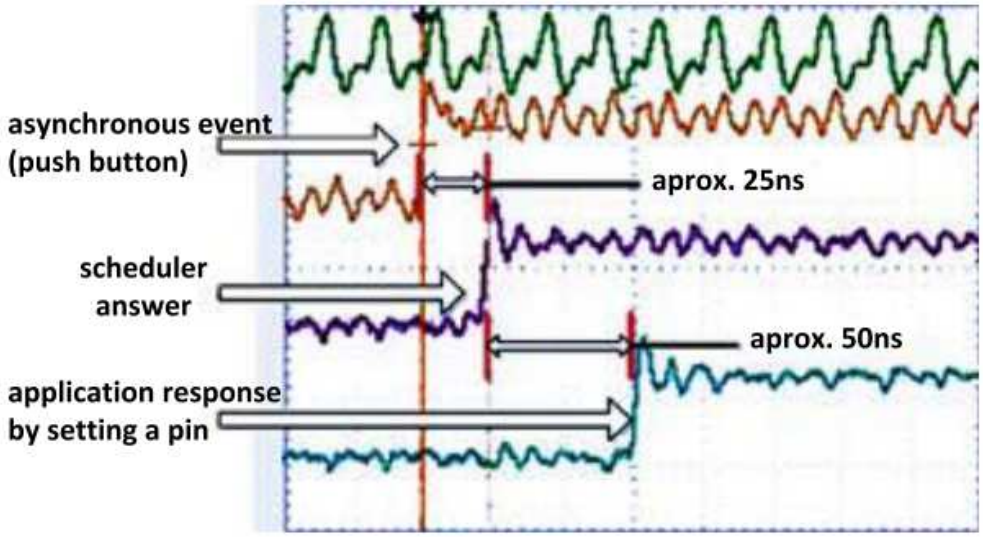
III معماری nHSE

nHSE (شکل 2a) یک ماشین حالت متناهی است که ورودی‌هایی برای رویداد دارد مانند وقفه‌ها، ضرب‌الاجل‌ها، تایمر دیده‌بان، تایمرها، انحصارهای متقابل، پیام‌ها و اجرای خود کفا شده (موقتاً بیکار). بعلاوه nHSEسیگنال‌هایی توانا برای زمان‌بندی استاتیک و دینامیک دارد و از سیگنال‌های تولید شده توسط اجرای دستورالعمل‌های ذخیره و بارگذاری جلوگیری می‌کند. خروجی‌ها، سیگنال‌های فعال‌سازی sCPUs تولید شده توسط nHSE می‌باشند. تنها یک چنین

خروجی می‌تواند در یک حافظه‌ی داده شده فعال باشد (همانطور که در شکل 2a دیده شد oi ≡ en_ pi pe_sC Pui). نمودار بلوکی(نمایش کلی مدار) شامل ثبات sCPU ID همراه با منطق هماهنگ‌سازی ، زمان‌بندی استاتیک، زمان‌بندی دینامیک و بلوک مربوط به رویداد می‌باشد. هر sCPU، ID منحصر به فردی دارد که برای مثال یک عدد صحیح از 0 تا n-1 می‌باشد. اگر n=4 باشد ما 4 ID 0، 1، 2، و 3 داریم. ثبات ID، sCPU فعال را در رخداد یک رویداد شناسایی می‌کند. اگر هیچ رویدادی فعال نباشد، سیستم در حالت تنبل (غیر



شکل 11. پاسخ زمان‌بند بسته به یک رویداد غیر همزمان



شکل 12. پاسخ نرم‌افزر و زمان‌بند HSE برای یک روبداد غیر همزمان

فعال)(idle) می‌باشد. زمای که رویداد رخ می‌دهد سیستم فعال می‌شود اما اگر sCPU متصل شده، رویداد را آشکار کند، آنگاه وظیفه، خود کفا شده خواهد شد. اگر اجرا ادامه داده شود، sCPU باید رویداد خود کفا شده را (شکل 2b و 2c) قبل از آشکار شدن رویداد رخ داده، فعال کند.

زمان‌بندی استاتیک، وظیفه‌گرا می‌باشد. الویت هر sCPUi، i می‌باشد، مشابه با sCPUi ID، این بذان معنی است که الویت‌ها در طول اجرای وظایف ثابت هستند. زمانی که پردازنده به یک منبع انرژی متصل است، زمان‌بندی استاتیک، غیر فعال است. زمان‌بندی دینامیک با یک ثبات الویت (PR) برای هر sC Pui ، n-1 , . . . , 1 = i ارائه می‌شود . بر اساس این ثبات‌ها، هر sCPUi ممکن است یک الویت در مقیاسی از 1 تا n-1 داشته باشد جایی که n-1 پایین‌ترین الویت می‌باشد.

sCPU0 معمولاً با بالاترین الویت (0) تعیین می‌شود و نمی‌تواند تغییر کند. الویت sCPU0 می‌توانند با استفاده از الگوریتم زمان‌بندی پویای اجرا شده هم در نرم‌افزار، در سطح sCPU0، و هم در سخت افزار، بصورت دینامیکی تغییر کند. زمانی که پردازنده به یک منبع انرژی متصل است، زمان‌بندی دینامیک غیر فعال است. در این حالت، تنها sCPU0 فعال باقی می‌ماند. رخداد یک رویدادی (اگر معتبر باشد)، در سطحی از یک sCPU0 نشان داده می‌شود. اگر sCPU0 بالاترین الویت را داشته باشد، زمان‌بند، اجرایش را معتبر خواهد کرد. در این حالت، هم سیستم، از حالت بیکار (idle) (غیر فعال) خود خارج می‌شود و هم sCPU دیگر با الویت‌های پایین‌تر حذف متوقف می‌شوند. بعلاوه، ما جزئیات تولید رویدادها و طراحی nHSE را ارائه می‌کنیم.

A. زمان‌بند استاتیک سخت‌افزاری

Nakano و همکارانش اشاره کردند که زمان‌بند سخت‌افزاری، عملکرد خط لوله‌ی CPU را تنزل می‌دهند. در حال حاضر، معماری با در نظر گرفتن تمام این معایب، در ارتباط با خط لوله‌ی CPU طراحی شده است. ساختار سخت‌افزار متعلق به هر sCPUi (تعبیه شده در بلوک منطقی nHSE) در شکل 2b نشان داده شده است. زمان‌بند، بطور مداوم رویدادهایی که با sCPU همراه می‌شوند را نظارت می‌کند.رویدادهای احتمالی sCPUi هستند: وقفه‌های تایمر (Tevi)، تایمر دیده‌بان (WDEvi)، دو وقفه‌ی استفاده شده برای سیگنال‌دهی ضرب‌الاجل (D1Evi and D2Evi)، وقفه‌های

متصل شده (IntEvi)، انحصارهای متقابل (MutexEvi)، هماهنگ‌سازی و رویدادهای ارتباطی وظیفه‌ی داخلی(SynEvi) و اجرای خود کفا برای sCPUi کنونی (lr_run_sC Pui).

هر زمان که یک منبع تولید یک رویداد یا یک وقفه آشکار شود، sCPUi کنونی ممکن است کنترل CPU را از دست دهد. رویدادهای فوق می‌توانند با استفاده از سیگنال‌های lr_enD1i, lr_enD2i, ،lr_enTi, lr_enWDi lr_enInti, lr_enMutexi, و lr_enSyniمعتبر شوند. تنها استثنا، lr_run_sCPUi می‌باشد. این سیگنال‌ها باید در یک ثبات ویژه، ثبات وظیفه‌ی نام برده (TR) ذخیره شود. سیگنال sCPUiEvi که برای سیگنال رخداد یک رویداد مورد انتظار استفاده می‌شود، با استفاده از سیگنال mr_stopCPUi فعال می‌شود. این بخشی از یک ثبات نظارتی است که تنها برای sCPU0 دردسترس می‌باشد.

sCPU0، تنها واحد اجرایی است که قادر به متوقف کردن sCPUi دیگر ($i \neq 0$) می‌باشد. بری هماهنگ‌سازی ما از فلیپ فلاپ D استفاده می‌کنیم که اطلاعات در مورد یک رویداد در حال تعلیق (pending) با افزایش زمان CPU ذخیره می‌کند. سیگنال‌های sCPUi در حال تعلیق، حاکی از کنترل در تمام sCPUi ID (sCPUi_ID) می‌باشد.این عمل با استفاده از نوشتن مقدار در گذرگاه (باس) نتیجه‌ی زمان‌بند انجام می‌شود،به شرطی که در اینجا، هیچ وظیفه‌ایی در اجرا وجود نداشته باشد، داشتن الویت بالاتر نسبت به sCPUi. این عمل با ستفاده از سیگنال‌های / sC PU_Ev0 sC PU_Evi – 1/ . . . مشخص می‌شود. یک نماینده‌ی بلوکی ساده از زمان‌بند محلی، قبلاً در شکل 2c نشان داده شده است.

شکل 3a، طراحی کلی زمان‌بند استاتیک را نشان ی‌دهد. طرح زمان‌بند نشان داده شده در شکل 3a، شامل، بلوک‌های عملکردی sCPUi_ready (قبلاً ارائه شده)، ثباتی که sCPUi ID بالاترین الویت را ذخیره می‌کند و یک رمزگشایی که sCPUi بالاترین الویت را فعال می‌کند می‌باشد. مدخل AND و فلیپ‌فلاپD از طرح، زمانی فعال هستند که در اینجا هیچ sCPUi فعال دیگری وجود نداشته باشد. سیگنال en_CPU بطور عمده می‌تواند بری ذخیره‌سازی انرژی استفاده شود.

فعالسازی یا غیر فعالسازی هر منبع sCPUi می‌تواند با en_pipe_sCPU0، از طریق سیگنال‌های en_pipe_sCPUn-1 انجام شود. در نتیجه، شمای پیشنهادی می‌تواند برای زمان‌بندی استاتیک استفاده شود به شرطی که هر وظیفه در یک sCPUi اجرا شود. در این حالت، الویت‌های استاتیک ، توسط IDهای وظایف شناسایی می‌شوند.

B. زمان‌بند دینامیک (پویای) سخت‌افزاری

تحت شرایط خاص، (برای مثال اگر یک سیاست زمان‌بندی دینامیک استفاده شود)، برخی از الگوریتم‌های زمان‌بندی می‌توانند منجر به بهبود عملکرد شوند. زمان‌بند دینامیک نشان داده شده در شکل 3b و 3c، امکان تنظیم کردن الویت را برای واحدهای زمان‌بندی sCPUi فراهم می‌می‌کند اما هیچ الگوریتم زمان‌بندی خاصی را اجرا نمی‌کند. این معماری سخت‌افزاری می‌تواند برای هر راه‌حل ارائه شئه در مراجع 8-15 بکار برده شود. راه‌حل‌ها می‌توانند شامل کوپردازنده‌های(کمک پردازنده‌ها) (میکروپردازنده‌ها طراحی شده برای تکمیل توانایی‌های پردازنده‌های اولیه) و اجراهای on-die هستند. ما نیاز یه تاکید ان sCPU0 داریم که معمولاً الویت 0 دارد، که آن را بالاترین الویت sCPU از سیستم می‌سازد. برای بقیه ما از یک ثبات خاص (ثبات PRIsCPUi_) استفاده می‌کنیم که الویت sCPUi مربوطه را برای $i = 1$ to $n-1$ ذخیره می‌کند. الویت با استفاده از یک رمزگشا، رمزگشایی می‌شود (شکل 3b) تولید یکی از سیگنال‌های en_pri_sC PUi_1,. . ., en_ pri_sC PUi_n – 1. همانطور که قبلاً گفته شد الویت صفر برای sCPU0 محفوظ می‌ماند. خروجی ثبات مشابه، برای انتخاب خروجی تسهیم‌کننده‌ی MUX از شکل 3 b استفاده می‌شود که نتیجه‌ی طرح الویت‌بندی را از سمت راست شکل جمع‌آوری می‌کند. مدخل OR (شکل 3c) انتخاب الویت را برای هر sCPUi (i = 1,. . ., n) میسر می‌سازد. مدخل AND، یک الویت خاص را معتبر می‌سازد و فلیپ-فلاپ Dبرای هماهنگ‌سازی با ساعت سیستم استفاده می‌شود. خروجی تسهیم کننده، مقدار sCPUi_ID_TS را کهID sCPUi در ورودی ثبات ID می‌باشد معتبر می‌سازد. ثبات ID مشابه قبلاً در شکل 3a توضیح داده شده است؛ ساختار سخت‌افزاری از شکل 3c در پیکربندی مشابه برای زمان‌بند دینامیک استفاده می‌شود.

اعتبارسنجی الویت، توسط سیگنال sCPU_Evi فعال می‌شود، تنها اگر sCPUi، رویداد را پیش‌بینی کند. en_CPU بعنوان یک سیگنال کلی ، بخشی از ثبات نظارتی که تمام sCPUi را به جزء sCPU0 غیر فعال می‌سازد استفاده می‌شود.سیگنال‌های pri_1,. . ., pri_n – 1، همه‌ی الویت‌های n−1 ممکن را نشان می‌دهند. ثبات‌های PRIsCPUi که الویت را ذخیره می‌کنند، می‌توانند بعنوان ثبات‌های محلی برای هر واحد sCPUi (i=0) دسترس شوند. دسترسی می‌تواند هدایت شود یا در حالت ناظر، sCPU0 می‌تواند ثبات‌ها را بخواد و یا بنویسد.

ثبات‌های PRIsCPUi، بطور مختصرPRs، در شکل 3 b نشان داده می‌شوند. این ثبات‌ها می‌توانند در هر sCPUi به جزء sCPU0 یافت شوند. به منظور استفاده از این ثبات‌ها، ما دستورالعمل کنترل Rj انتظار را پیشنهاد می‌دهیم، که برای رخداد هر رویدادی در ثبات Rj انتظار می‌کشد(بایت‌ها در 1 تنظیم می‌شوند). Rj بصورت خودکار به ثبات TR منتقل می‌شود. هر زمان که یک sCPUi ، بعد از یک دستوالعمل انتظار، دوباره اجرا می‌شود، Rj رویدادهای رخ داده را ذخیر خواهد کرد.یک روش کارامدتر و سریعتر، بر استفاده از یک یادمان (mnemonic) اختصاص یافته دلالت می‌کند که رخدادها را بعنوان یک مقدار فوری، رویددهای Rj انتظار، ذخیره می‌کند. رویدادهای مرد انتظار با دستورالعمل انتظار، در ثبات TRبارگذاری می‌شود. زمانی که وظیفه، دوباره استفاده می‌شود، این رخدادها در Rj بارگذاری می‌شود. کا این طراحی را با معرفی یک ثبات جدید به نام ER افزایش می‌دهیم، این ثبات خواندن سیگنال‌های نشان داده شده در شکل 2b را میسر می‌سازد.این، شناسایی رویدادهای رخداده‌ی متصل به sCPUi و بصورت اختیاری، منبع آنها (بدون خواندن ثبات TR) را ممکن می‌سازد. دستورالعمل‌های پیشنهادی برای این ثبات‌ها هستند:

movcr Ri, TRj movcr EV, Rj; movcr Ri, EV;movcr TR, Rj

در روش مشابه‌ایی، برای ثبات‌های PR، ما از دستورالعمل‌های زیر استفاده خواهیم کرد: movcr PR, Rj; movcr Rj, PR . ثبات‌های EV و TR، ثبات‌های کنترل واقع شده در هر sCPUi می‌باشند.

دستورالعمل انتظار، هماهنگ‌سازی اجرا را با چندین رویداد میسر می‌سازد. این رویدادها می‌توانند با کنترل نرم‌فزار با توجه به الویت داده شده توسط sCPUi آشکار شوند. اکثر RTOSs ها ، مکانیسمم‌های داخلی زیادی برای اشتراک-گذاری منبع، ارتباطات وظیفه‌ی داخلی و هماهنگ‌سازی دارند، اما تابعیت(functionality) به توابع‌ایی محدود

می‌شود که ای مکانیسم‌ها را اجرا می‌کند. برای مثال، یک وقفه، از بین بردن semaphore، و ورود یک پیام، نمی‌توانند بطور همزمان منتظر شوند. این راه‌حل پیشنهاد شده توسط این مقاله، بر این مسئله‌ی دقیق نظارت می‌کند.

C. رویدادهای وقفه

وقفه‌ها، مکانیسم‌های بسیار قدرتمندی هستند که اجرای دوره‌ایی، غیر دوره‌ایی، پراکنده‌ی رویدادهای مرتبط به عملیات CPU را میسر می‌سازند . این رویدادها با استفاده از لوازم جانبیON-chip (تراشه‌ی روشن) یا تراشه‌ی خاموش (OFF-chip) یا اجرای یک برنامه راه انداخته می‌شوند. با توجه به روش کلاسیک، کنترل‌کننده‌های (handlers) وقفه معمولاً غیر قابل گسیختن می‌باشد و معمولاً الویت بیشتری را نسبت به وظیفه‌ی بی‌درنگ دارند

یک راه‌حل این است که وقفه‌های handles (دسته‌ها)، بصورت ریسه‌ها در مرجع 16 ارائه می‌شوند. این برای سیستم عاملSun Solaris ارائه شد و آن بر اساس یکی‌سازی ریسه‌ها و وقفه‌ها در یک مدل تک می‌باشد. وقفه‌ها با استفاده از یک اطلاعات محدود به ریسه‌ها تبدیل می‌شوند. این، یک مدل هماهنگ‌سازی تک را در شالوده (kernel) ممکن می‌سازد.

مدل پیشنهاد شده در این مقاله، و توصیف شده در شکل 4 a مشابه با وقفه‌ها مانند روش ریسه‌ها می‌باشد. در این طراحی جدید، وقفه‌ها مانند رویدادی رفتار می‌کنند که به وظایف بی‌درنگ و بنابراین الویت جانشین آنها متصل می‌شوند. سیستم وقفه‌های p را دارد و برای هر یک از آنها، در اینجا یک ثبات کلی(INT_IDi_) با n بیت مفید وجود دارد که ID وظیفه را برای اینکه وقفه پیوسته شود ذخیره می‌کند . فعال‌سازی وقفه‌ی INTi(شکل 4a)، DECODER را تایید می‌کند که یکی از سیگنال‌های I NT _in − 1 ,. . ., I NT _i0 را فعال می‌کند. مدخل OR (شکل 4 a) می‌تواند تمام وقفه‌ها را از سیستم جمع‌آوری کند. اگر تمام ثبات INT_IDi _p(i = 0,. . . , p − 1) با مقدار i نوشته شود، آنها می‌توانند به sCPUi متصل شوند. به همین طریق، اگر هیچ یک از ثبات‌های INT_IDi _p(i = 0,. . . , p − 1) با یک مقدار i نوشته نشوند، هیچ وقفه‌ایی نمی‌تواند متصل شود. نقش فلیپ-فلاپ D هماهنگ‌سازی ظهور تصادفی رویداد وقفه‌ی INTi تولید کننده‌ی IntEvi (شکل 2b و 4a) می‌باشد. این، در تنزل لبه‌ی ساعت سیستم محاسبه می‌شود.

ویژگی‌های جالب و قوی طراحی بصورت زیر می‌باشد:

آن شامل کنترل‌کننده‌ی تخصیصی نمی‌باشد (این وقفه‌ها، الویت وظایف را به ارث می‌برند). یک وظیفه می‌توانند به هیچ، یک، چندین و حتی تمام p وقفه از سیستم متصل شود. یک برنامه‌نویس قادر به تخمین الویت وقفه‌های متصل شده به وظیفه‌ی مشابه می‌باشد. یک وقفه‌ی متصل شده به یک وظیفه می‌تواند یک وظیفه‌ی الویت پیین‌تر را معوق کند، آن هنوز نمی‌تواند اجرای وظیفه را معوق کند برای اینکه آن، به یک وظیفه‌ی الویت بالاتر متصل می‌شود. وقفه می‌تواند یک وظیفه شود یا می‌تواند به یک وظیفه‌ی تک متصل شود. بعلاوه، تمام وقفه‌ها می‌توانندبه یک وظیفه‌ی تک متصل شوند. وقفه‌ها، خط لوله‌ی سایر واحدهای sCPUi را تحت تاثیر قرار نمی‌دهند. این معماری بر ذخیره‌سازی و یا بازگرداندن هیچ زمینه‌ایی دلالت نمی‌کند. وقفه‌ها می‌توانند تو در تو شوند و الویت‌های وقفه‌ها می‌تواند دینامیک شوند (با استفاده از اتصال مجدد به وظیفه‌ی دیگر و یا توسط تغییر الویت وظایف برای اینکه متصل می‌شوند).

راه‌حل پیشنهادی دارای معایبی است مانندتعداد محدود سطوح تو در توی ممکن، محدود به تعداد sCPUs، یا فقدان بردارهای کنترل وقفه. اگر بیشتر وقفه‌ها به یکی از sCPU متصل شوند، مرتبه‌ی کنترل توسط نرم‌افزار تعیین می‌شود و این حالت می‌تواند منجر به تاخیرهای اضافی شود.

D. رخدادهای مربوط به زمان

چنانکه در شکل2 b نشان داده شده است، در اینجا 3 نوع رخداد مربوط به زمان وجود دارد: رخدادهای زمان پیش‌بینی (Tevi) 2- رخدادهای تایمر دیده‌بان(WDEvi) و 3)رخدادهای ضرب‌الاجل(D1Evi معادل است با یک هشدار و D2Evi معادل است با یک نقص). برای اجرا، هر sCPUi، دو تایمر اختصاصی دارد. یکی از آنها، 3 مقایسه‌کننده ,TEvi D1Evi, و D2Evi دارد در حالی که دیگری تنها یک مقایسه‌کننده‌ی استفاده شده برای WDEvi دارد. اگر دیده‌بان بطور دوره‌ایی تجدید نشود، رویداد WDEvi می‌تواند sCPUi را دوباره تنظیم کند(اگر فعال باشد). برای هر دو تایمر، معماری، ثبات‌های محلی دارد (در حافظه‌ی محلی هر sCPUi اجرا شده و با دستورالعمل‌های دسترسی حافظه‌ی نرمال یا طبیعی در دسترس می‌شود). بعلاوه این ثبات‌ها ممکن است بعنوان ثبات‌های نظارتی دیده شوند که می‌توانند با استفاده از sCPU0 با دستورالعمل دسترسی حافظه‌ی طبیعی، در دسترس شوند.

مقادیر ضرب‌الاجل هم با یک الگوریتم محلی بر روی sCPUi و هم با الگوریتم کلی محاسبه می‌شود، الگوریتم کلی در روی sCPU0 یا رویدادی با یک ترکیبی از دو تا اجرا می‌شود . زمانی که یک وظیفه در حال اجرا و یا در حال تعلیق باشد، معماری شامل دو تایمر برای شمارش چرخه‌های CPU می‌باشد، بنابراین یک تابع نرم‌افزاری می‌تواند اجرای یک وظیفه را در sCPUi ردیابی کند. دسترسی به این شمارنده‌ها می‌تواند در روش مشابه با تایمرهایی باشد که قبلاً معرفی شدند.

E. Mutexes (انحصارهای متقابل)

اجرای دو طرفه، یک موضوع مهم در ارتباط با منابع به شتراک گذاشته شده می‌باشد. پشتیبان سخت‌افزاری برای انحصار متقابل، پیشنهاد شده توسط طراحی ما ، در شکل 4b-eشان داده شده است. برای اجرا ما راه‌حل نشا داده شده در شکل 4b-e راپیشنهاد می‌کنیم که یک مجموعه از ثبات‌های کلی با دسترسی سریع را ارائه می‌کند (برای مثال، دسترسی می‌توند در مرحله‌ی اجرای EX اجرا شود.) MRF از یک مجموع از m ثبات ب طول n+1 بیت تشکیل می‌شود. آن شامل حالت انحصار متقابل در قابل‌توجه‌ترین بیت و ID وظیفه‌ی اختصاصی در پایین‌ترین بیت‌ها می‌باشد.ثبات‌های MRF از هر sCPU قابل دسترس شود و بنابراین آنها منابع به اشتراک گذاشته را برای تمام sCPUi هستند. هر sCPUi، یک بلوک سخت‌افزاری دارد چانکه در شکل 4c توصیف شده است که رخدادهای MutexEvi را تولید می‌کند هر زمانی که یک mutex بلوک شده منتشر شود . برای هر sCPUi، تصمیم مربوط به mutex در نظر گرفته شده، بر اساس سیگنال‌های lr_en_M0,. . . , lr_en_Mm – 1 ساخته می‌شود. ای سیگنال‌ها می‌توانند در ثبات‌های محلی به نام ثبات mutex فعال (EMR) ذخیره شوند. در اینجا یک یا چندین ثبات EMRi بسته به تعداد بیت‌های mutex اجرا شده در MRF وجود دارد. فلیپ فلاپ D برای هماهنگ‌سازی با ساعت CPU استفاده می‌شود و اطلاعات داخل عنصر، در لبه‌ی افزایشی ساعت CPU محکم نگه داشته می‌شود. همانطور که در شکل 4d نشان داده شد است، عملیات‌های بلوک(انسداد) و انتشار mutex در یک چرخه‌ی پردازنده‌ی تک منند یک عملیات اتمی انجام می‌شوند. یک تست و دستورالعمل مجموعه سیگنال‌هایTAS ، in_tasm_wr و Address_i) مقادیر قدیمی بیت mutex را از MRF (Mutex_i) و مجموعه‌های بیت Mutex_i می‌خوانند. اگر بیت Mutex_i در 1

تنظیم شود، mutex قابل دسترس در نظر گرفته می‌شود. n بیت آخر از ثبات MRF (سیگنال ID (Address_i دارنده‌ی mutex را نشان می‌دهند. این حالت با استفاده از سیگنال‌های Address_i,in_tasm_wr و Mutex_i از شکل 4e اطمینان می‌یابد. اگر بیت mutex صفر باشد، آنگاه آن در 1 تنظیم می‌شود و sCPUi ID که دستورالعمل را اجرا می‌کند، در ثبات MRF مناسب نوشته می‌شود(شکل 4e برای بیت‌های MRF متفاوت و سپس بیت mutex). اگر mutex خوانده شده توسط دستورالعمل 1 باشد و اگر آن با یک وظیفه تعیین نشود، مقدار خوانده‌شده از ثبات ID، ID دارنده‌ی mutex را نشان می‌دهد. دستورالعمل پاک کردن، Mutex_i را برای مقدار 0 با استفاده از سیگنال-های in_clrm_wr و Address_i (شکل 4d) نتیجه می‌دهد.

در کار با mutexes، دستورالعمل‌های زیر می‌توانند استفاده شوند: tst Rd، Rs(Rs شامل آدرس mutex می‌باشد و Rd شامل مقدار ثبات MRF انتخاب شده می‌باشد)، clrm Ri(Ri شامل آدرس mutex می‌باشد) movmr Rd، Rs (آن ثبات mutex را حرکت می‌دهد- Rs شامل آدرس مرتبط به mutex می‌باشد و Rd شامل مقداری از ثبات MRF بدون تاثیرگذاری مقدار ثبات می‌باشد) movcr EMRi، Ri (آن Ri را در EMRi می‌نویسد) و movcr Ri، EMRi (آن EMRi را در Ri می‌نویسد)ین دستورالعمل‌ها می‌توانند بر روی تمام sCPUi اجرا شوند و به همراه دستورالعمل انتظار ، هماهنگ‌سازی و دسترسی کلی به تمام mutexes را مطئن می‌سازند.

F. ارتباطات و هماهنگ‌سازی وظیفه‌ی داخلی

ارتباطات و هماهنگ‌سازی وظیفه‌ی داخلی، دو جنبه‌ی مهم خاص برای RTOSs می‌باشند. یک مجموعه از ثبات‌های کلی با دسترسی سریع برای پیاده‌سازی مکانیسم رویداد استفاده می‌شود. ERF از یک مجموعه از ثبات‌های (n + k2 + 1) بیت نشان داده شده در شکل 5 تشکیل می‌شود. این ثبات‌ها، حلت رویداد را در بیت بالاترین موقعیت ذخیره می‌کنند. N بیت زیر، ID را وظیفه‌ی منبع را ذخیره می‌کنند که رویداد را فعال کرد؛ n بیت بعدی ، ID وظیفه‌ی مقصد را برای اینکه رخداد نظارت شود نگه می‌دارند؛ در نهایت، k بیت انتهایی بعنوان قابل دسترس زمینه‌ی پیامک برای هر هدف استفاده می‌شوند؛ ERFمی‌تواند با یک رفتار خاص استفاده شود زمانی که یک رویدادفعال می‌شود و با یکی دیگر، زمانی که وظیفه‌ی مقصد، رویداد را به منظور کشف نهایی جا و چه پیامی فرستاده شده است می‌خواند. در

نتیجه، دو دستورالعمل مختلف استفاده می‌شوند. زمانی که یک وظیفه، فعال شدن یک رویداد را می‌خواهد، آن اطلاعاتی را آماده می‌کند که بخشی از یک ER می‌باشد و یک دستورالعمل انتظار را اجرا می‌کند. این وظیفه‌های فعال، بدون نیاز به مشخص کردن آدرس رویداداز ERF جای می‌گیرند.

بلوک (انسداد) سخت‌افزاری در شکل 6 نشان داده شده است و بطور خودکار، آدرس (با شروع از 0) اولین رویداد اختیاری را تولید می‌کند. بعلاوه این سیگنال‌های بلوکی سخت‌افزاری اگر تمام رویدادها فعال باشند(تنظیم کردن در 1). بعد از فعال‌سازی یک دستورالعمل انتظار ، سیگنال in_rdev_rd صفر می‌باشد و تسهیم‌کننده با خروجی‌های فلیپ فلاپ D فعال می‌شود چنانکه در بطوز شماتیک در شکل 6 نشان داده شده است. اگر Event_0 (مقدار ذخیره شده در ER0 در بیت n+k2) صفر باشد، آنگاه سیگنال /Event_0 مقدار منطق 1 دارد و وظایف نوشته شده در فلیپ فلاپ D در ارتباط با Event_0 جای می‌گیرند. اگر Event_0 در صفر تنظیم شود آنگاه Event_0 مقدار منطق 1 را دارد و نوشتن در دیگر فلیپ‌فلاپ D مهار می‌شود (شکل 6). مقدار Event_0 (مقدار منطق 1) در فلیپ فلاپ Dذخیره می‌شود. این سیگنال از طریق تسهیم‌کننده عبور خواهد کرد و 3 حالت بافر را فعال خواهد کرد. مقدار 0 آنگاه در خروجی DEMUX نوشت می‌شود که آدرس 0 را فعال می‌کند. اگر Event_0 مقدار منطق 1 را داشته باشد، فلیپ فلاپ D بعد از عبور از تسهیم‌کنندخ صفر نوشته خواهد شد، 3

Features	***nMPRA***	***hthreads [33]***	***ARPA-MT [30]***	***Kuacharoen [11]***
General register replicating	Yes	No	No	No
Pipeline register replicating	Yes	No	IF and ID	No
Tasks switching speed	1-3 processor cycles	1.7μs – 3.3μs 525-990 cycles (300 MHz)	3μs (72 cycles at 24 MHz)	125 cycles
Coprocessors	No	Yes	Yes	Yes -coupled on external processor bus
HW or HW/SW implementation	HW	HW/SW	HW/SW	HW/SW
Real parallel execution?	No	Yes for hardware threads	No	No

جدول 3. معماری RTOS و پردازنده

Features	*nMPRA*	*hthreads [33]*	*ARPA-MT [30]*	*Kuacharoen [11]*
Support for static scheduling	Yes, HW	Yes, HW	Yes, Cop2-OSC	Yes, HW
Support for dynamic scheduling	Yes, HW	-	Yes, Cop2-OSC	Yes, HW
Algorithms for static scheduling	round robin, priority based scheduling algorithms	FIFO, round robin, priority based scheduling algorithms	Fixed priority policy based on period/ minimum inter-arrival time (RM-Cop2-OSC)	Priority based, RM
Algorithms for dynamic scheduling	Yes, SW	-	Dynamic priority policy based on absolute deadline (EDF Cop2-OSC)	EDF
Scheduler activation	HW - distributed	external FPGA, HW, SW	Coprocessor - Cop2-OSC	FPGA external, HW,SW
The number of scheduled tasks	32	256 SW, 256 HW	128	-

جدول 4. مقایسه‌ی زمان‌بندها

Features	*nMPRA*	*hthreads [33]*	*ARPA-MT [30]*	*Kuacharoen [11]*
Specialized or distributed controller?	Distributed	Bypass Interrupt Scheduler(CBIS) - specialized	Coprocessor (Cop0-MEC)	Specialized
Interrupts as threads?	Yes	Yes	-	Yes
Interrupts can be attached to any task?	Yes	Yes, but a task can have attached a single interrupt	-	Yes
Interrupts inherit the task priority?	Yes	Yes	-	Yes
Interrupts affect pipeline?	No	Yes	-	Yes
Interrupt requires soft-ware context saving?	No	Yes	-	Yes

جدول 5. مقیسه‌ی کنترل‌کننده‌های وقفه

حالت بافر را مهار خواهد کرد و آدرس 0 را غیرفعال خواهد کرد. حرکت بر روی دومین فلیپ‌فلاپ D، یک مقدار منطقی از 1 برای Event_0، تجزیه وتحلیل Event_1 را معتبر خواهد کرد. روش مشابه برای تمام سیگنال‌ها تا زمان رخداد s−1 استفاده می‌شود.

در این حالت تمام رویدادها فعال هستند، دخل AND از فلیپ فلاپ D آخری یک سیگنال معتبر را تولید خواهد کردو این بدان معنی است که هیچ رویداد اختیاری (free) ترک نمی‌شود و در این حالت،a gr_en_mem_full تولید می‌شود. gr_en_mem_full همچنین توسط دستورالعمل‌های READ و WRITE استفاده می‌شود (همچنان که در شکل 7 نشان داده شده). به منظور سیگنالی که بیت‌های رخداد مشغول هستند.

شکل 7 تمام سیگنال‌های عمومی مورد نیاز برای WRITE (سیگنال‌های in_wrev_wr, Address_i and gr_ev_mem_full/) رخداد i مربوط به فلیپ‌فلاپ D و همچنین بیت‌های دیگر (bit_ij) ثبات Eri را نشان می‌دهد. عملیات A READ ،با فعال‌سازی سیگنال‌های in_wrev_wr, Address_i و hit بطور خودکار فلیپ فلاپ D را مجددا تنظیم خود کرد که مربوط به یک رویداد می‌شود،.اعتبار سازی این 3 مدخل حالت، خواندن مقدار بیت‌های bit_ij دیگر را میسر می‌سازد که بخشی از ثبات Eri می‌باشند.

این روش فعال‌سازی رویداد، تمام زمان لازم برای جستجوی یک رخداد اختیاری را حدف می‌کند. ثبات‌های ERF می‌توانند از هر sCPUi در دسترس شوند؛آنها منابع به اشتراک‌گذرده شده‌ایی برای تمام واحدهای sCPUi هستند. هر sCPUi یک بلوک سخت‌افزاری در Nhse دارد(چنانکه در شکل 8 نشان داده شده است). که برای تولید سیگنال-های SynEvi (شکل 2b) هر زمان که یک رخداد مورد انتظار فعال می‌شود استفاده می‌شود. با کمک سیگنال‌های lr_en_Evi0, . . . , lr_en_Evis – 1، هر sCPUi تصمیم می‌گیرد که رخداد مورد نظر قرار گرفته می‌شود. این سیگنال‌ها در ثبات رخداد فعال (EER) ثبات‌های محلی ذخیره می‌شوند. در اینجا یک یا چندین ثبات EERi می‌تواند بسته به تعداد رویداد اجرا شده در ERF وجود داشته باشد. یک فلیپ‌فلاپ D برای هماهنگ‌سازی با ساعت CPU استفاده می‌شود. اطلاعات بر روی لبه‌ی افزایشی ساعت CPU ذخیره می‌شود.

اگر یک وظیفه‌ی i((sCPUi) با استفاده از یک رویداد بیدار شود، آن برای شناسایی منبع ان رخداد مهم می‌باشد. جستجوی ERF برای یافتن منبع ممکن است یک دوره‌ی زمانی غیر قابل دسترس وقت بگیرد. برای پرهیز از این وضعیت، هر زمانی که دستورالعمل READ برای خواندن ثبات Eri از ERFخوانده شود، جستجو بر اساس یک اصل CAM انجام می‌شود همانطور که در شکل 9 مشاهده می‌شود. جستجو با آدرس 0 شروع و در اولین آدرس برای اینکه در انجا یک انطباق بین ID وظیفه‌ی مقصد و ID وظیفه‌ی کنونی وجود دارد پایان می‌یابد. یک دستورالعمل READ، می‌تواند انطباق، آشکار شدن منبع وظیفه‌ی رویداد را شناسایی کند و به سادگی با استفاده از خواندن محتویات ثبات ERF دریابد که کدام پیام ارسال شده است.

خروجی‌های بلوک‌های تسهیم‌کننده شامل ID وظیفه‌ایی می‌باشد که دستورالعمل و مقادیر مقصد . . ,Dest I D_0 Dest I D_s – 1 , . . را اجرا می‌کند (با استفاده از سیگنال in_rdev_rd) . اگر در اینجا انطباق وجود داشته باشد و رویداد فعال باشد، مدخل‌های سخت‌افزار بعدی برای تمام رخدادها با تعداد شاخص بزرگتر انسداد می‌یابند(این روش مشابه آن چیزی است که در شکل 6 نشان داده شده است). در شکل 6. سیگنال in_rdev_rd برای مجبور کردن تسهیم‌کننده برای استفاده از مقدار Hiti OR gr_rdi استفاده می‌شود (ثبات کلی را بخوانید).به جزء برای این ضمیمه، دیدگاه شماتیک نشان داده شده در شکل 6 در حال کار است همانطور که قبلاً توضیح داده شد. توضیحات مربوط به شکل 6 نیز باید دنبال شوند.

بری کار با رویدادها، دستورالعمل Ri رویداد می‌تواند استفاده شود (Ri شامل ID وظیفه‌ی منبع، ID وظیفه‌ی قصد و پیامی که در kk بیت آخر ذخیره می‌شود می‌باشد). بعد اجرا، Riشامل مقدار gr_en_mem_full می‌باشد که قبل از اجرای دستورالعمل، در پایین‌ترین موقعیت بیت بود. اگر این بیت مقدار 1 را داشته باشد، فعالسازی رویداد ب شکست روبرو می‌شود. clrev Ri بعد از اجری دستورالعمل، Ri، محتوایات اولین ثبات ERF را برای اینکه یک انطباق شناسایی شود نگه می‌دارد.اگر بیت رویداد، مقدار منطقی 0 را داشته باشد، این بدان معنا است که رویداد فعال نمی‌باشد. لیست رویداد می‌تواند بطور بازگشتی اسکن شود تا زمانی که تمام منابع رویداد آشکار شوند. این حالتی است برای تمام وظایفی که در حال کنترل یا نگه‌داری رویدادهای چندگانه هستند. دستورالعمل‌های زیر:mover Rd ، Rs (حرکت

ER، جایی کهRs شامل آدرس مربوط به رویداد می‌باشد- زمانی که مدخل‌های gr_rdi فعال شده-و Rd شامل مقدار ثبات ERF انتخاب شده- بدون اثرگذاری مقدار آن می‌باشد)، movcr EERi، Ri(آن Ri را در EERi می‌نویسد) و movcr Ri، EERi(آن EERi ر در Ri می‌نویسد) برای تمام sCPUi قابل دسترس می‌باشند. همراه با دستورالعمل انتظار، آنها می‌توانند هماهنگ‌سازی، دسترسی به رویدادها، و عبور پیام بین وظایف را مطمئن سازند.

G. ملاحظات بیشتر

بعدRESET، در زمان راه‌اندازی، nMPRA ، sCPU0 را فعال می‌کند و تمام sCPUi دیگر را غیر فعال می‌کند. sCPU0 شامل تمام نرم‌افزار قالب‌بندی (فرمت) و توالی‌های startup (برنامه‌هایی که در شروع یک برنامه اجرا می‌شود) بری تمام sCPUi دیگر (i=1, n-1) می‌باشد. sCPU0می‌تواند تمام تابعیت nMPRA مربوط به وقفه‌های سخت‌افزاری را که ممکن است منجر به خطاهای مهلک شود مورد عمل قرار دهد. PCs برای هر sCPUi با آدرس یک حلقه‌ی تله(trap loop) بارگذاری خواهد شد جایی که کنترل منتقل می‌شود. از این نقطه نظر، برنامه می‌تواند برای مکان‌های کدی مختلف به منظور اجرای نرم افزار جهش یابد و می‌تواند برای تابعیت خاص برای هرsCPUi توانا شود. هر sCPUi می‌تواند هم به کد یا رمز محلی و هم حافظه‌ی داده و به یک حافظه‌ی خارجی دسترسی یابد.

IV. اجرا و اعتبازسازی معماری nMPRA

این بخش، عملیاتی را توصیف می‌کند کهبرای آزمایش و اعتبارسازی nMPRA بکار رفته شدند. معماری در یک Virtex-6 FPGA ML605 ارزیابی Kit—Xilinx اجرا و اعتبارسازی شده است. رمز یا کد پردازنده‌ی در زبان توصیف سخت‌افزار مدار یک یکپارچه‌ی سرعت بسیار بالای استاندارد (VHDL) توصیف شده است. پردازنده‌ی nMPRA برای یک فرکانس کار 50 MHz اجرا شد. معماری nMPRA، 4 ثبات خط لوله (شامل ثباتPC) در مجموع 606 بیت را استفاده می‌کند. رونوشت این منابع برای 16 وظیفه، 1.18 KB از RAM را مورد نیاز می‌کند.

اگرچه، nMPRA یک معماری است که منابع به اشتراک گذاشته شد را درگیر می‌کند، هزینه‌های آن بسیار کارامدتر از معماری‌های تجاری دیگر است. باید توجه کرد که چنین اجراسازی برای یک تعداد معقولی از وظایف مناسب

می‌باشد.برای یک تعداد بزرگ، فرکانس به میزان قابل توجهی ، به دلیل ترکیب منطق با زمان‌های انتشار بالا ،بطور غیر معقولانه افزایش خواهد یافت.

جدول 1 حافظه‌ی مورد نیاز بری اجراسازی احتمالی 8، 16 و 32 sCPUs را ارائه می‌دهد. این جدول دارای به هدف نمایش مصرف حافظه برای نسخه‌های اجراسازی نهایی می‌باشد. با در نظر گرفتن داده‌های جدول I و II و به دلیل اینکه امروزه میکروکنترل‌ها از صدها RAM KB استفاده می‌کنند ما ممکن است بطور قطعی بپذیریم که الزامات حافظه‌ی مورد نیاز برای اجراسازی پردازنده‌ی nMPRA قابل دسترس‌تر می‌باشند. ما مثال میکروکنترل Renesas SuperH R5S72681W266FP را که RAM 2.6 MB برای استفاده‌ی کلی دارد، بدون در نظر گرفتن حافظه‌ی مورد نیاز برای اجراسازی پردازنده و سایر اجزای سخت‌افزاری ادغام شده با تراشه (chip) اتخاذ کردیم. در Virtex-6 FPGA ML605 تعداد ثبات‌های برشی، 14 334 می‌باشد و تعداد LUTs برشی، 17 749 برای 8 sCPU می‌باشد و تعداد ثبات‌های برشی 27 374 می‌باشد و تعداد LUTs برشی 33 571 برای 16 sCPU می‌باشد.

به منظور آزمایش عملکرد این پردازنده‌ها، ما یک برگرداننده همگذار را توسعه دادیم. این نرم‌افزارهمچنین از فایل‌های VHDL برای اعتباردهی اپوکد دستورالعمل‌ها استفاده می‌کند. این ابزار زمان استفاده شده توسط بازگرداننده در فرایند اضافه کردن دستوالعمل‌های جدید به فایل‌های VHDL را بهینه می‌کند. در موقعیت کنونی، یک کد ممکن است تنها در همگذار (برنامه‌ی مترجم) نوشته شود در حالی که بازگرداننده(یا مترجم) کد ماشین را به عنوان یک فایل نوشته شده‌ی VHDL تولید خواهد کرد. توسعه‌ی یک برنامه‌ی جدید به دلیل این حقیقت ضروری است که معماری پیشنهادی ، مجموعه دستورالعمل پردازنده‌ی MIPS را بسط می‌دهد. بعد از آزمایش عاملیت‌های پردازنده، ابزارهای گردآوری MIPS سنتی می‌توانند برای توسعه‌ی برنامه‌های بی‌درنگ استفاده شوند. اگر داده‌ها از طریق محیط حافظه‌ی عمومی تبادل یابند، ردیابی حضور دستورالعمل‌های ذخیره و بارگذاری در خط مونتاژ انجام می‌شود. تنها در این حالت، و در طول اجرای این دستورالعمل‌ها، واحد موتور زمان‌بندی سخت‌افزاری(HSE) برای عملکرد تعویض زمینه اجازه داده نمی‌شود(بنابراین داده‌ها بطور ثابت اطمینان داده می‌شوند. در ین زمینه، ما آزمایش زیر را ارئه می‌کنیم که تایید می‌کند این وضعیت، منجر به بالاترین چرخه‌ی تعویض- سه چرخه‌ی ساعت می‌شود.

آزمایش برنامه شامل دو وظیفه‌ی نوشته شده بصورت وظیفه‌ی 0 و وظیفه‌ی 1 می‌باشد. وظیفه‌ی 0، آغاز می‌شود که یک شماره‌شناسی شخصی (پین) I/O را در یک تنظیم می‌کند و یک رویداد را انتظار می‌شود(یک وقفه‌ی دوره‌ایی بوسیله‌ی یک تایمر تولید می‌شود). زمانی که رخداد روی می‌دهد ، وظیفه‌ی0 I/Oرا با مقدار صفر و برخی از دستورالعمل‌های هیچ عملیاتی (NO) را تنظیم می‌کند؛ آن پین را با مقدار 1 تنظیم می‌کند آن به حالت انتظار برای رویداد تولید شده‌ی تایمر بر می‌گردد. وظیفه‌ی 1، حداقل الویت آغاز می‌شود و در یک حلقه وارد می‌شود که شامل دستورالعمل وع sw می‌باشد.

در اینجا یک الویت بالایی از بکار گماشتن در حالت قفل وجود دارد که بطور دقیق jitter را از شکل 10 توضیح می‌دهد(یک حداکثر ضرب‌الجل 60 نانوثانیه- سه چرخه‌ی ماشین) چنانکه از شکل 10 ، زمانی که MPRA در فرکانسی از 50 Hzکار می‌کند ، حداکثر ضرب‌الاجل زمان‌بند 60 ns می‌باشد . فرکانس در شکل با $1/\Delta t$ ثبت می‌شود در حالی که دوره‌ی ساعت سیستم $\Delta t = 20\ ns$ می‌باشد. ساعت از کانال 1، فاز ساعت 1 می‌باشد و برای هماهنگ کردن حافظه و ثبات خط لوله استفاده می‌شود؛ ساعت از کانال 4 ، برای هماهنگ کردن زمان‌بند HSE استفاده می‌شود (ساعت فاز 240).

شکل 11 زمان پاشخ سیستم را برای یک وقفه‌ی خارجی نشان می‌دهد. در این برنامه، ثبات‌های ستفاده شده برای تنظیم یک پین، تنظیم و یا تنظیم مجدد می‌شوند. شکل 12 رویدادهای خارجی غیر همزمان را در کانال 1 نوسان‌سنج، نشان می‌دهد. در این حالت، رویداد با استفاده از یک تخته‌ی 60 ML را تولید می‌کند. پاسخ زمان‌بند، که در نتیجه‌ی تعویض یا سوئیچ وظیفه، در کانال 3 در شکل 12 نشان داده شده است. در حالی که پاسخ نرم‌افزار که پین را تنظیم می‌کند، در کانال 2 نشان داده می‌شود. در یک فرکانس 50 HZ، پاسخ زمان‌بند برای یک رویداد غیر همزمان می‌تواند تا 26.7 ns بسته به زمان رخدادرویداد بالارود. بلوک منطقی داخلی HSE به بیشتر از 6.7 ns برای رسید به توالی زمان‌بندی و طراحی مجدد زمینه نیاز ندارد. (شکل 12).

پاسخ نرم‌افزار در 50 ns بعد از تعویض وظیفه ایجاد می‌شود. قابل ملاحظه است که در 40 ns برای اجرای دستورالعمل SW و OR مورد نیاز می‌شوند. بعلاوه برای نظارت پاسخ‌های برنامه، ما از یک نوع پورت ثبت شده استفاده می‌کنیم که

از یک فلیپ‌فلاپ D داخلی برای ذخیره‌ی حالت استفاده می‌کند. این فلیپ فلاپ D در لبه افت کننده‌ی نوشته ساعت_ساعت فاز 0 نوشته می‌شود. این حالت ، دلیل دوم تاخیر می‌باشد. به منظور نظارت بر ساعت، سیگنال وقفه‌ی ناهمزمان ، و زمانبندی پاسخ، ما از پورت‌های P7، P6، و P5 ثبت نشده استفاده کردیم. این به این معنی است که این پورت‌ها از هیچ فلیپ فلاپ D داخلی اضافی برای ذخیره‌ی حالت استفاده نمی‌کنند و اینکه آنها، خروجی مرتبط با مقادیر منطقی را به طور مستقیم تولید خواهند کرد، ای مقادیر برای پورت نوشته می‌شوند. ما از این نوع از پورت‌ها، برای رسیدن به یک ارزیابی واقع بینانه از پاسخ زمانبندی و برای از بین بردن هر گونه تاخیر نتیجه شده از هماهنگ-سازی با ساعت سیستم در نوشتن پورت استفاده کردیم.

V. کارهای مرتبط

در این بخش ما تجزیه و تحلیل مختصری از نتایج به دست آمده در دو دهه گذشته را برای تسریع اجرای زمانبندها و الگوریتم شکلهای هندسی اولیه هسته‌ی بی‌درنگ (زمان- واقعی) در سخت افزار ارائه می‌کنیم. ما با پروژه ی FASTCHART پیشنهاد شده در [17] در سال 1991شروع می‌کنیم. منابع (غیر جبری) nondeterminism در سیستم‌های جاسازی شده(تعبیه شده‌ی) بی‌درنگ، توسط چرخه‌ی اجرای دستورالعمل مختلف به دلیل حضور این خط لوله، حافظه‌ی پنهان، وقفه خارجی غیر همزمان، زمان اجرای متغیر عملیات‌های RTOS، بر اساس تعدادی از وظایف و منابع داده می‌شوند. همه‌ی اینها می‌توانند با حرکت دادن عملیات RTOS در سخت افزار کاهش یابند. این، مفهوم اساسی FASTCHART می‌باشد.FASTCHART 64 وظایف را با هشت اولویت مختلف مدیریت می‌کند. در اینجا هیچ پشتیبانی برای منابع (mutexes، سمافور، و غیره) وجود ندارد. پیاده‌سازی به صورت یک پردازنده RISC ، بدون وقفه، خط لوله، دستورالعمل، و یا ذخیره‌سازی داده‌ها انجام شد؛ بنابراین، چرخه‌ی دستورالعمل قطعی شد. FASTHARD [9] بر اساس پروژه‌ی FASTCHART قبلی است [17]، اما آن یک هسته‌ی سخت‌افزاری است که پردازنده همه منظوره را پشتیبانی می‌کند. با استفاده از یک حافظه‌ی استاندارد، طرح گذرگاه آدرس / داده طراحی می‌شود. آن، از امکاناتی، از قبیل آمدگاه(rendezvous)، وقفه‌ی خارجی، شروع دوره‌ایی وظایف، و فعال‌سازی و خاتمه‌دادن به وظایف، بدون دخالت CPU پشتیبای می‌کند. بعد از FASTHARD

[9]، Adomat و همکارانش [18] یک واحد بی‌درنگ (RTU) را به عنوان هسته‌ی بی‌درنگ(زمان واقعی) مبتنی بر سخت‌افزار توصیف کردند. این، از 64 وظیفه، هشت سطح اولویت، وظایف دوره‌ایی با تاخیر نسبی، سمافورهای دوتایی، پرچم‌های رویداد، ناظران، و وقفه‌ها پشنیبانی می‌کند، و آن به طور همزمان توسط سه پردازنده همگن متصل به یک گذرگاه ماژول بالعکس اروپا (VME) مدیریت می‌شود.

لیند و همکاران [19] یک معماری مقیاس‌پذیر را برای برنامه زمان واقعی(بی‌درنگ) (SARA) توصیف کردند، که می‌تواند با RTU استفاده شود. SARA، یک سیستم قابل توسعه است، که از کارت‌های پردازنده‌ی شخصی با حافظه و کنترل‌کننده‌های گذرگاه استفاده می‌کند این کنترل کننده به یک مادربرد با استفاده از PCI فشرده متصل می‌شود. لی و همکارانش [20] RTU را به δ-سیستم [21] برای کوطراحی SoC / RTOS ادغام کردند. Nordstrom و همکارانش [22] هسته‌ی نرم‌افزار µC / OS-II RTOS را با یک نسخه‌ی تک پردازنده ی RTU، به منظور بهبود عملکرد، وفق دادند و در [23] آنها پیکربندی RTU را برای نسخه‌ی تک پردازنده اضافه کردند. هدف از این سیلیکون TRON (Stron) [8] افزایش سرعت فراخوان‌های سیستم RTOS پایه و کاهش حرکت نامنظم اتفاقی به منظور پیش‌بینی‌های دقیق زمان‌بندی می‌باشد. توسعه، بر اساس پروژه‌ی TRON، به خصوص مشخصات µITRON برای RTOSs، می‌باشد که فراخوان‌ها و ویژگی‌هایش در یک هسته سخت‌افزاری به نام Stron اجرا شده است. پردازنده‌ی Stron شامل مدیریت وظیفه، پرچم‌ها، سمافور و زمان‌سنج‌ها و مدیریت وقفه‌های خارجی می‌باشد. آن برای حافظه های خارجی طراحی می‌شود و آن ، به عنوان یک کمک پردازنده در نظر گرفت می‌شود. هسته‌ی SPRING در [24] توسعه و شرح داده می‌شود و به طور کامل رویکرد کاملاً متفاوتی با زمان‌بندی طبیعی RTOS دارد. هسته‌ی SPRING از زمان‌بندی پویا و تئوری بر اساس جستجوی انشعابی و الگوریتم های اکتشافی استفاده می‌کند.

با در نظر داشتن زمان اجرای وظایف، ضرب العجل ها، منابع و محدودیت اولویت، الگوریتم های زمان بندی ، یک طرح برنامه را با کمک تمام وظایفی که ضرب‌العجل‌شان را بدون گرفتن انسداد، در حین انتظار برای منابع می‌سازند. در [25]، یک پیاده‌سازی سخت‌افزاری به عنوان یک کمک پردازنده‌ی زمان‌بندی spring

برای هسته‌ی نرم‌افزاری ارائه شده در [24]. توصیف می‌شود. الگوریتم‌های زمان بندی پویا، تأثیر عمده ای بر روی بالا سری RTOSs دارند. برای کاهش زمان محاسبه‌ی اولویت در هنگام انتخاب یک وظیفه و برای ارائه‌ی استقلال زمانی در تعدادی از وظایف، یک کمک پردازنده‌ی زمان‌بندی پویا در FPGA اجرا شد، همانطور که در [26] و [27] توصیف شده است. آن، به عنوان یک شتاب دهنده ی زمان‌بندی ، با استفاده از برتری همسانی سخت‌افزار، توسعه داده شد. این می تواند برای بسیاری از پیکربندی الگوریتم هایی که اولین سستی (laxity) حداقل افزایش یافته (ELLF)، شرح داده شده در [26] پیشرفته‌ترین است. Vetromille و همکارانش [10] یک تست عملکرد مربوطه را بین سیستم‌هایی که از یک پردازنده‌ی تک، کمک پردازنده یا یک واحد سخت افزار خارجی اختصاصی برای اجرای زمانبندی استفاده می‌کنند ایجاد کردند. پارامترهای آزمایشی استفاده CPU، تعدادی از فقدان‌های ضرب العجل ، و تعدادی از سوئیچ های زمینه بودند. نویسندگان همچنین سربار اضافی را با توجه به ارتباطات خاص پردازنده-کمک پردازنده با زمانبند نرم افزاری که در یک پردازنده جداگانه اجرا شده است مورد بحث قرار دادند. با استفاده از ابزار سنتز رفتاری، چاندرا و همکارانش [28] چگونگی تولید خودکار HW-RTOS از منابع ECOS RTOS API POSIX، در نتیجه افزایش عملکرد سیستم به عنوان یک کل را توصیف کردند. H-هسته [29] از یک پردازنده‌ی خاص طراحی شده استفاده می‌کند که مجموعه‌ای از ثبات هر وظیفه و یک RTOS سخت افزار اختصاص داده دارد. آن، به 60٪ افزایش در عملکرد برای یک کاربرد خاص می‌رسد، مانند پردازش صدا چندتایی بی‌رنگ. ابزار H-هسته وظایف را بر اساس مدیریت اولویت، مدیریت وقفه ، بلوک های رویداد، صف‌ها، و مدیریت زمان اجرا می‌کند. سانگ و همکارانش [29] یک راه حل مبتنی بر پردازنده را توصیف کردند که می‌تواند برای رسیدن به یک عملکرد بهبود یافته با استفاده از کوطراحی مدرن FPGA از HW / SW برای یک کاربرد خاص طراحی شود. معماری پردازنده‌های بی‌درنگ چند رشته‌ای پیشرفته (ARPA-MT) است در [30] شرح داده شده است و از فن آوری مدرن FPGA برای رسیدن به یک پردازنده قابل پیش بینی و قابل تنظیم استفاده می‌کند. این پردازنده توسط دو کمک پردازنده‌ی سخت افزاری Cop0-MEC و Cop2-OSC همراه می‌شود. اولی ، خطای حافظه

و وقفه، را مدیریت می‌کند در حالی که ابزار دوم تمام ویژگی‌های استاندارد RTOS. را مدیریت می‌کند. اروک یک هسته‌ی بی‌درنگ نوشته شده در C ++. می‌باشد. در [31]، انگیزه، طراحی، و نتایج عملکرد به دست آمده پس از انتقال تمام قابلیت هسته به سخت افزار ارائه می‌شود. هدف ، بهبود جبر و عملکرد است. ARTESSO [32] یک RTOS اجرا شده در سخت افزار، به علاوه برخی از ماژول‌های TCP / IP خاص می‌باشد. ARTESSO با انگیخته می‌شود با استفاده از این حقیقت که یک پردازنده با کارایی بالا تعبیه شده ، مورد نیاز است اگر برنامه برای به دست آوردن توان داده ها برای سرعت 100 s / MB و یا بیشتر نیاز باشد. معماری ارائه شده هسته، محاسبات کنترلی، کپی حافظه، و چینش هدر(نگهدارنده) TCP / IP را به سخت افزار حرکت می‌دهد.

معماری جالب در [5]، پیشنهاد شده است که از یک خط لوله داخل سطحی ریسه‌ایی ، حافظه‌های مسیر چرکنویس،و یک کنترل‌کننده‌ی DRAM دارای عملکرد حافظه‌ی قابل فراگیری و قابل پیش بینی استفاده می‌کند. این معماری اجازه برنامه نویسی کنونی، حفظ زمان‌بندی آنها در همان زمان را میسر می‌سازد. proprieties. Kuacharoen و همکارانش [11] زمانبندی سخت افزار ی با قابلیت تنظیم را برای سیستم‌های نشان دادند. زمانبندی سخت افزاری با قابلیت تنظیم ، از سه الگوریتم زمان بندی پشتیبانی می‌کند: 1) مبتنی بر خصوصیات؛ 2) نرخ یکنواخت؛ و 3) اولین ضرب‌العجل قبلی. زمانبندی سخت افزاری همچنین یک کنترل کننده خاص را برای وقفه اجرا می‌کند. آن، دستورالعمل‌هایی مانند mutexes، سمافور، و غیره، اجرا نمی‌کنند چنانکه آنها از بخش نرم‌افزاری پیاده‌سازی داده شده باشد. اندروز و همکارانش [33] hthread ر توصیف کردند که یک RTOS هسته چند ریسه‌ایی سخت افزار / نرم افزار می‌باشد. این هسته، بخشی از یک مدل برنامه‌نویسی ریسه‌ایی هیبریدی توسعه یافته برای سیستم‌های ترکیبی است که شامل هر دو ریسه‌ی اجرایی مستقرنرم افزاری و مستقر سخت افزاری به صورت همزمان می‌شود. هسته ، تا 256 ریسه‌ی نرم افزار فعال ، 256 ریسه‌ی سخت‌افزاری فعال، 64 سمافور بلوکی یا نسدادی ،64 سمافور قفل -چرخش دوتایی ، اولویت پیشگیرانه، دور رابین، و الگوریتم زمان‌بندی FIFO را پشتیبانی می‌کند. هسته شامل یک کنترل کننده ی وقفه جدید به نام زمانبندی وقفه گذرگاه فرعی می‌باشد که معناشناسی وقفه ناهمزمان به همزمان، قابل کنترل ،

درخواست‌های ریسه‌ی مبتنی بر اولویت‌بندی ترجمه می‌کند. نرم افزارها به اجزای سیستم عامل مبتنی بر سخت افزار را از طریق API های آشنا، مانند ایجاد، پیوستن به، و خروج دسترسی دارند.

نتیجه می‌گیریم که از دلایل اصلی اجرای RTOS ، و یا بخش‌هایی از آن در سخت افزار عبارتند از: کاهش حافظه، کاهش اجرای بالاسری داد شده توسط توابع اساسی (زمان‌بندی و سوئیچینگ(تعویض) زمینه) و فراخوانی تابعRTOS ، کاهش یا حذف، حرکت نانظم اتفاقی و بهبود زمان پاسخ وقفه ها و امکان بهتر شدن کنترل رفتار آنها (وقفه به عنوان وظایف). همچنین می‌توانید توجه کنید که دو روند در شتاب دهنده‌های زمان‌بندی و طراحی RTOS مبتنی بر سخت افزار وجود دارد: یکی جزئی، بر اساس طراحی پردازنده های ویژه: FASTCHART [17]، H-هسته [29]، و ARPA [30] پروژه ها، و یک رشته اصلی، با استفاده از (کمک پردازنده‌ها) coprocessors، مانند [9] FASTHARD، 18] RTU] - [20]، [22]، [23]، [8] Stron، δ-چارچوب [21]، HW-ECOS [28]، [31] OReK_ COP، [32] AR-TESSO، hthreads [33]، و یا برنامه ریزی شتاب دهنده ها، مانند ELLF_ نقال [26]، [27] و یا Kuacharoen [11].

در پاراگراف زیر، ما در حال حاضر برخی از راه‌حل‌های پیشنهادی در این مقاله را با آنهایی که در بخش چهارم. شرح داده شده مقایسه می‌کنیم.

مقایسه اول، همانطور که در جدول III نشان داده شده است ، بر اساس معیارهای تکرار منابع، سرعت ارتباط وظیفه و پیاده‌سازی سخت افزار و یا سخت افزار-نرم افزار می‌باشد. هرچند آن ممکن است مانند یک راه حل ساده به نظر برسد، nMPRA تنها واحد تضمین تکرار ثبات‌های خط لوله با اثرات کاربردی بر روی سرعت سوئیچ کردن وظیفه می‌باشد. معماری ARPA-MT ،تنها تکرارIF و ID ثبات‌های خط لوله را برای هر وظیفه‌ی نمونه انجام می‌دهد. با این حال، آن ،برای سه سطح آخری خط لوله،-MA EX، و RB وارد رقابت می‌شود. اگر چه راه حل تکرار فایل حاوی ثبات‌ها، یک راه حل سخت افزاری گران قیمت است، آن همچنین یک فاکتور همگرا به سمت رسیدن به یک سرعت تعویض بین 1 و 3 چرخه‌ی ماشین می‌باشد، واقعیتی که تنها می‌تواند در nMPRA بدست بیاید. همانطور که می توان به وضوح در جدول دید، حتی برای یک پردازنده‌ی

فرکانس 50 مگاهرتز، سرعت تعویض وظایف NS 60 است. در زمان مشابه، nMPRA به سادگی یک اجرای سخت افزاری، بر خلاف پیاده‌سازی‌های دیگر است که ترکیبی سخت افزار-نرم افزار ی هستند که بطور استنباطی منجر به بازه زمانی تعویض طولانی‌تر می‌شود.معماری، اجرای همزمان وظایف سخت افزار و یک وظیفه‌ی نرم‌افزاری را میسر می‌سازد.

مقایسه‌ی دوم بر اساس با ویژگی های زمانبند است و در جدول IV توصیف می‌شود. راه حل‌های مقایسه شده به زمانبند استاتیک و دینامیک اجرا شده‌ی سخت افزاری، به جز برای hthreads اشاره می‌کنند. nMPRA تنها زمانبند اجرا شده در سطح پردازنده است و قادر به جمع‌آوری اطلاعات زمان‌بندی به طور مستقیم از سخت افزار، بدون تاثیر بر ارتباط گذرگاه ممکن می‌باشد چنانکه حالتی با راه حل های در نظر گرفته شده‌ی دیگر باشد .تمام راه حل ها، پیشنهاد الگوریتم زمان‌بندی استاتیک را پیشنهاد می‌کنند. از آنجا که زمان‌بندی انجام از سوئیچ سخت افزاری را به سمت بالاترین اولویت ریسه نجام می‌دهد، ما می‌توانیم بپذیرند که nMPRAسریع‌ترین است (آن از یک وظیفه به دیگری با تاخیر 1-3 چرخه دستگاه سوئیچ می‌کند). تا آنجا که به پیاده سازی الگوریتم‌های زمان‌بندی پویا مربوط می‌شود، ARPA-MT [30] و راه حل‌های Kuacharoen و همکارانش [11] بیشتر پیشرفته می‌شوند، چنانکه آنها سخت افزار اجرا شده باشند. در حال حاضر، nMPRA یک الگوریتم اجرا شده‌ی سخت افزاری پویا ندارد، چنانکه آن برای حمایت از اولویت‌های پویا با استفاده از زمانبندی پویا تهیه شود. همانطور که قبلا در بخش سوم-B، نشان داده شد ما می‌توانیم برخی از راه حل‌های اجرایی را در متون تخصصی ارائه دهیم. فعالیت زمانبندی از طریق سخت افزار به nMPRA اختصاص داده و اجرا می‌شود. چنانکه برای راه حل های دیگر، فعال سازی زمانبندی به کمک پردازنده، به چرخه‌های انتقال گذرگاه مورد نیاز و حتی مداخله نرم افزار منسوب می‌شود.

مقایسه‌ی سوم اشاره به سیستم وقفه دارد و در جدول V ارائه شده است. ما مفهوم وقفه را در ریسه‌های [11] و [33].دریافته‌ایم. مانند ARPA-MT [30]، هیچ مرجع صریحی برای مدیریت وقفه در اینجا وجور ندارد. در [11]،کنترل‌کننده‌ی وقفه به اختصار شرح داده می‌شود که آن ، هشت سطح وقفه را پشتبانی می‌کند. هر وقفه می تواند با یک وظیفه، واگذار شده به مدیریت سطوح مرتبط به وقفه‌ها در ارتباط باشد . هر وقفه ممکن است به عنوان

یک وقفه‌ی سریع (برنامه کنترل وقفه به حالت تعلیق در کار فعلی می‌باشد) و یا یک وقفه‌ی آرام (کنترل وظیفه در پایان خط اولویت قرار داده می‌شود). بر خلاف پیاده سازی‌های دیگر، قطع سیستم وقفه‌ی nMPRA بطور کامل اختصاص داده می‌شود، به طوری که یک وقفه می‌تواند تنها یک وظیفه متصل شود ، در حالی که یک کار می‌تواند به وقفه بیشتر متصل شود (حتی همه‌ی آنها). مزیت عمده‌ی nMPRA توسط این ایده شان داده می‌شود که آن هیچ تاثیری بر خط لوله ندارد ، NOR آن را برای ذخیره‌سازی ثبات مورد نیاز می‌سازد. به علاوه ، وقفه‌ها ، درست مانند رویدادهای دیگر در سیستم، به شیوهای واحدی توزیع می‌شوند.

مقایسه آخری در جدول VI نشان داده شده است و اشاره به اجرای هماهنگ سازی و ارتباطات شکلهای هندسی اولیه در میان وظایف دارد. nMPRA تنها معماری ، در میان آنهایی است که در حال حاضر ارائه شده‌اند، و شکلهای هندسی اولیه در سخت افزار را اجرا می‌کند. اگر یک رویداد رخ دهد، مانند آنهایی که در بخش سوم-F، توصیف شدند و اگر آن با یک وظیفه اولویت بالاتری نسبت به وظیفه فعلی همراه باشد، آنگاه وظیفه‌ی تعویض با یک تاخیر 1-3 چرخه رخ می‌دهد. nMPRA ممکن است به طور همزمان با تمام حوادث همگام‌سازی، با استفاده از فقط یک دستورالعمل انتظار پشتیبانی شود. هر یک از وقایع مورد انتظار که اولی را فعال می‌کند باید وظیفه را آغاز کند هنگامی که آن بالاترین اولویت می‌شود. به عنوان یک نتیجه، ما می‌توانیم بیان کنیم که nMPRA صف را برای سازه های بلوک کنترل وظیفه NOR حالات وظیفه اجر ا نمی‌کند.

نتیجه‌گیری

در این مقاله، ایده‌ی اولیه‌ی ارائه شده در مراجع 6 و 7، تعریف ویژگی‌های جدید اصلی برای معماری‌های nMPRA و nHSE توسعه می‌یابد. معماری زمان‌بندی با استفاده از رویدادها (تایمر، تایمر نگهبان، بی ضرب الاجل، وقفه، ناسازگاری دو جانبه و رویداد) پشتیبانی می‌شوند. دو سختار زمان‌بندی پیشنهاد می‌شود: یکی استاتیک ست که در آن ID و الویت کار برابر 0 بعنوان بالاترین الویت و n-1 بعنوان پایین‌ترین الویت می‌باشند.و نوع دیگر دینامیک می‌باشد که در آن

الویت و ID لزوماً مشابه نیستند.در اینجا الویت می‌تواند در داخل هر واحد sCPU برنامه‌ریزی شود. ین حالت، اجرای الگوریتم زمان‌بندی را با الویت‌ههای دینامیک میسر می‌سازد.

این مقاله یک راه‌حل ابتکاری را برای مکانیسم وقفه رائه می‌دهد. در این زمینه، وقفه‌ها، الویت و رفتاری از وظایف را برای اینکه متصل شود به ارث می‌برند. با استفاده از این روش، رفتار یک وقفه در زمینه‌ی برنامه‌ی بی درنگ قابل پیش‌بینی است (یک کار تنها با ستفاده از وقفه‌هایی که به وظایفی با الویت بالاتر متصل هستند می‌تواند قطع شود). معماری پیشنهادی، ویژگی جالب و قدرتمندی دارد که در بخش III-C توضیح داده شد.

یک سری ناسازگاری متقابل (انحصار متقابلes) اجرا شده در سخت‌افزار، برای دسترسی به منابع اشتراکی sCPUs ،در ساختار دخیل شده‌اند. دسترسی انحصار متقابل، در یک چرخه‌ی CPU تک انجام می‌شود و آن خودکار می‌باشد. روش پیشنهادی، کاری را میسر می‌سازد که در یک انحصار متقابل سخت‌افزار، برای متوقف کردن خود تا زمانی که انحصار متقابل بدون هیچ بالاسری از نرم‌افزار یا RTOS آزاد شود در حال انتظار است. یک سری رویدادهای اجرا شده بعنوان یک ERF ،به منظور استفاده شدن بعنوان بخشی از یک ارتباط بین وظیفه‌ایی یکپارچه و مکانیسم هماهنگ-سازی، در معماری دخیل شده‌اند. ERF همچنین بعنوان منبع به اشتراک گذاشته شده برای تمام sCPUi در نظر گرفته می‌شود. دسترسی برای انجام یک رویداد، بعنوان مرجع آدرس یک رویداد استفاده نمی‌شود (در اینجا هیچ عملیات جستجویی برای رویداد اختیاری وجود ندارد.)

همانطور که در بخش II-A رائه شد، آموزش انتظار بسیار قدرتمند است زیرا آن، هماهنگ‌سازی را با چندین رویداد فراهم می‌سازد. این رویدادها می‌توانند تحت کنترل نرم‌افزار به دنبال الویت‌بندی که توسط وظایف sCUP تحمیل می‌شود آشکار شوند. سرانجام، ما می‌تونیم نتیجه بگیریم که معماری nMPRA از جنبه‌های زیر، یک معماری بسیار قدرتمند است:

1. سوئیچ (تعویض) در بین وظایف، معمولاً در یک چرخه‌ی ساعت و در حداکثر سه چرفه‌ی ساعت انجام می‌شود. زمانی که CPU با حافظه‌ی کلی کار می‌کند.

2. واکنش سیستم به یک رویداد خارجی، از 1.5 چرخه‌ی ساعت تجاوز نمی‌کند در صورتی که رویداد به یک کار با الویت بالاتر از کار کنونی متصل شود.
3. خط لوله pipeline، ره‌اندازی مجدد نمی‌باشد؛ در اینجا هیچ نیازی به بازگرداندن یا ترمیم زمینه نمی‌باشد. فراخوان-های زیرروالSubroutine ، از طریق کپی پارامتر خودکار و ایجاد نقشه‌ی جدید مجموعه‌ی ثبات شتاب داده می‌شوند.
4. حافظه‌ی داخلی سرعت بالا-مبتنی بر پشته stack
5. دستورالعمل‌های قدرتمند برای به اشتراک گذاری منابع، هماهنگ سازی، و ارتباطات کار داخلیintertask . بعنوان کاری که در آینده می‌خواهیم انجام دهیم ما قصد داریم آزمون‌های میز خاص را برای معماری جدید به منظور مقایسه با پردازنده‌های منظم ایجاد کنیم (در حال حاضر، آزمون‌های میز موجود نمی‌توانند در معماری پیشنهادی اجرا شوند زیرا ما دستورالعمل‌های جدیدی را برای استفاده‌ی nHSE معرفی می‌کنیم). همچنین ما قصد داریم مفاهیم ارائه شده در این مقاله را در مکانیسم‌های RISC پیشرفته‌ی ساختارهای زمینه‌ی M (ARM) بکار ببریم.