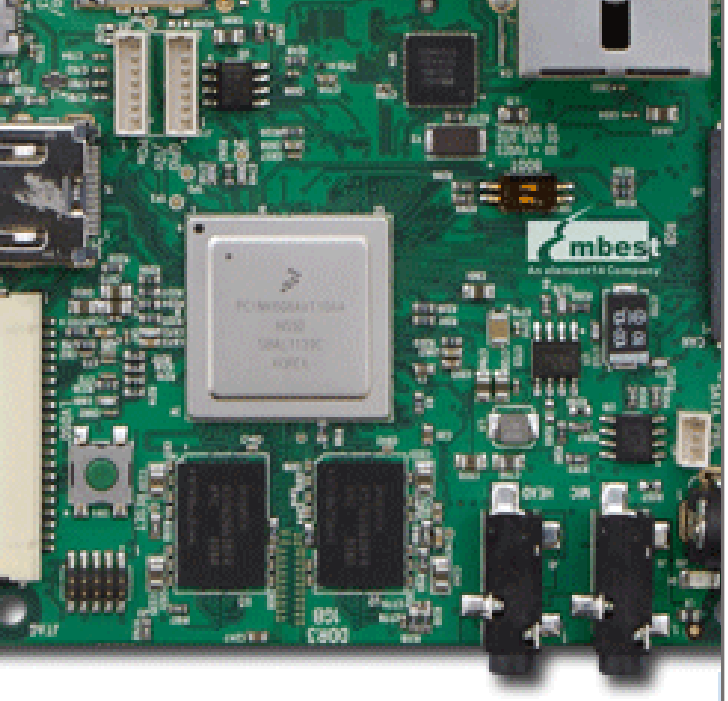


صنعت الکترونیک در چند دهه گذشته تحول چشم‌گیری داشته و حضور میکروکنترولرها به عنوان قلب تپنده دستگاههای هوشمند، در این تحول بسیار اثرگذاربوده‌اند. میکروکنترولرهای ۸ بیتی با قیمت مناسب و کارآیی قابل قبول در کنترل این دستگاه‌ها موفق بودند. با ظهور میکروکنترولرهای کارآمد ۳۲ بیتی ARM با قیمت‌های قابل رقابت و عرضه روزافزون آنان به بازار، رقابت شدیدی درگرفت که باعث پیشرفت هر چه بیشتر این سیستمها شده است.

مرجع کاربردی میکروکنترلرهای ARM

تالیف: س. سهرابی

تقدیم به پدر و مادرم

... و سایر قلم به دستانی که مقام دانش را به مراتب دنیایی ترجیح داده و افتخارشان به اندوخته های دانش شان و لذت شان در آموزش آن است.

توجه : کتاب پیش رو، نسخهٔ الکترونیکی کتاب اصلی می‌باشد که صرفا جهت دریافت رایگان از اینترنت و مطالعهٔ شما عزیزان، با کسب اجازه از نویسنده تهیه گردیده است. امید است که بهرهٔ کافی را از آن ببرید.

برای ارتباط بیشتر با نویسنده و نظرات و پیشنهادات لطفا از طریق پست الکترونیک زیر در تماس باشید:

sbsohrabi@gmail.com

پیش گفتار

سپاس می گوییم خداوند را که به ما توان نوشتن این نوشته را اعطا فرمود که بدین وسیله قدمی هر چند ناچیز در اعتلای سطح علمی دانشجویان و تمامی دوستداران علم الکترونیک برداریم.

صنعت الکترونیک در چند دهه گذشته تحول چشم‌گیری داشته است. حضور میکروکترولرها به عنوان قلب تپنده دستگاههای هوشمند، در این تحول بی‌تاثیر نبوده‌اند. میکروکنترولرهای ۸ بیتی به سادگی از پس این دستگاههای هوشمند برمی‌آمدند و به علت قیمت مناسب و کارآیی قابل قبول شان، مدتها مورد استفاده قرار می‌گرفتند. ظهور میکروکنترولرهای ۳۲ بیتی ARM با قابلیت های شگفت‌انگیز و قیمت‌های قابل رقابت باعث پیشرفت هر چه بیشتر این سیستمها شده است.

کتابی که پیش رو دارید میکروکنترولر های حرفه ای ARM (که امروزه به سرعت در حال همه گیر شدن هستند) را از جنبه های مختلف مورد بررسی قرار داده و با ارایه مثالهای کاربردی گوناگون، خواننده را به سمت کارِ عملی با این میکروکنترولرها هدایت می‌کند. میکروکنترولرهای ARM از ابتدای ورود به بازارهای جهانی، با سرعتی زیاد در حال جایگزین شدن میکروکنترولرهای ۸ بیتی می باشند. سرعت پردازش بالا، مصرفِ توانِ کم، ساختار داخلی و خارجی ۳۲ بیتی، ارتباط آسان با انواع مختلف وسایل جانبی (همانند USB، Ethernet، نمایشگرهای LCD، کارتهای حافظه و ...) از جمله مزایای این تراشه ها است. همین امر، در کنار قیمت قابل رقابت این میکروکنترولرها (در مقایسه با میکروکنترولر های ۸ بیتیِ پر کاربرد) آنها را به عنوان جایگزینی امن و بهینه معرفی می نماید.

در این مجموعه سعی بر این بوده است که از پرداختن به جزییات در صورت امکان پرهیز شود تا خواننده با سرعت به مفاهیم کتاب مسلط شود. گاها اشاره‌ای گذرا به جزییات شده است. البته در بیان مطالب کوتاهی صورت نگرفته است و سعی بر آن بوده است که کلیت قضیه با تمامی زیر و بمهایش گفته آید.

در این کتاب از به کار بردن ترجمه های نامفهوم تا حد ممکن پرهیز شده است و سعی بر این بوده است که از کلمات انگلیسی معادل در پانویس استفاده شود. تمامی پروژه‌های کتاب به همراه نقشه‌ی شماتیک، کُدهای آماده و برنامه‌های نرم‌افزاری مربوط شان در DVD همراه کتاب، آورده شده‌اند.

این نوشتار محصول چندین سال تلاش و تجربه علمی و عملی نویسنده‌ی کتاب در زمینه پردازنده ها و میکروکنترولرهای ARM می باشد. سعی بر این بوده است که کتابی، درخورِ

دوست داران علم الکترونیک، خصوصا مشتاقان میکروکنترولرهای پیشرفته، سیستمهای هوشمند و علم رباتیک ارایه گردد.

سازمان دهی کتاب

برای راحتی کار و تسهیل در امر خواندن و درک سریع مطالب، این کتاب به چندین بخش تقسیم شده است. در بخش اول بعد از معرفی میکروکنترولرها، به تاریخچه و معماریهای مختلف پردازنده های ARM نگاهی انداخته می شود.

بخش دوم خواننده را به سرعت با مفاهیم پایه‌ی پردازنده های ARM آشنا کرده و او را وارد دنیای ARM می کند. معرفی ساختار داخلی، ثبات ها ، معماری گذرگاه ها و روشهای رسیدگی به وقفه ها از اهداف اصلی این بخش است.

بخش سوم به معرفی میکروکنترولر های ARM ساخت شرکت‌های سازنده گوناگون، بیان ویژگیهای آنان، اختصاص دارد. اشاره به وسایل جانبی گوناگون و روشهای کار با این پردازنده ها از اهداف اصلی این فصل است.

بخش چهارم به محیطهای توسعه نرم‌افزاری اختصاص دارد. از برنامه نویسی به زبان اسمبلی و C/C++ گرفته تا کامپایل کردن کدها و روشهای برنامه ریزی این میکروکنترولرها، همگی در این بخش مورد بررسی قرار گرفته اند. به نحوی که خواننده از لحاظ نرم افزاری مشکلی نداشته و بعد از اتمام این بخش توانایی حل مشکلات نرم افزاری خود را دارد.

ارتباط با انواع مختلف موتورهای الکتریکی، شبکه های کامپیوتری و اینترنت، نمایشگرها، مدارات آنالوگ و سنسورها و مدارات دیجیتال گوناگون مورد بحث بخش پنجم این کتاب هستند. خوانده در این بخش با مجموعه ای از مثالهای کاربردی و پروژه های عملی روبرو است که اکثر آنها آزمایش شده و کاملا کاربردی هستند. با پایان این بخش خواننده دیگر مشکلی با این پردازنده ها نخواهد داشت و می تواند شروع به استفاده عملی از این میکروکنترولرها در پروژه های خود نموده و یا پردازنده های قدیمی خود را به این پردازنده ها تبدیل نماید.

بخش ششم و انتهایی کتاب به مباحث پیشرفته تری چون استفاده از پردازش سیگنال دیجیتال، بهینه سازی و استفاده از زبان برنامه نویسی C/C++ ، استفاده از سیستمهای عامل بلادرنگ، Linux و WinCE و استفاده از چند ریسمانی و پردازش بی‌درنگ ... اختصاص دارد. کسانی که سعی در گسترش دانش خود در زمینه‌های پیشرفته‌تری دارند، می توانند به این بخش مراجعه کنند.

هر یک از این بخش ها مشتمل بر چندین فصل بوده و در طی ارایه مطالب سعی شده است که یک سیر منظم آموزشی ارایه شود. به نحوی که از سردرگم شدن خواننده جلوگیری شود. در

انتهای بعضی فصلها، بخش های کوچکی به نام پرسش و پاسخ، اصطلاحات جدید و ایده‌های طراحی وجود دارند که برای محک زدن آموخته های خواننده، آورده شده‌اند. در ارایه جزییاتی که در خواننده سخت فهمی ایجاد می کنند، با احتیاط عمل شده است و در مواردی که نیاز به مطالعه‌ی فراتر از حدود این کتاب بوده، او را به مراجع مربوطه ارجاع داده ایم.
شما می‌توانید مثالهای کتاب، برنامه های کاربردی و بسته های به روزرسانی گوناگون را بر روی سایت اینترنتی کتاب نیز بیابید. برای اطلاعات بیشتر در این زمینه، به ضمایم مراجعه کنید.

خوانندگان این کتاب چه کسانی هستند؟

خواننده این کتاب طیف وسیعی از دانشجویان رشته‌های الکترونیک، سخت افزار، نرم افزار و سایر کسانی که الکترونیک و میکروکنترولر را به صورت حرفه‌ای یا سرگرمی دنبال می کنند را در بر می‌گیرد. این کتاب همچنین می‌تواند برای دانشجویان تحصیلات تکمیلی به عنوان یک مرجع کاربردی مطرح باشد.

DVD همراه کتاب

در DVD ارایه شده به همراه کتاب، مثال‌های گوناگونی (شامل مثال‌های داخل کتاب) آورده شده‌اند. DVD مذکور حاوی تمامی اسناد، برگه‌های اطلاعات و برنامه های ضروری برای راه اندازی میکروکنترولر های ARM می باشد.

قدردانی

در این‌جا شایسته است از خانواده ام که در مدت زمان نگارش این کتاب بنده را یاری نموده اند و سختی‌هایی چند را برای هدف بنده تحمل نموده‌اند تقدیر شایسته به عمل آورم. هر چند زبان در تقدیر آنان قاصر است، ولی در این مجال، تنها ابزار در دسترس قلم است. امید دارم که بتوانم زحماتشان را جبران کنم.
مسلما به رغم تلاشهای صورت گرفته در جهت درستی مطالب، این متن خالی از اشکال نیست. نظرات تمامی خوانندگان و صاحب نظران برای نویسندگان ارزشمند بوده و آنان را در بهبود هر چه بیشتر این کتاب یاری می رساند.

سالار سهرابی
زمستان ۸۹

قراردادها

اسامی ثباتها، اسامی خاص، وضعیتهای کاری و ... با *Italic* نشان داده می شوند.

بخش ۱

میکروکنترولرهای ARM

فصل اول

ظهور ARM در عصر مدرن

اهداف فصل

در این فصل شما با موارد زیر آشنا می‌شوید:

- ✓ ریزپردازنده‌ها چگونه پا بر عرصه‌ی دنیای دیجیتال گذاشتند؟
- ✓ پردازنده‌های ARM چگونه به وجود آمدند؟
- ✓ خانواده‌های گوناگون پردازنده‌های ARM کدام‌اند؟
- ✓ میکروکنترلرها چگونه قطعاتی هستند و یک سیستم تعبیه شده چیست؟
- ✓ میکروکنترلرهای ۳۲ بیتی چه مزایایی نسبت به انواع قدیمی‌تر خود دارند؟
- ✓ چه مسایل و تکنیک‌هایی در طراحی‌های ۳۲ بیتی دخیل هستند؟
- ✓ کاربرد وسیع میکروکنترلرهای ۳۳ بیتی در صنعت؛
- ✓ بازار فروش میکروکنترلرهای جدید.

۱.۱ ظهور ریزپردازنده‌ها[۱]

شرکت Intel ، در سال 1972 با ارایه پردازنده جدید خود به نام 8008، اولین پردازنده‌ی ۸ بیتی در دنیا را معرفی کرد. 8008 ، اوج یک پروژه‌ی طراحی بزرگ در شرکت Intel بود. این شرکت در حال طراحی پردازنده‌ای برای پایانه‌ی داده[۲] به سفارش شرکت Computer Terminals Corporation بود که این پردازنده‌ی جدید ۸ بیتی متولد شد.

در آوریل 1972، Intel اولین پردازنده‌ی ۸ بیتی خود را رسماً در بازار فروش قطعات الکترونیکی عرضه کرد. این پردازنده، پایه‌ی طراحی کیت کامپیوتری "Mark-8" بود.

تراشه‌ی 8008 ، آغازگر راه پردازنده‌های بسیار موفقی چون Intel 8080 (این پردازنده نسبت به 8008 بسیار بهبود یافته بود و به قطعات پشتیبانی کمتری احتیاج داشت) و Zilog Z80 بود. رقیب این پردازنده‌ها، تراشه‌ی 6800 شرکت Motorola و تراشه‌ی 6502 شرکت MOS Technology بودند که عمدتاً توسط همان متخصصان دو تراشه‌ی رقیب طراحی شده بودند. پردازنده‌ی 6502 ، در دهه‌ی 1980 بسیار فراگیر شد به نحوی که گوی سبقت را از Z80 محبوب، ربوده بود.

---

[۱] Microprocessor

[۲] Data Terminal

هزینه‌ی تمام شده پایین، ابعاد و بسته‌بندی کوچک تراشه‌ها و در بعضی مواقع دربر داشتن مدارات کمکی اضافی در سطح تراشه (همانند کنترل کننده‌ی حافظه‌ی دینامیک بر روی تراشه) ، منجر به انقلاب **کامپیوترهای شخصی**[۱]، در اوایل دهه‌ی 1980 و حرکت سریع ورود به رشد آن‌ها شد. تولید کامپیوترهای ارزان بهایی همانند Sinclair Zx-81 با قیمت فروش ۹۹ دلار، نمونه‌ای از این دستگاه‌هاست.

شرکت Western Digital Center (WDC) نمونه‌ی CMOS تراشه‌ی 6502 با نام 65C02 را در سال 1982 معرفی کرد و مجوز ساخت آن را به چندین کارخانه فروخت. کامپیوترهای شخصی شرکت Apple-lle ، از مصرف کنندگان اصلی آن بودند. شرکت WDC ، پیشگام فروش "مجوز ساخت" تراشه بود[۲].

۱.۲ پردازنده‌ی ARM

۱.۲.۱ شرکت سهامی تجاری ARM[۳]

شرکت سهامی تجاری ARM ، یک شرکت فنی ـ مهندسی بریتانیایی واقع در شهر Cambridge است. این شرکت، شهرت عمده‌ی خود را به خاطر پردازنده‌هایش کسب کرده است و محصولات توسعه نرم‌افزاریش را تحت عناوین تجاری Real View و KEIL ، به فروش می‌رساند. از دیگر محصولاتش می‌توان به سیستم ها و پلتفرم‌های نرم‌افزاری و "سیستم بر روی تراشه"[۴] اشاره کرد. قطعاً این شرکت را می‌توان یکی از بهترین و شناخته شده‌ترین شرکت‌های منطقه‌ی Silicon Fen (منطقه‌ای تجاری با فعالیت‌های حرفه‌ای الکترونیک و کامپیوتر حوالی شهر Cambridge که یکی از مهمترین مناطق مرتبط با فناوری، در اروپا است. این منطقه همکاری تنگاتنگی با دانشگاه Cambridge نیز دارد) دانست.

شکل‌گیری این شرکت حاصل یک فعالیت اقتصادی مشترک بین شرکت‌های Acorn Computers، Apple Computer (هم اکنون با نام Apple Inc) و VLSI Technology بود. پردازنده‌های تولیدی شرکت ARM ، نتیجه‌ی پیشرفت و توسعه تراشه‌های ابتدایی شرکت Acorn با معماری RISC بودند که امروزه در بسیاری از تراشه‌های با کاربرد خاص (ASIC) استفاده می‌شوند. این شرکت از پیشگامان و بزرگ‌ترین تولید کنندگان تراشه‌های تلفن‌های همراه به شمار می‌آید.

---

[۱] PC

[۲] در واقع، WDC تراشه نمی‌ساخت بلکه طرح و نقشه‌ی پردازه‌هایش را به شرکت‌های سازنده‌ی مدارات مجتمع (IC) می‌فروخت.

[۳] ARM Holdings

[۴] SOC

۱.۲.۲ تاریخچه

این شرکت، در سال 1990 با نام Advanced RISC Machines ، تأسیس شد و در سال 1993 اولین بهره‌برداری تجاری از این شرکت صورت گرفت. دو شعبه از آن در Silicon Valley (واقع در کالیفرنیای شمالی) و توکیو در سال 1994 آغاز به کار نمودند. در سال 1997 شرکت ARM با سرمایه گذاری در شرکت Palm chip وارد بازار هارددیسک و تولید تراشه‌های SOC شد و در سال 1998 نام خود را از Advanced RISC Machines Ltd ، به ARM Ltd تغییر داد. در همین سال نامش در فهرست بورس اوراق بهادار لندن و آمریکا قرار گرفت. ARM ، شرکت Micro Logic Solutions را نیز به عنوان مشاور نرم‌افزاری در سال 1999 خریداری کرد.
این مجموعه مراکز طراحی گوناگونی در کالیفرنیا، تگزاس، واشینگتن، نروژ، فرانسه، آلمان، تایوان و... دارد.

شکل ۱.۱ – شرکت ARM Holding واقع در شهر Cambridge

از مشخصات اصلی پردازنده‌های ARM ، مصرف توان کم آنان است که کاربرد آن‌ها را در طراحی دستگاه‌های قابل حمل (Portable Device) که از باتری استفاده می‌کنند، بسیار مناسب ساخته است. امروزه ARM بیشترین سهم در بازار محصولاتی همانند تلفن‌های همراه، سامانه‌های دستی[۱] را دارد[۲].

---

[۱] PDA

[۲] با سهمی معادل ٪۷۵ از تمام پردازنده‌های ساخته شده‌ی ۳۲ بیتی

پردازنده‌های ARM در بیشتر تلفن‌های همراه شاخته شده توسط شرکت‌های معروفی چون Nokia، Sony Ericsson و SAMSUNG به عنوان پردازنده‌ی مرکزی (CPU) استفاده می‌شود. در سامانه‌ی دستی iPad ساخت Apple و در دستگاه‌های بازی Nintendo، دستگاه‌های مسیریابی دستی (GPS)، دوربین‌های عکاسی دیجیتال، تلویزیون‌های دیجیتال، تجهیزات شبکه و ابزارهای ذخیره سازی داده، حضور فعال ARM دیده می‌شود. پردازنده‌ی WLAN دستگاه PSP شرکت Sony نیز یک ARM9 است.

همان‌طور که قبلاً نیز اشاره شد، برخلاف دیگر تولید کنندگان پر شهرت پردازنده‌های ۳۲ بیتی همانند AMD، Intel، FreeScale و Renesas، شرکت ARM ، پردازنده‌هایش را به عنوان مالکیت معنوی[۱] به فروش می‌رساند. بدین معنی که به جای ساختن پردازنده‌های سیلیکونی، طرح و نقشه‌ی پردازنده‌هایش را در معرض فروش گذارد. بنابراین شرکت‌های بسیاری به عنوان **سازندگان** تراشه‌های ARM وجود دارند. شرکت‌های Intel، Texas Instrument، FreeScale و Renesas همگی مجوز ساخت تراشه-های ARM را خریداری نموده‌اند. در سال 2007 تعداد ۲.۹ بیلیون تراشه بر مبنای پردازنده‌های ARM در سراسر دنیا ساخته شد.

آقای Warren East ، در اکتبر سال 2001 به عنوان مدیرعامل شرکت سهامی تجاری ARM انتخاب شد. او سالانه به میزان ۴۱۵,۰۰۰ پوند به عنوان حقوق و ۲۸۶,۵۰۱ پوند به عنوان پاداش به کارکنانش پرداخت کرد.

معماری ARM ، معرف یک پردازنده و مجموعه‌ای از دستورالعمل‌های ۳۲ بیتی است که توسط شرکت سهامی تجاری ARM طراحی و توسعه داده شد. پردازنده‌های معروف و پرکاربردی که توسط ARM Holdings ارایه شده‌اند، عبارت‌اند از ARM7، ARM9، ARM11 و Cortex.

تعدادی دیگر از پردازنده‌های ARM با مجوز ساخت این شرکت، ولی در شرکت‌های دیگر توسعه داده شده‌اند. از آن دسته می‌توان به: Strong ARM شرکت DEC، i.Mx شرکت Freescale، XScale از اینتل، Nintendo، NVidia Tegra، STEricsson Nomadik، Snapdragon شرکت Qualcomm، OMAP از TI، Apple A4 و Hummingbird شرکت سامسونگ اشاره کرد.

[۱] IP - Intellectual Property

شکل ۱.۲ – پردازنده Apple A4 به کار رفته در iPad، iPhone 4 و نسل چهارم iPod Touch

شرکت Acorn Computers Ltd ، بعد از کسب موفقیت نسبی در ساخت و تولید کامپیوترهای BBC Micro، به فکر توسعه و پیشرفت طرح‌های ساده و پایه بر مبنای پردازنده‌ی 6502 (ساخت MOS Technology) به‌منظور ربودن گوی سبقت از شرکتی همچون IBM در تولید PC هایش افتاد. رایانه‌های تجاری Acorn (Acorn Business Computer) نیاز به تعدادی از پردازنده‌های جدید مناسب برای پلتفرم BBC Micro داشتند. اما پردازنده‌هایی چون 68000 شرکت Motorola و 32016 ساخت National Semiconductor مناسب نبودند و پردازنده‌ی 6502 نیز قدرت لازم برای اجرای واسط‌های گرافیکی کاربر[۱] را نداشت.

Acorn ، بعد از آزمایش بر روی تراشه‌های موجود، نیاز به معماری جدیدی را حس می‌کرد. شرکت Acorn در آن زمان به طور جدی به فکر طراحی پردازنده‌ای مخصوص به خود را گرفت و مهندسانش با عضویت در پروژه‌ی RISC دانشگاه Berkeley ، اقدامی جدی در این مورد انجام دادند. مهندسان شرکت Acorn با نام‌های Steve Furber و Sophie Wilson ، با خود اندیشیدند که تولید این پردازنده‌ی ۳۲ بیتی جدید توسط تعدادی از دانش آموختگان و فارغ‌التحصیلان و بدون نیاز به منابع گسترده‌ی تحقیق و توسعه[۲]، امکان‌پذیر است.

Wilson ، اولین کسی بود که در این زمینه اقدام به توسعه مجموعه‌ای از دستورالعمل‌ها نمود. او با شبیه سازی دستورالعمل‌هایش در BBC Basic، مهندسان Acorn را متقاعد نمود که در مسیر صحیحی گام برداشته‌اند. اما، قبل از ادامه کار آن‌ها نیاز به منابع بیشتری داشتند. او، با راضی کردن مدیرعامل Acorn و در جریان گذاشتن پیشرفتی که حاصل شده بود، تیمی تحقیقاتی برای پیاده سازی مدل ارایه شده به صورت سخت‌افزاری، تشکیل داد.

---

[۱] GUI

[۲] R&D - Research and Development

۱.۲.۳ تولد Acorn RISC Machine ARM2

پروژه Acorn RTSC Machine ، در سال 1983 آغاز شد. شرکت VLSI Technology، با سابقه تجهیز شرکت Acorn به تراشه‌های ROM و تراشه‌های سفارشی دیگر، به عنوان شریک تجاری در ساخت تراشه‌های سیلیکونی انتخاب شد. VLSI اولین تراشه‌های سیلیکونی ARM را در 28 آوریل 1985 تولید نمود. این اولین تراشه‌ی تولیدی این خانواده از پردازنده‌ها بود و با نام ARM1 شناخته شد. نخستین نمونه‌ی تولیدی که در سال بعد در دسترس قرار گرفت، ARM2 نام داشت.

ARM2 گذرگاه داده ۳۲ بیتی، فضای آدرس‌دهی ۲۶ بیتی و ۱۶ ثبات ۳۲ بیتی داشت. ۴ بیت بالا و ۲ بیت پایین شمارنده‌ی برنامه[۱] به عنوان پرچم‌های وضعیت[۲]، انتخاب می‌شوند. به همین علت ۲۶ بیت برای آدرس‌دهی فضای حافظه‌ی کد باقی می‌ماند. از این رو، حافظه‌ی کد نمی‌توانست از 64MByte بیشتر باشد.

ARM2 ، احتمالاً ساده‌ترین و مفیدترین پردازنده‌ی ۳۲ بیتی آن‌زمان بود. در مقایسه با تراشه‌ی 68000 موتورولا با ۷۰,۰۰۰ ترانزیستور که ۶ سال از عمرش می‌گذشت، ARM2 تعداد ۳۰,۰۰۰ ترانزیستور داشت.

سادگی عمده آن مربوط به نداشتن میکروکد بود و همانند بیشتر CPU های آن‌زمان، از داشتن حافظه‌ی نهان[۳] محروم بود. این سادگی منجر به مصرف توان کم آن به‌رغم کارآیی بهترش نسبت به 80286 شرکت Intel می‌شد. پردازنده‌ی بعدی این خانواده یعنی ARM3، 4KB ، حافظه‌ی نهان به همراه داشت که منجر به افزایش کارآیی آن گردید.

شرکت‌های Apple Computer و VLSI Technology ، در اواخر دهه‌ی 1980 با همکاری Acorn، کار بر روی تراشه‌های جدیدی از خانواده ARM را آغاز کردند. این همکاری مشترک، منجر به ساخت ARM6 در اوایل سال 1992 شد. Apple ، پردازنده‌ی ARM610 را به عنوان قلب سامانه‌ی دستی جدید خود Apple Newton به کار برد. شرکت Acorn نیز در سال 1996، ARM610 را به عنوان پردازنده‌ی مرکزی در رایانه‌های Risc PC خود استفاده کرد.

شرکت DEC ، مجوز ساخت معماری ARM6 را خریداری و Strong ARM را تولید کرد. این پردازنده در فرکانس کاری 233MHZ کمتر از 1watt توان مصرف می‌کرد. خط تولید Strong ARM متعاقباً به شرکت Intel واگذار شد و Intel آن را جایگزین خط تولید تراشه‌های قدیمی i960 خود نمود.

شرکت Intel ، طرحی دیگر بر مبنای ARM6 به نام XScale را ساخت و تعداد زیادی از آن را به فروش رساند. این تراشه‌ی کارآمد بعد از مدتی به شرکت Marvell فروخته شد.

ابعاد پردازنده‌ی ARM در این مسیر تکاملی، تغییر چندانی نداشت. به نحوی که ARM2 دارای ۳۰,۰۰۰ ترانزیستور بود، در حالی که ARM6 فقط ۳۵,۰۰۰ ترانزیستور دارد. سیاست شرکت ARM همواره

---

[۱] PC – Program Counter

[۲] Status Flag

[۳] Cache

برپایه تولید و فروش مالکیت معنوی پردازنده‌هایش برای تولید CPU و یا میکروکنترلرهایی براساس ARM بوده است که در این بین موفق‌ترین آن‌ها پردازنده‌ی ARM7TDMI با فروش صدها میلیون نمونه بوده است. ایده‌ی کار به این صورت است که تولید کنندگان تراشه‌های سیلیکونی، هسته‌ی ARM را با اجزای اختیاری ترکیب کردند تا یک CPU کامل تهیه کنند. شرکت Atmel ، از مراکز طراحی پیشگام در تولید سیستم‌های تعبیه شده[۱] با هسته مرکزی ARM7TDMI بوده است.

شرکت ARM در سال 2005 تعداد ۱.۶ بیلیون تراشه را به فروش رساند که از این میزان، تعداد ۱ میلیارد آن در تلفن‌های همراه استفاده شد. میزان فروش این تراشه‌ها از سال 2008 به بیش از ۱۰ میلیارد رسیده و طبق پیش‌بینی‌های انجام گرفته، این تعداد در سال 2011 به ۵ میلیارد فروش در سال خواهد رسید.

پردازنده‌ی مورد استفاده در تلفن‌های همراه هوشمند، سامانه‌های دیجیتال دستی و دستگاه‌های قابل حمل از نمونه‌های منسوخ شده ARMv5 گرفته تا انواع هسته‌های Cortex با قابلیت‌های بالا هستند. پردازنده‌های XScale و ARM926 از نسخه ARMv5TE بوده و Strong ARM، ARM9TDMI و ARM7TDMI از نسخه ARMv4 هستند. حضور دو پردازنده‌ی اول در دستگاه‌های حرفه‌ای از پردازنده‌های دوم پررنگ‌تر است. در مقابل دستگاه‌های ارزان قیمت، از هسته‌های با مجوز ساخت ارزان‌تر بیشتر استفاده می‌کنند.

پردازنده‌های ARMv6 ، گامی فراتر از انواع ARMv5 برداشتند و در دستگاه‌های خاصی استفاده می‌شوند، در حالی که پردازنده‌های ARMv7 (یعنی Cortex) مصرف توان کمتر و سرعت بیشتری را نسبت به تمام نمونه‌های قبلی خود ارایه می‌کنند. هسته های Cortex-A ، در پردازنده‌های کاربردی[۲] که قبلاً از ARM9 ویا ARM11 بهره می‌بردند، استفاده می‌شوند. Cortex-R نیز مورد استفاده کاربردهای بی‌درنگ[۳] بوده و سری Cortex-M میکروکنترلرها را هدف قرار داده‌اند.

در سال 2009 بعضی سازندگان رایانه‌های Net book در رقابت با انواعی که از پردازنده‌های Atom شرکت Intel بهره می‌بردند، به استفاده از CPU های با هسته ARM ، روی آوردند.

۱.۲.۴ هسته‌های مختلف پردازنده‌های ARM

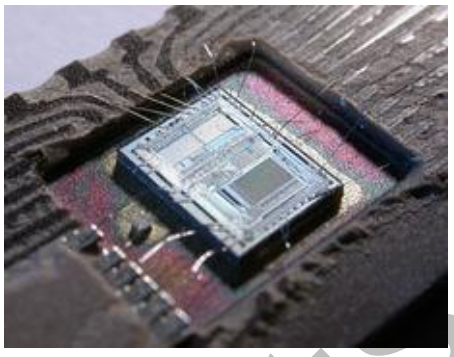
شرکت ARM ، فهرستی از فروشندگان عناصر نیم‌هادی را که در طرح‌های خود از ARM بهره برده‌اند را ارایه کرده است. KEIL نیز فهرستی جدیدتر و به‌روزتر را فراهم آورده است. بررسی مفصل خانواده‌های گوناگون پردازنده‌های ARM در ضمیمه الف، آمده‌اند.

---

[۱] Embedded System
[۲] Application Processor
[۳] Real-time Applications

۱.۳ میکروکنترلر چیست؟

میکروکنترلر (با نام‌های µC، uC و MCU نیز شناخته می‌شود)، یک رایانه‌ی کوچک بر روی مدار مجتمع[۱] است که شامل یک پردازنده، حافظه و وسایل ورودی/ خروجی قابل برنامه‌ریزی است. اغلب حافظه‌هایی از جنس Flash و یا OTPROM (حافظه‌ی غیر قابل پاک شدن) و RAM بر روی این تراشه‌ها موجودند. میکروکنترلرها برای سیستم‌های تعبیه شده طراحی و ساخته شده‌اند.



شکل ۱.۳ – نمای داخلی یکی از میکروکنترولرهای ابتدایی

ریزپردازنده‌ها که در کامپیوترهای شخصی و کاربردهای عمومی استفاده می‌شوند، برخلاف میکروکنترلر فاقد حافظه و دیگر مدارات جانبی هستند.
میکروکنترلرها در دستگاه‌های خود-کنترل شونده[۲] همانند سیستم کنترل موتورخودرو، قطعات کاشت[۳] در علم پزشکی، ریموت کنترل‌ها، ماشین‌های اداری، ابزارهای تغذیه و اسباب‌بازی‌ها استفاده می‌شوند. میکروکنترلر در مقایسه با سیستمی که از یک پردازنده‌ی مجزا به همراه حافظه و دستگاه‌های ورودی/ خروجی استفاده می‌کنند، طرحی اقتصادی و مقرون به‌صرفه را برای کنترل آن‌ها ارایه می‌کند. این در حالی است که حتی کارآیی بهتر و توانایی کنترل وسایل بیشتر نیز در آن‌ها گنجانده شده‌اند.
میکروکنترلرهای سیگنال- مرکبِ آنالوگ و دیجیتال[۴] نیز امروزه بسیار رایج هستند که کنترل وسایل غیردیجیتال را نیز ممکن ساخته‌اند. میکروکنترلرها دارای تنوع ساختی گوناگون با قابلیت‌های مختلف هستند. بعضی‌ها از کلمات[۵] ۴ بیتی استفاده کرده و در فرکانس کلاک 4KHZ کار می‌کنند. توان مصرفی در این کنترل کننده‌ها بسیار پایین است (در حد mw و یا حتی µw).

---

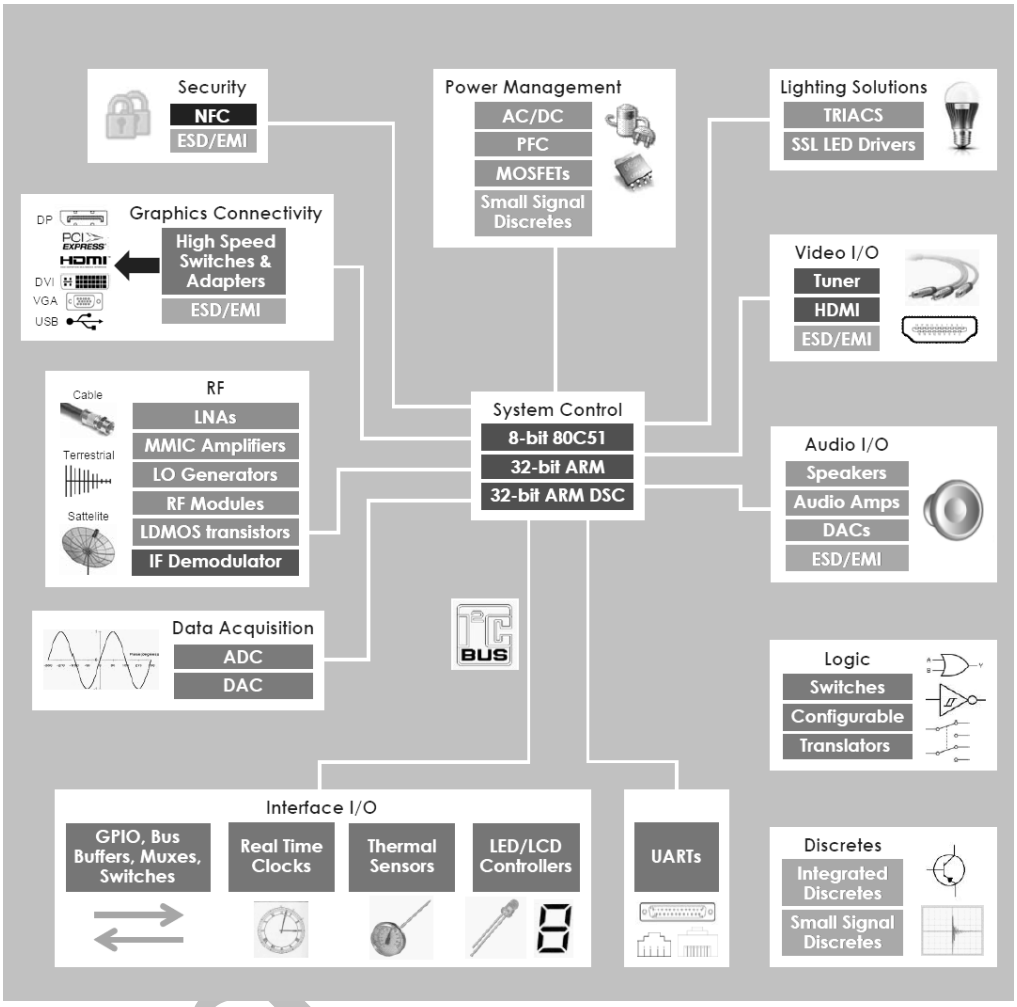
[۱] IC - Integrated Circuits
[۲] به صورت خودکار کنترل می‌شوند
[۳] Implant
[۴] Mixed Signal
[۵] Word

MCU ها، معمولاً از وقفه‌ها برای پاسخ‌گویی به فشار کلیدهای ورودی و امثال آن پشتیبانی می‌کنند. در حالتی که میکروکنترلر به خواب[1] رفته است (حالتی کـه فرکـانس پردازنـده و اجـزای جـانبی قطـع شـده است)، توان مصرفی تا حد nanoWatt نیز می‌رسد. این مصرف توان بسیار کـم، اسـتفاده از ایـن نـوع میکروکنترلرهـا را در وسـایل بـاتری‌دار، بسـیار مناسـب سـاخته اسـت. در جـایی دیگـر ممکـن اسـت میکروکنترلر نقش یک پردازنده‌ی قدرت‌مند سیگنال دیجیتال[2] را بازی کند یا مـدارهایی کـه از حساسـیت بالایی برخوردار بوده و نیاز مبرم به میکروکنترلری دارند که پاسخگویی بی‌درنگی داشته باشند.

۱.۳.۱ سیستم تعبیه شده

میکروکنترلر را می‌تـوان مجموعـه‌ای کامـل از یـک سـری تشـکیلات دانسـت کـه در قلـبش پردازنـده و درکنارش حافظه و وسایل جانبی قرار داشته و به عنوان یک سیسـتم **تعبیـه شـده** اسـتفاده مـی‌شـود. امروزه بیشتر میکروکنترلرهای موجود در درون دستگاه‌های دیگر جاسازی شده‌انـد. در دسـتگاه‌هـایی چون خودروهای شخصی، تلفن‌ها، وسایل جانبی مرتبط با رایانه و... این تراشه‌ها به چشـم مـی‌خورنـد. این وسایل، سیستم‌های تعبیه شده نامیده می‌شوند. در حالی که بعضی از سیستم‌های تعبیه شده بسیار تکمیل بوده و از انواع حافظه با یک پردازنده‌ی قدرت‌مند بهره می‌برند، بسیاری از آن‌ها کم‌تـرین حافظـه لازم را برای کد و داده در دسترس داشته و از نرم‌افزار سـاده‌ای اسـتفاده مـی‌کننـد. قطعـات ورودی و خروجی پرکاربردی که بـه میکروکنترلرهـا متصـل مـی‌شـوند، عبـارت‌انـد از کلیـد، سـولنوییـد، LED و نمایشگرهای LCD، قطعات فرکانس بالا و سنسورها برای جمع‌آوری داده‌هایی چون دما، رطوبت، شدت نور و ... .

[1] Sleep
[2] DSP – Digital Signal Processor



شکل ۱.۴ – شمای بلوکی یک سیستم تعبیه شده

۱.۳.۲ وقفه‌ها

میکروکنترلرها، می‌بایست پاسخی بی‌درنگ (قابل پیش‌بینی ولی نه لزوماً سریع) به رخدادهای[۱] موجود در سیستمی که تحت کنترلشان است، بدهند. وقتی رخدادهای خاصی روی می‌دهند، سیستم کنترل وقفه، پردازنده را مطلع کرده تا دستورالعمل جاری را متوقف ساخته و شروع به اجرای یک **روال سریع وقفه**-[۲] نماید. بسته به منبع تولید کننده وقفه و نوع آن، روال سریع دهنده به وقفه، پردازش لازم را انجام داده و سپس به روند اجرای اصلی برنامه باز می‌گردد. منابع تولید وقفه، عمدتاً به وسایل جانبی مربوط

---

[۱] Event

[۲] ISR - Interrupt Service Routine

هستند. به عنوان مثال، سرریز زمان سنج[1]، رویداد اتمام عملیات تبدیل آنالوگ به دیجیتال، تغییر سطح منطقی یکی از پایه‌های دیجیتال ورودی و یا دریافت داده روی کانال ارتباطی، می‌توانند از رویدادها و یا منابع تولید وقفه باشند. هنگامی که میکروکنترلر در مداری کار می‌کند که باتری داشته و مصرف توان در آن حیاتی است، میکروکنترلر در وضعیت خواب[2] قرار می‌گیرد و سپس با رخداد یک وقفه، از حالت سکون درآمده و شروع به اجرای عملیاتی خاص می‌کند.

۱.۳.۳ برنامه‌ها و یا کد

از آن‌جایی که استفاده از حافظه‌ی خارجی معمولاً هزینه‌بر است، کدهای نوشته شده برای میکروکنترلرها باید در فضای محدود حافظه داخلی جای بگیرند. کامپایلرها و اسمبلرها، وظیفه‌ی تبدیل زبان‌های سطح بالا و زبان اسمبلی را به کدهای فشرده‌ی ماشین[3] برای ذخیره سازی بر روی حافظه‌ی داخلی میکروکنترلر را دارند. بسته به دستگاه و محصول نهایی، برنامه میکروکنترلر می‌تواند به طور دایم بر روی تراشه قرار بگیرد. حافظه‌ی مورد استفاده برای این منظور می‌تواند از نوع ROM که فقط قابل خواندن هستند (یک‌بار برنامه‌ریزی می‌شوند) و یا انواعی که قابلیت پاک شدن را دارند، مانند Flash و یا E2PROM باشد.

۱.۳.۴ دیگر مشخصات میکروکنترلرها

میکروکنترلرها، عموماً تعداد زیادی پایه‌های ورودی و خروجی همه‌منظوره (GPIO) دارند. این پایه‌ها توسط نرم‌افزار به عنوان ورودی و یا خروجی تعریف می‌شوند. هنگامی که یک پایه به عنوان ورودی تعریف شود، برای خواندن سیگنال‌های خارجی و یا داده‌های سنسورها استفاده می‌شود. پایه‌ها در حالت خروجی برای راه‌اندازی رله‌ها، LED و یا موتورها و... مورد استفاده قرار می‌گیرند.
بسیاری از سیستم‌های تعبیه شده نیاز به خواندن اطلاعات سنسوری دارند که خروجی آنالوگ تولید می‌کنند. از آن‌جایی که طراحی پردازنده به گونه‌ای است که فقط توانایی پردازش داده‌های دیجیتال را دارد، میکروکنترلرها از وسیله‌ای به نام "مبدل آنالوگ به دیجیتال "[4] برای تبدیل کمیت‌های آنالوگ به دیجیتال استفاده می‌کنند. قابلیت دیگری که کمتر رایج است، وجود "مبدل دیجیتال به آنالوگ"[5] برروی تراشه است، که عکس عملیات فوق را انجام می‌دهد.

---

[1] Timer Over Flow
[2] Sleep State
[3] Machine Code
[4] ADC – Analog to Digital Converter
[5] DAC – Digital to Analog Converter

قابلیت‌های دیگری که بر روی این تراشه‌ها وجود دارند، تایمرها یا زمان سنج است. یکی از رایج‌ترین زمان سنج‌ها، "زمان سنج با قابلیت برنامه‌ریزی دوره‌ای"[۱] است. این زمان سنج- شمارنده از یک عدد از پیش تعیین شده تا مقدار صفر به سمت پایین و یا از صفر تا میزان حداکثر ظرفیت، شروع به شمارش می‌کند. هنگام رسیدن به عدد صفر و یا میزان حداکثر، معمولاً یک وقفه ایجاد می‌شود که از آن می‌توان برای زمان سنجی بین رویدادها استفاده کرد.
واحد پردازش زمان[۲] علاوه بر شمارش زمان، وظیفه تولید سیگنال‌های خروجی و یا استفاده از سیگنال-های ورودی برای شروع و یا پایان عملیات شمارش را دارد. بلوک تولید مدولاسیون پهنای پالس[۳] نیز در کنار زمان سنج به CPU این امکان را می‌دهد که بدون تلف نمودن وقت پردازنده مرکزی، به طور نیمه خودکار به کنترل دستگاه‌هایی چون مبدل‌های توان[۴]، بارهای مقاومتی[۵]، موتورها و... بپردازد.
واحد ارسال و دریافت داده آسنکرون[۶]، به سادگی امکان ارتباط سریال آسنکرون داده‌ها را برای CPU فراهم می‌کند. واحدهای دیگری نیز موجودند که برای ارتباطات سریالی از نوع SPI و یا I^2C و... مفید هستند.

۱.۳.۵ افزایش یک‌پارچگی و مجتمع سازی

در مقایسه با CPU های همه منظوره، میکروکنترلرها دارای حافظه‌ی داخلی RAM و حافظه‌ی غیرفرار ROM هستند. به همین علت نیازی به گذرگاه‌های خارجی برای اتصال حافظه اضافی نیست. استفاده از تعداد پایه‌های کمتر منجر به تولید تراشه‌هایی با بسته‌بندی کوچک‌تر و ارزان‌تر نیز می‌شود.
میکروکنترلر، تراشه‌ای منفرد با ویژگی‌های زیر است:

- پردازنده‌ی مرکزی- از پردازنده‌های سبک و کوچک ۴- بیتی گرفته تا انواع ۳۲ یا ۶۴ بیتی؛
- بیت‌های جداگانه ورودی و خروجی- کنترل هریک از پایه‌های ورودی و خروجی به تنهایی را ممکن ساخته است؛
- ورودی- خروجی‌های سریال همانند درگاه‌های سریال UART (Serial Port) ؛
- واسطه‌های ارتباطی دیگر همانند I^2C، SPI و CAN که برای ارتباط بین چندین دستگاه استفاده می‌شوند؛
- وسایل جانبی چون زمان سنج‌ها، شمارنده‌های رویداد، تولید کننده‌های شکل موج PWM و نگهبان[۷]؛

---

۱ PIT - Programmable Interval Timer
۲ TPU – Time Processing Unit
۳ PWM – Pulse Width Modulation
۴ Power Converter
۵ Resistive Loads
۶ UART
۷ Watchdog

- حافظه‌ی فرار برای ذخیره سازی داده‌ها (RAM) ؛
- حافظه‌ی غیرفرار برای ذخیره سازی برنامه‌ها (Flash، EPROM، E^2PROM و ROM) ؛
- تولید کننده‌های پالس کلاک ـ نوسان سازی برای تولید شکل موج متناوب از یک کریستال کوارتز، یک مدار RC و... ؛
- مبدل‌های آنالوگ به دیجیتال (در اکثر آن‌ها موجود است) ؛
- مدار واسط برای برنامه‌ریزی و عیب یابی برنامه‌ی روی تراشه.

این مجتمع سازی، به میزان قابل ملاحظه‌ای استفاده از تراشه‌های جانبی برای انجام عملیات مشابه را کاهش می‌دهد. کاهش این مدارات، تراشه‌ها و سیم‌بندی‌های اضافی، منجر به سادگی و کاهش هزینه‌های اضافی می‌گردد. در این مدل طراحی، هر پایه، دارای چندین کارکرد (مربوط به وسایل جانبی روی تراشه) است که توسط نرم‌افزار، هریک از آن‌ها را می‌توان انتخاب کرد. به عنوان مثال، یک پایه ممکن است هم یک پایه GPIO و هم یکی از پایه‌های واسط سریال باشد. در این روش، امکان قرار دادن قابلیت‌ها و کارکردهای بیشتری بر روی میکروکنترلر وجود دارد. (نسبت به حالتی که هر پایه، مخصوص یک کار ساخته شده باشد.)

از دهه‌ی 1970 ، میکروکنترلرها و استفاده‌شان همگانی شد. بعضی از میکروکنترلرها از معماری Harvard استفاده می‌کنند. بدین معنا که گذرگاه‌های[1] جداگانه‌ای برای دستورها و داده‌ها وجود دارند. در این معماری، دستورها و داده‌ها می‌توانند به طور هم‌زمان واکشی شوند که افزایش سرعت را در پی دارد. همچنین، استفاده از عرض بیت‌های مختلفی برای داده و دستورها ممکن است. به عنوان مثال، در سیستمی عرض بیت دستورها، ۲۶ بیت و عرض بیت داده‌ها و ثبات‌ها، ۸ بیت هستند. در مقابل، معماری دیگری به نام Von Neumann وجود دارد که درآن گذرگاه‌های داده و دستورالعمل، یکی هستند.

۱.۳.۶ بازار فروش میکروکنترلرها

نزدیک ٪۵۵ از تمام CPU های فروخته شده در دنیا، میکروکنترلرها و میکروپروسسورهای ۸ بیتی بوده‌اند. طبق آمار ارایه شده توسط گروه تحقیقاتی نیمه هادی Semico، در سال 2008 تعداد ۴ میلیارد میکروکنترلر ۸ بیتی فروخته شد.

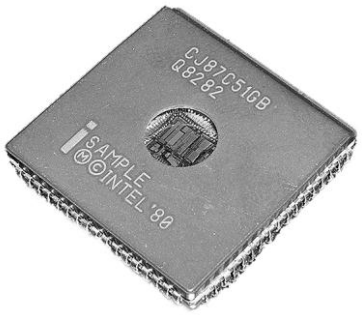
در یک خانه واقع در کشوری توسعه یافته، اغلب، دستگاه‌هایی با چندین میکروپروسسور وجود دارند. این در حالیست که در همان خانه بیش از ۲۰ میکروکنترلر در دستگاه‌های مختلف به کار گرفته شده‌اند. وسایلی همچون ماشین لباس‌شویی، اجاق‌های مایکروویو، تلفن‌ها و خودروهای شخصی، تعداد زیادی از این تراشه‌ها را در خود جای داده‌اند.

سازندگان و تولید کنندگان میکروکنترلرها، به طراحان امکانات متنوعی را ارایه می‌نمایند تا در طرح‌های خود از آنان استفاده کنند. میکروکنترلرهای ابتدایی، از حافظه‌های ROM استفاده می‌کردند که با یک‌بار

---

[1] Bus

برنامه‌ریزی دیگر قابلیت تغییر را نداشتند. با پیشرفت تکنولوژی، حافظه‌های EPROM جایگزین آن‌ها شدند. در بالای تراشه "پنجره"ای مخصوص این حافظه قرار گرفت. این نوع حافظه، قابلیت پاک شدن را با استفاده از اشعه ماورا بنفش (از طریق پنجره‌ی شیشه‌ای روی تراشه) را داشت. گام بعدی در طراحی جایگزینی این حافظه‌ها با انواعی به نام EEPROM در سال 1998 بود. این حافظه‌های جدید و حافظه‌های مشابه با نام Flash ، هزینه‌ی کمتری داشتند و از طریق مدارهای الکترونیکی می‌توانستند پاک شوند.



شکل ۱.۵ – میکروکنترولر 87C51 با EPROM داخلی

۱.۳.۷ محیط‌های برنامه نویسی

میکروکنترلرها، در اصل توسط زبان برنامه نویسی اسمبلی برنامه نویسی می‌شوند. ولی، با گذشت زمان زبان‌های برنامه نویسی سطح بالاتری نیز در دسترس قرار گرفتند. این زبان‌ها عمدتاً زبان‌های برنامه نویسی خاص منظور و یا نمونه‌های اصلاح شده زبان برنامه نویسی همه منظوره‌ی C هستند. کامپایلرها برای زبان‌های برنامه نویسی همه منظوره محدودیت‌ها و یا اصلاحاتی عرضه می‌کنند که تابع مشخصات میکروکنترلر مورد استفاده است. فروشندگان میکروکنترلرها عموماً ابزارهای برنامه نویسی مخصوص برای تراشه خود را نیز ارایه می‌کنند.

بعضی میکروکنترلرهای ابتدایی، از مفسر[۱] بر روی تراشه استفاده می‌کردند. به عنوان مثال، BASIC بر روی تراشه‌ی 8052 اینتل و FORTH برای تراشه‌ی Zilog Z8 از مفسر استفاده می‌کردند.

در بعضی از میکروکنترلرها، محیط‌های جامعی برای شبیه سازی کد میکروکنترلر در رایانه‌های شخصی نیز وجود دارند. MPLAB از شرکت Microchip نمونه‌ای از این شبیه سازها است. این قابلیت به طراح امکان می‌دهد که رفتار کامل میکروکنترلر را قبل از پیاده سازی مداری، کاملاً تجزیه و تحلیل کند.

[۱] Interpreter

اخیراً میکروکنترلرها به ابزار قدرت‌مندتری برای دیباگ و عیب‌یابی بر روی تراشه[1] و امولاتور داخل مدار[2] از طریق رابط JTAG مجهز شده‌اند. برنامه نویس از این طریق می‌تواند برنامه روی میکروکنترلر را عیب‌یابی و رفع خطا کند.
دسترسی به ابزارهای توسعه[3] برای راه‌اندازی سیستم‌های میکروکنترلری، یکی از ضروریات راه‌اندازی این سیستم‌ها و از عوامل مهم در انتخاب تراشه مورد نظر می‌باشند.

۱.۴ میکروکنترلرهای نسل بعد

میکروکنترلرها (MCUs)، دستگاه‌های مختلف از یک شوینده لباس خانگی گرفته تا یک وسیله‌ی حرفه‌ای چند رسانه‌ای را کنترل کرده یا به یکدیگر متصل می‌کنند. بازار ۲۰ سال گذشته‌ی میکروکنترلرها در دست میکروکنترلرهای ۸ بیتی بوده است. این در حالی است که افزایش کاربران حرفه‌ای و انتظارات آنان از دستگاه‌های تولیدی با قابلیت‌های سطح بالا، این بازار مصرف را به سمت میکروکنترلرهای ۱۰ بیتی و بعد از آن ۳۲ بیتی کشانده است. کارایی و قابلیت‌های این پردازنده‌ها می‌توانند پاسخ‌گوی نیازهای حرفه‌ای کاربران خود باشند.
گذر از دوران میکروکنترلرهای ۸ بیتی به سمت ۱۶ و ۳۲ بیتی به خوبی در جریان است. به نحوی که طبق تحقیقات انجام شده توسط Semico، در سال 2011 فروش میکروکنترلرهای ۳۲ بیتی به بیش از ۲ بیلیون قطعه با نرخ رشد سالانه‌ی ٪۱۸ خواهد رسید.

۱.۴.۱ محدودیت‌های میکروکنترلرهای ۸ و ۱۶ بیتی

میکروکنترلرهای ۸ و ۱۶ بیتی، برای دستگاه‌های سطح پایین و با انتظارات نسبتاً کم، بسیار مناسب هستند. دستگاه‌هایی یک‌پارچه و با حضور فقط یک پردازنده و تحت کنترل یک هسته نرم‌افزاری[4] ساده که برای انجام یک وظیفه ساده و خاص درنظر گرفته شده‌اند. نیاز به حافظه در این وسایل، محدود به حداکثر فضای قابل آدرس‌دهی در پردازنده‌های ۸ و ۱۶ بیتی است.
میانگین قیمت میکروکنترلرهای ۸ و یا ۱۶ بیتی با کارآیی متوسط، در حدود ۱ تا ۲ دلار است. این نرخ برای طراحی سیستم‌های ارزان قیمت با کارآیی حداقل، بسیار مناسب هستند.
در حالی که میکروکنترلرهای مذکور برای سیستم‌های ساده و سطح پایین مناسب هستند، امّا استفاده از آن‌ها در وسایلی که نیاز به محاسبات پیچیده و سنگین و وسایل جانبی گوناگون دارند، عملاً غیر ممکن است. محدودیت‌های پردازنده‌های ۸ و ۱۶ بیت را می‌توان در پهنای گذرگاه داده کم، بیشینه فضای قابل

---

[1] On-Chip Debug
[2] In-circuit Emulator
[3] Development Tools
[4] Kernel

آدرس‌دهی اندک، پشتیبانی محدود از زبان برنامه نویسی C و کمبود وسایل عیب‌یابی (debug) قدرت-مند دانست. میکروکنترلرهای ۱۶ بیتی علاوه بر موارد بالا، از نبود یک معماری استاندارد و عمومی نیز بی‌بهره‌اند.

۱.۴.۲ مزایای میکروکنترلرهای ۳۲ بیتی

برای بررسی برتری‌های یک میکروکنترلر ۳۲ بیتی بر نمونه ۸ بیتی‌اش، نمونه‌ای از کاربرد میکروکنترلرها در طراحی کنترل کننده‌ی موتور در صنعت یا در کاربردهای خانگی را درنظر بگیرید. سرعت و توان موتور توسط مدار مدولاتور پهنای پالس (PWM)، تثبیت می‌شود. به عنوان مثال، MCU ۸ بیتی توانایی اندازه‌گیری سرعت موتور و تغییر سیگنال PWM متناسب با آن را در هر ۱۰۰ms دارند. MCU ۳۲ بیتی در فرکانس کاری یکسان، توانایی معادل ۴ تا ۵ برابر MCU ۸ بیتی، با دقت بیشتر را خواهد داشت. زمان اضافی که از این طریق برای MCU ۳۲ بیتی حاصل می‌شود را می‌توان صرف اجرای هم‌زمان زیربرنامه‌های دیگر کرد. به عنوان مثال، تصحیح ضریب توان (Power Factor Correction-PFC) و یا اتصال به شبکه TCP/IP برای کنترل از راه دور و یا مشاهده‌ی پارامترها، از جمله وظایفی است که می‌تواند به طور هم‌زمان با کنترل موتور انجام شود.
MCU های ۳۲ بیتی، دارای توانایی محاسباتی بالا، فضای حافظه‌ی بیشتر و توان مصرفی کمتری هستند. این MCU ها از نرم‌افزارهای قدرت‌مند کاربر نیز پشتیبانی می‌کنند. قیمت نمونه‌ای از MCU های سطح بالای ۸ و یا ۱۶ بیتی با یک نمونه‌ی ۳۲ بیتی برابری می‌کند.

۱.۴.۳ مسایل درگیر با توسعه سیستم‌های ۳۲ بیتی

پردازنده‌های ۳۲ بیتی، دارای کارآیی بیشتری هستند و از نرم‌افزارهای نسل بعدی میکروکنترلرها پشتیبانی می‌کنند. مسأله‌ای که طراحان و کاربران را درگیر خود می‌کند، این‌است که کدام MCU ۳۲ بیتی علاوه بر برآورده کردن مشخصات وسیله‌ی مورد نظر در بهترین وضعیت ممکن، هزینه‌ی تمام شده را کاهش داده و ابزارهایی فراهم می‌آورد که از آن طریق طرحی مطمئن و بدون خطر و در کم‌ترین زمان ممکن برای تحویل محصول به بازار فروش داشته باشیم.
یک مسأله‌ی پیش‌روی طراحان، مدیریت توان مصرفی است. این مسأله در مورد دستگاه‌هایی که از باتری استفاده می‌کنند و یا دستگاه‌های خانگی و یا کنترل صنعتی هنگام "سازگاری با حداکثر توان مصرفی استاندارد" نمود بیشتری پیدا می‌کند.

توان مصرفی پویا[1] با $\mathbf{CV^2f}$ متناسب است که در آن f فرکانس کاری پردازنده است. بدیهی است که با فزونی فرکانس به‌منظور افزایش کارآیی پردازنده، توان مصرفی نیز بالا می‌رود. برای افزایش فرکانس کاری، ابعاد پردازنده می‌بایست کاهش یابد که این نیز منجر به اضافه شدن توان مصرفی می‌گردد.
حافظه‌ی فلش تعبیه شده (Embedded Flash) بیشترین سطح تراشه در اغلب MCU های امروزی را اشغال می‌کند. پس، بیشترین تأثیر بر افزایش قیمت پردازنده را دارد. جاسازی 256KB حافظه‌ی فلش به‌منظور اجرای یک سیستم‌عامل بی‌درنگ (RTOS)، الگوریتم‌های پیچیده و یا پروتکل‌های شبکه، امری عادی است. MCU ، باید دارای مشخصات و تجهیزاتی باشد که بتوان کمترین حجم کد برای کار با وسایل مختلف را بر روی آن جا داد و این امر، یعنی کاهش فلش مورد نیاز و هزینه‌ی تمام شده سیستم.
بعضی روش‌های کنترل وسایل مختلف از جمله موتورها، نیازمند انجام عملیاتی بحرانی و قطعی در زمان‌های خاص می‌باشد. برای یک MCU ۳۲ بیتی، داشتن یک حافظه‌ی نهان (Cache) چند مسیره (Multiway) و معماری خط لوله‌ی (Pipeline) چند طبقه به‌منظور افزایش کارآیی، امری ضروری است. یک چنین سیستمی به‌منظور عملکردی مطمئن و به دور از خطا، احتمالاً در یک بازه زمانی خاص می‌بایست به صدها وقفه پاسخ دهد. برای یک MCU مبتنی بر حافظه نهان، داشتن یک خط لوله سریع برای جلوگیری از خطاهای محتمل در کارکرد حافظه‌ی نهان (همانند Stall ها) و پاسخ‌دهی به وقفه‌ها و سایر استثنائات (Exception) امری ضروری است.
طرح‌های دیگری نیز وجود دارند که بر حافظه‌ی نهان مبتنی نیستند و در آن‌ها از SRAM استفاده می‌شود. پس، نیازی به این مراقبت‌های سخت گیرانه ندارد.

۱.۴.۴ مروری بر کاربردها و بازار فروش میکروکنترلرهای ۳۲ بیتی

وسایل صنعتی و لوازم مربوط به خودرو، دو نمونه از بزرگ‌ترین مصرف کنندگان MCU هستند.
در این بخش، پیرامون مسیر حرکت و روند رو به رشد استفاده از MCU ها در این دو حوزه، بحث می‌کنیم.

۱.۴.۴.۱ کاربرد در صنعت

محصولات صنعتی گوناگونی از MCU ها استفاده می‌کنند. به عنوان نمونه، می‌توان از موتورهای تک‌کاره، مبدل‌های توان، سیستم‌های شبکه‌ایِ ایمن و تجهیزات دارویی و پزشکی نام برد. این روزها مسیری رو به رشد از کاربرد MCU ها در حوزه مکاترونیک، راه خود را باز کرده است؛ جایی که سیستم‌های الکترونیکی هوشمند به جای و یا در کنار سیستم‌های مکانیکی قرار گرفته‌اند. به عنوان

[1] Dynamic

نمونه، تنظیم کننده‌ی دمای خانگی[1] از سیستم‌های هوشمندالکترونیکی برای کنترل خودکار دما و کاهش هرچه بیشتر هزینه انرژی مصرفی، بهره می‌برد.

بهره گیری از MCU ها در مکاترونیک، با افزودن نمایشگرهای LCD و یا استفاده از PWM به‌منظور کنترل هرچه بهتر سیستم کنترلی، کارکرد سیستم را افزایش می‌دهد. مسأله قابل توجه دیگر این است که می‌توان از یک سخت‌افزار یکسان برای کنترل چند سیستم مختلف استفاده کرد و یا فقط با تغییر نرم‌افزار، کارآیی‌های لازم را به آن اضافه کرد. امروزه وسایل دارای موتور به گونه‌ای طراحی می‌شوند که با سایر استانداردهای "سبز" همانند Energy Star® سازگار باشند. این استانداردها عموماً بر ایجاد مصالحه‌ی بهینه بین توان مصرفی و کارآیی ارایه شده، تأکید دارند. کارآیی MCU ها عموماً برحسب میلیون دستورالعمل در ثانیه Dhrystone [2] بر MHZ و بازدهی توان آن‌ها برحسب mw/DMIPS بیان می‌شوند. هرچه نرخ DMIPS/MHZ بیشتر باشد، بازدهی توان فزون‌تر بوده و فرکانس کاری کمتری برای رسیدن به همان کارآیی مورد نظر نیاز است.

ارتباط دستگاه‌های صنعتی با یکدیگر، روزبه‌روز افزایش می‌یابد. این ارتباطات، درون سازه‌ها و یا بیرون از آن و از طریق ارتباط بی‌سیم و یا باسیم، امکان‌پذیر است. در حالت ارتباط باسیم، MCU از ارتباط شبکه اترنت و راه‌اندازی پروتکل TCP/IP بر روی آن به‌منظور ایجاد یک ارتباط سالم و به دور از خطا بین دو نقطه استفاده می‌کند. در ارتباطات بی‌سیم نیز MCU همین کار را از طریق رابط MAC [3] شبکه اترنت به‌منظور ارسال اطلاعات باند پایه[4] بر روی Bluetooth یا Zigbee ، انجام می‌دهد. در هر دو مورد، MCU ای ۳۲ بیتی با قابلیت مدیریت حافظه[5] و توانایی اجرای RTOS ، انتخاب حتمی و مناسبی است.

هرچه اطلاعات منتقل شونده بر روی شبکه حساس‌تر باشند، نیاز بیشتری به وجود قابلیت‌های امنیتی بر روی MCU ها حس می‌شود. الگوریتم‌هایی چون AES، RSA، DES، ECC و... پیچیدگی بالایی دارند. MCU مورد نظر، باید کارآیی لازم بر اجرای این الگوریتم‌ها در جهت حفاظت از اطلاعات حساس را داشته باشد.

دستگاه‌های پزشکی عموماً دارای کارکردهای پیچیده‌تری نسبت به انواع صنعتی‌شان هستند. آن‌ها اغلب چندین کارکرد و ارتباط با سنسورهای مختلف را در خود جای داده‌اند. داده‌ها از سنسورهای مختلف جمع‌آوری شده و توسط مبدل‌های آنالوگ به دیجیتال با تکنولوژی سیگما- دلتا به مقادیر دیجیتال تبدیل می‌شوند. سپس، توسط تکنیک‌های پردازش سیگنال دیجیتال که اغلب میکروهای معمولی فاقد آن هستند،

---

[1] Thermostat
[2] DMIPS
[3] Medium Access Control
[4] Base Band
[5] Memory Management

پردازش می‌شوند. دستورهای ضرب و ضرب- جمع[1] از دستورهای پایه برای انجام عملیات و الگوریتم-های پردازش سیگنال دیجیتال هستند.
دستگاه‌های پزشکی دارای ارتباطات شبکه محلی LAN همانند آن‌چه در مورد دستگاه‌های صنعتی گفته شد نیز هستند.

۱.۴.۴.۲ صنایع خودرویی

MCU های ۳۲ بیتی، حضوری توان‌مند و رو به رشد در بازار قطعات خودرویی داشتند. این بازار، چند سال گذشته در دست MCU های ۸ و ۱۶ بیتی بود و از MCU های ۳۲ بیتی در کاربردهای حرفه‌ای‌تر و با کارآیی بالاتر استفاده می‌شد. تا این‌که اخیراً نیاز به پردازش بالا در صنایع خودرویی و همچنین نیاز به اجرای نرم‌افزارهای کنترلی هوشمندتر و تأمین امنیت بیشتر، تولید کنندگان را مجبور به بهره گیری از این MCU ها کرد. تقریباً ٪۵۰ ارزش یک خودروی نیمه حرفه‌ای، در سیستم الکترونیک آن و واحد کنترل الکترونیکی[2] اش نهفته است. ECU ها به سمت در دست گرفتن کنترل تمام اجزای خودرو از سیستم آسایش و مبلمان گرفته تا کنترل موتور و راه‌اندازی دستگاه‌های صوتی- تصویری، پیش می-روند.
کاربردهای نمونه‌ی یک MCU در حیطه‌ی سیستم‌های کنترلی خودرویی، عبارت‌اند از:

- بدنه و سیستم آسایشی:
  - پنجره‌های برقی؛
  - گرمایش و سیستم تهویه مطبوع؛
  - برف پاک‌کن‌ها و کنترل صندلی‌ها؛
  - در ورودی هوشمند و بدون کلید.
- امنیت و شاسی خودرو:
  - پایداری الکترونیکی (ESC)؛
  - کنترل کیسه‌های هوا (Airbag)؛
  - ترمزهای ضد قفل (ABS)؛
  - اندازه‌گیری فشار باد لاستیک‌ها؛
  - سیستم جلوگیری از برخورد و تصادف.
- سیستم محرکه و انتقال قدرت:
  - مدیریت موتور؛
  - کنترل جعبه دنده خودکار (سیستم انتقال قدرت)؛

---

[1] MAC – Multiply and ACcumulate
[2] ECU

- مدیریت سیستم هیبرید؛
- کنترل الکترونیکی انتقال توان بر روی چرخ‌ها.

- اطلاعات و مسیریابی:
  - مسیریابی؛
  - کنترل داشبورد.

سیستم محرکه و انتقال قدرت از لحاظ محاسباتی، پیچیده‌ترین بخش از بین تمام اجزای بالا به حساب می‌آید. در این بخش، نیاز به مدیریت انواع اطلاعات از سنسورهای موتور و کنترل بی‌درنگ در اجرای تصمیمات است. MCU مورد استفاده در این بخش، می‌بایستی برای کمترین تأخیر در پاسخ‌گویی به وقفه‌ها، پشتیبانی از واسطهای سریع آنالوگ و دیجیتال طراحی شده باشند، که دارای کارآیی بالا بوده و از دستورهای DSP بهره‌مند باشند. اغلب، پردازنده‌های با فرکانس کاری بیش از 300MHZ برای این منظور انتخاب می‌شوند. سیستم‌های خودرویی می‌بایست در طول عمر مفید خودرو، که عموماً بیش از ۱۰ سال است، ایمنی کامل سرنشینان را تأمین کرده و رفتاری قابل پیش‌بینی از خود به نمایش بگذارند.
ECU خودرویی بسته به شرایط و مکان قرارگیری‌اش درون خودرو می‌بایست توانایی کار در محدوده دمایی وسیع و شرایط محیطی گوناگون را داشته باشد. این قطعات باید محکم و ضربه پذیر بوده و توانایی تحمل ولتاژ/ جریان بیش از حد را داشته باشند. همچنین، دارای طول عمر میانگین تا خرابی[۱] بالایی بوده و تا دمای حداکثر $125^{\circ}C$+ کارکرد قابل قبولی از خود ارایه دهند.

۱.۵ خلاصه

در این فصل، با پردازنده‌های پر شهرت ARM و تاریخچه پیدایش شرکت ARM آشنا شدید و خانواده-های مختلف این پردازنده‌ها و کاربردهای هر کدام در دستگاه‌های گوناگون مختصراً معرفی شدند. میکروکنترلر را باید به عنوان عضوی اجتناب ناپذیر در سیستم‌های تعبیه شده امروزی دانست. بعد از بررسی کوتاه میکروکنترلر و الزاماتش، به بررسی میکروکنترلرهای ۳۲ بیتی نسل جدید با قابلیت‌های بالاتر پرداختیم.
امروزه، میکروکنترلرهای ۳۲ بیتی از اعضای اساسی سیستم‌های الکترونیکی حرفه‌ای و نیمه حرفه‌ای خودکار، نیمه خودکار و یا برنامه پذیر هستند. این تراشه‌ها از جنبه‌های گوناگون بررسی شدند و کاربردهای گوناگون آن‌ها را در حوزه‌ی دستگاه‌های صنعتی، پزشکی و صنعتِ خودرو دیدیم. در ادامه به بازار فروش این تراشه‌ها نیز نظری افکندیم.
در فصول بعد، با نگاهی به معماری ARM و جزئیات داخلی آن، به میکروکنترلرهای توان‌مند ARM که امروزه بسیار فراگیر شده‌اند، می‌پردازیم. این میکروکنترلرها را کاملاً بررسی کرده، وسایل جانبی آن را

---
[۱] Mean Time To Failure - MTTF

خواهیم شناخت و ضمن آشنایی با سازندگان مختلف و تراشه‌های گوناگون، نحوه‌ی کار با این میکروکنترلرها را خواهیم آموخت.

بخش ۲

معماری پردازنده ARM

فصل دوم

سیستمهای تعبیه شدهٔ ARM

اهداف فصل

با پایان این فصل شما با موارد زیر آشنا می‌شوید:

- ✓ فلسفه طراحی یک پردازنده‌ی RISC و تبدیل آن به پردازنده‌ی ARM؛
- ✓ مبانی طراحی پردازنده‌ی ARM و معماری داخلی گذرگاه‌های آن؛
- ✓ انواع حافظه، سیستم عامل؛
- ✓ خانواده‌های گوناگون پردازنده‌های ARM ، کدام‌اند؟

هسته پردازنده‌ی ARM بنیادگذار تعداد زیادی از سیستم‌های موفق ۳۲ بیتی بوده است. در حال حاضر ممکن است شما چنین سیستمی را در دست داشته باشید، حال آن‌که از آن بی‌اطلاع هستید!

پردازنده‌های ARM در بسیاری از سیستم‌های قابل حمل مانند گوشی‌های تلفن همراه، سازمان‌ده‌های دستی (Organizer) و چندین دستگاه پرکاربرد دیگر استفاده شده است.

پردازنده‌های ARM ، راه درازی را از اولین نمونه‌ی ارایه شده آن در سال 1985 پیموده‌اند. به طوری که تا سال 2001 میلادی، این شرکت توانست بیشتر از ۱ میلیون پردازنده را بفروشد!

این شرکت تمام پردازنده‌های خود را براساس طرح اولیه‌اش که بسیار ساده و قدرت‌مند بود، پیاده سازی کرد و طرح اولیه خود را پیوسته بهبود داد و توسعه بخشید. در حقیقت، ARM تنها یک پردازنده ساده نیست! ARM خانواده‌ای از پردازنده‌ها بر مبنای اصول طراحی و دستورالعمل‌های تقریباً یکسان است. ARM خانواده‌ی گسترده‌ای از پردازنده‌هاست! به عنوان مثال، ARM7TDMI یکی از توان-مندترین هسته‌های پردازشی ARM است! از مزایای اصلی این هسته، توان پردازشی ۱۲۰ میلیون دستورالعمل در ثانیه[۱]، مصرف توان کم و چگالی کد[۲] بالا است. این موارد از کلیدی‌ترین نکات، برای سیستم‌های قابل حمل[۳] هستند.

مبنای طراحی پردازنده‌های ARM ، معماری RISC [۴] است. در زیر، به بررسی معماری RISC و اصلاحات انجام شده روی آن، جهت دستیابی به پردازنده ARM، می‌پردازیم:

---

۱ Dhrystome MIPS
۲ Code density
۳ Portable Systems
۴ Reduced Instruction Set Computer

۲.۱ فلسفه طراحی یک پردازنده با معماری RISC

پردازنده‌های ARM از معماری RISC بهره‌منداند. در RISC ها، طراحی براساس دستورالعمل‌های ساده و در عین حال قدرت‌مند صورت گرفته است! که تنها در یک سیکل کاری پردازنده و در سرعت کلاک (فرکانس کاری پردازنده) بالا اجرا می‌شوند. هدف طراحی RISC ، کاهش پیچیدگی‌های سخت‌افزاری است. RISC با پیاده سازی نرم‌افزاری، توسط مجموعه‌ای از دستورالعمل‌های منعطف این امر را ممکن ساخته است. در RISC ها، عمده مسئولیت بر عهده‌ی کامپایلر است.

در مقایسه با RISC، سیستم‌های قدیمی‌تری به نام CISC هستند که بیشتر از سخت‌افزار سیستم برای اجرای دستورالعمل‌ها استفاده می‌کنند. به همین دلیل، دستورالعمل‌های سیستم CISC بسیار پیچیده‌تر است.

تفاوت‌های اصلی این دو سیستم در شکل ۲.۱ مصور شده است.

فلسفه طراحی RISC بر ۴ اصل استوار شده است:

۱. **دستورالعمل‌ها**: مجموعه‌ی دستورالعمل‌های پردازنده‌های RISC بسیار کم است. این دستورالعمل-ها شامل عملیات ساده‌ای است که تنها در یک سیکل ماشین قابل اجرا هستند. کامپایلر و یا پروگرامر، وظیفه‌ی پیاده سازی دستورالعمل‌های پیچیده و یا تبدیل آن‌ها به چندین دستورالعمل ساده را دارد. جهت این کار به **خطِ لوله**[۱] نیاز است. برای این که خط لوله درست و سریع کار کند، RISC دستورالعمل‌ها را با طول ثابت به خط لوله می‌فرستد.

این ویژگی RISC با معماری CISC که دستورالعمل‌هایش دارای طول متغیری هستند، در تقابل می-باشد؛

۲. **خط لوله**: در خط لوله، دستورالعمل‌ها به بلوک‌های کوچک‌تری تقسیم می‌شوند و به موازات یکدیگر اجرا می‌گردند. خط لوله در بهترین حالت، در هر سیکل ماشین یک مرحله به جلو می‌رود و دستورالعمل‌ها در یک طبقه‌ی خط لوله رمزگشایی[۲] می‌شوند. برخلاف CISC در معماری RISC نیازی به اجرای دستورالعمل‌ها توسط ریز_ برنامه‌هایی[۳] به نام micro code نیست؛

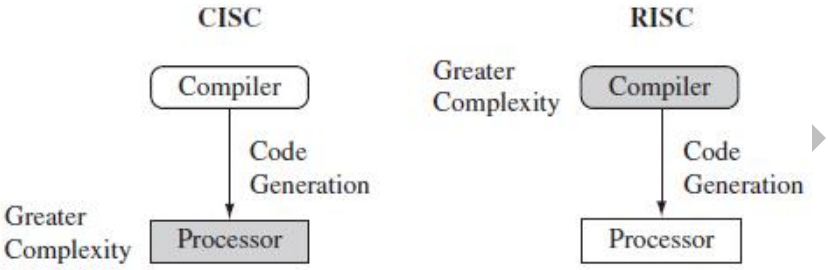
۳. **ثبات‌ها**: پردازنده‌های RISC دارای تعداد زیادی از ثبات‌ها با کاربر عمومی هستند. محتوای هر ثبات می‌تواند آدرس و یا داده باشد. ثبات‌ها به عنوان حافظه‌های محلی باسرعت در تمام عملیات دخیل هستند؛ در مقایسه، معماری CISC برای انواع عملیات، ثبات‌های جداگانه‌ای دارد؛

۴. معماری **بارگیری_ ذخیره** (Load-Store) : پردازنده، با داده‌های ذخیره شده در ثبات‌ها کار می-کند. وظیفه‌ی جابه‌جایی داده‌ها بین ثبات‌ها و حافظه‌ی خارجی، بر عهده‌ی دستورهای بارگیری و ذخیره است. از آن‌جایی که دسترسی به حافظه‌ی خارجی زمان‌بر است، پردازنده از ثبات‌ها برای

---

[۱] Pipeline
[۲] Decode
[۳] Mini Program

سرعت بخشیدن به محاسبات استفاده می‌کند. این در حالی است که CISC ها برای دستیابی مستقیم به حافظه، طراحی شده‌اند.
این قوانین، پردازنده‌های RISC را ساده‌تر ساخته‌اند. به همین دلیل است که RISC ها عموماً در فرکانس کاری بالاتری می‌توانند به کار گرفته شوند. CISC ها، دارای پیچیدگی بیشتری بوده و عموماً در فرکانس کاری کمتری به کار گرفته می‌شوند.



شکل ۲.۱ – مقایسه RISC و CISC

۲.۲ مبانی طراحی ARM

پردازنده‌های ARM ، اختصاصاً برای سیستم‌های کوچک و یا مصرف کم طراحی شده‌اند که با باتری تغذیه می‌شوند. مانند سیستم‌های قابل حمل از جمله: تلفن‌های همراه، سامانه‌های دستی دیجیتالی (PDA) و... .
سیستم‌های قابل حمل، علاوه بر مصرف توان، در میزان حافظه‌ی در دسترس و قیمت نیز، محدودیت دارند. این دستگاه‌ها نیازمند حافظه‌ای هستند که در فضای محدود قابل نصب باشند. پردازنده‌های ARM چگالی کد بالایی دارند و بدین ترتیب، میزان حافظه‌ی مورد نیاز را کاهش می‌دهند.
پردازنده‌ی ARM در سیستم‌های قابل حمل، از واسطهای حافظه‌ی ساده و ارزان قیمتی بهره می‌برد که در هزینه و قیمت تمام شده سیستم، نقش تعیین کننده‌ای بازی می‌کند. سطح تراشه‌ی مورد استفاده در سیستم‌های قابل حمل، از دیگر مسائلی است که درگیر آن هستیم. هرچه این مساحت کمتر باشد، فضای بیشتری برای دیگر مدارات در دسترس است و هزینه تمام شده‌ی آن نیز کمتر است.
ARM ها دارای سیستم اشکال‌زدایی پیشرفته هستند.
سیستم اشکال‌زدایی پیشرفته، از مداراتی است که به مهندسان دید درستی نسبت به وضعیت سخت‌افزار در حال اجرای نرم‌افزار مربوطه را می‌دهد. این سیستم باعث می‌شود مشکلات نرم‌افزاری با سرعت هرچه تمام‌تر برطرف شوند.

با استفاده از این تکنولوژی، زمان تحویل محصول به بازار[1] و هزینه‌های تمام شده به شدت کاهش می‌یابد. پردازنده‌ی ARM در بازار دارای محبوبیت است زیرا دارای سرعت بالا، ساختار ساده و مصرف توان قابل قبول است.

۲.۲.۱ دستورالعمل‌ها برای سیستم‌های تعبیه شده[2]

دستورالعمل‌های ARM ، شکل توسعه یافته‌ی معماری RISC است و نسبت به RISC چند خاصیت بیشتر دارد که همین امر ARM را برای سیستم‌های تعبیه شده، مناسب ساخته است. در زیر، به چند مورد از برتری‌های ARM نسبت به RISC اشاره می‌کنیم:

- زمان اجرای متفاوت برای برخی از دستورالعمل‌ها:
  تمامی دستورهای ARM در یک سیکل ماشین اجرا نمی‌شوند.
  مثلاً، دستورهای load-store-multiple بسته به تعداد ثبات‌های استفاده شده، در زمان‌های متفاوتی اجرا می‌شوند.
  ARM ، برای اجرای عملیات انتقال، به جای آن که از دسترسی‌های متعارفی (که وقت‌گیر است) استفاده کند، از مکان‌های متوالی حافظه بهره می‌برد که سریع‌تراند و چگالی کد را نیز افزایش می‌دهند؛
- استفاده از Inline barrel shifter برای بهره‌گیری از دستورهای پیچیده‌تر:
  Inline barrel shifter (ibs) ، سخت‌افزاری است که یکی از ثبات‌ها را برای استفاده‌ی دستورالعمل‌ها، پیش- پردازش می‌کند. سخت‌افزار (ibs) با پیاده‌سازی عملیاتی بسیار ساده ، چگالی کد و میزان کارآیی را بالا می‌برد؛
- مجموعه‌ی دستورهای ۱۶ بیتی (Thumb):
  شرکت ARM برای افزایش کارآیی و سازگاری پردازش‌هایش، مجموعه‌ای دیگر از دستورالعمل‌ها با نام Thumb را توسعه داد که ۱۶ بیتی بودند. ARM به کمک Thumb ، توانایی اجرای دستورهای ۱۶ و ۳۲ بیتی را پیدا کرده و همین امر باعث شد دستورهای ۱۶ بیتی نسبت به ۳۲ بیتی (با طول ثابت) به میزان 30% افزایش چگالی کد را به همراه داشته باشد؛
- اجرای شرطی دستورها[3]:
  دستورها وقتی اجرا می‌شوند که یک شرط کامل برقرار باشد. با شرطی شدن، تعداد دستورهای انشعاب[4] به شدت کاهش می‌یابد و بر افزایش چگالی کد، تأثیرگذار است؛
- دستورهای بهبود یافته:

---

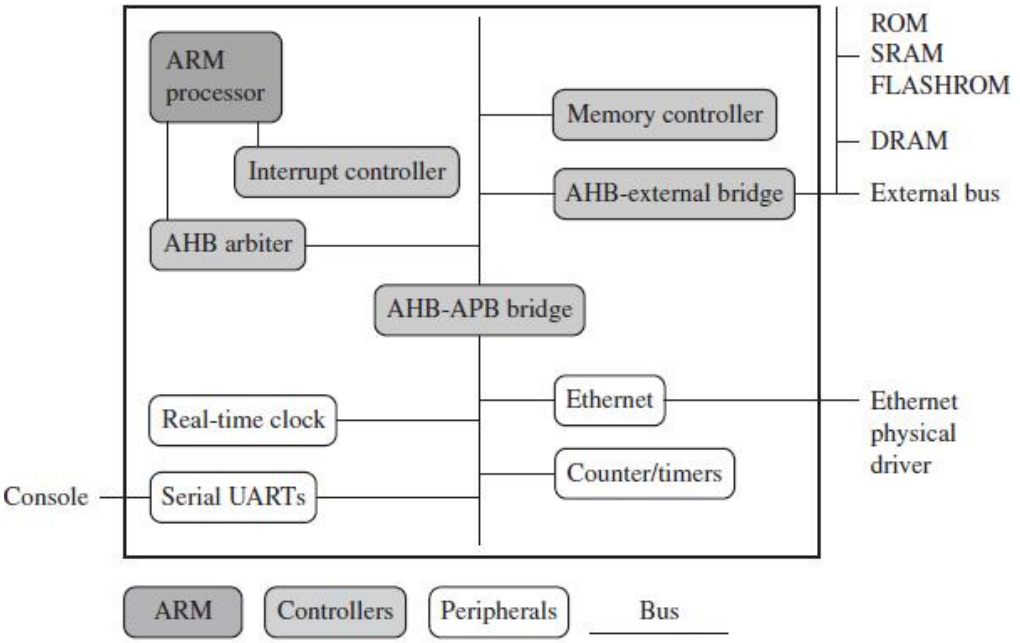
[1] Time to Market
[2] Embedded System
[3] Conditional Execution
[4] Branch

- یکی از دستورهای بهبود یافته که به هسته ARM افزوده شده است، دستورهای پردازش سیگنال دیجیتال (DSP) است که همانند ضرب 16×16 بیتی عمل می‌کند. DSP باعث افزایش کارآیی می‌شود. این خواص اضافی، پردازنده‌های ARM را به یکی از پرکاربردترین پردازنده‌های ۳۲ بیتی تبدیل کرده است و امروزه تعداد زیادی از کارخانه‌های ساخت قطعات و محصولات الکترونیکی براساس ARM ، تولیدات خود را به بازار عرضه می‌کنند.

۲.۳ سخت‌افزار یک سیستم تعبیه شده

سیستم‌های تعبیه شده، طیف گسترده‌ای از محصولات را دربرمی‌گیرند. این سیستم‌ها می‌توانند ساده‌ترین و پیچیده‌ترین اجزا را کنترل کنند؛ از کنترل یک سنسور در خط تولید کارخانه‌ای گرفته تا کنترل بی‌درنگ[۱] در پرنده‌های فضایی سازمان NASA . تمام این دستگاه‌ها، ترکیبی از نرم‌افزار و سخت‌افزارند. انتخاب هر یک ازاین اجزا، براساس کارآیی مورد نظرشان و همچنین قابلیت ارتقاء در آینده صورت می‌گیرد.

شکل ۲.۲، یک دستگاه نمونه با به کار گیری هسته‌ی ARM را نشان می دهد. هر یک از مستطیل ها در شکل بالا نشان دهنده‌ی یک کارکرد یا یک مشخصه‌ی دستگاه بوده و خطوط اتصال دهنده، گذرگاه‌های حامل داده هستند.



[۱] Real time

شکل ۲.۲ – دستگاهی با استفاده از پردازنده‌ی ARM

این دستگاه را می توانیم به چهار جزء سخت‌افزاری تقسیم کنیم:

- **پردازنده‌ی ARM** ، سیستم تعبیه شده را کنترل می کند. نسخه‌های مختلفی از هسته پردازنده ARM با مشخصه‌های گوناگون برای کنترل دستگاه مورد نظر وجود دارند. پردازنده ARM شامل یک هسته پردازشی (برای پردازش دستورها و دستکاری داده‌ها) به همراه اجزای جانبی برای ارتباط با گذرگاه‌های خارجی می‌باشد.
- **کنترل کننده‌ها**، وظیفه‌ی هماهنگی بین بلوک‌های کاری مهم سیستم را بر عهده دارند. دو کنترل کننده رایج، کنترل کننده‌ی حافظه و کنترل کننده وقفه هستند؛
- **وسایل جانبی**، تمامی ارتباطات ورودی و خروجی که بیرون از تراشه قرار دارند را مدیریت می‌کنند؛
- **گذرگاه**، برای برقراری ارتباط بین اجزای مختلف دستگاه مورد استفاده قرار می گیرد.

۲.۳.۱ فناوری گذرگاه ARM

سیستم‌های تعبیه شده، از فناوری متفاوتی نسبت به آن‌چه در مورد رایانه‌های x86 صدق می‌کند، استفاده می‌کنند. سیستم گذرگاه پر کاربرد رایانه‌های شخصی، با نام PCI دستگاه‌های متفاوتی از جمله کارت‌های تصویر، کنترل کننده‌های دیسک سخت[۱] و... را به گذرگاه پردازنده‌ی x86 متصل می‌کند. این گذرگاه از لحاظ فناوری، بیرون_ تراشه[۲] و بر روی برد مادر یک رایانه شخصی قرار دارد.

در مقایسه، یک سیستم تعبیه شده از گذرگاهی بر روی تراشه، جهت اتصال وسایل گوناگون به پردازنده، بهره می‌برد. دو کلاس متفاوت از دستگاه‌هایی که به گذرگاه وصل می‌شوند، وجود دارند. پردازنده‌ی ARM فرمانده گذرگاه[۳] بوده، در حالی‌که وسایل جانبی فرمان‌بر گذرگاه[۴] هستند. فرمانده، آغاز کننده یک انتقال داده بر روی گذرگاه است. در حالی‌که فرمان‌برها هنگام درخواست داده، اطلاعاتی را بر روی گذرگاه قرار می‌دهند.

گذرگاه دو سطح ساختاری دارد. اولین آن مربوط به مشخصات الکتریکی آن، یعنی سطوح ولتاژ و عرض داده (۱۶، ۳۲ یا ۶۴ بیت) و دومی مربوط به پروتکل برقراری ارتباط است. (مجموعه‌ای از قوانین حاکم بر نحوه‌ی ارتباط بین پردازنده و وسایل جانبی را پروتکل می‌نامیم.) ARM به ندرت قوانین الکتریکی گذرگاه را وضع می‌کند. در عوض، به طور منظم بر پروتکل گذرگاه نظارت دارد.

---

[۱] HDD

[۲] Off-Chip

[۳] Bus Master

[۴] Bus Slave

۲.۳.۲ پروتکل گذرگاه AMBA

ساختار گذرگاه میکروکنترولری پیشرفته یا AMBA ، در سال 1996 معرفی و در بین پردازنده‌های ARM مقبولیت گسترده‌ای پیدا کرد. اولین گذرگاه‌های معرفی شده‌ی AMBA، گذرگاه سیستمی ARM (ASB) و گذرگاه وسایل جانبی ARM (APB) بودند. بعدها ARM طرحی دیگر برای گذرگاه به نام گذرگاه با کارآیی بالای ARM (AHB) را معرفی کرد.

AHB ، توانایی انتقال داده با سرعت بالاتر نسبت به ASB را دارد. طراحی AHB براساس گذرگاه با مرکزیت مالتی پلکس شونده است. در حالی‌که ASB ، یک گذرگاه دوطرفه‌ی ساده است. این تغییر، گذرگاه AHB را بسیار سریع‌تر ساخته و آن را به عنوان اولین گذرگاهی که از پهنای گذرگاه ۶۴ و ۱۲۸ بیت پشتیبانی می‌کند، شناسانده است.

ARM دو گونه‌ی مختلف از AHB را عرضه داشت: AHB چند لایه[۱] و AHB-Lite. برخلاف طرح اولیه-ی AHB که در آن گذرگاه همواره یک فرمانده داشت، در ساختار AHB چند لایه، چندین فرمانده فعال در یک زمان می‌توانند بر روی گذرگاه به فعالیت بپردازند. AHB-Lite نسخه‌ای ساده و خلاصه شده از AHB بوده که اجازه حضور یک فرمانده‌ی فعال بر روی گذرگاه را می‌دهد.

اتصال چندگانه‌ی موجود بر روی AHB چند لایه توانایی اجرای داده را افزایش داده است. با این‌حال AHB وAHB چند لایه از یک پروتکل استفاده می‌کنند.

قطعه‌ی نمونه‌ی نشان داده شده در شکل ۲.۲، سه گذرگاه دارد: گذرگاه AHB برای وسایل جانبی با کارآیی بالا، گذرگاه APB برای وسایل جانبی کندتر و یک گذرگاه سوم برای ارتباط با وسایل خارجی که مخصوص این قطعه می‌باشد.

۲.۳.۳ حافظه

هر سیستم تعبیه شده، باید نوعی حافظه برای ذخیره‌ی کد و اجرای آن داشته باشد. هنگام تصمیم گیری در مورد سلسله مراتب حافظه، پهنای داده و نوع حافظه، آن‌ها را با یکدیگر از لحاظ قیمت، کارآیی و توان مصرفی مقایسه می‌کنیم. مثلاً، اگر حافظه‌ای، دو برابر سریع‌تر از آن‌چه هست باشد، توان مصرفی آن نیز به همان اندازه افزایش می‌یابد.

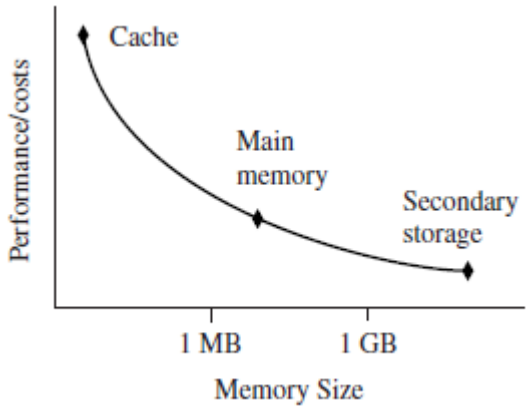
---

[۱] Multi Layer

۱.۳.۳.۱ سلسله مراتب[۱]

همه‌ی سیستم‌های رایانه‌ای به نوعی از حافظه‌ها استفاده می‌کنند. هر کدام از این حافظه‌ها در مکانی از سلسله مراتب قرار دارند. در شکل ۲.۲ قطعه‌ای را می‌بینیم که از حافظه‌های خارجی پشتیبانی می‌کند. در درون پردازنده امکان استفاده از "حافظه‌ی نهان" برای افزایش کارآیی حافظه وجو دارد. (هرچند در این شکل نشان داده نشده است.)

شکل ۲.۳ مصالحه‌ای بین کارآیی و حجم حافظه‌ی در دسترس را نشان می‌دهد. بدین صورت که حافظه-های نهان دارای بیش‌ترین سرعت دسترسی هستند و لی حجم زیادی ندارند، در عوض حافظه‌های خارجی دارای حجم زیاد و البته با سرعت دسترسی پایین‌تر هستند.



شکل ۲.۳ – مصالحه‌ای بین کارآیی و حجم حافظه‌ی در دسترس

توجه

به عنوان یک قانون کلی هرچه حافظه به پردازنده نزدیک‌تر باشد، گران‌تر بوده و ظرفیت کم‌تری نیز دارد.

حافظه‌ی نهان بین هسته‌ی پردازنده و حافظه‌ی اصلی قرار می‌گیرد. هدف اصلی آن افزایش سرعت انتقال داده بین پردازنده و حافظه است. حافظه‌ی نهان موجب افزایش کارآیی می‌شود و در عوض، این عیب را دارد که زمان اجرای دستورها در آن غیر قابل پیش‌بینی است.

---

[۱] Hierarchy

نکته

حافظه نهان اگرچه کارآیی کل سیستم را افزایش می‌دهد ولی، به پاسخ‌دهی بی‌درنگ سیستم کمکی نمی‌کند.

دقت کنید که در بسیاری از کاربردهای عادی سیستم‌های تعبیه شده، نیازی به حافظه‌ی نهان و قابلیت آن نیست. استفاده از حداکثر فواید آن، بستگی به سیستم‌عامل مورد استفاده دارد و نیز خود وسیله‌ای است که از این پردازنده استفاده می‌کند. در کل، استفاده از این حافظه محدود به کاربردهای خاص می‌باشد.

حافظه‌ی اصلی، معمولاً اندازه بزرگی بین 256KB تا 256MB (یا حتی بیشتر) دارد. این اندازه بستگی به دستگاه و کاربرد مورد استفاده دارد. این حافظه معمولاً به صورت تراشه‌ای جداگانه در کنار پردازنده‌ی اصلی قرار می‌گیرد. دستورهای بارگیری[1] و ذخیره سازی[2] به غیر از حالاتی که با حافظه‌ی نهان سروکار دارند، دسترسی مستقیم به حافظه‌ی اصلی دارند.

ذخیره کننده‌های ثانویه، بزرگ‌ترین و کندترین انواع حافظه هستند. درایوهای CD-ROM و دیسک‌های سخت، از این دست ذخیره کننده‌های داده هستند. امروزه این نوع حافظه‌ها و یا ذخیره کننده‌های داده، تا اندازه‌های بیش از 500GB نیز در دسترس هستند.

۲.۳.۳.۲ پهنای حافظه[3]

پهنای حافظه، تعداد بیتی است که حافظه در هر بار دسترسی به آن، باز می‌گرداند. (اعداد نمونه‌ی پهنای حافظه عبارت‌اند از: ۸، ۱۶، ۳۲ یا ۶۴ بیت) پهنای حافظه تأثیر مستقیمی بر روی کارآیی کلی و قیمت‌های نسبی سیستم دارد.

اگر از یک سیستم بر مبنای پردازنده‌ی ARM ۳۲ بیتی و بدون حافظه‌ی نهان که به یک حافظه‌ی با پهنای ۱۶ بیت متصل است، استفاده کنید؛ آن‌گاه پردازنده در هر دستورالعمل، دوبار واکشی اطلاعات خواهد داشت. هر واکشی داده، نیاز به دو دستور بارگیری ۱۶ بیتی دارد. کاهش کارآیی سیستم در این‌جا کاملاً مشهود است. اما در عوض حافظه‌ی ۱۶ بیتی ارزان‌تر از انواع ۳۲ بیتی است.

---

[1] Load

[2] Store

[3] Memory Width

در این‌جا، اگر پردازنده در حالت Thumb کار کند، در حین اتصال بـه یـک حافظـه‌ی ۱۶ بیتـی، کـارآیی بیشتری از خود نشان خواهد داد. این افزایش کارآیی، مرهون اتصال مناسب با پهنای حافظه‌ی هماهنـگ است. در این حالت، تنها به یک دستور بارگیری نیازمندیم.

نکته

استفاده از دستورالعمل‌های Thumb با حافظه‌های ۱۶ بیتی، افزایش کارآیی و کاهش هزینه‌ها را به همراه خواهد داشت.

در جدول ۱.۱، تعداد سیکل‌های پردازنده‌ی ARM برای حافظه‌های مختلف آورده شده‌اند.

۲.۳.۳.۳ انواع حافظه

امروزه انواع مختلفی از حافظـه‌هـا وجـود دارنـد. در ایـن بخـش بـه تعـدادی از انـواع حافظـه‌ای کـه در سیستم‌های ARM رایج هستند، اشاره می‌کنیم.

حافظه‌ی فقط خواندنی[۱] (ROM) کم انعطاف‌ترین نوع حافظه در بین تمامی آن‌هاست. محتوای این حافظه فقط یک بار برنامه‌ریزی می‌شود و قابلیت تغییـر مجـدد را نـدارد. ROM هـا در دسـتگاه‌هـای بـا تعـداد تولیدی بالا که در آن‌ها نیاز به هیچ‌گونه اصلاح و یا بـه روز رسـانی نـداریم، بسـیار کـاربرد دارنـد. در بسیاری از دستگاه‌ها نیز از ROM برای نگه‌داری کدهای اولیه (همانند کدهای Boot) استفاده می‌شود.

Instruction size	8-bit memory	16-bit memory	32-bit memory
ARM 32-bit	4 cycles	2 cycles	1 cycle
Thumb 16-bit	2 cycles	1 cycle	1 cycle

جدول ۱.۱

ROM های فلش، قابلیت نوشتن و خواندن دایمی را دارند. اما سرعت نوشـتن در آن‌هـا بـالا نیسـت. بـه همین علت، نمی‌توان از آن‌ها در جایی که نیاز به تغییر مداوم اطلاعات هسـت اسـتفاده کـرد. اسـتفاده از آن‌ها بیشتر در نگه‌داری کدهای پردازنده (که حجم زیادی ندارند) و یا نگه‌داری اطلاعات بلند مدت هنگام قطع تغذیه است. نوتن به درون این حافظه‌ها و یا پاک کردنشـان، کـاملاً نـرم‌افـزاری بـوده و نیـازی بـه تغییرات سخت‌افزاری ندارند، این به معنای هزینه‌های ساخت پایین می‌باشد.

[۱] Read Only Memory

حافظه‌های RAM دینامیک[1] از رایج‌ترین انواع RAM مورد استفاده در سیستم‌هاست و قیمت این حافظه-ها نیز به ازای هر مگابایت داده، نسبت به انواع دیگر RAM پایین‌تر است.
DRAM، یک حافظه‌ی دینامیک یا پویا است. بدین معنا که بار الکترونیکی ذخیره شده در این حافظه، در هر چند میلی ثانیه یک‌بار باید به‌روز رسانی شوند[2]. پس، مشخصاً در کنار این نوع حافظه به یک کنترل کننده‌ی مخصوص آن نیز نیاز خواهیم داشت.
SRAM ها از DRAM های متداول سریع‌تر هستند. اما این سرعت در ازای اندازه‌ی بزرگ تر SRAM ها حاصل می‌شود. (رابطه‌ی مستقیمی بین سطح تراشه و سرعت وجود ندارد. در حقیقت عیب آن‌ها بزرگ بودن اندازه‌شان است.) SRAM ها استاتیک یا ایستا هستند. بنابراین نیازی به مدار به‌روز رسانی ندارند. سرعت دسترسی به SRAM بیشتر از انواع DRAM است. به علت قیمت بالای SRAM ها، استفاده‌شان محدود به نگه‌داری اطلاعات کم است که نیاز به سرعت دسترسی بالایی دارند.
DRAM های سنکرون که آن‌ها را با نام SDRAM می‌شناسیم، یکی از زیر شاخه‌های DRAM ها هستند که در سرعت کلاک بیشتری نسبت به آن‌ها کار می‌کنند. در رابطه با حافظه‌های پر کاربرد SDRAM در بخش‌های بعدی، صحبت‌های بیشتری شده است.

۲.۳.۴ وسایل جانبی[3]

سیستم‌های تعبیه شده، همگی به نوعی با دنیای خارج ارتباط دارند. این وظیفه بر عهده‌ی وسایل جانبی است که در پردازنده‌ی اصلی قرار دارد. با استفاده از این وسایل جانبی، اطلاعات سنسورها و ورودی‌های کاربر خوانده شده و خروجی‌های مورد نظر به سمت نمایشگرها، رله‌ها و... هدایت می-شوند.
هر یک از این وسایل جانبی، مخصوص انجام یک وظیفه خاص طراحی می‌شوند. بعضی از آن‌ها بر روی تراشه‌ی اصلی و برخی در بیرون قرار دارند. این وسایل، کارکردهای گوناگونی از یک ارتباط سریال داده گرفته تا ارتباط با شبکه بی‌سیم WLAN را بر عهده دارند.

نکته	تمامی وسایل جانبی در ARM، بر روی حافظه نگاشته شده‌اند. (Memory Mapped یا

[1] Dynamic Random Access Memory - DRAM
[2] Refresh
[3] Peripherals

دارای نقشه‌ی حافظه) یعنی کار کردن با وسایل جانبی همانند نوشتن روی و یا خواندن از حافظه است. برای دستکاری ثبات‌های درون وسایل جانبی از یک آدرس پایه‌ی[1] مربوط به وسیله، به همراه یک آفست[2] که اشاره به یک ثبات خاص دارد، استفاده می‌کنیم. روش کار در شکل ۲.۳.۱ آمده است.

کنترل کننده‌ها، انواع خاصی از وسایل جانبی هستند که سطوح بالاتری از کارکردها را بـر عهـده دارنـد. دو نوع اصلی کنترل کننده‌ها، کنترل کننده‌های حافظه و کنترل کننده‌های وقفه هستند.

۲.۳.۴.۱ کنترل کننده‌های حافظه[3]

این کنترل کننده‌ها، انواع مختلفی از حافظه‌را به گذرگاه پردازنده متصل می‌کنند. در هنگام شروع بـه کـار سیستم، کنترل کننده‌ی حافظه برای حافظه‌ی متصل به سیستم پیکره‌بندی می‌شود. تنظـیم‌هـای DRAM باید در نرم‌افزار صورت بگیرد. به عنوان مثال، زمـان‌بنـدی‌هـای حافظـه و نـرخ بـه‌روز رسـانی قبـل از استفاده‌ی DRAM ، باید تنظیم شوند.

۲.۳.۴.۲ کنترل کننده‌های وقفه

وقتی یک وسیله‌ی جانبی و یا یک تراشه نیاز به توجه پردازنده دارد، برای پردازنده یک وقفه ایجاد مـی-کند. کنترل کننده‌ی وقفه وظیفه مدیریت وقفه‌ها با رعایت تقدم آن‌ها توسط تنظیم بیت‌های مربوطه را بـر عهده دارد.

دو نوع کنترل کننده‌ی وقفه برای پردازنده‌ی ARM موجود هستند: کنتـرل کننـده‌ی وقفـه‌ی اسـتاندارد و کنترل کننده‌ی وقفه‌ی برداری (VIC).

کنترل کننده‌ی وقفه‌ی استاندارد، هنگام درخواست وقفه توسط وسیله‌ی خارجی، وقفه‌ی مورد نظر را بـه پردازنده می‌دهد. این کنترل کننده را می‌توان برای ماسک کـردن (یـا نادیـده گـرفتن) عـده‌ای از وقفـه‌هـا برنامه‌ریزی کرد.

VIC از چند جهت از یک کنترل کننده‌ی استاندارد وقفه، حرفه‌ای‌تر است. VIC وقفه‌هـا را نسـبت بـه هـم تقدم می‌دهد و هنگام رخداد وقفه، توانایی تشخیص منبع تولید کننده‌ی وقفه را دارد.

---

[1] Base Address

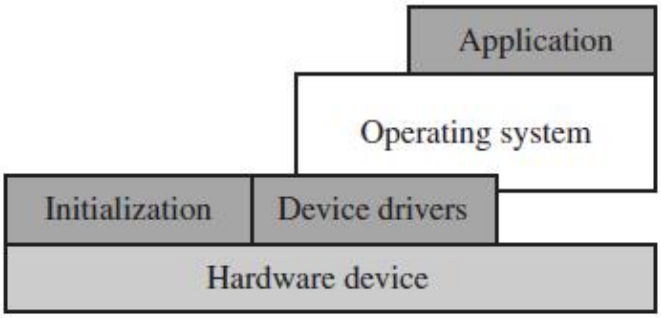
[2] Offset

[3] Memory Controllers

۲.۴ نرم‌افزار سیستم تعبیه شده

یک سیستم تعبیه شده به نرم‌افزاری برای راه‌اندازی نیاز دارد. شکل ۲.۴، چهار جزء اصلی نرم‌افزار مربوطه برای کنترل یک سیستم تعبیه شده را نشان می‌دهد. هر جزء نرم‌افزاری از لحاظ پیچیدگی یک "سطح" از لایه‌های دیگر بالاتر است. این لایه‌ها به صورت پشته بر روی یکدیگر قرار دارند.

کد مقداردهی اولیه[۱]، نخستین کد اجرا شونده بر روی هر سیستم است و نوع آن بستگی به سخت‌افزار هدف دارد. وظیفه‌ی این قسمت تنظیم مشخصات مهم اجزای اصلی سیستم قبل از سپردن کنترل به سیستم‌عامل است.



شکل ۲.۴ – چهار جزء اصلی نرم‌افزار کنترلی یک سیستم تعبیه شده

سیستم‌عامل، یک ابر ساختار برای کنترل سیستم‌های کنترلی فراهم می‌آورد. در بسیاری از سیستم‌ها نیازی به وجود سیستم‌عامل نبوده و صرفاً از زمان‌بندی وظایف کاری[۲] استفاده می‌کنند.

راه‌اندازها[۳] یک واسط نرم‌افزاری برای راه‌اندازی وسایل جانبی فراهم می‌آورند. در نهایت خود برنامه‌ی کاربردی[۴] وظیفه‌ی اصلی را انجام می‌دهد.

توجه

---

۱ Initialization Code

۲ Task Scheduling

۳ Drivers

۴ Application

اجزای نرم‌افزاری هم می‌توانند از روی ROM و هم از روی RAM اجرا شوند. قسمت‌هایی از کد که بر روی ROM قرار می‌گیرند، ثابت بوده و آن را "میان‌افزار[۱] " می‌نامند.

۲.۴.۱ کد مقداردهی اولیه[۲] یا کد Boot

این قسمت از کد، پردازنده را از حالت ریست به آمادگی لازم برای اجرای سیستم‌عامل می‌رساند. وظیفه‌ی این بخش، تنظیم مشخصات ابتدایی کنترل کننده‌های حافظه، حافظه‌ی نهان و دیگر وسایل جانبی ضروری است.

در یک سیستم ساده‌تر، همین وظیفه به تنظیم یک زمان‌بند ساده و یا یک برنامه‌ی نگهبان عیب‌یابی[۳] تبدیل می‌شود. در مورد سیستم‌عامل، فرآیند بوت به سه وظیفه ساده تقسیم می‌شود: پیکره‌بندی اولیه‌ی سخت‌افزار، تشخیص عیب‌ها و بوت کردن.

پیکره‌بندی سخت‌افزاری، سخت‌افزار را برای بارگیری image سیستم‌عامل آماده می‌سازد. اگرچه بعضی مشخصات، استاندارد هستند ولی برای انطباق با هر سیستم‌عامل جدید، باید تنظیمات جداگانه‌ای داشته باشند. مثال آن نقشه‌ی حافظه در مثال ۱.۱ است که مجدداً سازماندهی شده است.

بخش تشخیص عیب نیز یک بخش ضروری برای تشخیص کارکرد صحیح اجزای مختلف سخت‌افزاری و نشان دادن پیغام‌هایی در ارتباط با عملکرد و مشخصات آن‌ها می‌باشد.

بوت کردن، شامل بارگیری یک فایل image و سپردن کنترل به دست سیستم‌عامل است. فرآیند بارگیری image ، شامل کپی کردن اطلاعات فایل مذکور به درون RAM است. سپس پردازنده، فایل موردنظر را اجرا می‌کند. بعضی از فایل‌های image فشرده شده‌اند تا حجم کمتری را اشغال کنند. در این صورت، فرآیندهای فوق شامل از فشردگی درآوردن[۴] فایل مزبور نیز می‌شود.

مثال ۱.۱

مقداردهی اولیه‌ی حافظه و سازماندهی آن، از وظایف مهم ابتدایی کدهای بوت است. دلیل آن این است که هر سیستم‌عامل خاص، نقشه‌ی حافظه‌ی خود را طلب می‌کند.

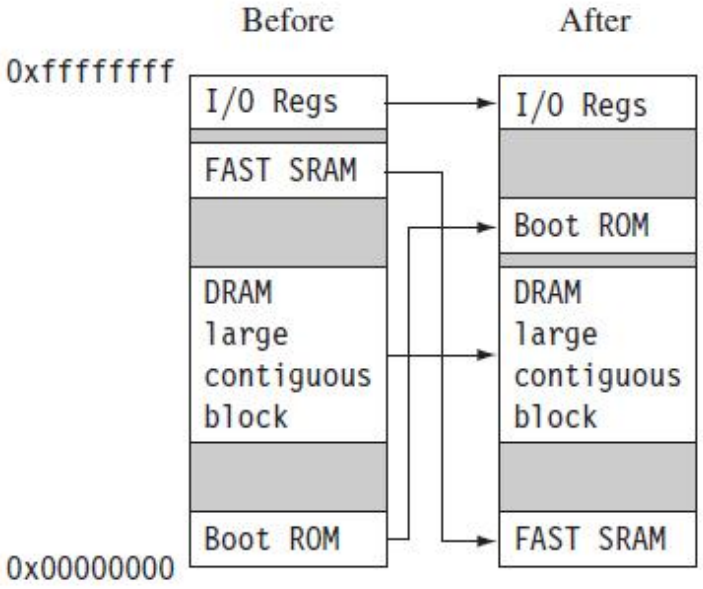
---

۱ Firmware

۲ Initialization

۳ Debug Monitor

۴ Decompress



شکل ۲.۵ – نقشه‌ی حافظه، قبل و بعد از سازماندهی

شکل ۲.۵ نقشه‌ی حافظه، قبل و بعد از سازماندهی را نشان می‌دهد. فرآیند "نقشه نگاری مجدد حافظه[۱]" از مسایل رایج در سیستم‌های ARM است. هنگام آغاز به کار سیستم، کدهای اولیه در ROM قرار دارند. آدرس شروع ROM نیز 0x0000 0000 است. بعد از فرآیند remap، حافظه‌ی RAM در آدرس 0x0000 0000 قرار گرفته که امکانات زیادی را در اختیار قرار می‌دهد. به عنوان مثال، جدول بردار[۲] (که بعداً در مورد آن بحث خواهیم کرد) را می‌تواند مجدداً مقداردهی کرده و در RAM جای داد.

۲.۴.۲ سیستم‌عامل

فرآیند مقداردهی اولیه، سخت‌افزار را برای شروع به کار سیستم‌عامل، مهیا می‌کند. سیستم‌عامل سازماندهی منابع سیستمی را بر عهده دارد. این منابع عبارت‌اند از:وسایل جانبی، حافظه و زمان پردازش. بعد از سازماندهی ذکر شده، برنامه‌های اجرا شده تحت سیستم‌عامل، به راحتی در این محیط اجرا گردیده و از لوازم مورد نیازشان بهره می‌برند.

---

[۱] Memory Remapping

[۲] Vector Table

بر روی پردازنده‌های ARM ، بیش از ۵۰ سیستم‌عامل، پشتیبانی می‌شوند. سیستم‌عامل‌ها را می‌توان به دو دسته اصلی تقسیم کرد: سیستم‌عامل‌های بی‌درنگ[۱] و سیستم‌عامل‌های پلتفرم[۲].
RTOS ها، زمان پاسخ‌دهی دقیقی را ضمانت می‌کنند. دستگاهی که از سیستم بی‌درنگ سخت[۳] پشتیبانی می‌کند، پاسخ‌دهی دقیق و ضمانت شده‌ای را دارد. در مقابل سیستم بی‌درنگ نرم[۴] نیاز به زمان پاسخ خوبی دارد (نه لزوماً دقیق و ضمانت شده).

نکته

سیستم‌هایی که از RTOS استفاده می‌کنند، عموماً از وجود عنصر ذخیره کننده‌ی ثانویه[۵] بی‌بهره‌اند. در حالی‌که سیستم‌عامل‌های پلتفرم، شامل این عناصر است.

سیستم‌عامل پلتفرم به یک واحد مدیریت حافظه برای مدیریت برنامه‌های بزرگی که بی‌درنگ نیستند، نیاز دارند. سیستم‌عامل Linux نمونه‌ای از این سیستم‌هاست.
البته، سیستم‌های عاملی نیز موجودند که از هر دو عامل بهره می‌برند. بدین معنا که هم واحد مدیریت حافظه[۶] داشته و هم پاسخ‌دهی بی‌درنگی دارند.

۲.۵ خلاصه و نتیجه گیری

طراحی پردازنده‌ای که فقط براساس مبانی RISC باشد، تنها دارای کارآیی بالایی است. ولی ARM از معماری RISC تغییر یافته‌ای بهره می‌برد که علاوه بر کارآیی بالا، چگالی کد بالا و مصرف توان کم را نیز ارایه می‌کند.
یک سیستم تعبیه شده شامل هسته‌ی پردازنده‌ای است که با حافظه‌ی نهان، حافظه‌های گوناگون و وسایل جانبی احاطه شده است. کنترل سیستم به دست نرم‌افزار سیستم‌عامل بوده که وظایف کاربردی و اجرایی را مدیریت می‌کند.

---

۱ RTOS
۲ Platform
۳ Hard Real-time
۴ Soft Real-time
۵ Secondary Storage
۶ MMU

نکته‌ی کلیدی معماری RISC، افزایش کارآیی با استفاده از کاهش پیچیدگی دستورها است. از دیگر مشخصات این معماری افزایش سرعت اجرای دستورها با بهره‌گیری از خط لوله، فراهم آوردن تعداد زیادی از ثبات‌ها برای ذخیره سازی اطلاعات در نزدیکی پردازنده و استفاده از ساختار بارگیری- ذخیره سازی است.

فلسفه طراحی پردازنده‌ی ARM ، دارای ایده‌هایی علاوه بر آن‌چه RISC ارایه کرده است نیز هستند:

- امکان اجرای چند سیکلی بعضی دستورهای خاص که مصرف توان، مساحت تراشه و اندازه‌ی کد را بهینه می‌کند؛
- استفاده از Barrel Shifter برای افزایش قابلیت بعضی دستورهای خاص؛
- از دستورهای Thumb و Thumb-2 برای بهبود کارآیی بهره می‌برد؛
- با امکان اجرای شرطی دستورها، چگالی کد و کارآیی، افزایش چشم گیری دارد؛
- از دستورهای بهبود یافته‌ای برای انجام مجموعه‌ای از عملیات پردازش سیگنال، بهره می‌برد.

فصل سوم

اصول مقدماتی پردازنده ARM

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ ثبات وضعیت فعلی برنامه و اجزای آن؛
- ✓ ثبات‌های عمومی چیست و کاربردشان در کجاست؟
- ✓ خط لوله چیست؟ انواع پردازنده‌های ARM از چه خط لوله‌ای استفاده می‌کنند؟
- ✓ افزونه‌های حافظه‌ی هسته؛
- ✓ خانواده‌های مختلف پردازنده‌ی ARM .

زیر گروه‌های گوناگونی از پردازنده ARM وجود دارند که در بعضی، سرعت حافظه اصلی افزایش داشته و در برخی دیگر، مجموعه‌ای از دستورهای جدید اضافه شده‌اند. در این قسمت به بررسی هسته-ی پردازنده‌ی ARM می‌پردازیم.

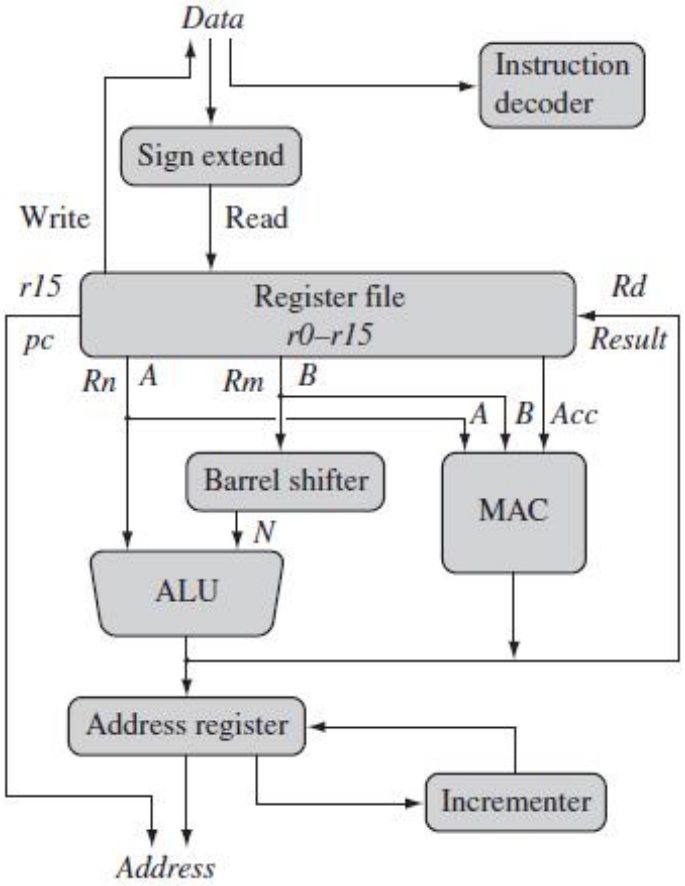
ابتدا، مروری بر هسته‌ی پردازنده خواهیم داشت. سپس، به نحوه انتقال داده بین قسمت‌های مختلف می‌پردازیم. مدل برنامه نویسی ARM را از دیدگاه یک برنامه نویس نرم‌افزاری پردازنده‌ی ARM بررسی می‌کنیم. این مدل کارکرد پردازنده و نحوه ارتباط بین اجزای مختلف را تشریح می‌کند. زیر گروه‌ها موضوع بحث، ادامه‌ی این بخش است. سپس، تاریخچه و نحوه نام‌گذاری نسخه‌های مختلف ARM و تغییرات ایجاد شده در مجموعه دستورهای پردازنده را به ترتیب زمانی بررسی می‌کنیم. در قسمت انتهایی چندین پردازنده‌ی معروف این دسته‌بندی‌ها را به تفصیل معرفی می‌کنیم.

پردازنده‌ی ARM از دیدی برنامه نویس، شامل بخش‌های گوناگون مداری و کاربردی است که توسط گذرگاه داده به یکدیگر متصل شده‌اند. (شکل ۳.۱) خطوط این شکل نمایانگر گذرگاه‌ها؛ جهت پیکان‌ها نشانگر سمت و سوی انتقال داده؛ اشکال مستطیلی محتوی یک بخش کاربردی و یا مکانی برای ذخیره داده می‌باشد. این شکل علاوه بر بیان گردش داده‌ها و جهت آن‌ها، شمایی کلی از یک پردازنده‌ی ARM و اجزای آن را نشان می‌دهد.

داده از طریق **گذرگاه داده** به هسته پردازنده وارد می‌شود. این داده می‌تواند یک دستورالعمل و یا یک داده واقعی باشد. در شکل ۳.۱، معماری ARM با ساختار Von-Neumann (که در آن داده‌ها و دستورالعمل‌ها از یک مسیر و گذرگاه مشترک استفاده می‌کنند) به تصویر کشیده شده است. در مقابل اگر از ساختار Harvard استفاده می‌کردیم، این دو گذرگاه، گذرگاه‌هایی جداگانه بودند.

رمزگشای دستورالعمل‌ها[1]، دستورها را قبل از اجرا، ترجمه می‌کند. هر دستور اجرا شده، به گروه خاصی از مجموعه‌ی دستورالعمل‌ها تعلق دارد.

پردازنده ARM همانند تمام پردازنده‌های RISC از ساختار ذخیره- بارگیری بهره می‌برد. این بدین معنا است که پردازنده دارای دو نوع دستورالعمل برای انتقال داده از یا به سمت پردازنده است. (یا به سمت داخل پردازنده و یا از سمت پردازنده به خارج) دستور Load داده را از حافظه بر روی ثبات کپی کرده و دستور Store بالعکس داده را از ثبات‌ها بر روی حافظه کپی می‌کند. هیچ دستوری برای پردازش مستقیم و کار بر روی حافظه‌ی سیستم وجود ندارد. پردازش داده‌ها، صرفاً بر روی ثبات‌ها انجام می‌شود.



شکل ۳.۱ – ساختار داخلی پردازنده‌ی ARM با ساختار Von-Neumann

[1] Instruction Decoder

داده‌ها بر روی مجموعه‌ای از ثبات‌ها به نام Register File قرار می‌گیرند (مجموعه‌ای از ثبات‌های ۳۲ بیتی). از آن جایی که ARM پردازنده‌ای ۳۲ بیت است، داده‌های درون ثبات‌های ۳۲ بیتی را به عنوان مقادیر علامت‌دار و بدون علامت ۳۲ بیتی، در نظر می‌گیرد.

دستورالعمل‌های ARM عموماً دارای دو ثبات منبع با نام‌های Rn و Rm و یک ثبات مقصد به نام Rd هستند. عملوند (Operand) های مبدأ از فایل ثبات‌ها و به ترتیب از گذرگاه‌های Aو B خوانده می‌شوند. ALU (واحد محاسبه و منطق) یا MAC (واحد ضرب و انباره) مقادیر Rn و Rm را از گذرگاه‌های A و B خوانده و نتیجه را محاسبه می‌کنند. دستورهای پردازش داده، مقدار Rd را مستقیماً بر روی فایل ثبات می‌نویسند. دستورهای بارگیری و ذخیره ازALU برای تولید آدرسی که باید در ثبات آدرس نگه-داری شده و سپس بر روی گذرگاه آدرس قرار داده شوند، استفاده می‌کنند.

از ویژگی‌های مهم ARM این‌است که بر روی ورودی Rm از یک Barrel Shifter استفاده می‌کند. داده Rm قبل از ورود به ALU پردازش می‌شود. ALU و Barrel Shifter به همراه یکدیگر برای محاسبه طیف وسیعی از عبارات و آدرس‌ها مناسب هستند.

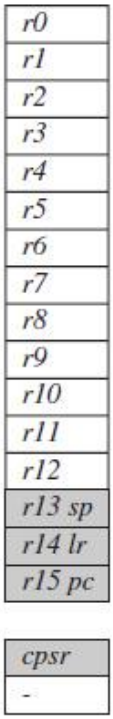
Rd در خروجی ALU از طریق **گذرگاه نتیجه**، بر روی فایل ثبات نوشته می‌شود. در مورد دستورهای Load و Store، قبل از خواندن و یا نوشتن مقدار ثبات بعدی، ثبات آدرس توسط افزایش دهنده، افزایش می‌یابد تا عملیات بعدی بر روی مکان بعدی حافظه صورت پذیرد. اجرای دستورها تا زمانی که یک وقفه یا یک استثنا (exception) رخ ندهد، ادامه می‌یابد.

اکنون با دید کلی که از پردازنده به دست آوردیم، کمی در احوال داخلی پردازنده‌ی عمیق شده و به بررسی ثبات‌ها، ثبات وضعیت فعلی پردازنده (CSPR) و خطوط لوله (Pipeline)، می‌پردازیم.

۳.۱ ثبات‌ها

ثبات‌های با کاربرد عمومی[۱]، نگه‌دارنده داده و یا آدرس هستند. مشخصه آن‌ها در نام‌گذاری، حرف r به همراه شماره ثبات مورد نظر می‌باشد. به عنوان مثال، نام ثبات شماره۴، *r4* است. شکل ۳.۲ ثبات‌هایی که در حالت کاربری *user* فعال هستند را نشان می‌دهد. (این حالت یک حالت محافظت شده است که هنگام اجرای برنامه‌های کاربردی، فعال است). پردازنده در هفت حالت کاربری می‌تواند فعالیت کند که مختصراً آن‌ها را توضیح خواهیم داد. تمام ثبات‌های نشان داده شده ۳۲ بیتی هستند.

[۱] General-Purpose



شکل ۳.۲ – ثبات‌های فعال در حالت *user*

حداکثر، ۱۸ ثبات فعال وجود دارند: ۱۶ ثبات داده و ۲ ثبات نشانگر وضعیت پردازنده. ثبات‌های داده برای برنامه نویس با نام‌های *r0* تا *r15* شناخته می‌شوند.

پردازنده‌ی ARM سه ثبات دارد که دارای کاربردهای خاص هستند. آن‌ها عبارت‌اند از: *r13* و *r14* و *r15*. عموماً نام‌های دیگری برای این سه ثبات در نظر گرفته می‌شود تا آن‌ها را از ثبات‌های دیگر تمایز دهد.

ثبات‌های سایه‌دار در شکل ۳.۲ نام‌های پر کاربرد و رایج را برای این سه ثبات نشان می‌دهد.

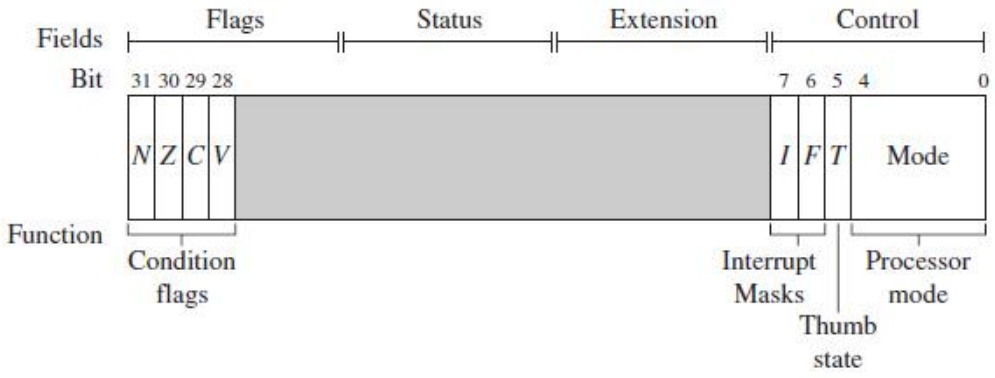
- ثبات *r13* از قدیم به عنوان اشاره‌گر پشته (Stack Pointer-SP) شناخته می‌شده و اشاره به بالای آن دارد؛
- ثبات *r14* با نام ثبات اتصال (Line Register-LR) شناخته شده و نگه‌دارنده‌ی آدرس بازگشت هنگام فراخوانی روال‌ها (Subroutine) است؛
- ثبات *r15* شمارنده‌ی برنامه (Program Counter-PC) بوده و حاوی آدرس دستور بعدی است که پردازنده می‌بایست اجرا کند.

ثبات‌های *r13* و *r14* بسته به محتوایشان می‌توانند به عنوان ثبات‌های با کاربرد عمومی نیز استفاده شوند. هنگام استفاده از یک **سیستم‌عامل**، به کاربران *r13* به عنوان یک ثبات با کاربرد عمومی، کار

خطرناکی است. زیرا، سیستم‌های عامل عموماً از آن به عنوان اشاره‌گر به محتوای پشته، استفاده می‌کنند. در ARM مجموعه ثبات‌های *r0* تا *r13*، نابسته و یکسان هستند (به این معنا که هر دستوری که با عملوند *r0* اجرا می‌شود، می‌تواند عیناً با ثبات‌های دیگری از این مجموعه نیز به کار گرفته شود) دستورهایی موجودند که با ثبات‌های *r14* و *r15* به طریقی مخصوص برخورد می‌کنند.
علاوه بر این ۱۶ ثبات داده، ۲ ثبات وضعیت برنامه، یعنی *cpsr* و *spsr*، نیز وجود دارند (ثبات وضعیت فعلی و وضعیت ذخیره شده برنامه). فایل ثبات تمامی این ثبات‌ها را دربر دارد. اما، ثبات‌های در دسترس با توجه به حالت کاربری انتخاب شده توسط کاربر، مشخص می‌شوند.

۳.۲ ثبات وضعیت فعلی برنامه *cpsr*

هسته ARM برای مشاهده‌ی وضعیت داخلی پردازنده و کنترل آن از *cpsr* استفاده می‌کند. *cpsr* یک ثبات ۳۲ بیتی مخصوص است که در درون فایل ثبات قرار دارد. شکل ۳.۳ یک نقشه کلی از ثبات وضعیت برنامه را نشان می‌دهد. نواحی سایه‌دار برای کاربرد احتمالی آینده در نظر گرفته شده است.
cpsr ، به چهار بخش ۸ بیتی پرچم‌ها[۱]، وضعیت[۲]، توسعه[۳] و کنترل تقسیم شده است. در طرح‌های فعلی دوبخش "توسعه" و "وضعیت" برای استفاده‌های بعدی درنظر گرفته شده‌اند. بخش "کنترل" شامل بیت‌های مربوط به وقفه و حالت کاری پردازنده بوده و بخش "پرچم" شامل بیت‌های مربوط به پرچم‌های شرط[۴] است.
تعدادی از هسته‌های پردازنده ARM ، بیت مخصوص اضافی‌تری نیز دارند. به عنوان مثال، بیت 'J' در پردازنده‌های با تکنولوژی Jazelle یافت می‌شود. توضیح کامل بیت‌های *cpsr* در ضمایم آمده است.



[۱] Flags
[۲] Status
[۳] extension
[۴] Condition Flags

شکل ۳.۳ – ثبات وضعیت برنامه

۳.۲.۱ حالت‌های کاری پردازنده[۱]

مشخص کننده‌ی ثبات‌های فعال و حقوق دسترسی به ثبات *cpsr* هستند. هر حالت کاری پردازنده یا دارای امتیاز است یا بدون امتیاز. حالت دارای امتیاز[۲]، دست‌یابی کامل برای نوشتن- خواندن بر روی *cpsr* را می‌دهد. حالت بدون امتیاز[۳] فقط اجازه‌ی خواندن از بخش کنترلی *cpsr* را داده ولی اجازه‌ی خواندن- نوشتن بر روی بیت‌های شرط را می‌دهد.

تعداد هفت حالت کاری برای پردازنده موجود است. شش حالت دارای امتیاز (*abort* و *FIQ* و *IRQ* و *Supervisor* و *System* و *Undefined*) و یک حالت بدون امتیاز (*User*) می‌باشد.

هنگامی که یک دسترسی ناموفق به حافظه از جانب پردازنده صورت می‌گیرد، پردازنده وارد حالت *abort* می‌شود. حالت‌های *Fast Interrupt Request* و *Interrupt Request* منطبق با دوسطح وقفه‌ی موجود در پردازنده‌های ARM هستند. حالت *Supervisor* حالت فعال پردازنده، بلافاصله بعد از آغاز به کار است. هسته‌ی سیستم‌های عامل نیز عموماً در این حالت کار می‌کنند. حالت *System* شاخه-ای از حالت *User* است که اجازه‌ی دسترسی کامل خواندن و نوشتن را بر روی *cpsr* می‌دهد. هنگامی که پردازنده با دستورالعملی برخورد می‌کند که تعریف نشده است، وارد حالت کاری *Undefined* می-شود. حالت *User* مورد استفاده برنامه‌های کاربردی است.

۳.۲.۲ ثبات‌های بانک شده[۴]

شکل ۳.۴ تمامی ۳۷ ثبات موجود در فایل ثبات را نشان می‌دهد. ۲۰ ثبات از ۳۷ ثبات یاد شده از دید برنامه در زمان‌های مختلف مخفی هستند. این ثبات‌ها را با نام **ثبات‌های بانک شده** می‌شناسیم و در شکل مذکور با سایه‌دار شدن، از دیگر ثبات‌ها متمایز شده‌اند.

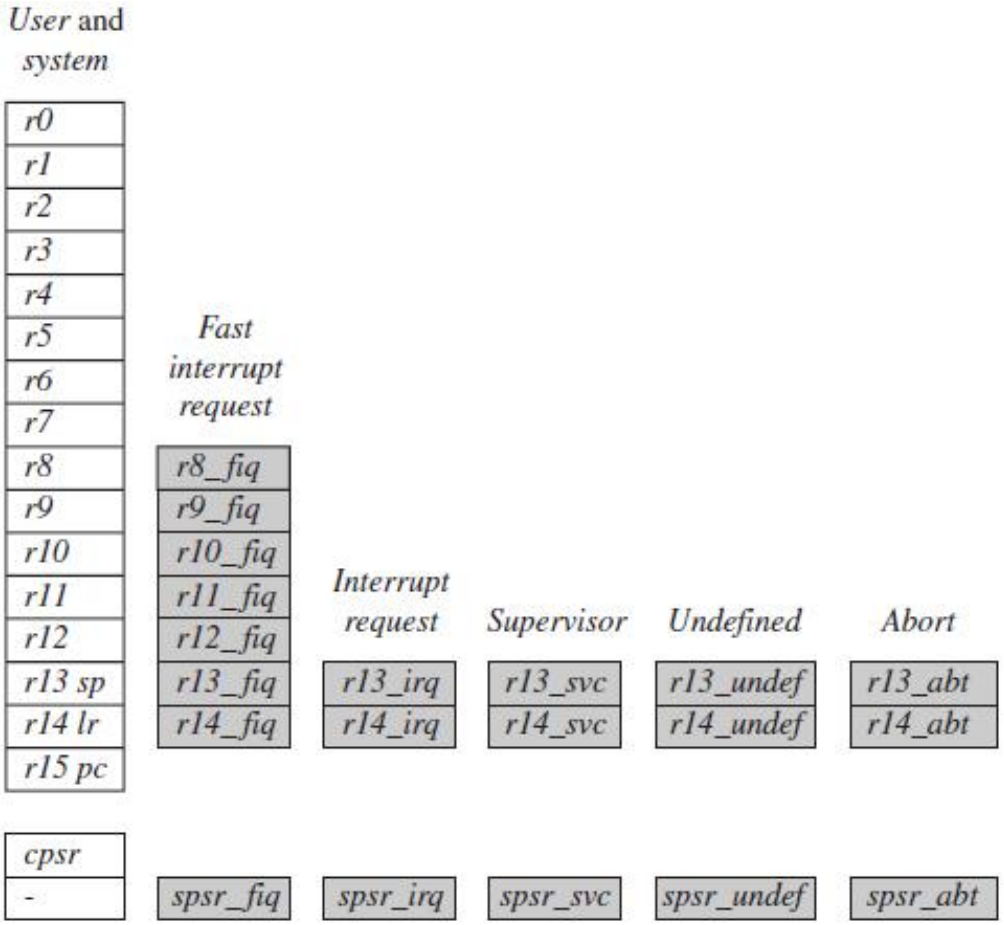
---

۱ Processor Modes

۲ Privileged

۳ non-Privileged

۴ Banked Registers



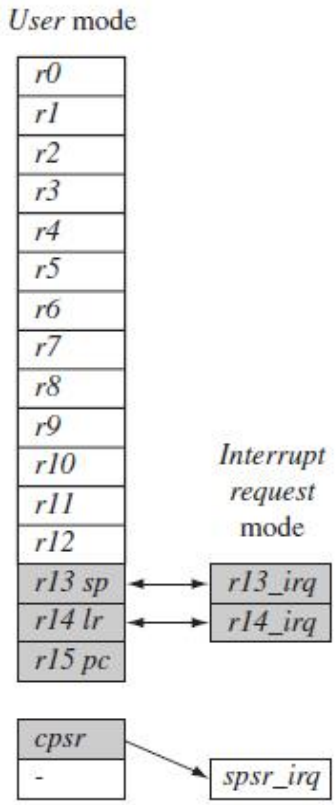
شکل ۳.۴ – تمامی ۳۷ ثبات موجود در فایل ثبات

هر یک از ثبات‌های بانک شده، هنگامی که پردازنده در حالت خاصی قرار دارد، در دسترس هستند. برای مثال، حالت کاری *Abort*، ثبات‌های بانک شده‌ی *r8_abt*، *r14_abt* و *spsr_abt* را دارد.

توجه

برای نام‌گذاری ثبات‌های بانک شده در حالات مختلف پردازنده از پسوند *mode_* استفاده می‌کنیم. mode در این‌جا مخفف شده‌ی حالت کاری پردازنده است.

حالت کاری پردازنده با نوشتن در بیت‌های مخصوص در درون *cpsr*، به راحتی قابل تغییر هستند. تنها حالت کاری که این مکان را ندارد، حالت *User* است. همگی حالات کاربری به جز حالت *System*، مجموعه‌ای از ثبات‌های بانک شده دارند که زیر مجموعه‌ای از ۱۶ ثبات اصلی هستند. ثبات‌های بانک شده نگاشتی یک‌به‌یک از ثبات‌های حالت *User* هستند. هنگامی که حالت کاری پردازنده تغییر می‌کند، ثبات بانک شده‌ی حالت جدید، جایگزین ثبات متناظر حالت قبل می‌شود.



شکل ۳.۵ – تغییر حالت پردازنده هنگام وقوع یک استثنا

برای مثال، وقتی پردازنده در حالت *IRQ* است، دستورهایی که اجرا می‌کنید همچنان از ثبات‌های *r13* و *r14* استفاده می‌کنند، اگرچه در حقیقت این ثبات‌ها، ثبات‌های بانک شده‌ی *r13_irq* و *r14_irq* هستند. ثبات‌های حالت *User* (یعنی *r13_user* و *r14_user*) با دستکاری این ثبات‌ها تغییر نمی‌کنند. برنامه‌ها، به ثبات‌های *r0* تا *r12* همانند قبل دسترسی عادی دارند.

حالت کاری پردازنده با نوشتن در بیت‌های مربوط، واقع در ثبات *cpsr* قابل تغییر هستند. (منوط به این-که پردازنده در حالت ممتاز باشد). راه دیگر استفاده از وقفه‌ها برای تغییر حالت پردازنده است.

استثناها و وقفه‌های زیر، باعث تغییر حالت کاری پردازنده می‌شدند: *reset*، *IRQ*، *FIQ*، *Software Int*، *Data Abort*، *Prefects Abort* و *Undefined Instruction*.

استثناها و وقفه‌ها، روند اجرای ترتیبی برنامه را متوقف ساخته و به مکانی خاص پرش می‌کنند. در مورد وقفه‌ها و استثناها به صورت مفصل بحث خواهیم کرد.

شکل ۳.۵ هنگامی که یک وقفه، اجبار به تغییر حالت می‌کند را نشان می‌دهد. شکل مذکور، موقعیتی را نشان می‌دهد که یک سیگنال وقفه‌ی خارجی، پردازنده را از حالت *User* به حالت *IRQ* می‌برد. این تغییر باعث بانک شدن ثبات‌های *r13* و *r14* می‌شود. ثبات‌های حالت *User*، در این‌جا با ثبات‌های *r13_irq* و *r14_irq* جایگزین می‌شوند. توجه کنید که *r14_irq* شامل آدرس بازگشت و *r13_irq* حاوی اشاره‌گر پشته[۱] حالت کاری *IRQ* هستند.

در شکل ۳.۵ حضور یک ثبات جدید به نام "ثبات حالت ذخیره شده برنامه" (*spsr*) را می‌بینیم که مقدار *cpsr* حالت قبلی را در خود ذخیره می‌کند. هنگام بازگشت به حالت *User*، دستوری خاص وجود دارد که محتویات ذخیره شده در *spsr_irq* را به درون *cpsr* بازمی‌گرداند.

نکته

spsr فقط در حالات ممتاز در دسترس هستند و می‌توانند دستکاری شوند. درحالت کاری *User* هیچ اثری از *spsr* نیست.

Table 2.1 Processor mode.

Mode	Abboreviation	Privileged	Mode [4:0]
Abort	abt	yes	10111
Fast interrupt request	fiq	yes	10001
Interrupt request	irq	yes	10010
Supervisor	svc	yes	10011
System	sys	yes	11111
Undefind	und	yes	11011
User	usr	no	10000

جدول ۲.۱

[۱] Stack Pointer

نکته‌ی دیگری که باید به آن توجه کنید، این است هنگامی که یک برنامه به طور دستی اقدام به تغییر حالت کاری می‌کند (با نوشتن در درون *cpsr*)، محتویات *cpsr* به درون *spsr* کپی نمی‌شود. عملیات ذخیره سازی مزبور تنها هنگامی رخ می‌دهد که استثنا یا وقفه رخ داده باشد.

پنج حالت کاری فعال پردازنده که پنج بیت متناظر پایین *cpsr* را اشغال نموده‌اند، در شکل ۳.۳ آورده شده‌اند. هنگام اتصال تغذیه به پردازنده، حالت کاری پیش‌فرض *Supervisor* است که یک حالت ممتاز است. شروع به کار در حالت ممتاز یک مزیت به حساب می‌آید. زیرا، مقداردهی اولیه دسترسی کامل به *cpsr* داشته و می‌تواند پشته‌ی مربوط به هر حالت را تنظیم کند.

جدول ۲.۱ حالات مختلف و الگوی باینری متناظرشان را نشان می‌دهد. ستون آخر الگویی از بیت‌ها که در *cpsr* قرار می‌گیرند را نشان می‌دهد.

۳.۲.۳ مجموعه‌ی دستورالعمل‌ها و وضعیت پردازنده[۱]

حالت پردازنده، مشخص می‌کند که کدام مجموعه دستورها اجرا می‌شوند. ۳ مجموعه‌ی دستورالعمل وجود دارند که عبارت‌انداز:ARM ، Thumb و Jazelle.

دستورهای ARM تنها وقتی فعال هستند که پردازنده در حالت ARM قرار داشته باشد. دستورهای Thumb نیز در حالت Thumb ، فعال هستند. دستورهای Thumb، ۱۶ بیتی هستند. امکان اجرای ترتیبی و ترکیبی دستورهای ARM ، Thumb و Jazelle وجود ندارد.

بیت‌های J (Jazelle) و T (Thumb) درون *cpsr* مشخص کننده‌ی حالت پردازنده هستند. وقتی هر دوی J و T، "0" باشند، پردازنده در حال ARM قرار دارد. هنگام اتصال تغذیه، پردازنده در وضعیت ARM قرار دارد. اگر بیت T='1' باشد، پردازنده در وضعیت Thumb قرار دارد. برای تغییر وضعیت، از دستورالنشعاب خاصی استفاده می‌شود. جدول ۲.۲ مشخصات مجموعه دستورهای Thumb و ARM را مقایسه می‌کند.

طراحان ARM ، مجموعه‌ی سومی از دستورها به نام Jazelle را نیز معرفی نمودند. Jazelle ، دستورهای بایت- کد[۲] ۸ بیتی جاوا را به صورت ترکیبی از سخت‌افزار و نرم‌افزار اجرا می‌کنند. برای اجرای دستورهای جاوا نیاز به جزئیات فناوری Jazelle و نمونه‌ای اختصاصی شده از JVM (ماشین مجازی جاوا) خواهید داشت. توجه کنید که تنها قسمتی از دستورهای جاوا در سخت‌افزار پیاده شده‌اند و بقیه باید توسط نرم‌افزار شبیه سازی شوند.

Table 2.2 ARM and Thumb instruction set features.

	ARM (cpsr T = 0)	Thumb (cpsr T = 1)
Instruction size	32 – bit	16 – bit
Core instructions	58	30
Conditional execution(1)	most	only branch instructions
Data processing instruction	access to barrel shifter and ALU	separate barrel shifter and ALU instructions
Program status register	read – write in privileged mode	no direct access

[۱] State

[۲] Byte code

Register usage	15 general – purpose registers	8 general – purpose registers
	+ pc	+ 7 high register + pc

جدول ۲.۲

Table 2.3 Jazelle instruction set features.

	Jazell (cpsr T=0 , J=1)
Instruction size	8 - bit
Core instructions	Over 60% of the java bytecodes are implemented in hardware: The rest of the codes are implemented in software.

جدول ۲.۳

دستورهای Jazelle بسته بوده و آزادانه در دسترس قرار ندارند. جدول ۲.۳ مشخصات مجموعه‌ی دستورهای Jazelle را ارایه می‌دهد.

۳.۲.۴ ماسک‌های وقفه

ماسک‌های وقفه، به‌منظور جلوگیری از ایجاد وقفه توسط منابع خاص استفاده می‌شود. دو سطح وقفه *IRQ* و *FIQ* در ARM وجود دارند. بیت‌های ماسک I و F (بیت‌های ۶ و ۷ *cpsr*) کنترل کننده این فرآیند ماسک گذاری هستند. هنگامی این بیت‌ها '1' شوند، وقفه‌ی مربوط شان '1' می‌شود.

۳.۲.۵ پرچم‌های شرط

پرچم‌های شرط (بیت‌های مشخص کننده یک موضوع را معمولاً پرچم می‌نامیم)، بر اثر نتیجه‌ی حاصل از عملیات مقایسه و دستورهایی در ALU که پسوند S دارند، تنظیم می‌شوند. برای مثال، اگر حاصل نتیجه‌ی SUBS در یک ثبات، برابر صفر باشد، پرچم Z در *cpsr* برابر '1' می‌شود. این دستور تفریق خاص مشخص برای دستکاری *cpsr* فراهم آورده شده است.

جدول ۲.۴

Table 2.4 Condition flags.

Flag	Flag name	Set when
Q	Saturation	the result causes an overflow and/or saturation
V	Overflow	the result causes a signed overflow
C	Carry	the result causes an unsigned carry
Z	Zero	the result is zero,frequently used to indicate equality
N	Negative	Bit 31 of the result is a binary 1

در پردازنده‌هایی که از عملیات پردازش سیگنال دیجیتال (DSP) بهره می‌برند، بیت Q معرف رخداد سرریز[۱] و یا اشباع[۲] نتیجه‌ی حاصله در دستور DSP مورد نظر است. این پرچم فقط توسط سخت‌افزار '1' می‌شود. ولی برای پاک کردنش، باید در درون *cpsr* بنویسید.

بیشتر دستورهای ARM با استفاده از این بیت‌های شرط می‌توانند به صورت شرطی اجرا شوند. جدول ۲.۴، کلیه‌ی پرچم‌های شرط را به همراه توضیحات‌شان و عامل تغییر دهنده‌شان نشان می‌دهد.

شکل ۳.۶، یک مقدار نمونه از *cpsr* را نشان می‌دهد که در آن هر دو افزونه‌ی Jazelle و DSP در آن فعال هستند.

توجه

برای این‌که نحوه‌ی خواندن بیت‌ها واضح‌تر باشند، از این به بعد پرچم‌هایی را که دارای مقدار '1' هستند، با حروف بزرگ (مثلاً پرچم صفر Z) و آن‌هایی راکه دارای مقدار صفر هستند، با حروف کوچک (مثلاً پرچم صفر z) نشان خواهیم داد.

---

[۱] Overflow

[۲] Saturation

در مثال، شکل ۳.۶ تنها بیت '1' شده بیت C بوده و بقیه بیت‌های *nzvq* همگی صفر هستند. پردازنـده در وضعیت ARM قرار دارد. (هر دو بیت t و j صفر هستند) وقفه‌های *IRQ* فعال و وقفه‌های *FIQ* غیر فعال هستند.

در پایـان همـان‌طـور کـه مـی‌بینیـد، پردازنـده در حالـت کـاری *Supervisor* قـرار دارد.(زیـرا mode[4:0]=10011)

شکل ۳.۶ – مثال برای وقتی که *cpsr = nzCvqjiFt_SVC*.

جدول ۲.۵

Table 2.5 Condition mnemonics.

Mnemonic	Name	Condition flags
EQ	equal	Z
NE	Not equal	z
CS HS	carry set/unsigned higher or same	C
CC LO	carry clear/ unsigned lower	c
MI	minus/negative	N
PL	plus/positive or zero	n
VS	overflow	V
VC	no overflow	v
HI	unsigned higher	zC
LS	unsigned lower or same	Z or c
GE	signed greater than or equal	NV or nv
LT	signed less than	Nv or nV
GT	signed greater than	NzV or nzv
LE	signed less than or equal	Z or Nv or nV
AL	always (unconditional)	ignored

۳.۲.۶ اجرای شرطی[۱]

اجرای شرطی، این‌که هسته‌ی پردازنده دستوری اجرا خواهد کرد یا نه را تحت کنترل دارد.
بیشتر دستورهای یک مشخصه‌ی شرط دارند که اجرا یا عدم اجرا شدن آن دستور را با توجه به پرچم‌های شرط، مشخص می‌کند. قبل از اجرای دستور، پردازنده مشخصه‌ی شرط دستورالعمل را با پرچم‌های شرط درون *cpsr* مقایسه کرده و اگر با یکدیگر انطباق داشته باشند، دستور اجرا خواهد شد.
مشخصه‌ی شرط بعد از نماد دستورالعمل قرار می‌گیرد و در درون دستورالعمل رمز می‌شود. جدول ۲.۵ نمادهای اجرای شرطی دستورها را فهرست می‌کند. اگر شرطی موجود نباشد، حالت پیش‌فرض استفاده از نماد AL (برای اجرای همیشگی دستور) است.

۳.۳ خط لوله[۲]

خط لوله، مکانیزمی است که پردازنده‌ی RISC برای اجرای دستورها به کار می‌بندند. با استفاده از خط لوله، سرعت اجرای دستورها افزایش می‌یابد. در خط لوله، هنگامی که دستورهایی در حال اجرا و یا رمز گشایی هستند، به طور موازی، دستور دیگری در حال واکشی است. یک روش بررسی خط لوله، تصور کردن خط مونتاژ یک خودرو است که وظیفه‌ی هر مرحله‌ی انجام وظایف خاص است، تا به یک محصول نهایی بیانجامد.

شکل ۳.۷ – خط لوله سه طبقه ARM7

شکل ۳.۷ یک خط لوله‌ی ۳ طبقه را نشان می‌دهد:

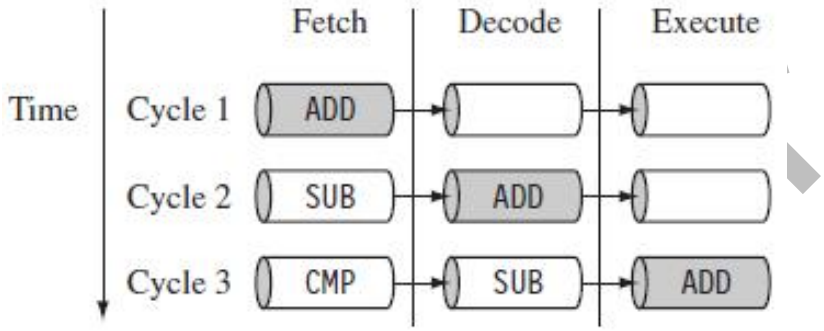
- واکشی[۳] یک دستور را از حافظه بارگیری می‌کند؛
- رمز گشایی[۴] دستوری که باید اجرا شود را مشخص می‌کند (از حالت رمز شده در می‌آورد)؛
- اجرا[۵]، دستور مورد نظر را پردازش کرده و نتیجه‌ی حاصل را بر روی ثبات می‌نویسد.

شکل ۳.۸ خط لوله را در یک مثال ساده که سه دستور به آن وارد شده‌اند، نشان می‌دهد.
پس از پر شدن خط لوله (بعد از سیکل سوم) دو دستورالعمل **فقط در یک سیکل** ماشین اجرا می‌شود.

---

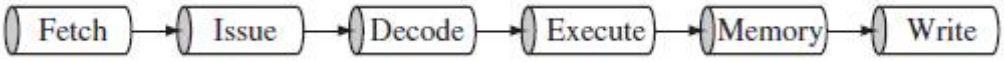
۱ Conditional Execution
۲ Pipe Line
۳ Fetch
۴ Decode
۵ Execute

اگر طول خط لوله افزایش یابد، میزان کاری که در هر مرحله انجام می‌شود کاهش می‌یابد به پردازنده اجازه می‌دهد در فرکانس کاری بالاتری کار کند. این به معنی افزایش کارآیی است. تأخیر اولیه‌ی سیستم به علت زمانی که طول می‌کشد تا خط لوله پرشود، نیز طولانی است. افزایش طول خط لوله به معنای افزایش وابستگی[1] بین طبقات خط لوله نیز هست. برای کاهش این وابستگی، می‌توانید کدهایی با استفاده از زمان‌بندی دستورها[2] بنویسید.



شکل ۳.۸ – ترتیب اجرای دستورها در خط لوله

شکل ۳.۹ – خط لوله پنج طبقه ARM9



شکل ۳.۱۰ – خط لوله شش طبقه ARM10

طراحی خط لوله‌ی هر خانواده‌ی ARM با دیگری متفاوت است. برای مثال ARM9 در خط لوله‌اش ۵ طبقه دارد. همان‌طور که در شکل ۳.۹ آمده است، ARM9 دو طبقه‌ی بازنویسی و حافظه را نیز به خط لوله اضافه کرده است. این افزایش طبقات به معنی حصول کارآیی 1.1 DhrystoueMIPS که ٪۱۳ بیشتر از ARM7 است، می‌باشد. بیشترین فرکانس قابل کارکرد ARM9 نیز از ARM7 بالاتر است.

ARM10 نیز با اضافه نمودن طبقه‌ی ششم، طول این خط لوله را نیز افزایش داده است. خط لوله ARM10 در شکل ۳.۱۰ آمده است. کارآیی قابل حصول در این‌جا به 1.3 MIPS به ازای هر MHZ رسیده است. این معنی ٪۳۴ افزایش کارآیی نسبت به ARM7 است.

اگرچه خط لوله ARM9 و ARM10 با ARM7 تفاوت دارند، اما کماکان مشخصات اجرایی یکسانی دارند. بدین معنی که کدهای نوشته شده برای ARM7، بر روی ARM9 و ARM10 نیز اجرا خواهند شد.

---

[1] Dependency

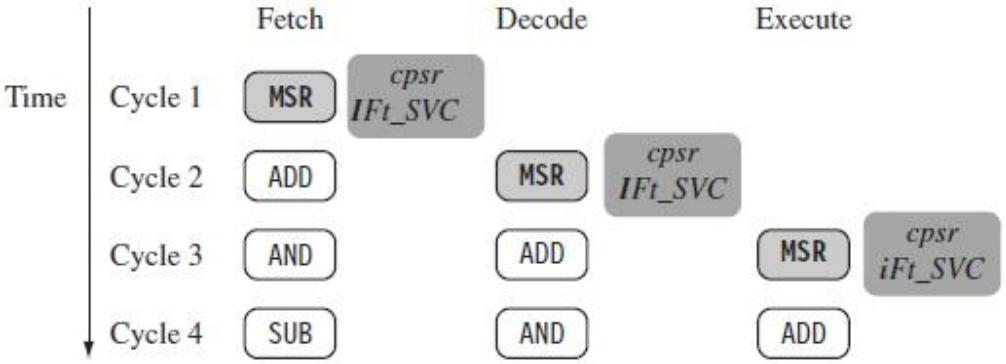
[2] Instruction Scheduling

۳.۳.۱ مشخصات اجرایی خط لوله

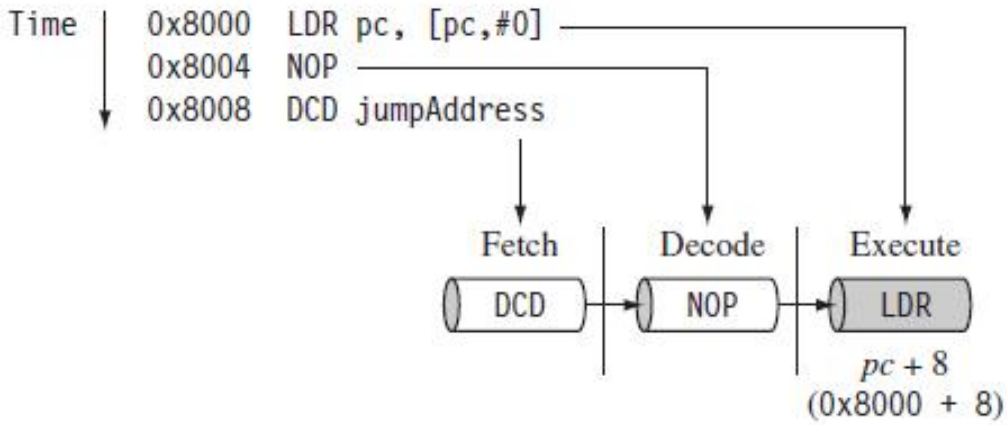
در خط لوله، اجرای یک دستور هنگامی به پایان می‌رسد که از مرحله‌ی اجرا گذشته باشد. برای مثال در ARM7، دستور اول وقتی اجرا شده است که چهارمین دستور واکشی شود.

شکل ۳.۱۱ ترتیبی از دستورات را در خط لوله‌ی ARM7 نشان می‌دهد. دستور MSR برای توانمند سازی وقفه‌های *IRQ* به کار رفته است. البته هنگامی این اتفاق می‌افتد که دستور مذکور، طبقه اجرا را سپری کرده باشد. این دستور بیت I را در *cpsr* پاک می‌کند تا به وقفه‌های *IRQ* اجازه رخداد بدهد. هنگام ورود دستور ADD به خط لوله، وقفه‌ها توانمند شده‌اند.

شکل ۳.۱۲ استفاده از خط لوله و شمارنده برنامه PC را نشان می‌دهد. در مرحله‌ی "اجرا"، PC حاوی آدرس دستور جاری به علاوه‌ی ۸ بایت می‌باشد. به عبارت دیگر، PC همراه به آدرس دستور اجراشونده به علاوه دو دستور بعد اشاره دارد. دانستن این مسأله در محاسبه‌ی آفست‌های نسبی مهم است و از مشخصات ساختاری دو خط لوله‌ای به حساب می‌آید. توجه کنید که در وضعیت Thumb، محتوای PC، آدرس دستورالعمل به علاوه‌ی ۴ است.



شکل ۳.۱۱ – ترتیب اجرای دستورهای ARM



شکل ۳.۱۲ – مثال برای وقتی که pc = address + 8.

سه مشخصه‌ی دیگر برای خط لوله وجود دارند که بد نیست در این‌جا به آن اشاره‌ای بکنیم.

اول این‌که اجرای یک دستور انشعاب و یا دستکاری مستقیم PC به‌منظور ایجاد انشعاب، باعث می‌شود که پردازنده‌ی ARM تمام خط لوله را پاک کند.

دوم این‌که از پیش‌بینی انشعاب[۱]، استفاده می‌کند. این افزونه، دفعات رخداد عملیات خالی کردن خط لوله[۲] را کاهش می‌دهد. این کار با پیش‌بینی وجود انشعاب و بارگیری آدرس مناسب انشعاب قبل از بارگیری دستور بعد است.

و نکته‌ی سوم این‌که هنگام رخداد وقفه، دستور جاری تا انتها اجرا شده ولی پردازنده‌ی دستورهای بعدی را با محتویات جدول بردار[۳] چه می‌کند.

۳.۵ افزونه‌های هسته[۴]

افزونه‌هایی که در این بخش به آن پرداخته می‌شود، شامل اجزای استانداردی هستند که در کنار هسته‌ی پردازنده قرار می‌گیرند. وظیفه‌ی آن‌ها افزایش کارآیی، مدیریت منابع و فراهم آوردن کارکردهای اضافی برای افزایش کارآیی در بعضی کاربردهای خاص است.

سه افزونه وجود دارند که در این بخش به آن‌ها خواهیم پرداخت: حافظه‌ی نهان[۵] و TCM، مدیریت حافظه (MMU) و واسط کمک پردازنده[۱].

---

۱ Branch Prediction

۲ Pipe Line Flush

۳ Vector Table

۴ Core Extentions

۵ Cache

۳.۵.۱ حافظه‌ی نهان و TCM

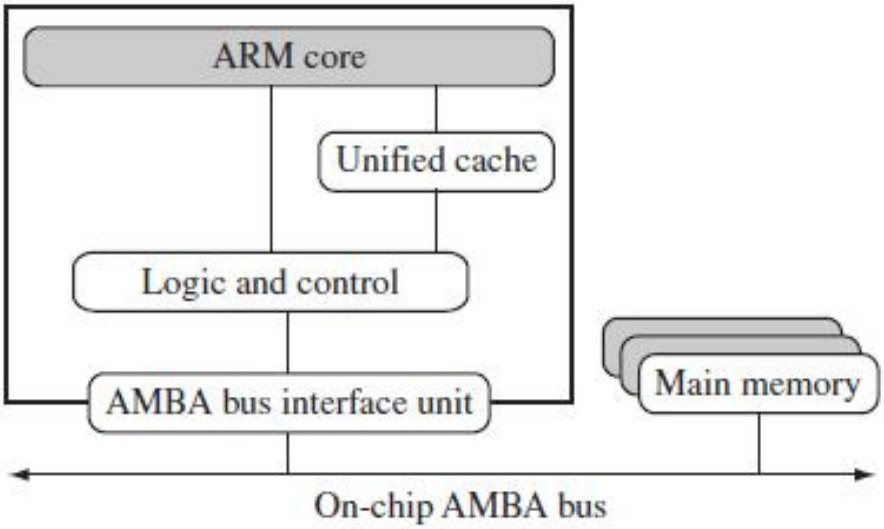
"حافظه‌ی نهان"، بلوک حافظه‌ای سریع است که بین حافظه‌ی اصلی و هسته قرار می‌گیرد. وظیفه‌ی آن افزایش کارآیی واکشی دستورالعمل‌ها از بعضی انواع حافظه‌های خارجی است. پردازنده با استفاده از حافظه نهان این امکان را می‌یابد که در اکثر مواقع در حال اجرای دستورها باشد و درگیر حافظه‌های خارجی کُند و زمان‌های تأخیر آن‌ها نمی‌شود. بیشتر پردازنده‌های ARM از حافظه‌ی نهان تک سطحی (همان L1) که در درون پردازنده قرار دارد، بهره می‌برند. بدیهی است، بسیاری از سیستم‌های تعبیه شده از مزایایی که حافظه‌های نهان ارایه می‌دهد، استفاده نخواهند کرد.

ARM ، دو نوع حافظه‌ی نهان دارد. اولین مدل آن در هسته‌های با معماری Von Neumann پیدا می‌شود. در این نوع حافظه‌ی نهان، هر دو اطلاعات مربوط به داده و دستورها در یک حافظه‌ی نهان یکپارچه در دسترس قرار می‌گیرند. (نمونه‌ی این معماری در شکل ۳.۱۳ آمده است). در مقایسه با معماری بالا، معماری دیگری نیز برای Harvard درست شده است که از دو حافظه‌ی نهان مجزای داده و دستورالعمل استفاده می‌کند.

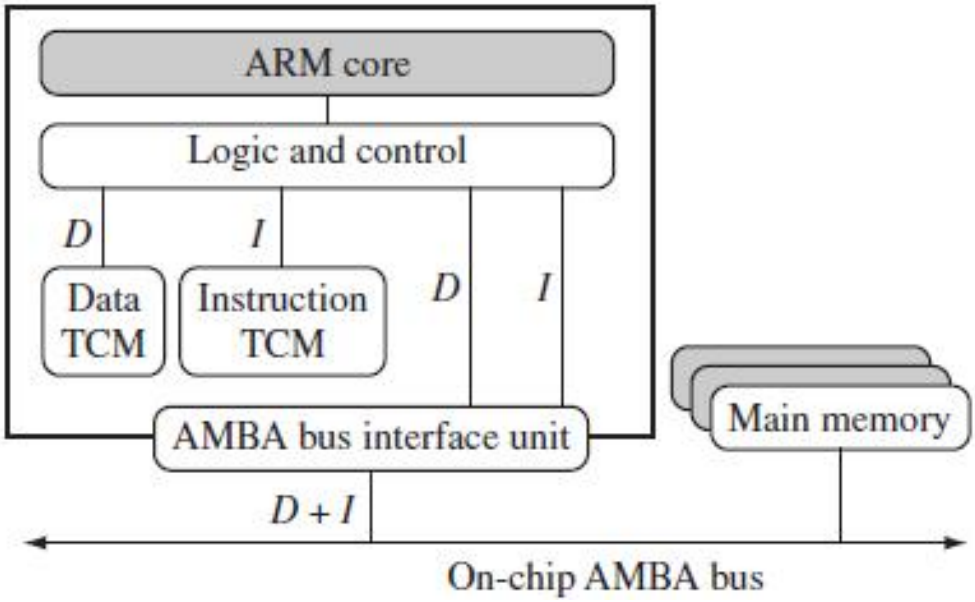
حافظه‌ی نهان، کارآیی کلی سیستم را افزایش می‌دهد که این مسأله در ازای قابل پیش‌بینی نبودن زمان اجرای دستورها است. برای جبران این معزل، از نوعی دیگر حافظه به نام TCM استفاده می‌کنیم. TCM یک SRAM سریع است که در نزدیکی پردازنده قرار گرفته است. مزیت این حافظه، قابل پیش‌بینی بودن زمان واکشی داده‌ها و دستورالعمل‌هاست (این مسأله برای الگوریتم‌های بی‌درنگ که نیاز به یک رفتار زمانی مشخص دارند، امری حیاتی است). TCM به عنوان حافظه‌های سریع در نقشه‌ی حافظه‌ی پردازنده قرار می‌گیرند و می‌توانند استفاده شوند. نمونه‌ای از یک پردازنده به همراه TCM در شکل ۳.۱۴ نشان داده شده است.

---

[۱] Coprocessor



شکل ۳.۱۳ – معماری ساده شده‌ی Von Neumann به همراه حافظه‌ی نهان



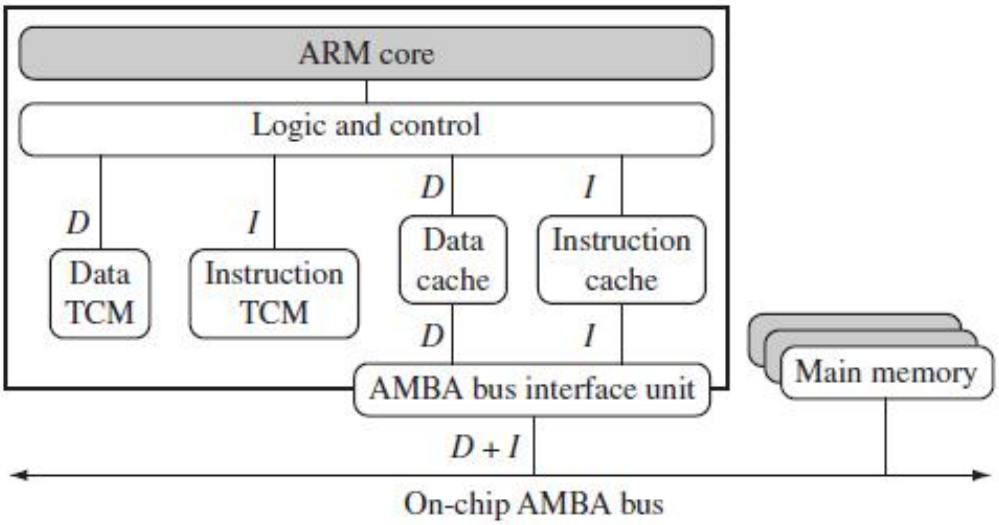
شکل ۳.۱۴ – معماری ساده شده‌ی Harvard به همراه TCM

با ترکیب هر دو تکنولوژی فوق، می‌توان به کارآیی بهینه و بهتری دست یافت. نمونه‌ای از پیاده سازی مورد اخیر در ARM را در شکل ۳.۱۵ می‌بینیم.

۳.۵.۲ مدیریت حافظه

سیستم‌های تعبیه شده، اغلب از چندین حافظه استفاده می‌کنند. ما، اغلب به دنبال روشی برای مدیریت، سازماندهی و حفاظت از این نوع حافظه‌ها از دسترس‌ها ی ناخواسته توسط برنامه‌ها هستیم. این خصایص تنها با استفاده از کمک سخت‌افزار "مدیریت حافظه" ممکن شده است.

هسته‌های ARM در سه نوع: بدون محافظت، واحد محافظت از حافظه (MPU) با محافظت محدود و واحد مدیریت حافظه (MMU) با محافظت کامل، ساخته شده‌اند.



شکل ۳.۱۵ – معماری ساده شده‌ی Harvard به همراه TCM و حافظه‌ی نهان

در فصول بعدی، در مورد هر یک از سه مورد گفته شده صحبت خواهیم کرد.

۳.۶ نسخه‌های مختلف معماری

هر معماری از پردازنده‌های ARM از یک معماری مجموعه‌ی دستورالعمل‌ها (ISA) ی خاص استفاده می‌کند. اگرچه یک ISA ممکن در چندین پردازنده پیاده شده باشد.

توسعه‌ی ISA برای نیاز بازار فروش انجام شده است. یکی از مزایای این توسعه این بوده است: کدهایی که برای اجرا بر روی نسخه‌های پایین‌تر معماری نوشته شده‌اند، به آسانی بر روی نسخه‌های بالاتر نیز اجرا می‌شوند. قبل از توضیح نحوه‌ی پیشرفت نسخه‌های مختلف، بد نیست نگاهی به مجموعه و اختصارات نام‌گذاری این پردازنده‌ها بیاندازیم. این علائم مشخص کننده‌ی پردازنده‌ها و توانمندی‌های پایه‌ی آن‌ها هستند.

۳.۶.۱ مجموعه‌ی علائم

ARM از علائم شکل ۳.۱۶ برای توضیح در مورد نحوه‌ی پیاده سازی پردازنده‌اش استفاده می‌کند. حروف بعد از کلمه‌ی "ARM" ، مشخص کننده‌ی این خصوصیات هستند.

ARM{x}{y}{z}{T}{D}{M}{I}{E}{J}{F}{-S}

x—family
y—memory management/protection unit
z—cache
T—Thumb 16-bit decoder
D—JTAG debug
M—fast multiplier
I—EmbeddedICE macrocell
E—enhanced instructions (assumes TDMI)
J—Jazelle
F—vector floating-point unit
S—synthesizible version

شکل ۳.۱۶ – روش نام‌گذاری پردازنده‌های ARM

این اعداد و حروف، با گذشت زمان در حال تغییر هستند. با پیشرفت تکنولوژی، ویژگی‌های جدیدی نیز به این پردازنده‌ها افزوده می‌شوند. توجه کنید که این علائم، اطلاعات مربوط به نسخه‌ی معماری را دربر ندارند. در این‌جا لازم است که چند نکته در رابطه با این علائم شرح داده شوند.

- تمامی هسته‌های ARM بعد از ARM7TDMI شامل خصیصه‌ی TDMI هستند. حتی اگر در نام-گذاری آن‌ها ذکر نشده باشد؛
- خانواده‌ی پردازنده[۱] اشاره به گروهی از پردازنده‌ها دارد که از مشخصات سخت‌افزاری یکسانی استفاده کرده‌اند. به عنوان مثال، ARM7TDMI، ARM740T و ARM720T همگی دارای مشخصات خانواده‌ای یکسانی بوده و به خانواده‌ی ARM7 تعلق دارند؛
- JTAG در استاندارد IEEE با شماره‌ی 1149.1 توضیح داده شده است. مورد استفاده‌ی آن در این‌جا ارسال و دریافت اطلاعات عیب‌یابی (یا دیباگ[۲]) بین هسته‌ی پردازنده و تجهیزات آزمایش کننده‌ی سیستم است؛

---

[۱] Processor Family

[۲] Debug

- ماکروسل Embedded ICE، سخت‌افزار عیب‌یابی مقیم در پردازنده است که وظیفه‌اش تنظیم و به کارگیری نقاط شکست[۱] و نقاط نگهبان[۲] است؛
- قابل سنتز[۳] به این معناست که هسته‌ی پردازنده به صورت کد منبع[۴] در اختیار سازندگان قرار گرفته است. آنان نیز با استفاده از ابزارهای طراحی، این کد را به سخت‌افزار مورد نظر تبدیل نموده‌اند.

۳.۶.۲ سیر تکامل معماری

از اولین نمونه پردازنده ARM ارایه شده در سال 1985، معماری ARM دایماً در حال تکامل و پیشرفت است. جدول ۲.۷ مهم‌ترین بهبودهای ساختاری از اولین نسخه تاکنون را نشان می‌دهد.
یکی از مهم‌ترین تغییرات ایجاد شده در ISA، ارایه‌ی دستورهای Thumb از اوایل نسخه‌ی ARMv4 (پردازنده‌ی ARM7TDMI) بوده است.
جدول ۸.۲ قسمت‌های مختلف cpsr و در دسترس بودن بعضی مشخصات را در یک ISA خاص نشان می‌دهد.

۳.۷ خانواده‌های پردازنده ARM

شرکت ARM ، پردازنده‌های گوناگونی را طراحی کرده است که براساس همان مشخصات سخت-افزاری‌شان خانواده‌های مختلفی قرار می‌گیرند. این خانواده‌ها برپایه‌ی هسته‌های ARM7، ARM9، ARM10، ARM11 و ARM-Cortex بنا شده‌اند. اعداد 7، 9، 10 و 11 و حروف Cortex نشان دهنده‌ی طرح‌های مختلف پردازنده هستند. ARM8 ساخته شده، اما به علت مقبول نیافتادنش به سرعت از چرخه‌ی محصولات کنار گذاشته شد. افزایش شماره از 7 به 11 و به سمت Cortex، به معنای افزایش کارآیی و رضایت بخش بودن محصولات است.
جدول ۲.۹ مقایسه‌ای نسبی بین مشخصات خانواده‌های ذکر شده می‌کند.

Table 2.7 Revision history.

Revisin	Example core implementation	ISA enhancement

[۱] Break Point
[۲] Watch Point
[۳] Synthesizable
[۴] Source Code

ARMv1	ARM1	First ARM processor 26-bit addressing
ARMv2	ARM2	32-bit multiplier 32-bit coprocessor support
ARMv2a	ARM3	On-chip cache Atomic swap instruction Coprocessor 15 for cache management
ARMv3	ARM6 and ARM7DI	32-bit addressing Separate cpsr and spsr New modes-undefined instruction and abort MMU support-virtual memory
ARMv3M	ARM7M	Signed and unsigned long multiply instructions
ARMv4	StrongARM	Load-store instructions for signed and unsigned halfwords/bytes New mode-system Reserve SWI space for architecturally defined operations 26-bit addressing mode no longer supported
ARMv4T	ARM7TDMI and ARM9T	Thumb
ARMv5TE	ARM9E and ARM10E	Superset of the ARMv4T Extra instructions added for changing state between ARM and Thumb Enhanced multiply instructions Extra DSP-type instructions Faster multiply accumulate
ARMv5TEJ	ARM7EJ and ARM926EJ	Java ecceleration
ARMv6	ARM11	Improved multiprocessor instructions Unaligned and mixed endian data handling

	New multimedia instructions

جدول ۲.۷

Table 2.8 Description of the cpsr

Parts	Bits	Architectures	Description
Mode	4:0	all	Processor mode
T	5	Armv4T	Thumb state
I & F	7:6	all	Interrupt masks
J	24	ARMv5TEJ	Jazelle state
Q	27	ARMv5TE	Condition flag
V	28	all	Condition flag
C	29	all	Condition flag
Z	30	all	Condition flag
N	31	all	Condition flag

جدول ۲.۸

Table 2.9 Description of the cpsr

	ARM7	ARM9	ARM10	ARM11
Pipeline depth	Three-stage	Five-stage	Six-stage	Eight-stage
Typical MHz	80	150	260	335
mW/MHz(a)	0.06 mW/MHz	0.19mW/MHz (+cache)	0.5mW/MHz (+cache)	0A mW/MHz (+cache)
MIPS(b)/MHz	0.97	1.1	1.3	۱.۲
Architecture	Von Neumann	Harvard	Harvard	Harvard
Multilier	8×32	8×32	16×32	16×32

(a) Watts/MHz on the same 0.13 mocron process.
(b) MIPS are Dhrystone VAX MIPS.

جدول ۲.۹

در هر خانواده‌ی ARM، انواع مختلفی که شامل حافظه‌ی نهان، TCM و واحد مدیریت حافظه هستند، وجود دارند. شرکت ARM ، دایماً در حال توسعه‌ی تعداد شماره‌ها و افزونه‌های اضافه شونده به پردازنده‌هایش است.

پردازنده‌هایی دیگر که ISA مربوط به ARM را اجرا می‌کنند نیز وجود دارند. پردازنده‌هایی چون Strong ARM و Xscale از شرکت Intel که قبلاً نیز به آن اشاره شده، از این دسته‌اند.

جدول ۲.۱۰ مشخصات مختلف پردازنده‌های گوناگون را فهرست کرده است. در ادامه‌ی این بخش خانواده‌های ARM را بیشتر مورد مطالعه قرار خواهیم داد.

Table 2.10 Description of the cpsr

CPU core	MMU/MPU	Cache	Jazelle	Thumb	ISA	E (a)
ARM7TDMI	none	none	no	yes	v4T	no
ARM7EJ-S	none	none	yes	yes	v5TEJ	yes
ARM720T	MMu	Unifined__8K cache	no	yes	v4T	no
ARM920T	MMU	Separate__16K/16K D+I cache	no	yes	v4T	no
ARM922T	MMU	Separate__8K/8K D+I cache	no	yes	v4T	no
ARM926EJ-S	MMU	Separate__ cache and TCMs configurable	yes	yes	v5TEJ	yes
ARM940T	MPU	Separate__4K/4K D+I cache	no	yes	v4T	no
ARM946E-S	MPU	Separate__ cache and TCMs configurable	no	yes	v5TE	yes
ARM960E-S	none	Separate__ TCMs configurable	no	yes	v5TE	yes
ARM1020E	MMU	Separate__ 32K/32K D+I cache	no	yes	v5TE	yes
ARM1022E	MMU	Separate__ 16K/16K D+I cache	no	yes	v5TE	yes
ARM1026EJ-S	MMU and MMP	Separate__ cache and TCMs configurable	yes	yes	v5TE	yes
ARM1136J-S	MMU	Separate__ cache and TCMs configurable	yes	yes	v6	yes
	MMU	Separate__ cache and TCMs configurable	yes	yes	v6	yes

(a) E extension provides enhanced multiply instructions and saturation.

جدول ۲.۱۰

۳.۷.۱ خانواده‌ی ARM7

ARM7 ساختاری از نوع معماری Von Neumann دارد که در آن هر دو گذرگاه مربوطه‌ی داده و دستورها یکی هستند. خط لوله‌ی این هسته، ۳ مرحله‌ای بوده و مجموعه‌ی دستورها ARMv4T را اجرا می‌کند. ARM7TDMI ، اولین مجموعه از گروهی جدید از پردازنده‌ها بود که در سال 1995 روانه‌ی بازار شد. هم اکنون این هسته بسیار رایج بوده و در بسیاری از سیستم‌های تعبیه شده استفاده می‌شود. نسبت "کارآیی به توان مصرفی" در این پردازنده خیلی خوب است. مجوز ساخت پردازنده ARM7TDMI توسط بسیاری از سازندگان مطرح نیمه هادی و تراشه‌های سیلیکونی در جهان خریداری شده است. از دلایل اقبال این پردازنده حضور دستورهای Thumb، دستورهای ضرب سریع و فناوری عیب‌یابی Embedded ICE در این پردازنده است.

نوع دیگری از این پردازنده، پردازنده‌ی ARM7TDMI-S است. این پردازنده‌ی جدید، علاوه بر دارا بودن تمامی مشخصات ARM7TDMI، قابل سنتز کردن نیز هست.

ARM720T ، منعطف‌ترین پردازنده‌ی این خانواده است زیرا در درون خود دارای MMU است. داشتن MMU ، به معنای پشتیبانی از سیستم‌عامل‌های پیشرفته چون Linux و WinCE شرکت Microsoft است. این پردازنده، دارای حافظه‌ی نهان یکپارچه‌ی ۸ کیلوبایتی نیز هست.

گونه‌ای دیگر، پردازنده‌ی ARM7EJ-S است. این پردازنده کاملاً متفاوت است. این هسته قابل سنتز بوده و دارای خط لوله‌ی ۵ مرحله‌ای است. این هسته دستورهای جاوا را اجرا کرده، از دستورهای بهبود یافته پشتیبانی کرده (دستورهای ARMSTEJ) و در عین حال از وجود واحد محافظت از حافظه، بی‌بهره است.

۳.۷.۲ خانواده‌ی ARM9

خانواده‌ی ARM9 ، در سال 1997 ارایه شد. به خاطر ۵ طبقه‌ای بودن خط لوله، این پردازنده در فرکانس کاری بالاترین سطح به ARM7 می‌تواند کار کند. طبقات اضافی خط لوله کارآیی بیشتری را نسبت به خط لوله‌ی ۳ مرحله‌ای ارایه کرده‌اند. سیستم حافظه با طراحی مجدد از معماری Harvard پیروی می‌کند که گذرگاه‌های جداگانه‌ای را برای داده (D) و دستورالعمل (I) اختصاص داده است.

اولین پردازنده‌ی این خانواده ARM920T بود که دارای یک حافظه‌ی نهان جداگانه D+I به همراه یک MMU بود. این پردازنده در سیستم‌های عاملی که نیاز به پشتیبانی از حافظه‌ی مجازی[۱] دارند، به راحتی به کار گرفته می‌شود. ARM922T گونه‌ای از ARM920T بوده که اندازه‌ی حافظه‌ی نهان D+I در آن نصف شده است. ARM940T دارای حافظه‌ی نهان کوچک‌تر و یک واحد MPU است. این

---

[۱] Virtual Memory

پردازنده برای سیستم‌هایی که از سیستم‌های عامل پلتفرم استفاده نمی‌کنند، طراحی شده است. هر دو پردازنده‌ی ARM920T و ARM940T ، دستورهای ساختار V4T را اجرا می‌کنند.
پردازنده‌های بعد خانواده‌ی ARM9 بر مبنای هسته‌ی ARM9E-S ساخته شده‌اند. این هسته نمونه قابل سنتز هسته‌ی ARM9 به همراه افزونه‌ی E است. دو مدل مختلف از این هسته وجود دارند:
ARM946E-S و ARM966E-S . دستورهای اجرایی بر روی این پردازنده‌ها، دستورهای V5TE هستند. این پردازنده‌ها از مدار ETM برای دنبال کردن اجرای دستورها به صورت بی‌درنگ، پشتیبانی می‌کنند. ETM هنگام عیب‌یابی مدارات حساس از لحاظ زمان اجرا، بسیار کاربرد دارد.
ARM946E-S دارای حافظه‌ی نهان، TCM و MPU است. اندازه‌ی حافظه‌ی نهان و TCM قابل تنظیم هستند. در مقابل ARM966E-S حافظه‌ی نهان و MPU ندارد ولی TCM های قابل تنظیم دارد.
آخرین هسته در مجموعه‌ی ARM9، پردازنده‌ی ARM926EJ-S بود که در سال 2000 ارایه شد. این پردازنده برای دستگاه‌های قابل حمل کوچک که قابلیت اجرای جاوا داشتند، طراحی شد. در این دسته وسایل می‌توان به PDA ها و تلفن‌های همراه 3G اشاره کرد. ARM926EJ-S اولین پردازنده‌ی ارایه شده توسط ARM بود که از تکنولوژی Jazelle استفاده می‌کرد. این پردازنده دارای MMU، TCM های قابل تنظیم و حافظه‌های نهان D+I است.

۳.۷.۳ خانواده‌ی ARM10

ARM10 ، در سال 1999 پا بر عرصه‌ی تولیدات شرکت ARM گذاشت. این پردازنده، برای کارآیی بالا طراحی شده بود. به همین علت از یک خط لوله‌ی ۶ طبقه بهره می‌برد. این پردازنده همچنین از یک واحد "نقطه اعشاری برداری[۱]" اختیاری استفاده می‌کرد که یک طبقه بر خط لوله اضافه می‌کرد.
VFU ، کارآیی عملیات اعشاری را به شدت افزایش می‌دهد و استاندارد IEEE 754-1985 نیز سازگار است.
ARM1020E ، اولین پردازنده‌ای بود که از هسته‌ی ARM10E استفاده می‌کرد. این پردازنده دارای 32KB حافظه‌ی نهان D+I، واحد VFU اختیاری و یک MMU است. ARM1020E دارای یک گذرگاه دوگانه‌ی ۶۹ بیتی بر افزایش کارآیی نیز هست.
ARM1026EJ-S ، خیلی شبیه ARM926EJ-S است. با این تفاوت که دارای هر دو واحد MPU و MMU می‌باشد. این پردازنده دارای کارآیی ARM10 با قابلیت‌ها و انعطاف‌پذیری ARM926EJ-S است.

[۱] Vector Floating-Point VFU

۳.۷.۴ خانواده‌ی ARM11

ARM1136J-S ، در سال 2003 متولد شد. هدف از تولید این پردازنده، دستگاه‌هایی با کارآیی بالا و مصرف توان بهینه بود. این پردازنده، اولین هسته‌ای بود که از دستورهای ARMv6 استفاده می‌کرد. این پردازنده دارای خط لوله‌ی ۸ طبقه است و برای عملیات ریاضی، بارگیری و ذخیره سازی، خط لوله‌ی جداگانه‌ای دارد. افزونه‌ی SIMD (تک دستورالعمل و چندین داده) دستورهایی هستند که در ARMv6 ارایه شدند و کاربردهایی در عملیات چندرسانه‌ای دارند. این دستورها، اختصاصاً برای افزایش کارآیی پردازش تصویر ایجاد شده‌اند.ARM1136JF-S همان ARM1136J-S به اضافه‌ی یک واحد VFU برای سرعت بخشیدن به عملیات نقطه‌ی اعشاری است.

فصل چهارم

رسیدگی به استثنا و وقفه

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ استثناها و وقفه‌ها چیستند ؟
- ✓ تقدم استثنا و جدول بردار؛
- ✓ رسیدگی به وقفه‌ها به چند روش صورت می‌گیرد؟
- ✓ استثناهای FIQ و IRQ.

رسیدگی کننده‌های به استثنا، در قلب سیستم‌های تعبیه شده جای دارند. وظیفه شان رسیدگی به خطاها، وقفه‌ها و دیگر رویدادهای تولید شده توسط سیستم‌های خارجی است. روال‌های رسیدگی کننده‌ی کارآمد، کارآیی کل سیستم را به میزان چشم‌گیری افزایش می‌دهند. فرآیند انشعاب و روش رسیدگی به این رویدادها از کارهای زمان‌بر و پیچیده است.

در این بخش، روش تئوری و تجربی رسیگی به استثناها (و در حالت خاص روش‌های رسیدگی به وقفه‌ها در ARM) را بررسی خواهیم کرد. پردازنده‌ی ARM، هفت حالت استثنا دارد که می‌توانند روند اجرای ترتیبی دستورها را متوقف بسازند: *Data Abort*، *Fast Interrupt Request*، *Interrupt Request*، *Prefetch Abort*، *Software Interrupt*، *Reset* و *Undefined Instruction*.

این فصل به سه زیربخش کلی تقسیم می‌شود:

- رسیدگی به استثناها – بررسی روش‌های خاصی که پردازنده‌ی ARM به استثناها رسیدگی می‌کنند؛
- وقفه‌ها – پردازنده‌ی ARM ، وقفه را به عنوان حالت خاصی از استثنا درنظر می‌گیرد. این بخش درخواست وقفه را شرح داده و با تعریف یک سری عبارات رایج، مکانیزم‌های رسیدگی به وقفه را توضیح می‌دهد؛
- روش‌های رسیدگی به وقفه – در بخش انتهایی مجموعه‌ای از روش‌های رسیدگی به وقفه‌ها به همراه مثال‌هایی از شیوه‌های به کار بردن هر روش را آورده‌ایم.

۴.۱ رسیدگی به وقفه

استثنا، حالتی است که نیاز به متوقف ساختن روند اجرای ترتیبی دستورها دارد. مثال آن وقتی است که هسته‌ی ARM ریست می‌شود. "رسیدگی به استثناها"، روش‌های پردازش این استثناهاست. بیشتر استثناها یک برنامه‌ی رسیدگی کننده‌ی نرم‌افزاری دارند. (برنامه‌ای که هنگام رخداد استثنا اجرا می‌شود) برای مثال استثنای *Data Abort* یک روال رسیدگی کننده به نام **`Data Abort Handler`** دارد.

رسیدگی کننده، ابتدا منشأ ایجاد استثنا را پیدا کرده، سپس به آن خدمات می‌دهد. خدمات رسانی یا در روال رسیدگی کننده و یا از طریق پرش به یک زیربرنامه‌ی رسیدگی کننده، صورت می‌گیرد. استثنای "ریست" در این مورد با بقیه فرق دارد و هنگام مقداردهی اولیه‌ی سیستم، اجرا می‌شود. این بخش مباحث رسیدگی به استثناها را این‌گونه دنبال می‌کند:

- حالت‌های کاری پردازنده‌ی ARM و استثناها؛
- جدول بردار[۱]؛
- تقدم استثناها؛
- آفست ثبات لینک[۲].

۴.۱.۱ حالت‌های کاری پردازنده‌ی ARM و استثناها

جدول ۹.۱، استثناهای پردازنده‌ی ARM را در فهرستی آورده است. هر استثنا، پردازنده را به حالت خاصی وارد می‌کند. علاوه بر روش فوق، با دستکاری *cpsr* نیز می‌توانیم حالت‌های کاری پردازنده را عوض کنیم. حالت‌های کاری *User* و *System* تنها حالاتی هستند که از طریق استثنا نمی‌توان به آن‌ها وارد شد. (تنها راه ورود به آن‌ها از طریق دستکاری *cpsr* است).

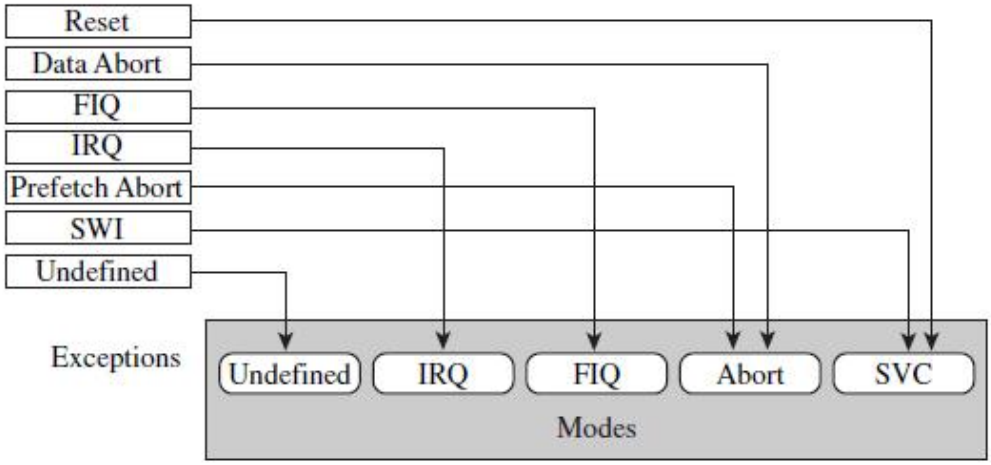
وقتی یک استثنا باعث تغییر حالت کاری پردازنده می‌شود، هسته‌ی پردازنده به طور خودکار:

- *cpsr* را در درون *spsr* حالت استثنا ذخیره می‌کند؛
- PC را در درون *lr* حالت استثنا ذخیره می‌کند؛
- *cpsr* را برای حالت استثنا تنظیم می‌کند؛
- PC را با آدرس زیربرنامه‌ی رسیدگی به استثنا پر می‌کند.

استثنا	حالت کاری	هدف اصلی
Fast Interrupt Request	FIQ	fast interrupt request handling
Interrupt Request	IRQ	interrupt request handling

[۱] Vector Table
[۲] Link Register offset

protected mode for operating systems	SVC	SWI and Reset
virtual memory and/or memory protection handling	abort	Prefetch Abort and Data Abort
software emulation of hardware coprocessors	undefined	Undefined Instruction



شکل ۴.۱ – استثناها و حالت‌های کاری مرتبط

شکل ۴.۱، شمایی ساده از استثناها و حالت‌های کاری مرتبط را نشان می‌دهد.

توجه

هنگام رخداد استثنا، پردازنده‌ی ARM همواره به حالت ARM تغییر حالت عملکردی می-دهد.

۴.۱.۲ جدول بردار

جدولی است که پردازنده‌ی ARM هنگام رخداد استثنا به آن انشعاب می‌کند. این آدرس‌ها عموماً شامل یکی از دستورهای انشعاب به یکی از شکل‌های زیر هستند:

- B <address>ـ دستور انشعاب که انشعابی نسبی[۱] نسبت به مقدار فعلی PC فراهم می‌آورد؛

[۱] Relative

- LDR PC, [PC, # Offset] ـ دستور بارگیری ثبات که آدرس روال رسیدگی کننده را از حافظه به درون PC بارگیری می‌کند. آدرس مقداری ۳۲ بیتی است که در نزدیکی جدول بردار ذخیره شده است. استفاده از این دستور، تأخیری به همراه دارد، هرچند امکان پرش به هر آدرس ۳۲ بیتی وجود دارد؛
- LDR PC, [PC, # -0xff0] – دستور بارگیری ثبات که آدرس یک سرویس دهنده‌ی وقفه را از آدرس 0xffff f030 به درون PC بارگیری می‌کند؛
- MOV PC, # immediate ـ این دستور، یک مقدار ثبات را به درون PC کپی می‌کند.

وجود دستورهای دیگر در جدول بردار امکان‌پذیر است. برای مثال، روال رسیدگی کننده به FIQ ممکن است از آدرس 0×1C+ شروع شود. از آن‌جایی که FIQ در انتهای جدول بردار قرار دارد، زیربرنامه‌ی رسیدگی به *FIQ* می‌تواند در همین آدرس بردار قرار گیرد. جدول ۹.۲ استثنا، حالت کاری و آفست جدول بردار را برای هر استثنا نشان می‌دهد.

استثنا	حالت کاری	آفست جدول بردار وقفه
Reset	SVC	+0x00
Undefined Instruction	UND	+0x04
Software Interrupt (SWI)	SVC	+0x08
Prefetch Abort	ABT	+0x0C
Data Abort	ABT	+0x10
Not assigned	--	+0x14
IRQ	IRQ	+0x18
FIQ	FIQ	+0x1C

مثال ۹.۱

شکل ۴.۲ یک جدول بردار را نشان می‌دهد.ورودی مربوط به *Undefined Instruction* ، یک دستور انشعاب است که به روال رسیدگی کننده‌ی دستور تعریف نشده پرش می‌کند. بردارهای دیگری از پرش غیرمستقیم به آدرس خود با استفاده از دستور LDR، استفاده می‌کنند.

```
0x00000000: 0xe59ffa38  RESET: > ldr  pc, [pc, #reset]
0x00000004: 0xea000502  UNDEF:   b    undInstr
0x00000008: 0xe59ffa38  SWI   :  ldr  pc, [pc, #swi]
0x0000000c: 0xe59ffa38  PABT  :  ldr  pc, [pc, #prefetch]
0x00000010: 0xe59ffa38  DABT  :  ldr  pc, [pc, #data]
0x00000014: 0xe59ffa38   -    :  ldr  pc, [pc, #notassigned]
0x00000018: 0xe59ffa38  IRQ   :  ldr  pc, [pc, #irq]
0x0000001c: 0xe59ffa38  FIQ   :  ldr  pc, [pc, #fiq]
```

شکل ۴.۲ – یک جدول بردار نمونه

توجه کنید که رسیدگی به *FIQ* نیز در همان محل انجام نمی‌شود بلکه از دستور LDR به‌منظور پرش به آدرس خود استفاده می‌کند.

۴.۱.۳ تقدم استثناها[۱]

از آن‌جا که امکان رخداد هم‌زمان استثناها در پردازنده وجود دارد، پردازنده باید مکانیزمی برای ایجاد تقدم در رسیدگی به آن‌ها داشته باشد. جدول ۹.۳ فهرستی از استثناهای مختلف به همراه سطح تقدم آن‌ها را نشان می‌دهد. استثنای ریست، بالاترین تقدم را داشته و هنگام اتصال تغذیه به سیستم رخ می‌دهد. بعد از ریست، استثنای *Data Abort* دارای بالاترین تقدم نسبت به دیگر استثناهاست. پایین‌ترین سطح تقدم مشترکاً مربوط به دو استثنای "وقفه‌ی نرم‌افزاری" و "دستورالعمل تعریف نشده" است. بعضی از استثناها، وقفه را از طریق بیت‌های I و F درون *cpsr* غیرفعال می‌کنند.

استثنای ریست، بالاترین تقدم را داشته و همواره بعد از فراخوانی‌اش، اجرا می‌شود. روال رسیدگی به ریست، سیستم را مقداردهی اولیه کرده و حافظه و حافظه‌ی نهان را پیکره‌بندی می‌کند. مقداردهی اولیه‌ی وقفه‌های خارجی، باید قبل از فعال سازی *FIQ* و *IRQ* انجام بگیرد. بدین وسیله، امکان رخداد وقفه‌های ناخواسته، قبل از تنظیم روال رسیدگی کننده وجود ندارد. روال رسیدگی به ریست، وظیفه‌ی پیکره‌بندی پشته‌ی[۲] مربوط به تمامی حالات پردازنده را نیز دارد.

فرض بر این‌است که در طول اجرای چند خط ابتدای برنامه، هیچ‌گونه استثنا و یا وقفه‌ای رخ نمی‌دهد.

استثنای "داده‌ی غیرعادی[۳]" هنگامی رخ می‌دهد که کنترل کننده‌ی حافظه و یا MMU ، یک دسترسی نادرست به آدرس حافظه را تشخیص می‌دهند. (برای مثال، هنگامی که حافظه‌ی فیزیکی در آدرس مورد نظر موجود نباشد) یا زمانی که کد در حال اجرا، اقدام به خواندن و نوشتن حافظه بدون حقوق دسترسی کافی بنماید. هنگام رخداد یک استثنای *Data Abort*، استثنای *FIQ* نیز می‌تواند اتفاق بیافتد. (زیرا در این حالت *FIQ* غیرفعال نیست). بعد از این‌که *FIQ* به طور کامل سرویس داده شد، کنترل برنامه به روال رسیدگی *Data Abort* برمی‌گردد.

استثنا	تقدم	بیت I	بیت F
Reset	1	1	1
Data Abort	2	1	—
Fast Interrupt Request	3	1	1

[۱] Exception Priorities

[۲] Stack

[۳] Data Abort

—	1	4	Interrupt Request
—	1	5	Prefetch Abort
—	1	6	Software Interrupt
—	1	6	Undefined Instruction

استثنای "درخواست وقفه ی سریع" (*FIQ*) هنگامی رخ می‌دهد که یک وسیله‌ی خارجی پایه‌ی *nFIQ* را به طرز مناسبی تحریک کند. *FIQ* بالاترین سطح وقفه در سیستم است. هنگام رسیدگی به *FIQ*، ورود به استثنای *IRQ* و *FIQ* غیرممکن می‌شود. بنابراین، هیچ وسیله‌ی خارجی نمی‌تواند در کار پردازنده وقفه ایجاد کند، مگر این‌که *IRQ* یا *FIQ* مجدداً توسط نرم‌افزار فعال شده باشند.

استثنای *IRQ* نیز همانند *FIQ* است. با این تفاوت که توسط پایه‌ی خارجی *IRQ* فعال شده و دارای دومین سطح تقدم در بین وقفه‌هاست.

استثنای *Prefetch Abort* هنگامی که درخواستی برای واکشی یک دستورالعمل از حافظه با مشکل مواجه می‌شود، رخ می‌دهد. این استثنا هنگامی رخ می‌دهد که دستورالعمل در مرحله‌ی "اجرا" ی خط لوله باشد و هیچ‌یک از استثناهای بالا رخ نداده باشند.

وقفه‌ی نرم‌افزاری (SWI) هنگامی رخ می‌دهد که دستور SWI اجرا شود. در هنگام ورود به روال رسیدگی کننده، *cpsr* به حالت *Supervisor* تنظیم می‌شود. اگر سیستم مورد نظر از SWI تودرتو استفاده کند، ثبات‌های *r14* (*lr*) و *spsr* باید در حافظه ذخیره شوند، تا از ایجاد مشکل در سیستم جلوگیری به‌عمل بیاید.

اگر یک دستور در مرحله‌ی "اجرا" ی خط لوله به عنوان دستور صحیح ARM و یا Thumb تشخیص داده نشود، یک استثنای *Undefined Instruction* رخ می‌دهد.

هر دو استثنای SWI و *Undefined Instruction* دارای سطح تقدم یکسانی هستند. زیرا این دو هیچ‌گاه به طور هم‌زمان اتفاق نمی‌افتند؛ این سطح تقدم یکسان ایجاد مشکل نمی‌کند.

۴.۱.۴ آفست‌های ثبات لینک

هنگامی که یک استثنا رخ می‌دهد، مقدار ثبات لینک بر مبنای مقدار فعلی PC، به مقدار جدید تنظیم می‌شود. برای مثال، هنگام رخداد استثنای *IRQ*، *lr* به آدرس آخرین دستور اجرا شده به علاوه‌ی ۸ اشاره می‌کند. باید دقت شود که در روال رسیدگی کننده به استثنا، مقدار *lr* دستکاری نشود. زیرا *lr* به آدرس بازگشت از این روال اشاره دارد. روند رسیدگی به وقفه تا بعد از پایان دستور در حال اجرا به تعویق می‌افتد. بنابراین، آدرس بازگشت باید به دستور بعد، *lr* - 4، اشاره کند. جدول ۹.۴، فهرستی از آدرس‌های مفید برای استثناهای مختلف را فراهم آورده است.

سه مثال بعد، روش‌های مختلف بازگشت از *IRQ* و یا *FIQ* را نشان می‌دهد.

استثنا	آدرس	مورد استفاده

***lr* is not defined on a Reset**	—	Reset
points to the instruction that caused the Data Abort exception	*lr* - 8	Data Abort
return address from the FIQ handler	*lr* - 4	FIQ
return address from the IRQ handler	*lr* - 4	IRQ
points to the instruction that caused the Prefetch Abort exception	*lr* - 4	Prefetch Abort
points to the next instruction after the SWI instruction	*lr*	SWI
points to the next instruction after the undefined instruction	*lr*	Undefined Instruction

جدول ۹.۴

مثال ۹.۲

این مثال نشان می‌دهد که روش نمونه‌ی بازگشت از روال رسیدگی کننده به *IRQ* و یا *FIQ* ، استفاده از دستورالعمل SUBS است.

```
handler
       <handler code>
       ...
       SUBS pc, r14, #4 ; pc=r14-4
```

کد ۹.۱

به علت وجود S در انتهای دستورالعمل SUB، وجود PC در ثبات مقصد، *cpsr* به طور خودکار از درون *spsr* کپی می‌شود.

مثال ۹.۳

این مثال روشی دیگر را نشان می‌دهد که در ابتدای روال رسیدگی کننده، مقدار آفست را از ثبات لینک (*r14*) کم می‌کند.

```
handler
       SUB r14, r14, #4 ; r14-=4
       ...
       <handler code>
       ...
       MOVS pc, r14 ; return
```

کد ۹.۲

بعد از اتمام سرویس‌دهی، بازگشت به اجرای طبیعی دستورها با کپی کردن *r14* به PC و بازگردانی محتویات *spsr* به *cpsr* صورت می‌گیرد.

مثال۹.۴

مثال آخر، از پشته‌ی وقفه برای ذخیره سازی ثبات لینک استفاده می‌کند.

```
handler
       SUB r14, r14, #4 ; r14-=4
       STMFD r13!,{r0-r3, r14} ; store context
       ...
       <handler code>
       ...
       LDMFD r13!,{r0-r3, pc}^ ; return
```

کد ۹.۳

برای بازگشت به حالت عادی، از دستور LDM برای بارگیری PC استفاده می‌شود. نماد ^ در انتهای باعث بازگردانی محتویات *spsr* به *cpsr* می‌گردد.

۴.۲ وقفه‌ها

دو نوع وقفه در پردازنده‌ی ARM وجود دارند. اولین نوع آن باعث شروع یک استثنا توسط یک وسیله-ی خارجی می‌شود، یعنی *FIQ* و *IRQ*. نوع دوم با استفاده از دستورالعمل SWI و به طور نرم‌افزاری انجام می‌شود. هر دو نوع وقفه باعث توقف روند اصلی برنامه می‌شوند.
در این بخش بر روی وقفه‌های *FIQ* و *IRQ* تمرکز می‌کنیم.

۴.۲.۱ اختصاص دادن وقفه‌ها

طراح سیستم در مورد این‌که کدام سخت‌افزار جانبی، کدام وقفه را ایجاد می‌کند، باید تصمیم بگیرد. بسته به سیستم تعبیه شده مورد استفاده، پیاده سازی نرم‌افزاری و یا سخت‌افزاری، انتخاب می‌شوند.
پردازنده‌ی ARM دو درخواست وقفه *FIQ* و *IRQ* دارد. یک کنترل کننده‌ی وقفه‌ی مطلوب، تعداد زیادی از منابع وقفه خارجی را به این دو منبع وقفه متصل می‌کند.
در هنگام اختصاص وقفه‌ها، طراحان سیستم مجموعه‌ای از استانداردهای عملی را به شرح زیر استخراج کرده‌اند:

- وقفه‌های نرم‌افزاری برای فراخوانی روال‌های ممتاز[۱] سیستم‌عامل رزرو شده‌اند. برای مثال، دستور SWI در متن برنامه، برای تبدیل حالت کاری کاربر[۲] به حالت کاری ممتاز استفاده می‌شوند؛ شوند؛

[۱] Privileged Routines
[۲] User

- درخواست‌های وقفه عموماً برای وقفه‌های با کاربرد عمومی اختصاص داده می‌شوند. برای مثال، یک زمان سنج تناوبی، وقفه‌ای برای ذخیره‌ی محتویات ثبات‌ها ایجاد می‌کند. به علت این‌که این وظیفه چندان بحرانی نیست، از IRQ به جای FIQ برای آن استفاده می‌کنیم. (که بالتبع تأخیر بیشتری را نیز به همراه دارد.) ؛
- درخواست‌های وقفه‌ی سریع عموماً تنها برای یک خط خاص استفاده می‌شوند که زمان پاسخ‌دهی سریعی را طلب می‌کنند. به عنوان مثال، در مورد DMA که برای انتقال بلوک‌های داده از حافظه را بر عهده دارد. بنابراین، در یک سیستم تعبیه شده، در طراحی سیستم‌عامل، استثنای FIQ برای یک کار خاص استفاده می‌شود، در حالی‌که استثنای IRQ برای سایر فعالیت‌های سیستم‌عامل (که عمومی‌تر هستند) ، استفاده می‌شود.

۴.۲.۲ تأخیر وقفه[۱]

سیستم‌های تعبیه شده باید راه‌حلی اساسی برای یک مشکل اساسی به نام "تأخیر وقفه" داشته باشند. تأخیر وقفه، میزان تأخیر بین دریافت سیگنال وقفه‌ی خارجی و واکشی اولین دستور از روال سرویس‌دهی وقفه است[۲].

تأخیر وقفه بستگی به ترکیبی از سخت‌افزار و نرم‌افزار دارد. بدین معنا که معمار سیستم باید تعادلی در طراحی سیستم برقرار نماید. به نحوی که چندین وقفه‌ی رسیده به طور هم‌زمان را با کمتر میزان تأخیر رسیدگی کند. اگر وقفه‌ها به موقع رسیدگی نشوند، سپس می‌توان زمان پاسخ کُندی از سیستم انتظار داشت.

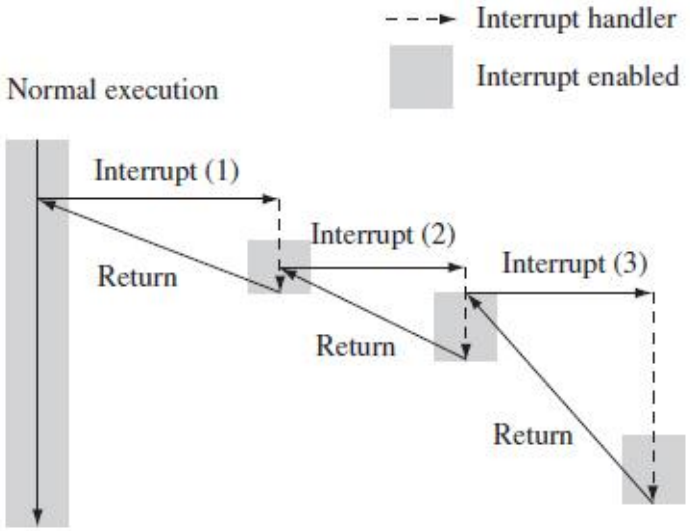
روال‌های رسیدگی کننده‌ی نرم‌افزاری، دو روش برای مینیمم کردن تأخیر وقفه دارند. روش اول استفاده از روال رسیدگی کننده‌ی تودرتو[۳]، (که اگر در حال سرویس‌دهی به یک وقفه باشند، امکان دریافت و سرویس‌دهی به وقفه‌های دیگر را نیز فراهم می‌آورد) در شکل ۴.۳ است. با ایجاد توانایی دریافت وقفه‌ها درست بعد از سرویس‌دهی به وقفه تولید شده و قبل از پایان یافتن روال رسیدگی کننده‌ی وقفه، این امر محقق می‌شود. هنگامی که یک وقفه‌ی تودرتو سرویس داده شد، کنترل سیستم به روال سرویس‌دهنده‌ی اصلی بازمی‌گردد.

---

[۱] Interrupt Latency

[۲] Interrupt Service Routine-ISR

[۳] Nested Interrupt Handler



شکل ۴.۳ – یک وقفه تودرتوی ۳ سطحی

روش دوم، استفاده از "تقدم‌دهی" است. برنامه‌ریزی شما براین اساس است کـه کنتـرل کننـده‌ی وقفـه، وقفه‌های با تقدم یکسان و یا پایین‌تر نسبت به وقفه‌ی فعلی (که در حال سرویس‌دهی می‌باشد) را نادیده می‌گیرد. بنابراین، تنها وقفه‌های باتقدم بالاتر امکان ایجاد وقفه در کار روال سرویس‌دهنده‌ی فعلی‌تان را دارد.

پردازنده، زمان خود را صرف رسیدگی به وقفه‌های باتقدم پایین می‌کند. این امر تا زمانی ادامه دارد کـه وقفه‌ای باتقدم بالاتر فرارسد. بنابراین، وقفه‌های باتقدم بالاتر دارای میانگین " تأخیر وقفه‌ی" پایین‌تـری نسبت به وقفه‌های باتقدم پایین‌تر هستند.

۴.۲.۳ استثناهای *IRQ* و *FIQ*

استثناهای *IRQ* و *FIQ* ، هنگامی رخ می‌دهند که ماسک وقفه‌ی خاصی در *cpsr* پاک شده باشد.

از آن‌جایی که پردازنده‌ی ARM قبلاً از پریـدن بـه روال رسـیدگی وقفـه، دسـتور فعلـی را در مرحلـه‌ی "اجرا" ی خط لوله کامل می‌کند، یک عامل مهم در طراحی وقفه‌های با پاسـخ‌دهـی بـالا، بررسـی میـزان تأخیر اجرای هر دستورالعمل، در خط لوله است. (بعضی از دستورها در مرحله‌ی "اجـرا" ی خـط لولـه زمان بیشتری را مصرف می‌کنند.)

با فرض این‌که وقفه‌ها ماسک نشده‌اند، وقفه‌های *FIQ* و *IRQ* پردازنده‌ها را وادار به طی نمـودن رونـدی استاندارد می‌کنند:

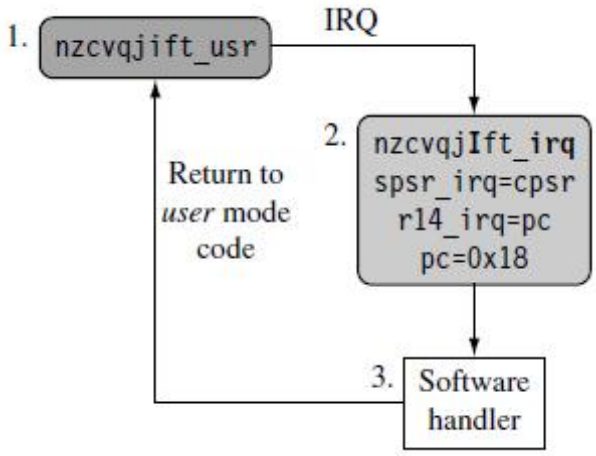
۱. پردازنده به حالت خاصی از درخواست وقفه، تغییر حالت می‌دهد. این حالت بستگی بـه منبـع تولیـد کننده‌ی وقفه دارد؛

۲. *cpsr* ، مربوط به حالت قبلی به درون *cpsr* حالت درخواست وقفه‌ی جدید، ذخیره می‌شود؛
۳. PC ، در درون *lr* حالت درخواست وقفه‌ی جدید، ذخیره می‌شود؛
۴. وقفه (یا وقفه‌ها) ناتوان می‌شوند. استثنای *IRQ* و یا هر دو استثنای *IRQ* و *FIQ* در *cpsr* ناتوان می‌شوند. این کار از ایجاد وقفه توسط منابع یکسان جلوگیری به عمل می‌آورد؛
۵. پردازنده به ورودی خاصی از جدول بردار انشعاب می‌زند.

روند ارایه شده بسته به نوع وقفه‌ی تولید شده، کمی متفاوت است. هر دو وقفه‌ها را در زیر با مثالی تشریح خواهیم کرد. در مثال اول، اتفاقاتی که هنگام رخداد *IRQ* رقم می‌خورند و در مثال دوم همین اتفاقات را هنگام رخداد *FIQ* بررسی خواهیم کرد.

مثال ۹.۵

شکل ۴.۴ نشان می‌دهد که هنگام رخداد استثنای *IRQ* (وقتی پردازنده در حالت "کاربر" است) چه اتفاقاتی می‌افتد. پردازنده از حالت ۱ شروع می‌کند. در این مثال، بیت‌های هر دو استثنای *IRQ* و *FIQ* در *cpsr* توانا شده‌اند. هنگامی که یک *IRQ* رخ می‌دهد، پردازنده به حالت ۲ حرکت می‌کند. این گذر، بیت *IRQ* را برابر '1' می‌کند. که خود به معنی ناتوان سازی هرگونه استثنای بعدی است. اگرچه *FIQ* هنوز توانایی تولید وقفه را دارد (این به دلیل تقدم بیشترش نسبت به *IRQ* است). حالت پردازنده‌ی *cpsr* به حالت *IRQ* تغییر می‌کند. *cpsr* حالت *User* به طور خودکار به درون *spsr_irq* کپی شده است.



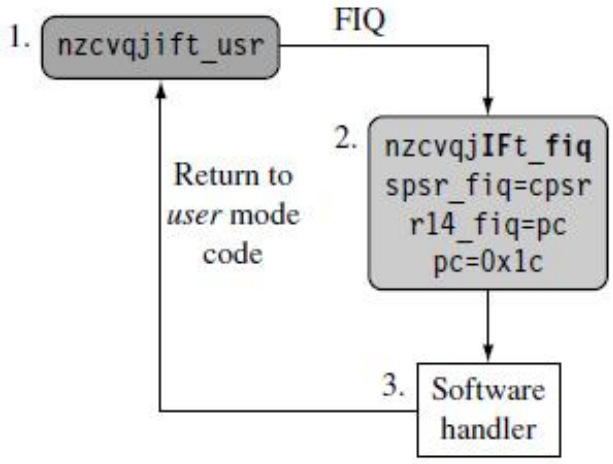
شکل ۴.۴ – درخواست وقفه (IRQ)

هنگام شروع وقفه، *r14_irq* حاوی مقدار PC می‌شود. سپس PC ، برابر آدرس ورودی IRQ + 0×18 در جدول وقفه می‌شود.

در مرحله‌ی ۳، رسیدگی کننده نرم‌افزاری کنترل را در دست گرفته و روال سرویس وقفه‌ی مناسب را برای سرویس‌دهی به منبع وقفه، فراخوانی می‌کند. بعد از کامل شدن عملیات، حالت پردازنده به حالت اصلی *User* در شماره‌ی ۱ بازمی‌گردد.

مثال ۹.۶

شکل ۴.۵ مثالی از استثنای *FIQ* را نشان می‌دهد. فرآیند اجرایی در این‌جا همانند آن‌چه در مثال بالا دیدیم، است. با این تفاوت که علاوه بر *IRQ*، جلوی رخداد تمام وقفه‌های *FIQ* دیگر را نیز می‌گیرد. یعنی، در حالت ۳، هر دو وقفه *IRQ* و *FIQ* غیرفعال هستند.



شکل ۴.۵ – درخواست وقفه سریع (FIQ)

تغییر حالت به *FIQ*، بدین معنا است که احتیاجی به ذخیره سازی ثبات‌های *r8* تا *r12* نیست (زیرا در این حالت *FIQ*، ثبات‌های مذکور بانک شده‌اند). این ثبات‌ها را می‌توان برای نگه‌داری داده‌های لحظه‌ای از قبیل شمارنده‌ها، اشاره‌گر بافر و... استفاده کرد.

این مسایل، *FIQ* را برای سرویس‌دهی به یک وقفه‌ی با تأخیر کم، تقدم بالا و نشأت گرفته از تنها یک منبع، مناسب می‌سازد.

۴.۲.۳.۱ فعال یا غیرفعال ساختن استثناهای *FIQ* و *IRQ*

پردازنده‌ی ARM ، روندی ساده برای فعال یا غیرفعال ساختن وقفه‌ها دارد. این روند شامل *cpsr*، وقتی پردازنده در حالت ممتازاست، می‌باشد.

جدول ۹.۵ چگونگی فعال شدن وقفه‌های *FIQ* و *IRQ* را نشان می‌دهد. این روند شامل سه دستور ARM می‌شود.

اولین دستور، MRS، محتویات *cpsr* را به درون ثبات *r1* کپی می‌کند. دستور دوم، بیت ماسک *FIQ* یا *IRQ* را پاک می‌کند. دستورسوم، محتویات به‌روز شده‌ی ثبات *r1* را مجدداً به درون *cpsr* کپی می‌کند (درخواست وقفه را توانا می‌سازد). پسوند "C-" محتویات کنترلی بیت‌های *cpsr* (یعنی *cpsr*[7:0]) به-

روز رسانی می‌شوند. جدول ۹.۶، روندی مشابه برای ناتوان سازی و یا ماسک کردن وقفه را نشان می-دهد.

نکته

دانستن این نکته مهم است که فقط هنگامی درخواست وقفه فعال و یا غیرفعال می‌شود که MSR در خط لوله، مرحله‌ی اجرایی‌اش را به پایان برساند.

مقدار cpsr	IRQ	FIQ
Pre	*nzcvqj**I**Ft_SVC*	***nzcvqjIFt_SVC***
Code	enable_irq MRS r1, cpsr BIC r1, r1, #0x80 MSR cpsr_c, r1	enable_fiq MRS r1, cpsr BIC r1, r1, #0x40 MSR cpsr_c, r1
Post	*nzcvqj**i**Ft_SVC*	***nzcvqjIft_SVC***

جدول ۹.۵

مقدار cpsr	IRQ	FIQ
Pre	*nzcvqj**if**t_SVC*	***nzcvqjift_SVC***
Code	disable_irq MRS r1, cpsr BIC r1, r1, #0x80 MSR cpsr_c, r1	disable_fiq MRS r1, cpsr BIC r1, r1, #0x40 MSR cpsr_c, r1
Post	*nzcvqj**If**t_SVC*	***nzcvqjiFt_SVC***

جدول ۹.۶

برای توانا ساختن و یا ناتوان کردن هر دو استثنای *FIQ* و *IRQ*، دستور دوم را باید کمی تغییر دهیم. مقدار عددی در دستورهای پردازش داده BIC یا ORR برای توانا سازی و یا ناتوان سازی هر دو وقفه باید با عدد 0×C0 جایگزین شود.

۴.۲.۴ طراحی و پیاده سازی پشته‌ی وقفه‌ی پایه

روال‌های رسیدگی بـه وقفـه، اسـتفاده‌ی گسـترده‌ای از پشـته مـی‌کننـد. هـر حالـت اسـتثنا دارای ثبـاتی مخصوص برای اشاره‌گر پشته می‌باشند. طراحی پشته‌های استثنا بستگی به عوامل زیر دارند:

- احتیاجات سیستم‌عامل_ هر سیستم‌عامل نیاز خاص خود را برای طراحی پشته دارد؛
- سخت‌افزار_ سخت‌افزار هدف بر روی حداکثر اندازه‌ی پشته و مکان قرارگیری آن محـدودیت‌هـایی اعمال می‌کند.

در طراحی پشته دو تصمیم باید گرفته شود:

- مکان قرارگیری_ مشخص می‌کند که از کجـای حافظـه پشـته شـروع بـه افـزایش مـی‌کنـد. بیشـتر سیستم‌های بر مبنای ARM از پشته‌ای استفاده می‌کنند که به سمت پایین گسترش مـی‌یابـد. بـالای پشته در آدرس حافظه‌ی بالاتری قرار دارد؛
- اندازه‌ی پشته بستگی به نوع روال رسیدگی کننده (تودرتو و یا ساده) دارد. سـاختار پیـاده سـازی وقفه‌های تودرتو، به حافظه‌ی بیشتری نیاز دارند.

طراحی خوب برای پشته نیازمند این‌است که هیچ وقت با پیغـام خطـای سـرریز وقفـه[۱] روبـه‌رو نشـویم (یعنی وقتی که پشته پیش ازحافظه‌ی تخصیص یافته‌اش گسترش یابد). این خطا باعث ناپایداری در روند برنامه و یا سیستم‌عامل مورد استفاده می‌شود. تعداد تکنیک‌های نرم‌افزاری برای فهمیدن زمان سـرریز شدن وقفه و همچنین روش‌هایی برای رسیدگی و جبران این حالت سرریز وجـود دارد کـه مـی‌تـوان از آن‌ها بهره برد.

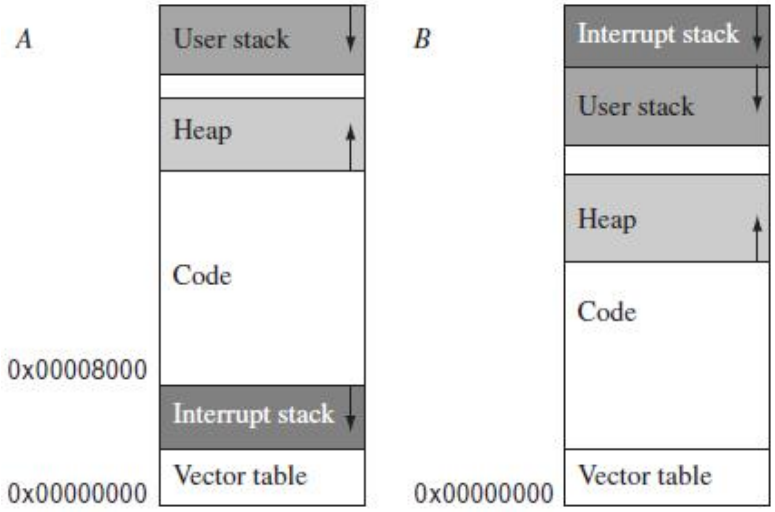
دو روش اصلی عبارتنـد از ۱) اسـتفاده از واحـد محافظـت از حافظـه و ۲) فراخـوانی یـک زیربرنامـه‌ی "بررسی کننده‌ی سرریز پشته" در ابتدای هر روال سرویس‌دهنده‌ی وقفه.

قبل از فعال سازی وقفه‌ها، پشته‌ی حالت *IRQ* باید تنظیم شود (معمـولاً در برنامـه‌ی مقـداردهی اولیـه‌ی سیستم). دانستن اندازه‌ی اشغال شونده حافظه برای پشته، مسأله‌ی مهمی اسـت زیـرا در همـین مراحـل اولیه‌ی مقداردهی، اندازه‌ی پشته مورد نیاز اختصاص داده می‌شود.

شکل ۴.۶، دو نوع نقشه‌ی حافظه نمونه در فضایی خطی را نشان مـی‌دهـد. اولـین مـورد A، نمونـه‌ای از طراحی قدیمی پشته را که در آن پشته، در فضای زیرین سگمنت کد قـرار مـی‌گیـرد را نشـان مـی‌دهـد. نقشه‌ی دوم B ، قرارگیری پشته‌ی وقفه در بالای حافظه بعد از پشته‌ی کاربر را نشان می‌دهد. مزیـت B بر A ، این‌است که هنگام وقوع سرریز پشته، B جدول بـردار را دچـار اخـتلال نمـی‌کنـد. پـس، سیسـتم شانس تصحیح خود را در صورت وقوع چنین حالتی دارد.

---

[۱] Stack Overflow



شکل ۴.۶ – یک نقشه‌ی حافظه‌ی نمونه

مثال ۹.۷

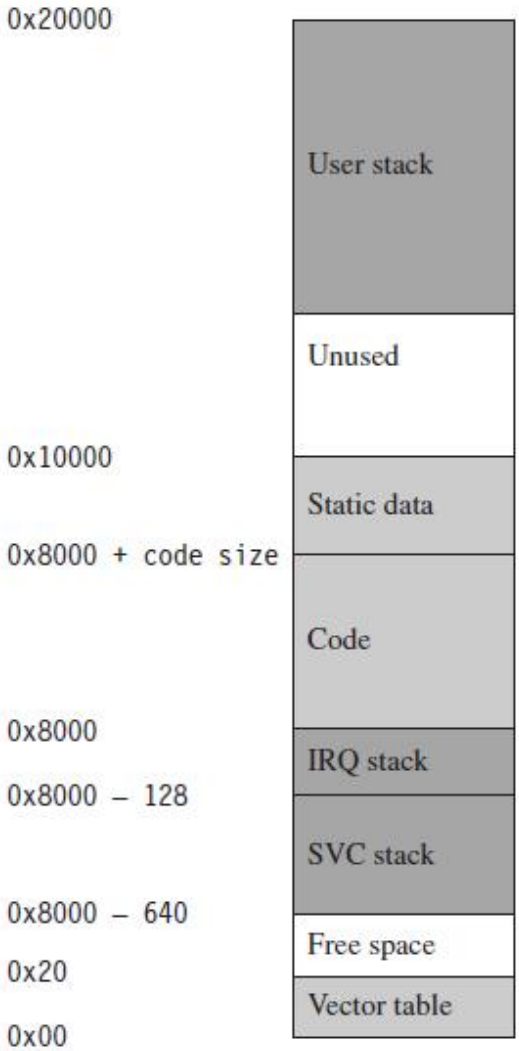
برای هر حالت پردازنده، پشته‌ای باید اختصاص یابد. همواره بعد از ریست شدن پردازنده، این عمل انجام می‌شود. شکل ۴.۷، یک نوع پیاده سازی با استفاده از نقشه‌ی A را نشان می‌دهد. برای راحتی در انجام تخصیص حافظه، مجموعه‌ای از تعاریف در ابتدا قرار داده شده‌اند که نواحی مختلف حافظه را با آدرس‌های مطلق نام‌گذاری می‌کنند.

برای مثال، پشته‌ی کاربر برچسب USR-Stack را داشته و دارای آدرس 0×20000 می‌باشد. پشته‌ی *Supervisor* دارای آدرسی معادل ۱۲۸ عدد کمتر از پشته‌ی *IRQ* است.

```
USR_Stack EQU 0x20000
IRQ_Stack EQU 0x8000
SVC_Stack EQU IRQ_Stack-128
```

برای راحتی در امر تغییر حالات پردازنده، ما هر بیت مربوط به حالت کاری مختلف پردازنده را نیز توسط اسم مفهوم‌تری تعریف می‌کنیم. این برچسب‌ها می‌توانند برای تغییر *cpsr* استفاده شوند.

```
Usr32md EQU 0x10 ; User mode
FIQ32md EQU 0x11 ; FIQ mode
IRQ32md EQU 0x12 ; IRQ mode
SVC32md EQU 0x13 ; Supervisor mode
Abt32md EQU 0x17 ; Abort mode
Und32md EQU 0x1b ; Undefined instruction mode
Sys32md EQU 0x1f ; System mode
```



شکل ۴.۷ – نمونه‌ی پیاده‌سازی با استفاده از نقشه‌ی A

برای امنیت بیشتر، یک دستور define نیز برای ناتوان سازی *IRQ* و *FIQ* در *cpsr*، تعریف شده است.

```
NoInt EQU 0xc0 ; Disable interrupts
```

No Int ، هر دو وقفه را با '1' کردن بیت‌های متناظر، ماسک می‌کند.

کد مقداردهی اولیه با تعیین ثبات‌های پشته برای هر حالت پردازنده آغاز می‌شود. ثبات پشته‌ی r13 یکی ثبات‌هایی است که به هنگام تغییر حالت، همواره بانک شده است. کد مورد نظر ابتدا پشته را مقداردهی اولیه می‌کند. برای ایمنی بیشتر، بهتر است در ابتدای برنامه اطمینان حاصل شود که چگونه وقفه‌ای رخ نخواهد داد. با OR کردن No Int و حالت جدید، این اطمینان حاصل خواهد شد.

پشته‌ی هر حالت باید تنظیم شود. در این‌جا مثالی برای تنظیم و برپایی سه پشته‌ی مختلف، بعد از ریست شدن پردازنده را با هم می‌بینیم.

- پشته‌ی حالت *Supervisor*ـ هسته‌ی پردازنده در حالت *Supervisor* شروع به کار می‌کند. پس تنظیم پشته‌ی حالت *Supervisor* شامل بارگیری ثبات *r13_SVC* با آدرس اشاره شده توسط SVC_New Stack می‌باشد. برای این مثال، مقدار مورد نظر SVC_Stack است؛

```
        LDR r13, SVC_NewStack ; r13_svc
        ...
SVC_NewStack
        DCD SVC_Stack
```

- پشته‌ی حالت *IRQ*ـ برای تنظیم پشته‌ی *IRQ*، حالت پردازنده باید به حالت *IRQ* تغییر یابد. این کار با کپی کردن الگوی بیت‌های مربوط به *cpsr* به درون *r2* و سپس کپی کردن *r2* به *cpsr* انجام می‌شود. این عملیات، بلافاصله ثبات *r13_irq* را در دسترس قرار می‌دهد که می‌توانیم مقدار IRQ_NewStack را به آن بدهیم؛

```
        MOV r2, #NoInt|IRQ32md
        MSR cpsr_c, r2
        LDR r13, IRQ_NewStack ; r13_irq
        ...
IRQ_NewStack
        DCD IRQ_Stack
```

- پشته‌ی حالت کاربردـ روش عمومی تنظیم پشته حالت کاربرد، تنظیم آن در مرحله‌ی آخر است. زیرا در حالت کاربرد، روش مستقیمی برای تنظیم مقدار *cpsr* وجود ندارد. راه‌حل جایگزین قرار دادن حالت پردازنده، در حالت کاری *System* است. زیرا این دو حالت با یکدیگر از مجموعه ثبات‌های یکسانی استفاده می‌کنند.

```
        MOV r2, #Sys32md
        MSR cpsr_c, r2
        LDR r13, USR_NewStack ; r13_usr
        ...
USR_NewStack
        DCD USR_Stack
```

استفاده از پشته‌ی جداگانه برای هر حالت کاری به جای استفاده از تنها یک پشته‌ی مشترک، این مزیت اصلی را به دنبال دارد؛ زیرا برنامه‌ها دارای خط از تمام سیستم ایزوله هستند و می‌توانند به صورت جداگانه از تمام سیستم، عیب‌یابی شوند.

۴.۳ روش‌های مختلف رسیدگی به وقفه

در این بخش نهایی، درمورد روش‌های رسیدگی به وقفه بحث خواهیم کرد. روش‌هایی که بررسی خواهیم کرد از این قرارند:

- روال رسیدگی کننده‌ی ساده (غیر تودرتو) که وقفه‌ها را به صورت پشت سر هم سرویس‌دهی می‌کند؛
- روال رسیدگی کننده‌ی تودرتو که چندین وقفه را بدون درنظر گرفتن تقدم، رسیدگی خواهد کرد؛
- روال رسیدگی کننده‌ی بازگشتی[۱] چندین روال وقفه را با رعایت حق تقدم، رسیدگی می‌کند.

۴.۳.۱ رسیدگی کننده به وقفه‌های غیر تودرتو[۲]

ساده‌ترین وقفه، وقفه‌ای است که "تودرتو" نباشد (در این سیستم، هنگام رسیدگی به وقفه، وقفه‌های دیگر غیرفعال هستند. این موضوع تا زمانی که رسیدگی به وقفه‌ی جاری کامل شود، ادامه دارد). از آن‌جایی که هنگام رسیدگی به این نوع وقفه، وقفه‌های دیگر غیرفعال هستند، این نوع سیستم‌ها کاربردهای گسترده‌ای ندارند. به خصوص در وسایلی که نیاز به چندین وقفه با سطوح تقدم مختلف است.

شکل ۴.۸، وقایعی را که هنگام رخداد این نوع وقفه اتفاق می‌افتد و شیوه‌ی رسیدگی به آن را شرح می‌دهد:

۱. غیرفعال کردن وقفه (یا وقفه‌ها)ـ هنگام رخداد یک وقفه‌ی *IRQ*، پردازنده‌ی ARM از رخداد هرگونه وقفه‌ی IRQ دیگر جلوگیری می‌کند. حالت پردازنده به مقدار جدید عوض شده و *cpsr* مربوط به حالت قبلی به *spsr* حالت جدید کپی می‌شود. سپس، پردازنده، PC را برای اشاره به یکی از ورودی‌های جدول بردار تنظیم کرده و شروع به اجرای دستورها می‌کند. این دستور، PC را برای اشاره به روال رسیدگی کننده‌ی وقفه‌ی مورد نظر، به مقدار جدیدی تغییر می‌دهد؛

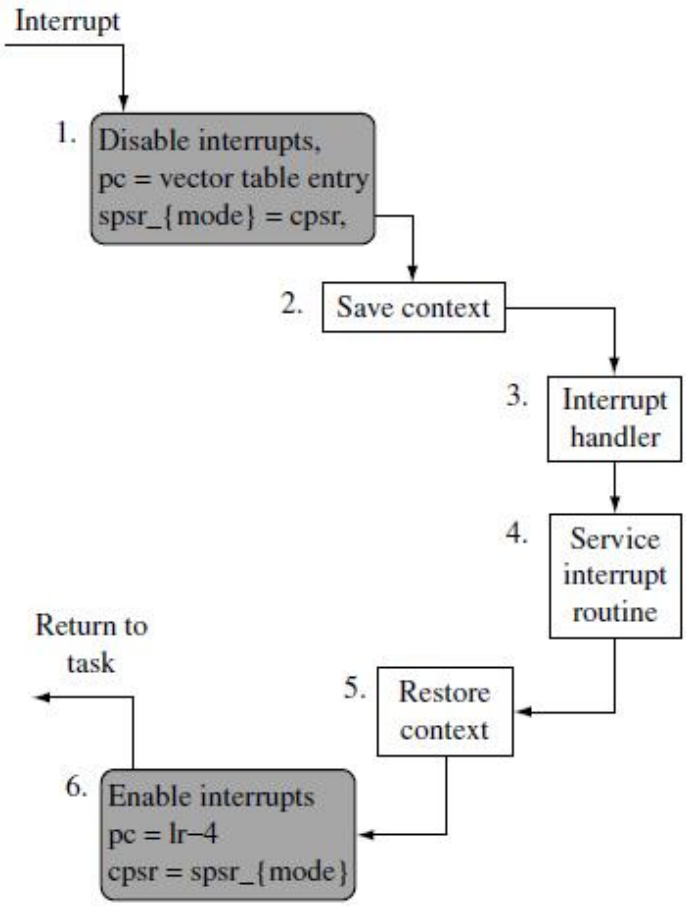
۲. ذخیره وضعیت فعلی و ثبات‌ها[۳]ـ به هنگام ورود، کد روال رسیدگی کننده، زیرمجموعه‌ای از ثبات‌های بانک نشده‌ی حالت فعلی پردازنده را ذخیره می‌کند؛

---

[۱] Reentrant

[۲] Non nested

[۳] Context



شکل ۴.۸ – رسیدگی کننده‌ی ساده به وقفه‌ی غیر تودرتو

۳. رسیدگی کننده به وقفه- روال رسیدگی کننده، منبع وقفه‌ی خارجی را مشخص کرده و ISR مناسب را اجرا می‌کند؛
۴. روال سرویس دهنده‌ی وقفه (ISR)- ISR به منبع وقفه‌ی خارجی سرویس داده و وقفه را ریست می‌کند؛
۵. بازگردانی وضعیت ثبات‌ها- ISR به روال رسیدگی کننده بازگشته و مقادیر ثبات‌ها را بازمی-گرداند؛
۶. فعال سازی وقفه‌ها- نهایتاً برای بازگشت از روال رسیدگی کننده، *spsr* حالت فعلی به درون *cpsr* بازگردانده می‌شود. سپس، PC به مقدار آدرس دستور بعد از دستوری که وقفه ایجاد شد، تنظیم می‌شود.

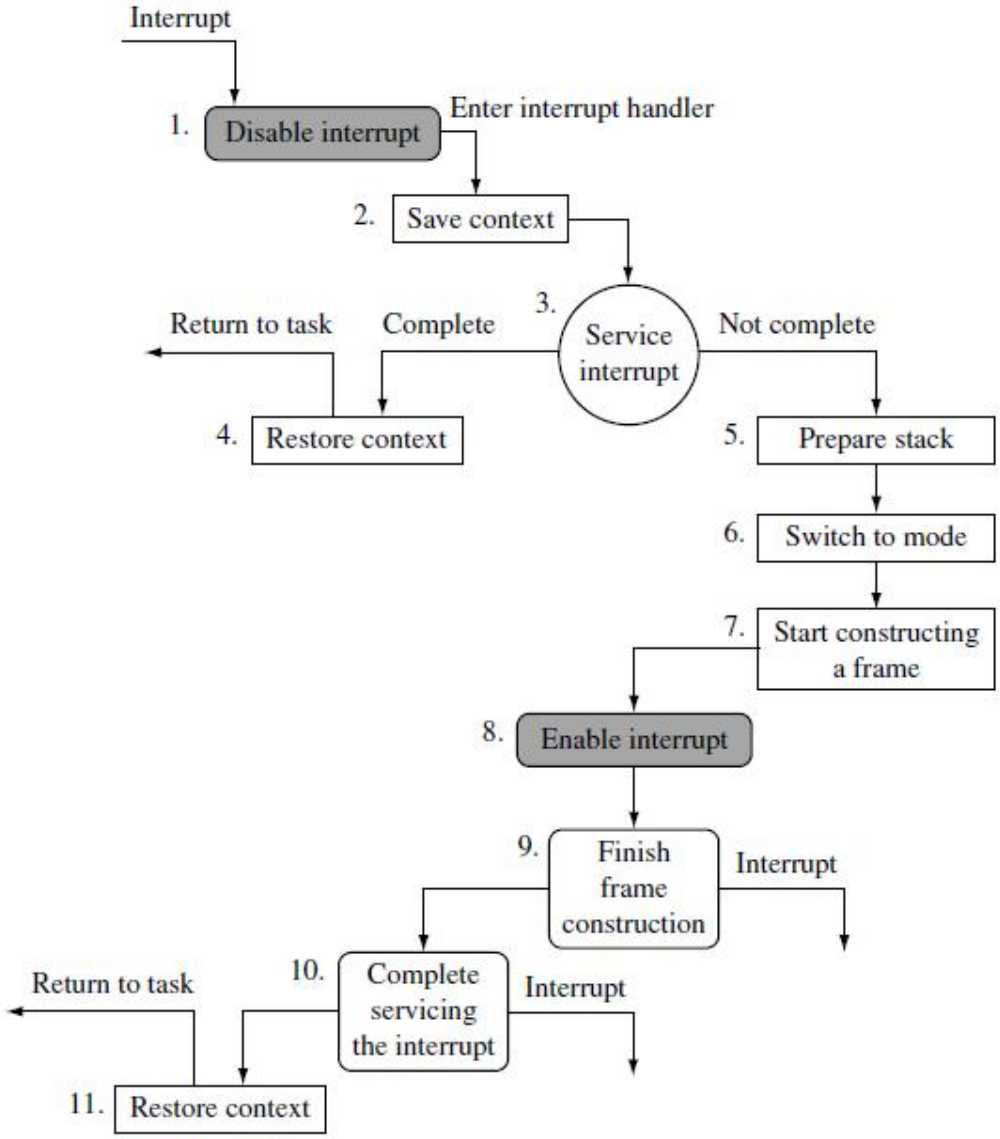
مشخصات این نوع رسیدگی به وقفه

- به وقفه‌های مجزا به صورت جداگانه و پشت سر هم رسیدگی می‌کند؛
- تأخیر در رسیدگی به وقفه‌ها در این‌جا زیاد است؛
- مزایا: پیاده سازی و عیب‌یابی آسان؛
- معایب: برای پیاده سازی سیستم‌های پیچیده با چندین سطح تقدم وقفه مناسب نیستند.

۴.۳.۲ رسیدگی کننده به وقفه‌های تودرتو[۱]

وقفه‌های تودرتو، این امکان را در اختیار می‌گذارند که در روال رسیدگی کننده، وقفه‌ی دیگری بتواند رخ بدهد. این کار از طریق فعال نمودن وقفه قبل از تکمیل ISR، ممکن می‌شود. پیچیدگی یک سیستم بی-درنگ از این طریق افزایش می‌یابد، ولی در عین حال کارآیی بیشتری به دست خواهد آمد. این پیچیدگی امکان ایجاد مشکلات ظریفی را افزایش می‌دهد. پس، در طراحی یک چنین سیستم‌هایی باید دقت کافی را مبذول داشت تا با مشکلات جدی روبه‌رو نشویم. این کار با محافظت از محتویات ثبات‌های مهم و ذخیره‌ی به موقع آن‌ها، امکان‌پذیر می‌شود. نکته‌ی مهم در این‌جاست که هنگام ذخیره سازی این ثبات-های مهم، نباید هیچ‌گونه وقفه‌ای رخ دهد.
وقفه‌ی سرریز پشته نیز مشکل ساز است.
اولین هدف روال‌های رسیدگی کننده‌ی تودرتو، پاسخ گویی سریع به وقفه‌هاست. هدف دوم این‌است که اجرای طبیعی دستورها، هنگام پاسخ گویی به وقفه‌ها، به تأخیر نیافتد.
افزایش پیچیدگی در این سیستم به این معنا است که طراحان باید بین بازدهی و ایمنی، تعادل برقرار کنند تا کمترین مشکلات را در اجرای سیستم داشته باشند.

[۱] Nested



شکل ۴.۹ – رسیدگی کننده به وقفه‌ی تودرتو

شکل ۴.۹ یک روال رسیدگی کننده‌ی تودرتو را نشان می‌دهد. پیچیدگی این روال، نسبت به نوع غیرتودرتوی بخش قبل، کاملاً آشکار است. کد ورودی روال رسیدگی کننده‌ی تودرتو همانند نمونه‌ی غیرتودرتوی آن است. مگر در هنگام خروج که روال رسیدگی کننده باید بررسی کند که وقفه‌ها را مجدداً فعال سازد یا نه.

مشخصات روش رسیدگی به وقفه‌ی تودرتو:

- به چندین وقفه، بدون درنظر گرفتن تقدم، می‌تواند رسیدگی کند؛
- میزان تأخیر در رسیدگی به وقفه در حد متوسط تا زیاد؛
- مزیت: می‌تواند وقفه‌ها را قبل از اتمام سرویس به یک وقفه‌ی منفرد، فعال کند که کاهش تأخیر وقفه را به همراه دارد؛
- عیب: به تقدم وقفه‌ها اهمیت نمی‌دهد. بنابراین، وقفه‌های باتقدم پایین، به راحتی کار وقفه‌های باتقدم بالا را مختل خواهند کرد.

بخش ۳

میکروکنترولرهای ARM و وسایل جانبی

فصل پنجم

میکروکنترولرهای مبتنی بر ARM

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ میکروکنترولرهای ARM چگونه به وجود آمدند ؟
- ✓ انواع مختلف میکروکنترولرهای ARM؛
- ✓ سازندگان مختلف و خانواده‌های تولیدی توسط آنان؛
- ✓ بررسی سایر مشخصات داخلی میکروکنترولرهای ARM .

امروزه، میکروکنترولرهای مختلفی در سراسر دنیا با استفاده از هسته ARM ساخته می‌شوند. به ادعای سازندگان، سالانه بیش از میلیون‌ها عدد از این تراشه‌ها به فروش می‌رسد. استقبال گسترده و روزافزون از ARM ، مرهون مشخصات تحسین برانگیز آن‌ها در ساخت دستگاه‌های قابل حمل است. همین مسأله، شرکت‌های سازنده را ترغیب می‌کند که مدل‌های بیشتری را روانه‌ی بازار کنند. شرکت‌های معروف سازنده‌ی نیمه هادی‌ها، قطعات و ادوات سیلیکونی همانند Atmel، NXP(Philips)، ST Microelectronics، Texas Instrument، Qualcomm و... ، از بزرگ‌ترین سازندگان میکروکنترولر بر مبنای پردازنده‌ی ARM هستند. این شرکت‌ها با **خرید مجوز ساخت تراشه** از شرکت ARM و قرار دادن وسایل جانبی ضروری در کنار تراشه، اقدام به ساخت **میکروکنترولرهای ARM** نموده‌اند.

در این فصل، گذاری مفصل و در عین حال ساده بر انواع مختلف میکروکنترولرهای ARM ساخته شده توسط شرکت‌های نام‌برده خواهیم داشت. این میکروکنترولرها به راحتی در بازارهای الکترونیک یافت می‌شوند. از جمله علل دیگر پرکاربرد شدن این میکروکنترولرهای ۳۲ بیتی، قیمت‌های قابل رقابت آن‌ها در مقایسه با میکروکنترولرهای قدیمی‌تر ۸ و یا ۱۶ بیتی در عین قابلیت‌های زیادشان است.

۵.۱ میکروکنترولرهای ARM شرکت Atmel با نام‌گذاری AT91

شرکت Atmel ، ارایه کننده میکروکنترولرها و دیگر مدارات مجتمع سیلیکونی برای چندین دهه‌ی گذشته بوده است. از میکروکنترولرهای ۸ بیتی خانواده AT89C5X و

AT89S5X گرفته تا میکروکنترولرهای ۸ بیتی موفق AVR (البته این موفقیت نظر نویسنده و طیف وسیعی از کاربران در حوزه کاربری و بازار فروش است) تولیدات این شرکت فقط به میکروکنترولرها محدود نبوده و همانند دیگر سازندگان مطرح نیمه هادی در سراسر جهان، اقدام به ساختن تراشه‌هایی چون حافظه‌ی Flash و E^2PROM، حافظه‌ی RAM، تراشه‌های فرکانس بالا، تراشه‌هایی مخصوص اتوماسیون و... نموده است. با پیشرفت سریع تکنولوژی در چند سال اخیر، این شرکت اقدام به ساخت میکروکنترولرهای ARM نمود. میکروکنترولرهای قدرت‌مند ۳۲ بیتی به همراه وسایل جانبی متنوع و کارآمد همانند پورت USB، راه‌انداز شبکه‌ی Ethernet ، راه‌انداز LCD و بسیاری از مدارهای حرفه‌ای دیگر که در جای خود مورد بررسی قرار می‌گیرند.

این میکروکنترولرها به راحتی امکان راه‌اندازی یک سامانه‌ی دستی دیجیتال را دارند. به عنوان مثال، یک سامانه‌ی دستی که قابلیت اتصال به اینترنت را داشته باشد، به همراه نمایشگرهای LCD و توانایی‌های بسیار دیگر را درنظر بگیرید که با باتری تغذیه شده و به راحتی در دست جای می‌گیرد.

Atmel ، در حال حاضر میکروکنترولرهایی با استفاده از هسته‌های ARM7، ARM9 و Cortex-M3 تولید می‌کند. در نام‌گذاری این قطعات از روش خاصی استفاده می‌کند. در زیر، این شیوه نام‌گذاری را می‌بینیم.

شرکت Atmel ، در نام‌گذاری میکروکنترولرهای ARM از پیشوند AT91 استفاده می‌کند. این نام-گذاری شبیه پیشوند AT90 است که در AVR ها استفاده می‌شد.

همان‌طور که در شکل می‌بینیم، میکروکنترولرهای با هسته ARM7TDMI را با پیشوند AT91SAM7، میکروکنترولرهای با هسته‌ی ARM9 را با پیشوند AT91SAM9 و سری جدیدتر میکروکنترولرهایش از خانواده Cortex-M3 را با پیشوند AT91SAM3 می‌شناسد.

در حال حاضر، این شرکت توسعه‌ی مسیر میکروکنترولرهایش در همین سه خانواده را مدنظر دارد. هم اکنون ARM9 های استفاده شده بر روی تراشه‌های AT91SAM9G و AT91SAM9M، آن‌ها را مبدل به پیشرفته‌ترین تراشه‌های این خانواده با امکانات فراوان و سرعت کلاک بالاتری نموده است. در گروه Cortex-M3 نیز تراشه‌هایی به همراه امکانات جانبی جدید همانند USB سرعت بالا (High Speed-480 Mbps) در حالت‌های کاری Device و Host و شبکه اترنت و... را مورد هدف قرار داده است.

جدول ۴.۱ نام‌گذاری میکروکنترولرهای AT91

AT91	SAM	7X	256	-AU		
				-AU	بسته بندی LQFP100	محدوده دمایی -40°C...85°C
				-AC	بسته بندی TFBGA100	
			128			
			256			
			512			

		7S
		7X
	SAM	
	CAP	
AT91		

AT91	SAM	926	0	-AU	
				-QU	بسته بندی PQ208 محدوده دمایی
				-CU	بسته بندی BGA217 -40°C...85°C
			0		
			1		
			3		
		926			
	SAM				
AT91					

AT91SAM		9G	20	-AU	
				-QU	بسته بندی PQ208 محدوده دمایی
				-CU	بسته بندی BGA217 -40°C...85°C
			10		
			20		
			45		
			46		
		9G			
AT91SAM					

AT91SAM		9M	10	-AU	
				-QU	بسته بندی PQ208 محدوده دمایی
				-CU	بسته بندی BGA217 -40°C...85°C

		10		
		11		
	9M			
AT91SAM				

AT91SAM		9R	10	-AU	
				-QU	بسته بندی PQ208 محدوده دمایی -40°C...85°C
				-CU	بسته بندی BGA217
			64		
		9R			
		9RL			
AT91SAM					

AT91SAM		9XE	256	-AU	
				-AU	بسته بندی LQFP100 محدوده دمایی -40°C...85°C
				-AC	بسته بندی TFBGA100
			128		
			256		
			512		
		9XE			
AT91SAM					

۵.۲ شرکت NXP و میکروکنترولرهای سری LPC

شـــرکت فیلیـــپس، شـــرکت NXP [۱] را در 31 آگوسـت سـال 2006 تأسـیس نمـود و تمـام تولیدات نیمـه هـادی خـود را بـه شـرکت جدیـد واگذار کرد. NXP از بزرگ‌ترین تولیـد کننـدگان مدارات آنالوگ و دیجیتال (Mixed Signal) است. این شرکت، محصولاتی در زمینه مدارهای مخابراتی

[۱] Next eXPerience

(RF)، آنالوگ، مدیریت تغذیه، حوزه امنیت و پردازش دیجیتال ارایه کرده است. تراشه‌های تولیدی این شرکت در حوزه‌های وسیعی از قبیل اتوماسیون، شناسایی، ارتباطات بی‌سیم، نورپردازی، کاربردهای صنعتی و پزشکی، صنعت موبایل و... کاربرد دارند. دفتر مرکزی این شرکت در هلند بوده و دارای ۲۸۰۰۰ نیروی کار فعال در ۲۵ کشور جهان می‌باشد. فروش سالانه ثبت‌شده‌ی شرکت در سال 2009 به میزان 3,8 بیلیون دلار آمریکا بوده است.

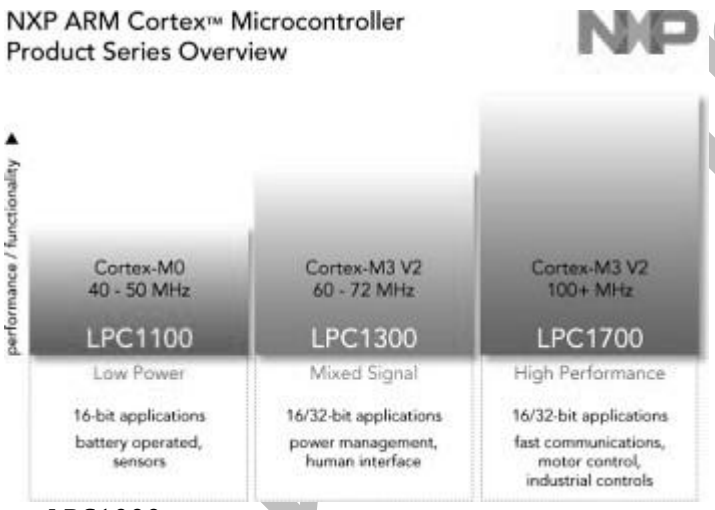
شرکت NXP ، هم‌اکنون از تولید کنندگان اصلی میکروکنترولرهای ARM به حساب می‌آید. میکروکنترولرهای تولیدی این شرکت طیف وسیعی از هسته‌های ARM از قبیل ARM7TDMI-S، ARM9 و CortexM0/ M1/ M3/ M4 را شامل می‌شود.
میکروکنترولرهای ARM شرکت NXP ، در خانواده‌های LPC1000، LPC2000، LPC3000 و LPC4000 تولید می‌شوند. این میکروکنترولرها طیف گسترده‌ای از ادوات جانبی از قبیل ADC، DAC، USB، Ethernet، راه‌اندازهای LCD، کنترل کننده‌های حافظه و... را شامل می‌شوند.
این تراشه‌ها، کارآیی خوبی از خود نشان داده‌اند و در بسیاری از کاربردهای صنعتی کارکرد قابل قبولی را ارایه کرده‌اند.این شرکت با ارایه طیف وسیع میکروکنترولرهای ARM به همراه مجموعه‌ای کامل از وسایل راه‌انداز، عیب‌یابی و برنامه‌ریزی پشتیبانی خوبی از این میکروکنترولرها به عمل آمده است.

۵.۲.۱ میکروکنترولرهای خانواده‌ی LPC1000

مشخصات کلیدی آن‌ها عبارتند از:

- هسته‌ی مرکزی براساس ARM Cortex است:
  - هسته‌ی تازه متولدشده‌ی Cortex-M0؛
  - نسخه‌ی دوم هسته‌ی Cortex-M3؛
- سرعت کارکرد از روی RAM و یا فلش تا فرکانس 100 MHz؛
- مصرف توان کم؛
- کنترل کننده وقفه‌ی جدید Wakeup Interrupt Controller (WIC)؛
- واحد محافظت از حافظه (MPU)؛
- وسایل جانبی در دسترس همچون:

- Ethernet، CAN، I^2S و USB Host /OTG /Device؛
- I^2S FMT، SPI/SSP، UARTS؛
- ADC ۱۲ بیتی با فرکانس تبدیل 1 MHz؛
- RTC کم‌مصرف؛
- PWM مخصوص کنترل موتور و واسط Quadrature Encoder؛
- در بسته‌بندی‌های مختلف در دسترس است.

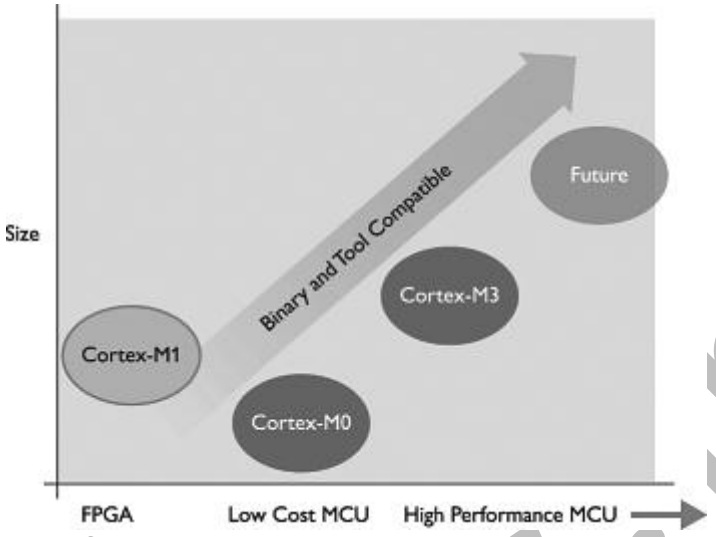


شکل ۵.۱ – مقایسه‌ی میکروکنترولرهای خانواده‌ی LPC1000

تفاوت Cortex M0 و Cortex M3 در چیست؟

Cortex M ، از سری میکروکنترولرهای کارآمد به شمار می‌آید. Cortex M0 ، مجموعه‌ای در ادامه‌ی Cortex M3 بوده که مزایای زیر را ارایه می‌دهد:

۱. بازدهی انرژی؛
۲. بازدهی کارآیی؛
۳. راحتی استفاده.



شکل ۵.۲ – مقایسه‌ی بین اعضای خانواده‌ی Cortex-M

در مورد اول، Cortex M0 به میزان 60DMIPS/mw (در مقایسه با Cortex M3 که 31DMIPS/mw است) ارایه می‌دهد. Cortex M0 همچنین نسبت به بسیاری از میکروکنترولرهای ۱۶/۸ بیتی نیز بهینه‌تر عمل می‌کند. به عنوان مثال، در دستگاه‌هایی که با باتری کار می‌کنند، این میکروکنترولر خیلی سریع‌تر دستور را اجرا کرده و به حالت خواب (Sleep) بازمی‌گردد که این بهینه سازی در مصرف انرژی را به دنبال دارد.

در مورد دوم، Cortex M0 چگالی که کد را به میزان زیادی نسبت به پردازنده‌های ۱۶/۸ بیتی افزایش داده است.

در مورد سوم، رفتار مشخص و زمان‌بندی معلوم این هسته، کاربرد آن را همانند میکروکنترولرهای ۱۶/۸ بیتی بسیار ساده ساخته است. استفاده از نرم‌افزارهای مخصوص و برنامه‌ریزی ساده از مزایای این هسته به حساب می‌آیند.

۵.۲.۱.۱ میکروکنترولرهای خانواده‌ی LPC1100(L)

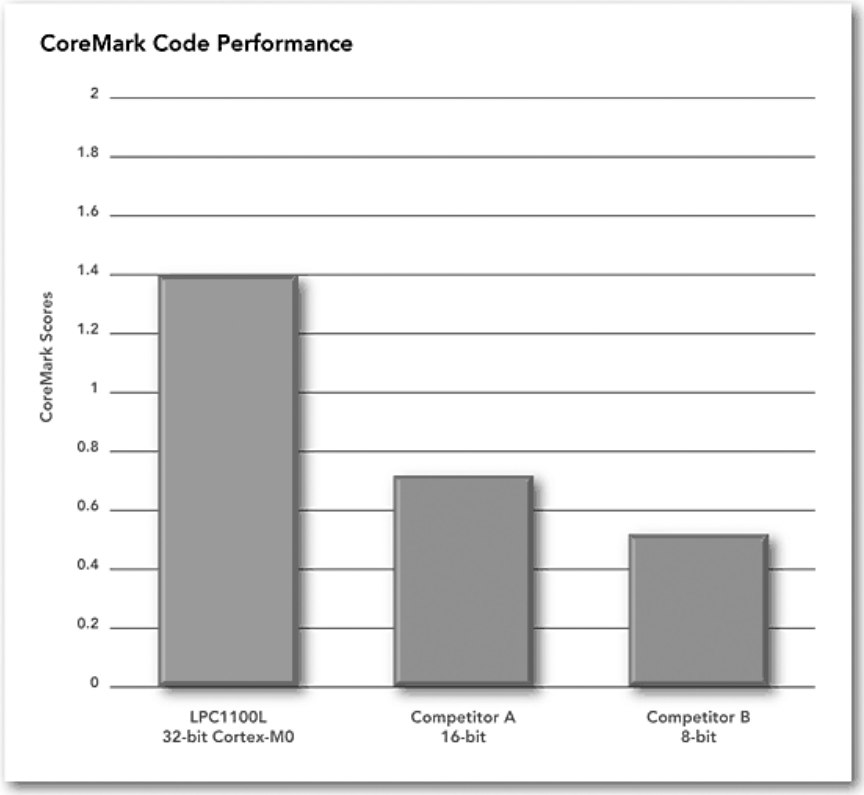
خانواده‌ی میکروکنترولرهای LPC1100 (L)، ارزان‌ترین میکروکنترولرهای ۳۲ بیتی در بازار هستند که با این قیمت غیرقابل رقابت، کارآیی بسیار بهتری را نسبت به میکروکنترولرهای ۱۶/۸ بیتی ارایه می‌دهند. LPC1100 (L) نقطه‌ی شروع بسیار خوبی برای کاربران ۱۶/۸ بیتی است. سادگی استفاده، قیمت بسیار مناسب و اندازه‌ی کوچک، این میکروکنترولرها را بهترین جایگزین برای انتقال ساده از ۱۶/۸ بیت به ۳۲ بیت ساخته است.

کوچک‌ترین، کم‌مصرف‌ترین و بازده‌ترین میکروکنترولر از لحاظ مصرف انرژی است. استفاده از این میکروکنترولرها در دستگاه‌هایی که از باتری استفاده می‌کنند، نشان‌دهنده‌های دیجیتالی هوشمند، کنترل موتور و... توصیه شده است.

شکل ۵.۳ – LPC1100 کم‌مصرف‌ترین ۳۲ بیتی

کارآیی ممتاز

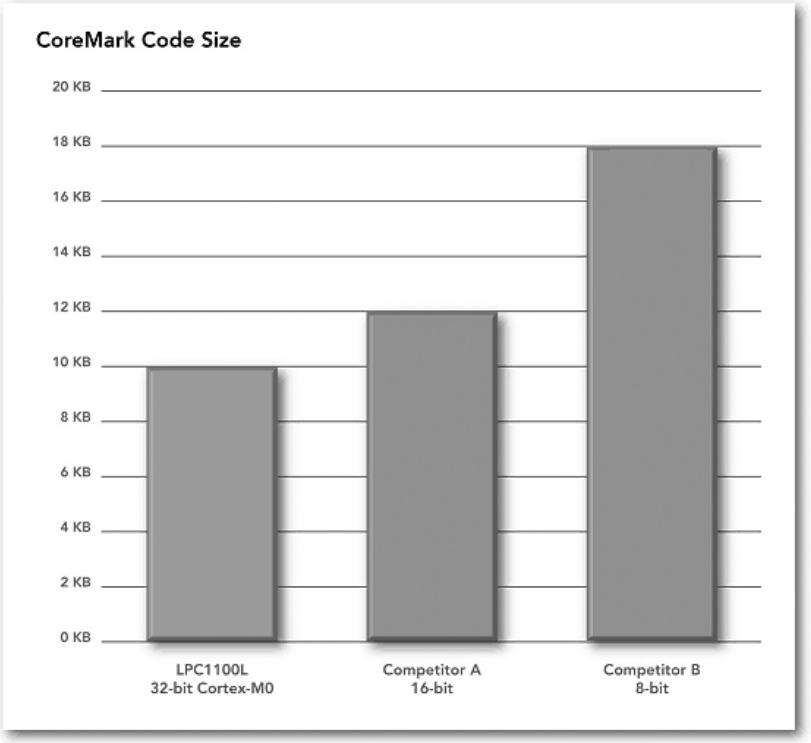
LPC1100 ، با ارایه‌ی کارآیی 45DMIPS/mw در مقایسه با کارآیی زیر 1DMIPS/mw در میکروکنترولرهای ۸ بیتی و تا چندین DMIPS/mw در میکروکنترولرهای ۱۶ بیتی، نه تنها برای اجرای وظایف کنترلی ساده بلکه برای اجرای الگوریتم‌های پیچیده نیز بسیار مناسب‌اند. نیاز به زمان کم برای اجرای دستورهای بیشتر، مستقیماً به معنای مصرف انرژی کمتر است. این سطح از کارآیی با فرکانس کاری حداکثر 50 MHz و با مصرف توان کمتر از 10 mA ، قابل حصول شده است.



شکل ۵.۴ – بررسی کارآیی LPC1100

کوچک‌ترین اندازه‌ی کد

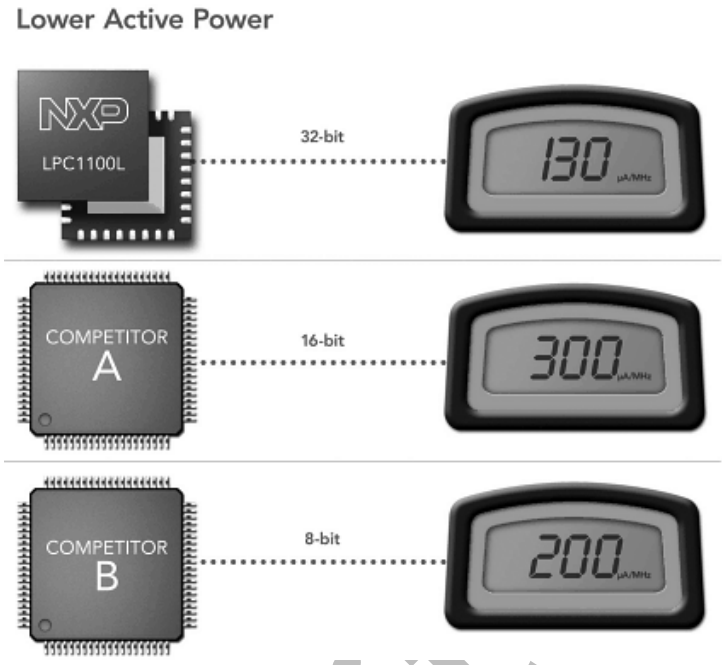
افسانه‌ای در مورد میکروکنترولرهای ۱۶/۸ بیتی حاکم است که مبتنی بر کوچک تر بودن اندازه کد در آن‌هاست. آزمایشهایی استاندارد براساس معیارهای استاندارد Core mark حاکی از این است که LPC1100 تا سقف 50-40 درصد کاهش اندازه‌ی کد را دربر دارد.



شکل ۵.۵ – بررسی اندازه‌ی کد LPC1100

کمترین مصرفِ توان فعال

میکروکنترولرهای ۳۲ بیتی LPC1100 ، مصرف توان فعالی معادل 130mA/MHz را ارایه می‌دهند. استفاده از توابع از پیش‌نوشته شده برای کنترل توان مصرفی، به راحتی امکان استفاده از نمونه‌هایی مشخص از مدیریت مصرف توان را می‌دهند. این میکروکنترولرها بدون از دست دادن کارآیی در محدوده ولتاژ 1.8 V تا 3.6 V به راحتی کار می‌کنند.



شکل ۵.۶ – مصرفِ توان کم

۵.۲.۱.۲ خانواده‌ی میکروکنترولرهای LPC13xx

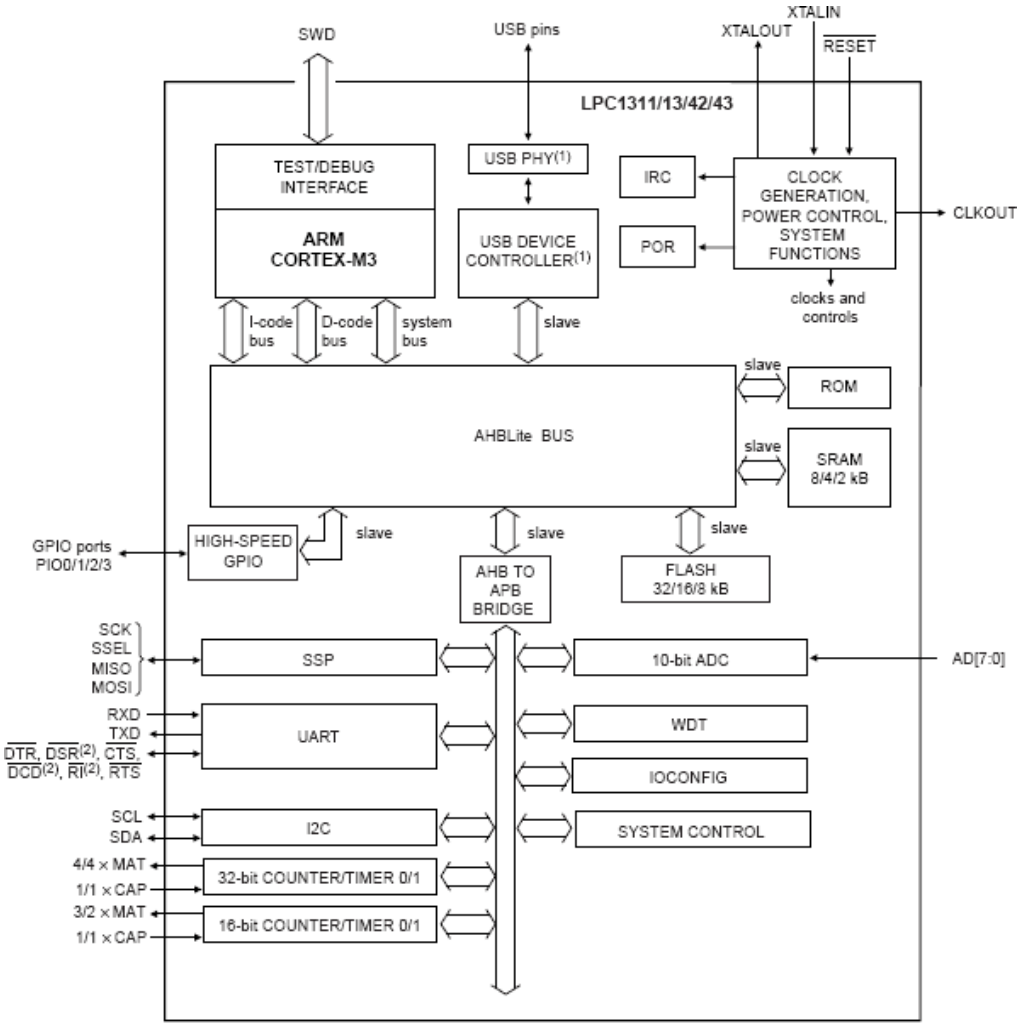
مشخصات کلیدی،

- پردازنده‌ی ARM Cortex M3 با فرکانس کاری 72 MHz؛
- تا سقف 32 KB حافظه‌ی فلش و تا سقف SRAM 8 KB؛
- کنترل کننده‌ی USB Device با PHY روی تراشه (LPC 134x)؛
- ADC ۱۰ بیتی ۱۰ کاناله؛
- واسط UART، کنترل‌کننده‌ی SPI و I^2C؛
- تا سقف ۴۲ پایه‌ی GPIC؛
- ۴ زمان سنج به همراه WDT (زمان سنج نگهبان)؛
- واحد مدیریت توان با پشتیبانی از حالت‌های خواب عمیق؛
- تولید کلاک مجتمع شده بر روی تراشه.

میکروکنترولرهای LPC 13xx میکروکنترولرهای براساس ARM Cortex-M3 بوده که توانایی مجتمع سازی وسایل جانبی متعدد را به همراه مصرف توان اندک، در یک‌جا گنجانده‌اند.

این میکروکنترولرها، در فرکانس حداکثر تا 72 MHz کار می‌کنند.هسته‌ی پردازنده‌ی Cortex M3 از یک خط لوله‌ی ۳ طبقه‌ای استفاده می‌کند. ساختار داخلی‌اش Harvard بوده و دارای گذرگاه‌های مجزا برای دستورالعمل، داده و وسایل جانبی است. این هسته همچنین دارای یک واحد Perfetch داخلی است.

شمای کلی این میکروکنترولرها را در شکل زیر می‌بینید.



شکل ۵.۷

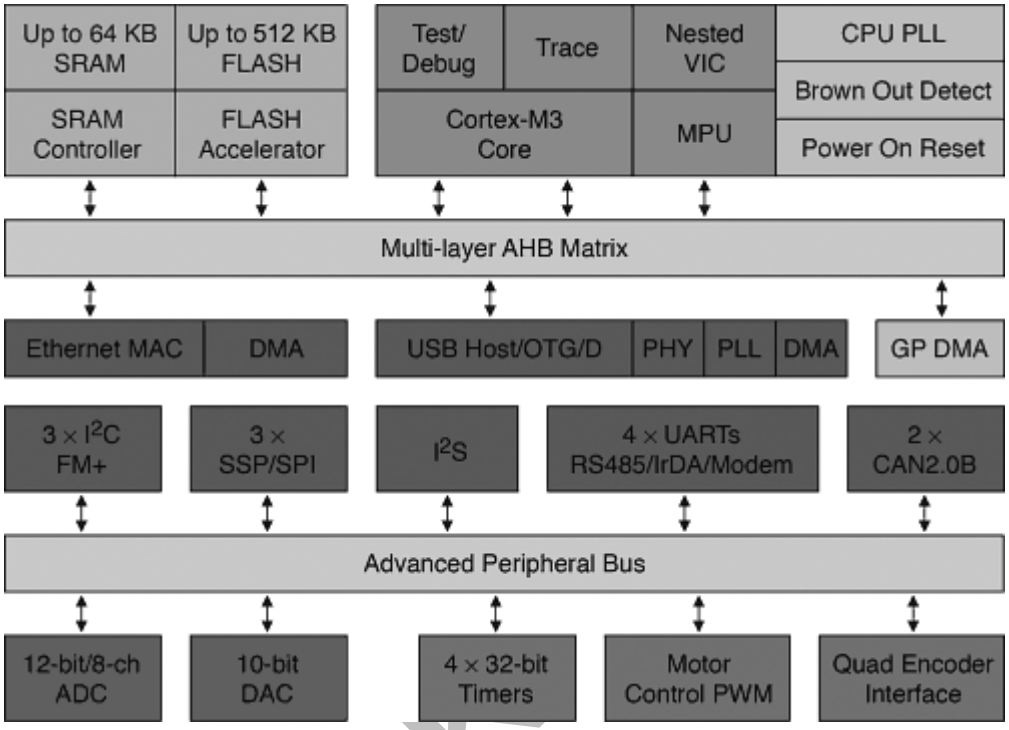
۵.۲.۱.۳ میکروکنترولرهای خانواده‌ی LPC17xx

مشخصات کلیدی:

- عملکرد تا فرکانس 100 MHz (در مورد LPC1769 تا 120 MHz)؛
- کنترل کننده‌ی وقفه‌ی برداری تودرتو (Nested VIC)؛
- واحد محافظت از حافظه (MPU)؛
- ۴ حالت توان مصرفی کم: Sleep، Deep-Sleep، Power-Down و Deep Power-Down.

سری LPC 17xx دارای ادوات جانبی چون Ethernet، USB Host/ Device/ OTG، CAN20B بوده؛ تا فرکانس کاری 120 MHz کار کرده؛ دارای 512 KB حافظه‌ی فلش؛ تا 64 KB حافظه‌ی SRAM؛ ADC ۱۲ بیتی؛ DAC ۱۰ بیتی و نوسان‌ساز RC داخلی است.

شمای کلی LPC 17xx را در زیر می‌بینید.



شکل ۵.۸

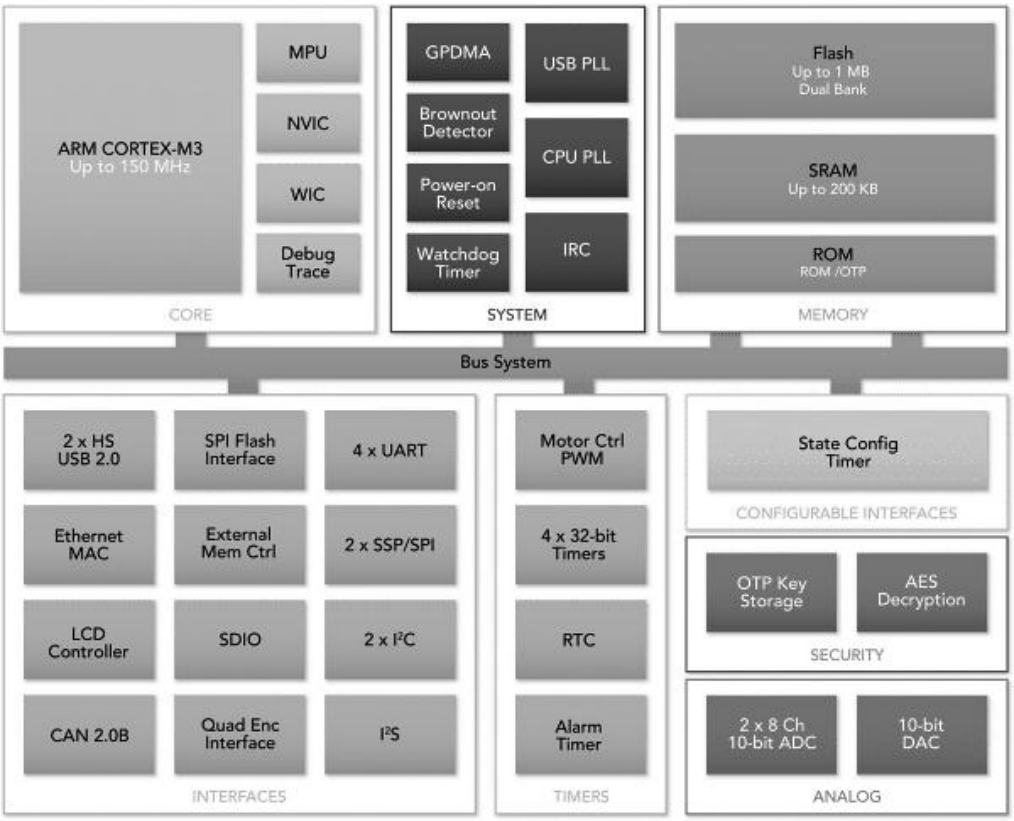
۵.۲.۱.۴ میکروکنترولرهای خانواده LPC18xx

مشخصات کلیدی:

- عملکرد در فرکانس 150 MHz؛
- دو بانک حافظه‌ی فلش تا سقف 1 MB؛
- تا سقف 200 KB حافظه‌ی SRAM داخلی؛
- وسایل جانبی ابداعی همانند واسط حافظه‌ی فلش SPI و زمان سنج SCT؛
- دو عدد USB سرعت بالا؛
- واحد محافظت حافظه (MPU).

این سری از میکروکنترولرهای Cortex M3 با کارآیی بالا، دارای USB سرعت بالا، کنترل کننده‌ی LCD، CAN و سرعت عملکرد 150 MHz هستند.

شمای کلی این میروکنترولرها را در زیر می‌بینیم.



شکل ۵.۹

۵.۲.۲ میکروکنترولرهای خانواده‌ی LPC2000

مشخصات کلیدی این میکروکنترولرها را در زیر می‌بینیم:

- هسته‌ی مرکزی آن‌ها براساس ARM7TDMI-S با ولتاژ کاری 1.8 V است (البته LPC29xx براساس ARM968E-S ساخته شده)؛
- حداکثر سرعت کار تا سقف 80 MHz؛
- ارتباطات سریال همانند: CAN، LIN، I^2C، UART و SPI؛
- ارتباط USB، I^2C و SD/ MMC؛
- نسخه‌های بسیار کارآمد با گذرگاه دوگانه‌ی AHB نیز موجودند؛
- کنترل کننده‌های LCD در بعضی تراشه‌ها موجودند؛

- ADC و DAC های ۱۰ بیتی؛
- مصرف توان کم؛
- بعضی تراشه‌ها در بسته‌بندی ظریف LQFP 7mm×7mm نیز ارایه شده‌اند.

این خانواده از میکروکنترولرها به چندین زیرگروه تقسیم می‌شوند که در ادامه متن به بررسی آن‌ها می‌پردازیم. این گروه‌ها عبارت‌اند از:

- گروه LPC2103، LPC2102، LPC2101. مشخصه کلیدی: قیمت اندک، میکروکنترولرهای ظریف و کوچک؛
- گروه LPC214x. مشخصه‌ی کلیدی: میکروکنترولرهای با پورت USB-Device نسخه 2.0؛
- گروه LPC23xx. مشخصه کلیدی: میکروکنترولرها با گذرگاه AHB دوگانه؛
- گروه LPC24xx. مشخصه کلیدی: میکروکنترولرها با گذرگاه AHB دوگانه و واسط حافظه‌ی خارجی؛
- گروه LPC29xx. مشخصه کلیدی: سریع‌ترین میکروکنترولر با هسته‌ی ARM968 به همراه CAN و LIN.

۵.۲.۲.۱ میکروکنترولرهای LPC2103 و LPC2102 و LPC2101

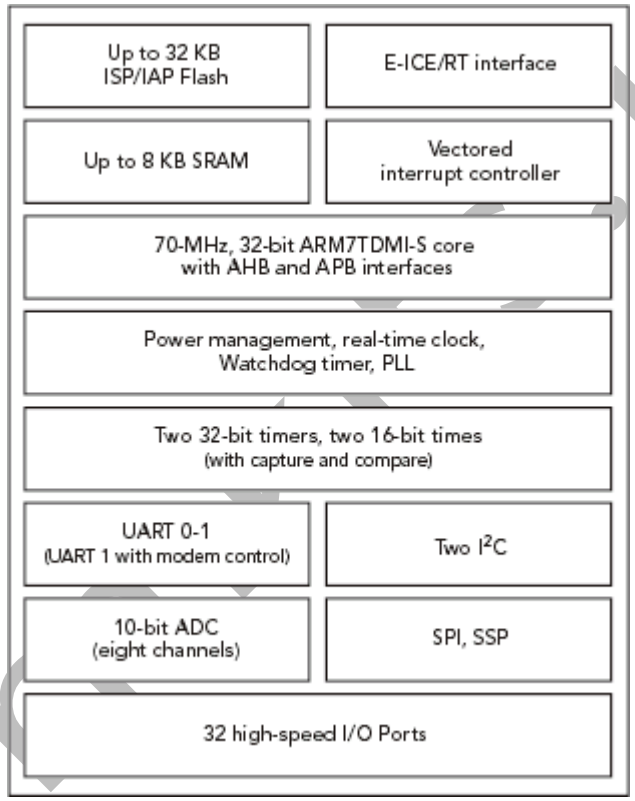
مشخصات کلیدی:

- معماری هسته ARM7 ۳۲- بیتی؛
- قیمت بسیار پایین (از 1.47 دلار آمریکا شروع می‌شود)؛
- کارآیی سریع در فرکانس کاری 70 MHz و سرعت اجرای دستورها 63 Dhrystone MIPS؛
- بسته‌بندی ظریف LQFD 7mm×7mm؛
- جایگزینی ایده‌آل برای سیستم‌های مبتنی بر میکروکنترولرهای ضعیف ۸ و ۱۶ بیتی؛
- به همراه وسایل جانبی ارزشمند و طراحی با مصرف پایین.

میکروکنترولرهای LPC2103، LPC2102، LPC2101 ، دربردارنده پردازنده‌ی ARM7TDMI-S در فرکانس کاری 70 MHz به همراه 8 تا 32 KB حافظه‌ی فلش بدون تأخیر (Zero wait-state) می‌باشند. ساده‌ترین تراشه این گروه یعنی LPC2101، ارزان‌ترین آن‌ها نیز هست. قیمت این قطعه در سفارشهای با تعداد 10,000 به بالا، از قیمت 1,47 دلار آمریکا شروع می‌شود. این قطعه جایگزینی مناسب برای سیستم‌های ۸ و ۱۶ بیتی با کارآمدی کمتر است.

در کنار این خصیصه، سرعت بالا به همراه مصرف توان پایین و بسته‌بندی ظریف با ابعاد 7mm× 7mm آن را برای گستره‌ای وسیع از کاربردها مناسب می‌سازد.

این پردازنده‌های ارزان قیمت با تعدادی از زیرسیستم‌های جانبی مفید همانند ADC های ۱۰ بیتی، ۴ تایمر، چندین I^2C، SPI و واسط سریال UART در یک بسته‌بندی جای گرفته‌اند.
این سری میکروکنترولرها با تجهیز به حالت‌های جدید مصرف توان کم (Power-Saving) و پایه‌های ورودی خروجی دیجیتال با سرعت بالا دست طراح را در طراحی بازتر کرده‌اند.
وسایل جانبی و کدهای نوشته شده برای پردازنده با تمام میکروکنترولرهای خانواده LPC2000 (که بیش ۳۰ عضو هستند) ،سازگاراند. شمای کلی میکروکنترولرهای این خانواده را در شکل ۵.۱۰ می-بینیم.



شکل ۵.۱۰

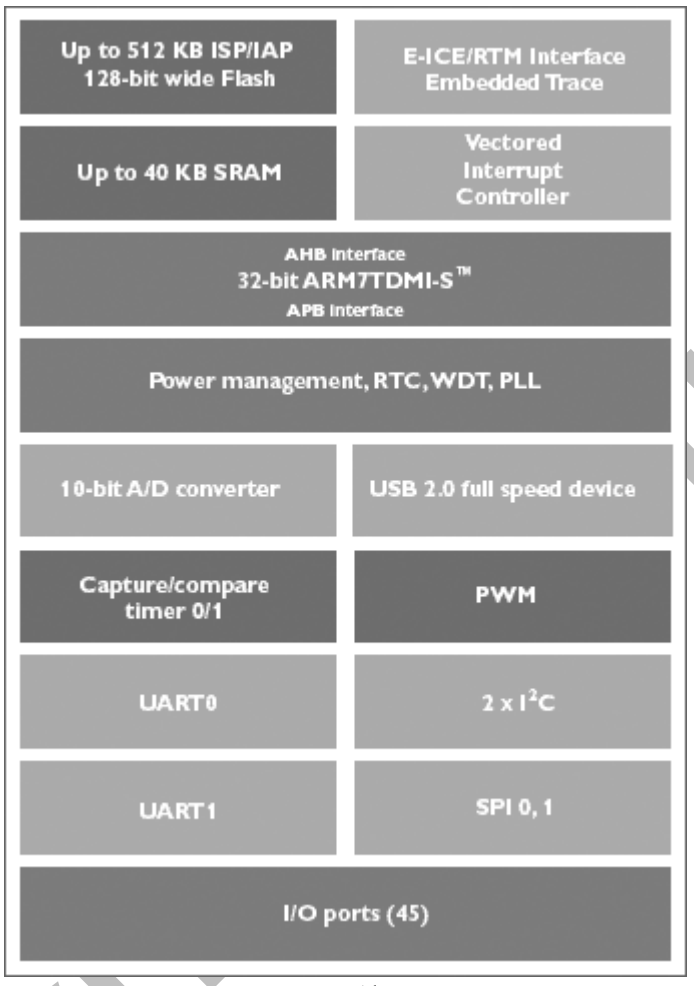
۵.۲.۲.۲ میکروکنترولرهای سری LPC214x

مشخصات کلیدی:

- معماری هسته ۳۲ بیتی ARM7؛
- دارای USB-Device با سرعت Full-Speed هستند؛

- حافظه‌ی فلش سریع بر روی تراشه با حجم حداکثر تا 512 KB؛
- حافظه‌ی SRAM تا سقف 40 KB؛
- ۴۵ پایه I/O سریع با حداکثر سرعت سوئیچینگ 15 MHz؛
- کنترل کننده‌ی وقفه‌ی برداری (VIC)؛
- بسته‌بندی بسیار کوچک LQFP با ابعاد 10mm×10mm؛
- مناسب برای وسایل سرگرمی، ارتباط و نمایش‌دهنده‌ها.

میکروکنترولرهای سری LPC214x ، تنها میکروکنترولرهای ARM7 کاملاً سازگار با USB 2.0 و دارای گواهینامه‌ی تأیید از USB.Org هستند. این قطعات از USB-Device با سرعت Full Speed پشتیبانی کرده، دارای 32 end_point به همراه 2 KB حافظه‌ی SRAM برای این end point ها می‌باشند. همچنین، دارای واسط DMA برای USB با 8 KB حافظه‌ی RAM مخصوص به خود (در تراشه‌های LPC2146 و LPC2148) بوده و از تمامی روش‌های انتقال داده (یعنی Control، Interrupt، Bulk و Iso Chronous) پشتیبانی می‌کند. این میکروکنترولرها طراحان را قادر می‌سازد که با صرف کمترین هزینه (معادل 3.60 دلار برای LPC2141) ، دستگاه‌های تولیدی خود را مجهز به USB نمایند.
علاوه بر USB، این میکروکنترولرهای کم مصرف دارای 32 KB تا 512 KB حافظه‌ی فلش بدون تأخیر، حافظه‌ی SRAM از 8 KB تا 40 KB، ADC ۱۰- بیتی، DAC ۱۰- بیتی و یک RTC کم مصرف هستند.
شمای کلی این خانواده از میکروکنترولرها را در شکل ۵.۱۱ می‌بینیم.


Up to 512 KB ISP/IAP
128-bit wide Flash
E-ICE/RTM Interface
Embedded Trace
Up to 40 KB SRAM
Vectored
Interrupt
Controller
AHB Interface
32-bit ARM7TDMI-S™
APB Interface
Power management, RTC, WDT, PLL
10-bit A/D converter
USB 2.0 full speed device
Capture/compare
timer 0/1
PWM
UART0
2 x I²C
UART1
SPI 0, 1
I/O ports (45)

شکل ۵.۱۱

۵.۲.۲.۳ میکروکنترولرهای سری LPC23xx

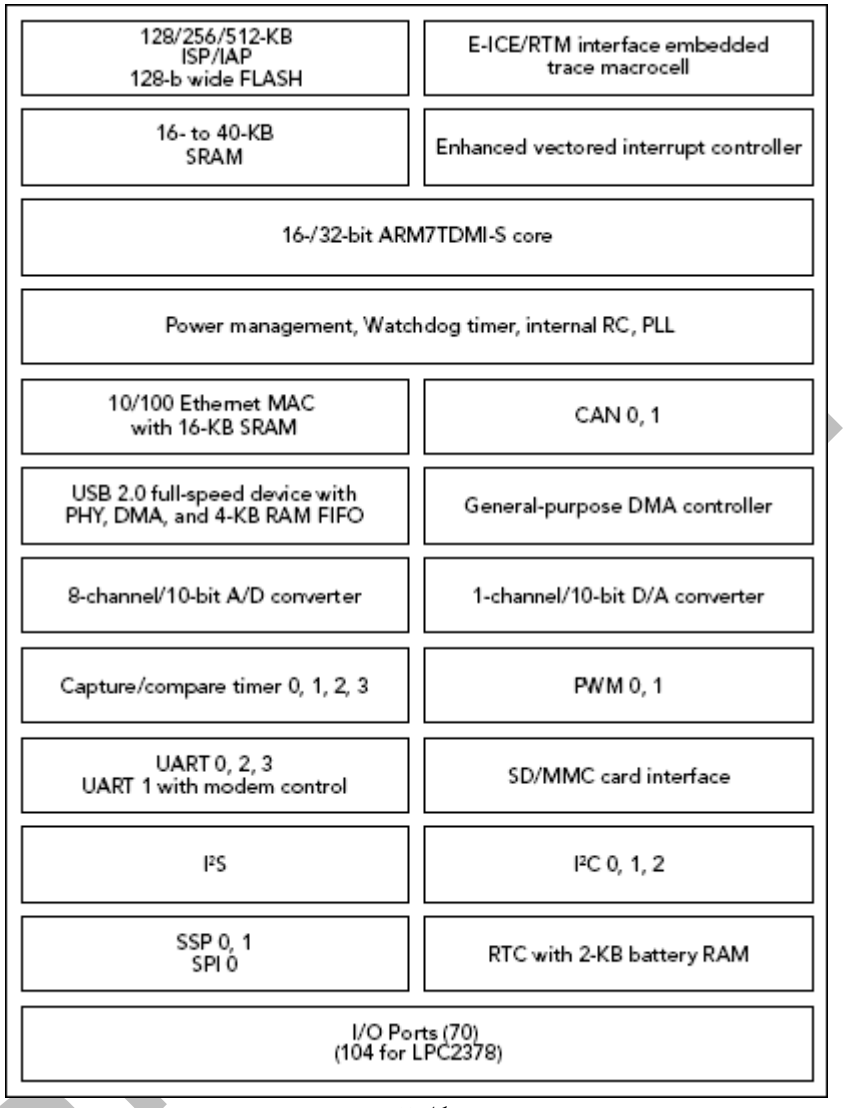
مشخصات کلیدی:

- معماری هسته ARM7 ۳۲ بیتی؛
- عملکرد تا فرکانس 72 MHz (64 Dhrystone MIDS)؛
- تا سقف 512 KB حافظه‌ی فلش بـر روی تراشـه و 58 KB حافظه‌ی SRAM؛
- رابط شبکه‌ی Ethernet MAC به همراه DMA؛
- رابط USB Device نسخه‌ی 2.0 به همراه PHY و DMA؛
- رابط CAN 2.0B با دو کانال؛
- کنترل کننده‌ی DMA همه‌منظوره؛
- رابط I^2S، سه رابط I^2C، سه رابط SPI/SSP، چهار رابط UARTs؛
- نوسان‌ساز داخلی RC با فرکانس 4 MHz (IRC) با دقت ٪۱.

میکروکنترولرهای سری LPC23xx ، در فرکانس کاری 72 MHz کار می‌کنند. دارای حافظه‌ی فلـش تـا سقف 512 KB با حالت تأخیر صفر[۱] هستند. مهم‌ترین توانایی این میکروکنترولرها این‌اسـت کـه در آن-واحد می‌توانند USB، CAN، Ethernet و یک برنامه‌ی کاربردی را راه‌اندازی کنند. مهم‌ترین دلیل این توانمندی استفاده از ۲ گذرگاه AHB در یک ساختار ARM7 است.

شمای کلی این میکروکنترولر را در شکل ۵.۱۲ می‌بینیم.

[۱] Zero Wait-State


128/256/512-KB
ISP/IAP
128-b wide FLASH
E-ICE/RTM interface embedded
trace macrocell
16- to 40-KB
SRAM
Enhanced vectored interrupt controller
16-/32-bit ARM7TDMI-S core
Power management, Watchdog timer, internal RC, PLL
10/100 Ethernet MAC
with 16-KB SRAM
CAN 0, 1
USB 2.0 full-speed device with
PHY, DMA, and 4-KB RAM FIFO
General-purpose DMA controller
8-channel/10-bit A/D converter
1-channel/10-bit D/A converter
Capture/compare timer 0, 1, 2, 3
PWM 0, 1
UART 0, 2, 3
UART 1 with modem control
SD/MMC card interface
I²S
I²C 0, 1, 2
SSP 0, 1
SPI 0
RTC with 2-KB battery RAM
I/O Ports (70)
(104 for LPC2378)

شکل ۵.۱۲

۵.۲.۲.۴ میکروکنترولرهای سری LPC24xx

مشخصات کلیدی:

- فرکانس عملکرد تا 72 MHz، معماری ARM7TDMI-S با گذرگاه AHB دوگانه؛
- مدار واسط برای اتصال حافظه‌های خارجی SRAM، SDRAM و Flash؛
- واسط Ethernet MAC با DMA و واسطه‌ی MII/RMII؛
- واسط USB Device/ Host/ OTG نسخه‌ی 2.0 به همراه PHY و DMA؛
- در بعضی شماره‌ها کنترل کننده LCD TFT/STN گنجانده شده است؛
- محدوده گسترده‌ای از وسایل جانبی همانند CAN، I^2S، ADC، PWM و... .

میکروکنترولرهای سری LPC24xx ، براساس معماری گذرگاه دوگانه‌ی AHB ساخته شده‌اند. این سری همانند LPC23xx دارای CAN، USB و Ethernet می‌باشند. علاوه بر آن LPC24xx واسط USB/ Host/ OTG، گذرگاه خارجی و تعداد بیشتری I/O را در اختیار گذاشته است. این مشخصات، آن را به قدرت‌مندترین میکروکنترولرهای ARM7 در بازار مبدل کرده است.

جدیدترین میکروکنترولرهای این سری با شماره‌ی LPC247x افزون بر مزایای گفته شده، دارای کنترل کننده‌ی LCD از نوع STN/TFT با رزولوشن رنگ ۲۴ بیت نیز هستند.

LPC2468

این میکروکنترولر، دارای حافظه‌ی فلش سریع روی تراشه‌ی 512 KB می‌باشد. این تراشه برای ارتباطات چندگانه، بین چند دستگاه انتخاب خوبی است. LPC2468 دارای کنترل کننده‌ی داخلی Ethernet MAC با سرعت 10/100 Mbps، کنترل کننده‌ی USB از نوع Device/Host/OTG با حافظه‌ی اندپوینت[۱] 4 KB، ۴ رابط UART، ۲ کانال CAN و مجموعه‌ای از ارتباطات سریال دیگر است.

[۱] End Point

LPC2470 و LPC2478

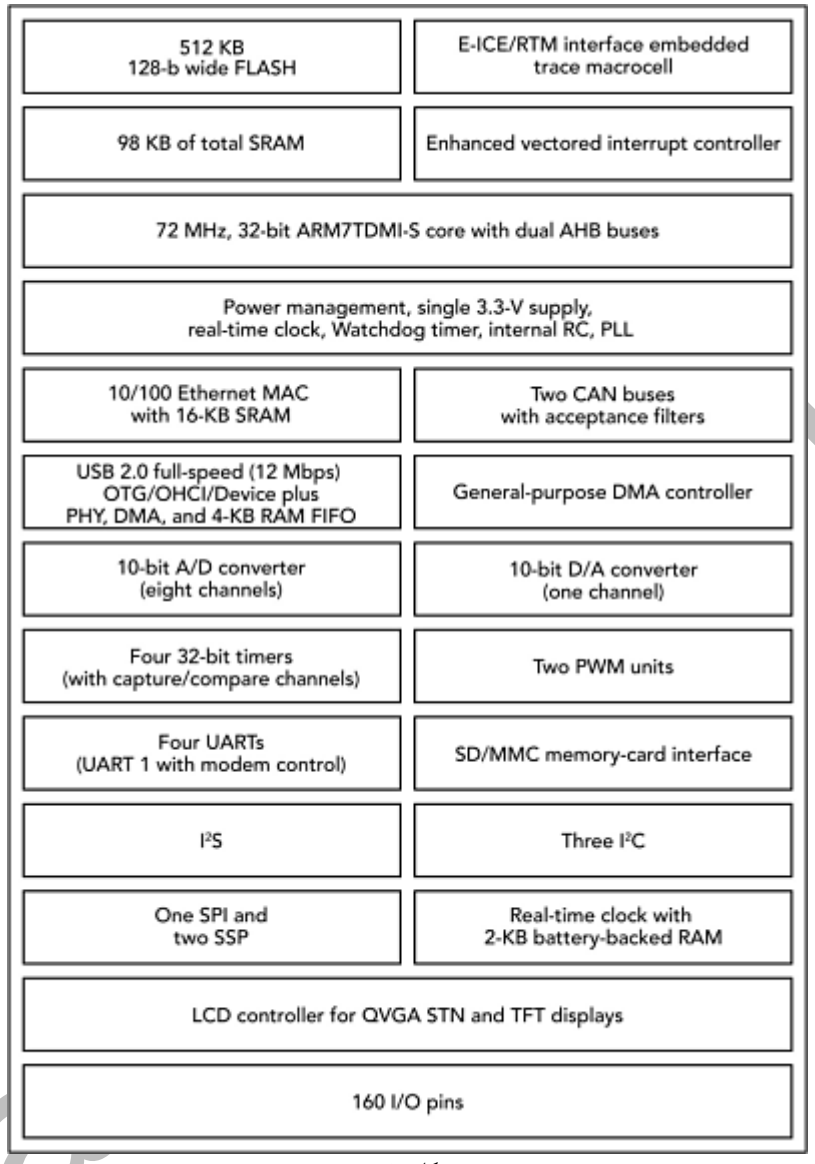
این دو میکروکنترولر، کنترل کننده‌ی LCD را نیز به مجموعه‌ی فوق اضافه می‌کنند. این کنترل کننده توانایی کنترل LCD های STN[1] و TFT[2] با رزولوشن نمایش 1024×768 پیکسل را با قدرت نمایش رنگ از تک‌رنگ تا رزولوشن نمایش رنگ ۲۴ بیت را دارد.

پشتیبانی از Cursor سخت‌افزاری نیز جزیی از توانایی‌های آن است. این توانمندی برای دستگاه‌های پایانه‌ی پرداخت (POS)، دستگاه‌های صنعتی و تجهیزات پزشکی ایده‌آل است. علاوه بر این، کنترل کننده LCD از قالب‌بندی داده‌های Win CE نیز پشتیبانی می‌کند.

شمای کلی این میکروکنترولرها را در شکل ۵.۱۳ می‌بینیم.

[1] Super Twisted Nematic

[2] Thin-Film Transistor


512 KB
128-b wide FLASH
E-ICE/RTM interface embedded
trace macrocell
98 KB of total SRAM
Enhanced vectored interrupt controller
72 MHz, 32-bit ARM7TDMI-S core with dual AHB buses
Power management, single 3.3-V supply,
real-time clock, Watchdog timer, internal RC, PLL
10/100 Ethernet MAC
with 16-KB SRAM
Two CAN buses
with acceptance filters
USB 2.0 full-speed (12 Mbps)
OTG/OHCI/Device plus
PHY, DMA, and 4-KB RAM FIFO
General-purpose DMA controller
10-bit A/D converter
(eight channels)
10-bit D/A converter
(one channel)
Four 32-bit timers
(with capture/compare channels)
Two PWM units
Four UARTs
(UART 1 with modem control)
SD/MMC memory-card interface
I²S
Three I²C
One SPI and
two SSP
Real-time clock with
2-KB battery-backed RAM
LCD controller for QVGA STN and TFT displays
160 I/O pins

شکل ۵.۱۳

۵.۲.۲.۵ میکروکنترولرهای سری LPC29xx

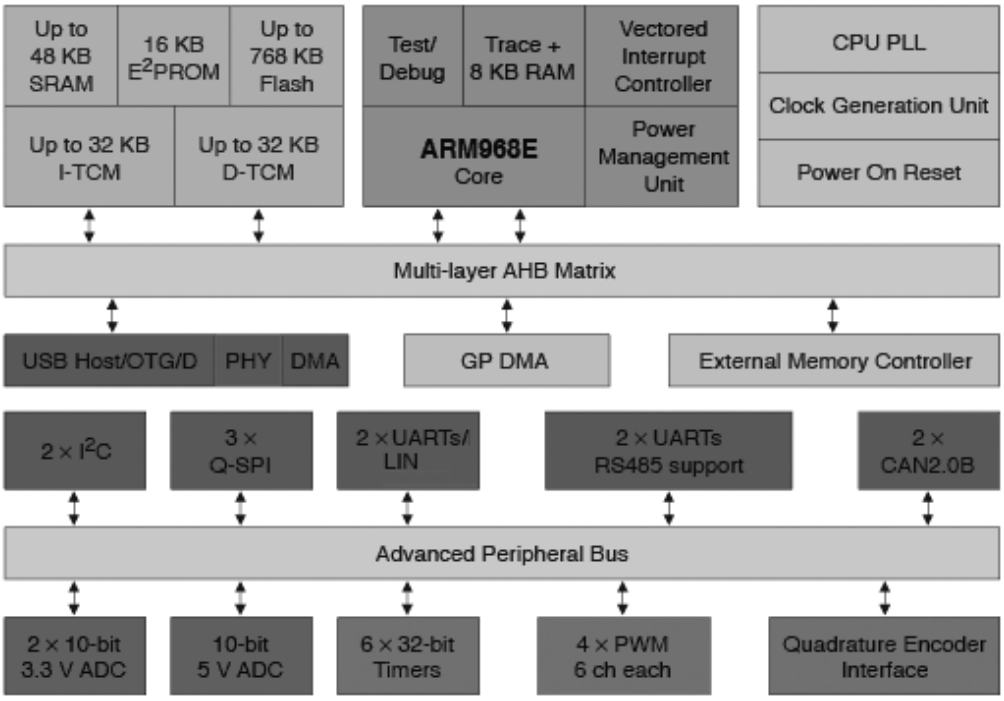
مشخصات کلیدی:

- هسته‌ی ARM968E-S ۳۲ بیتی با فرکانس کاری 125 MHz؛
- تا سقف 768 KB حافظه‌ی برنامه از نوع فلش؛
- تا سقف 48 KB حافظه‌ی SRAM؛
- حافظه‌های از نوع TCM (32 KB برای دستورها و 32 KB برای داده‌ها)؛
- 16 KB حافظه‌ی EEPROM؛
- کنترل کننده‌ی USB Host/ OTG/ Device؛
- دو کنترل کننده‌ی CAN 2.0B و دو کنترل کننده‌ی LIN 2.0 Master؛
- دو 3V ADC و یک 5V ADC؛
- چندین واسط I^2C ، Q-SPI و UART؛
- PWM مخصوص کنترل کننده‌ی موتور و واسط Quadrature Encoder؛
- واسط کنترل کننده‌ی حافظه‌ی خارجی.

میکروکنترولرهای سری LPC29xx ، با فرکانس کاری 125 MHz، سریع‌ترین میکروکنترولرهای ARM968 موجود در بازار هستند. این میکروکنترولرها کاربرد متنوعی در راه‌اندازهای صنعتی، سیستم HVAC، دستگاه‌های شوینده، فروش خودکار و دستگاه‌های کنترل موتور دارند.

علاوه بر سرعت عملکرد بالای این تراشه‌ها، سری LPC29xx مشخصاتی از قبیل USB Host/ OTG/ Device، 16 KB حافظه‌ی EEPROM، UART با پشتیبانی از RS485 و LIN و کنترل کننده‌ی موتور را دارا است. سری LPC29xx دارای مشخصات یک فی همانند خانواده‌ی LPC23xx است. بنابراین، این خانواده‌ی جدید به عنوان جایگزینی مناسب، سریع و با مشخصات اضافه مطرح است. این سری با حافظه‌ی فلش تا سقف 768 KB، حافظه‌ی RAM تا سقف 56 KB و دو عدد TCMبا حجم 32 KB تجهیز شده است.

شمای کلی این میکروکنترولرها را در شکل زیر می‌بینیم.



شکل ۵.۱۴

۵.۲.۳ میکروکنترولرهای سری LPC3000

این خانواده جزو پیشرفته‌ترین میکروکنترولرها با هسته‌ی ۳۲/۱۶ بیتی ARM926EJ-S است. فرکانس کاری این میکروکنترولرها بالغ بر 200 MHz بوده و با تکنولوژی 90 nm ساخته شده‌اند.

۵.۲.۳.۱ میکروکنترولرهای سری LPC31xx

این سری، ارزان‌ترین ARM9 به همراه USB 2.0 OTG و موتور رمزگشایی اختیاری و سیستم قدرت-مند آنالوگ است.

سیستم‌های تعبیه شده هم‌اکنون از کارآیی بالا، قیمت کم و توان مصرفی اندک و اندازه فیزیکی کوچک تراشه‌ها بسیار بهره‌مند می‌شوند. کاربردهای تجاری و دستگاه‌هایی که نیاز به اتصال USB، موتورهای رمزکننده‌ی AES دارند، منابع تغذیه (PSU)، کدکننده-دیکودکننده‌های استریو و شارژرهای باتری Li-Ion از جمله مصرف کنندگان اصلی این میکروکنترولرهای پیشرفته‌اند.

مشخصات کلیدی:

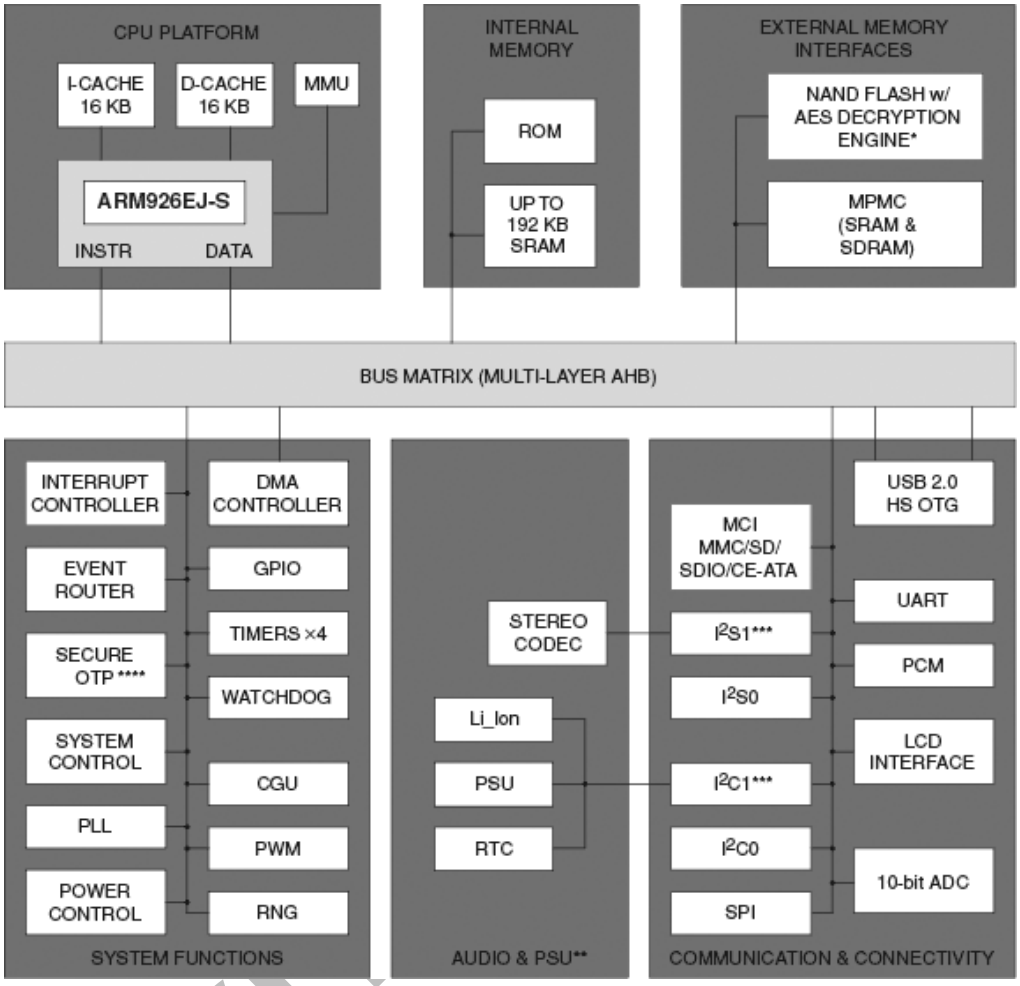
- هسته‌ی ARM926EJ-S با فرکانس کاری حداکثر 270 MHz؛
- USB سرعت بالا (High Speed) از نوع OTG به همراه PHY بر روی تراشه؛
- موتور رمزگشایی ۱۲۸ بیتی AES (LPC3143/LPC315x)؛
- تا سقف 192 KB حافظه‌ی SRAM داخلی؛
- اتصالات کلاک و مقیاس‌بندی دینامیک کلاک؛
- کنترل کننده‌ی فلش NAND به همراه ECC ۸ بیتی؛
- واسط کارت‌های حافظه‌ی MMC/SD/SDIO/CE-ATA؛
- واسط LCD؛
- مجموعه‌ای گسترده از ارتباطات سریال.

خانواده‌ی LPC313x با مجموعه‌ی گسترده از خصوصیات بهبود یافته، برای گستره‌ای وسیع از کاربردها، مشتمل بر کنترل کننده‌های راه دور پیشرفته، مترجم‌های جیبی، دستگاه‌های صوتی و تصویری خانگی، تجهیزات موبایل و رایانه‌ی شخصی، تجهیزات اتوماسیون صنعتی، سیستم‌های POS و... می‌باشند. البته، فهرست دستگاه‌هایی که از این میکروکنترولر پیشرفته استفاده می‌کنند، بسیار گسترده‌تر از آن چیزی است که در این‌جا نام برده‌ایم.

سری LPC314x خصوصیاتی علاوه بر آن‌چه در LPC313x بود را اضافه کرده است. این مشخصات عبارت‌اند از موتور رمزگشایی AES و حافظه‌ی ایمن OTP. این سری با فرکانس معادل 270 MHz می‌تواند کار کند (این فرکانس ٪۵۰ بیشتر از فرکانس کاری LPC313x است).

سری LPC315x با فرکانس 150 MHz کار می‌کند، دارای یک Codec استریو به همراه تقویت کننده هدفون کلاس AB و یک واحد منبع تغذیه (PSU)، یک RFC کم مصرف و شارژر باتری لیتوم یون است.

شمای کلی این خانواده از میکروکنترولرها را در شکل ۵.۱۵ می‌بینید.


CPU PLATFORM
I-CACHE 16 KB
D-CACHE 16 KB
MMU
ARM926EJ-S
INSTR
DATA
INTERNAL MEMORY
ROM
UP TO 192 KB SRAM
EXTERNAL MEMORY INTERFACES
NAND FLASH w/ AES DECRYPTION ENGINE*
MPMC (SRAM & SDRAM)
BUS MATRIX (MULTI-LAYER AHB)
INTERRUPT CONTROLLER
DMA CONTROLLER
EVENT ROUTER
GPIO
SECURE OTP ****
TIMERS ×4
WATCHDOG
SYSTEM CONTROL
CGU
PLL
PWM
POWER CONTROL
RNG
SYSTEM FUNCTIONS
STEREO CODEC
Li_Ion
PSU
RTC
AUDIO & PSU**
MCI MMC/SD/ SDIO/CE-ATA
I2S1***
I2S0
I2C1***
I2C0
SPI
USB 2.0 HS OTG
UART
PCM
LCD INTERFACE
10-bit ADC
COMMUNICATION & CONNECTIVITY

شکل ۵.۱۵

۵.۲.۳.۱ میکروکنترولرهای سری LPC3180/01

مشخصات کلیدی:

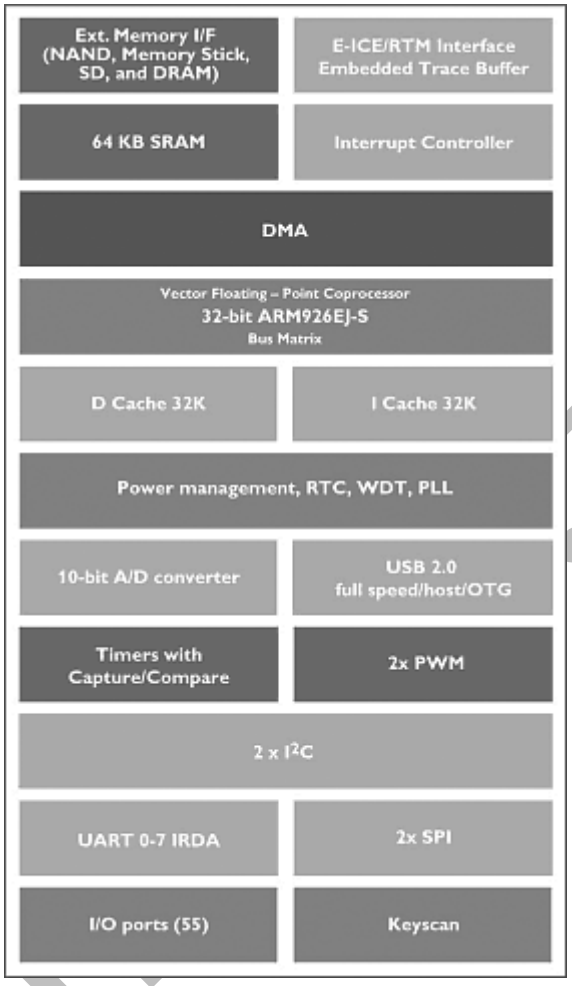
- هسته‌ی ARM926EJ-S؛
- تکنولوژی ساخت 90nm؛
- سرعت کار بالا: 208 MHz؛
- حالت کاری توان پایین (تاحد کمینه‌ی 0.9V)؛
- کمک پردازنده منحصربه‌فرد VFU برای عملیات اعشاری؛
- USB از نوع OTG؛
- کمک پردازنده‌ی Java byte-Code بر روی تراشه؛
- ۷ واسط UART، SPI و I^2C.

LPC3180/01 کارآیی بالا به همراه توان مصرفی کم را عرضه می‌کند. با سرعت عملکرد تا 208 MHz این میکروکنترولرها برای طیف وسیعی از دستگاه‌ها کاربرد دارند. دستگاه‌هایی چون POS ها، تجهیزات صنعتی، پزشکی، GPS، روبوتیک و... از عمده مصرف کنندگان این تراشه هستند.

این میکروکنترولرها براساس هسته‌ی پرمصرف و شناخته شده‌ی ARM926EJ-S ساخته شده است. MMU واقع بر روی تراشه، این میکروکنترولر را برای بسیاری از سیستم‌های عامل همانند Linux و Win CE مناسب ساخته است.

کمک پردازنده‌ی سخت‌افزاری اعشاری، محاسبات معمولی را به میزان ۴ تا ۵ برابر سرعت می‌بخشد. و در حالت برداری بسیار بهینه‌تر عمل می‌کند.

شمای کلی این میکروکنترولر را در شکل ۵.۱۶ می‌بینیم.



شکل ۵.۱۶

۵.۲.۳.۳ میکروکنترولرهای سری LPC32x0

مشخصات کلیدی:

- هسته‌ی ۳۲ بیتی ARM9EJ-S با فرکانس کاری 266 MHz؛
- کمک پردازنده‌ی برداری نقطه‌ی اعشاری (VFU)؛
- تا سقف 256 KB حافظه‌ی داخلی SRAM و 32 KB حافظه‌ی نهان I و 32 KB حافظه‌ی نهان D؛
- کنترل کننده‌ی حافظه‌های خارجی SRAM، Flash، DDR/SDR SDRAM؛
- کنترل کننده‌ی Ethernet MAC10/100؛
- کنترل کننده‌ی USB Host/ Device؛

- کنترل کننده‌ی LCD با پشتیبانی از STN/TFT؛
- واسط کارت حافظه‌ی SD؛
- مجموعه‌ای وسیع از ارتباطات سریال.

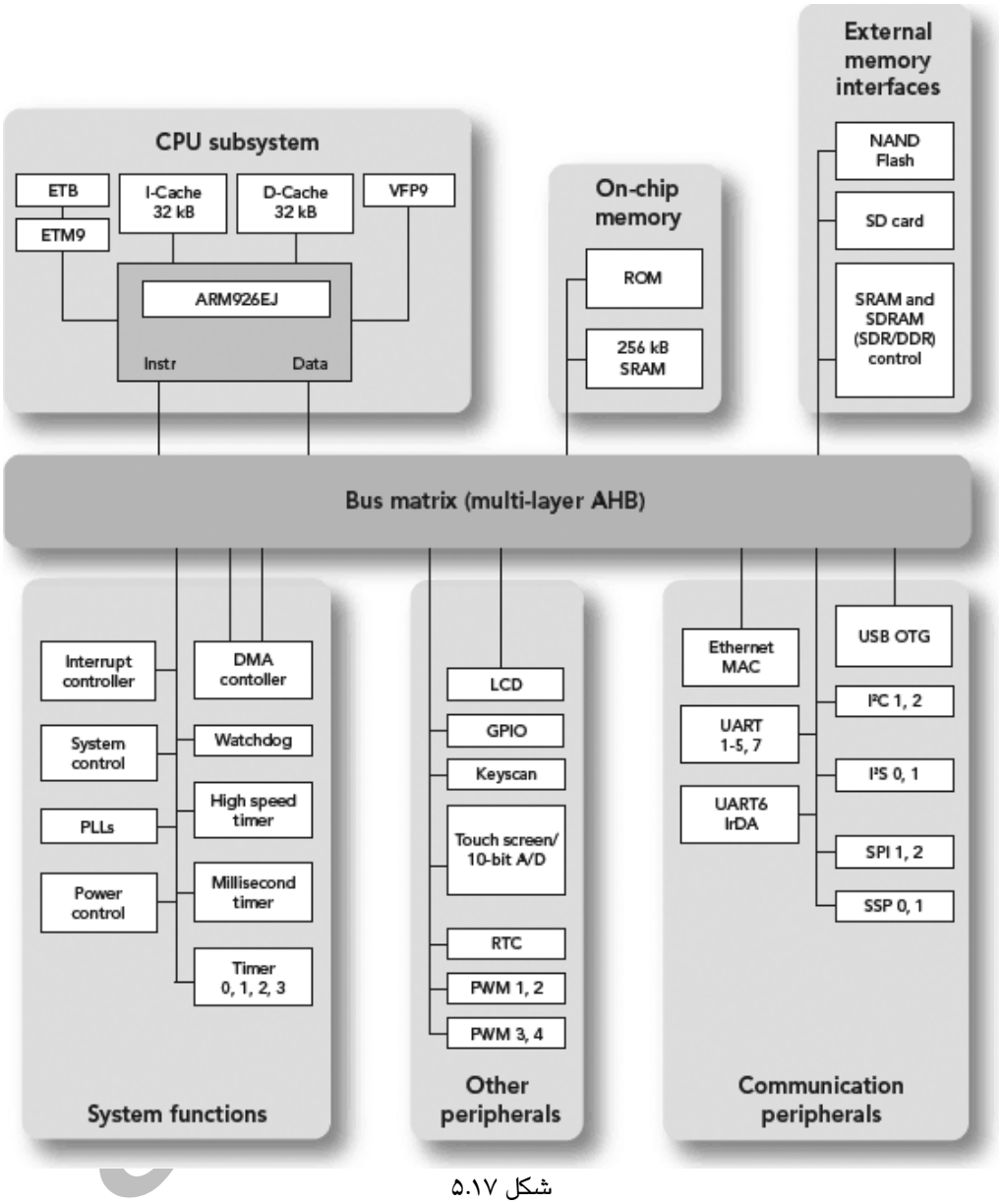
خانواده‌های میکروکنترولرهای LPC32x0 با تکنولوژی ساخت 90nm، هسته‌ی ARM926EJ-S با فرکانس کاری 266 MHz وکمک پردازنده‌ی VFU به یکی از کاراترین، کم مصرف‌ترین پردازنده‌ها مبدل گشته‌اند. کمک پردازنده، کارایی پردازشی را در حالت اسکالر به میزان ۴ تا ۵ برابر افزایش می‌دهد. این میزان در حالت برداری بسیار بیشتر است.

میکروکنترولرهای LPC3240 و LPC3250 یک کنترل کننده‌ی EMAC 10/100 به همراه DMA را در خود جای داده‌اند. یک کنترل کننده‌ی LCD منعطف برای راه‌اندازی LCD های STN/TFT در تراشه‌های LPC3230 و LPC3250 گنجانده شده است.

ارتباطات سریال گوناگونی برای این میکروکنترولرها فراهم آمده است. میکروکنترولرهای LPC32x0 دارای یک مبدل ADC ۱۰ بیتی سه کاناله با سرعت تبدیل 400 KHz و یک رابط صفحه‌ی لمسی می‌باشد. ساختار گذرگاه ۷-لایه‌ی ۳۲ بیتی و 104 MHz گذرگاه AHB برای هر یک از فرماندهان گذرگاه[۱]، یک گذرگاه جداگانه فراهم آورده است (یعنی D-Cache، I-Cache، دو DMA، کنترل کننده USB و کنترل کننده‌ی LCD).

شمای کلی این میکروکنترولرها را در شکل ۵.۱۷ می‌بینید.

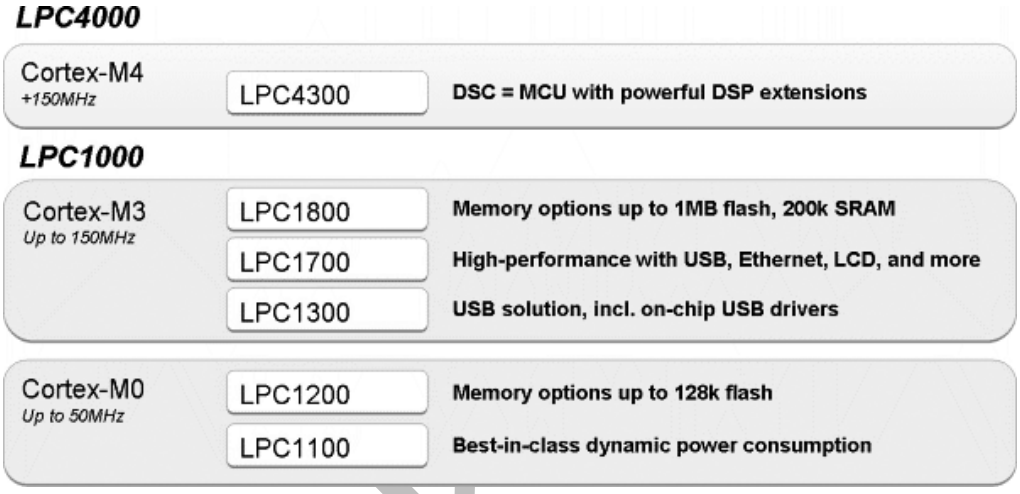
[۱] Bus Master


CPU subsystem
ETB
ETM9
I-Cache 32 kB
D-Cache 32 kB
VFP9
ARM926EJ
Instr
Data
On-chip memory
ROM
256 kB SRAM
External memory interfaces
NAND Flash
SD card
SRAM and SDRAM (SDR/DDR) control
Bus matrix (multi-layer AHB)
Interrupt controller
DMA contoller
System control
Watchdog
PLLs
High speed timer
Power control
Millisecond timer
Timer 0, 1, 2, 3
System functions
LCD
GPIO
Keyscan
Touch screen/ 10-bit A/D
RTC
PWM 1, 2
PWM 3, 4
Other peripherals
Ethernet MAC
UART 1-5, 7
UART6 IrDA
USB OTG
I²C 1, 2
I²S 0, 1
SPI 1, 2
SSP 0, 1
Communication peripherals

شکل ۵.۱۷

۵.۲.۴ میکروکنترولرهای سری LPC4000

نسل جدید کنترل کننده‌های سیگنال دیجیتال[۱] که براساس هسته‌ی ARM Cortex-M4 شکل گرفته‌اند را در میکروکنترولرهای سری LPC4000 می‌بینیم. معماری Cortex-M4 مشخصاً برای بازار پردازنده-های DSC طراحی شده است. ترکیب این ساختار از مشخصات یکی از بهترین معماری‌های میکروکنترولری، یعنی Cortex-M3 دستورهای DSP خاص به دست آمده است. سری LPC4000 نسل جدید و رو به رشد میکروکنترولرهای براساس Cortex-M شرکت NXP است.



شکل ۵.۱۸

۵.۲.۴.۱ میکروکنترولرهای سری LPC43xx

مشخصات کلیدی:

- هسته‌ی ARM Cortex-M4 ۳۲ بیتی با فرکانس کاری 150 MHz؛
- کمک پردازنده‌ی ۳۲ بیتی نامتقارن ARM Cortex-M0 با فرکانس کاری 150 MHz؛
- دو بانک بزرگ حافظه‌ی فلش با اندازه‌ی 1 MB ؛
- وسایل جانبی ابداعی و جدیدی شامل واسط حافظه‌ی فلش SPI، زمان سنج با حالت قابل تنظیم (SCT) و GPIO سریال؛

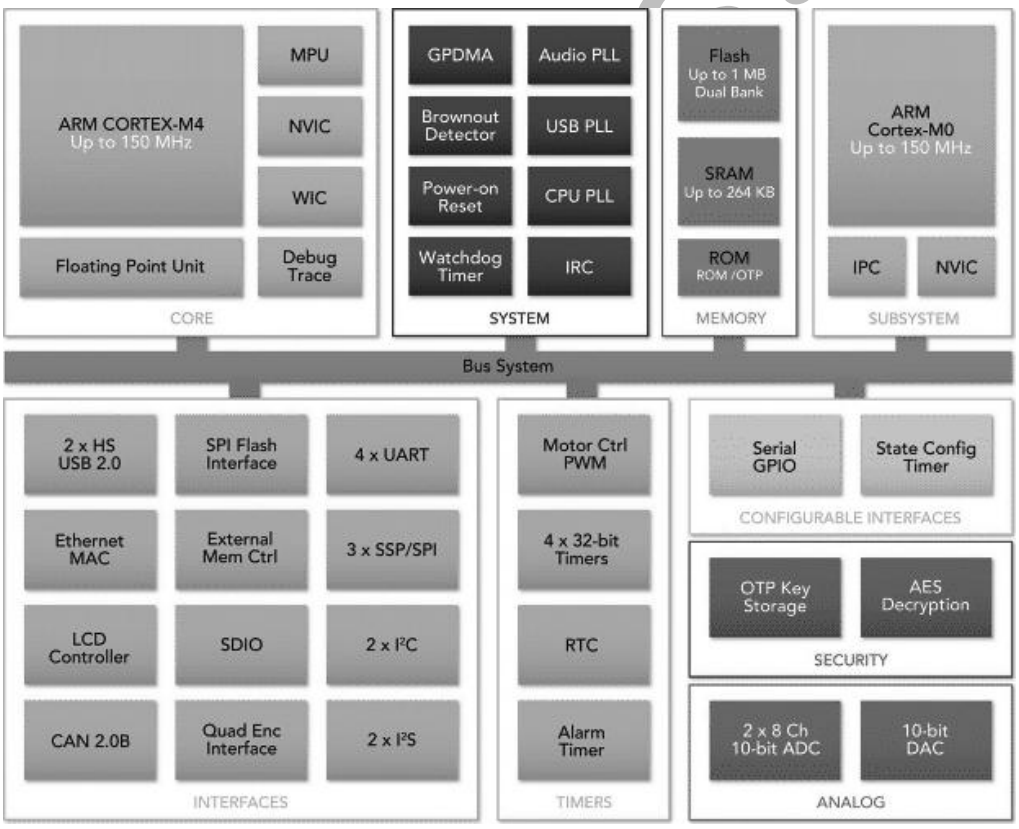
---

[۱] Digital Signal Control - DSC

- دو عدد USB سرعت بالا (یکی از آن‌ها دارای PHY بر روی تراشه است) ؛
- واحد محافظت از حافظه (MPU).

LPC43xx اولین معماری کنترل کننده سیگنال دیجیتال دو هسته‌ای نامتقارن است که از Cortex-M4 و Cortex-M0 استفاده می‌کند. این تراشه‌ی مزایای پردازش سیگنال دیجیتال و یک میکروکنترولر را در یک بسته‌بندی جای می‌دهد. پردازنده‌ی Cortex-M4 مزایای یک میکروکنترولر را با فناوری پردازش سیگنال دیجیتال کارآمدی که از MAC تک‌سیکل، دستورهای SIMD دستورهای ریاضی اشباع و واحد نقطه‌ی اعشاری بهره می‌برد، تلفیق می‌کند. کمک پردازنده‌ی Cortex-M0 بسیاری از عملیات نقل و انتقال‌های داده و وظایف I/O را از عهده‌ی Cortex-M4 برمی‌دارد و بدین وسیله پهنای داده بیشتری را برای پردازش، در اختیار Cortex-M4 قرار می‌دهد.

با مشخصات گفته شده و ساختار دو هسته‌ای LPC43xx، این میکروکنترولر در بسیاری کاربردها چون کنترل موتور، مدیریت توان، اتوماسیون صنعتی، روبوتیک و... به کار گرفته می‌شود.



شکل ۵.۱۹

۵.۳ میکروکنترولرهای شرکت Texas Instrument Inc

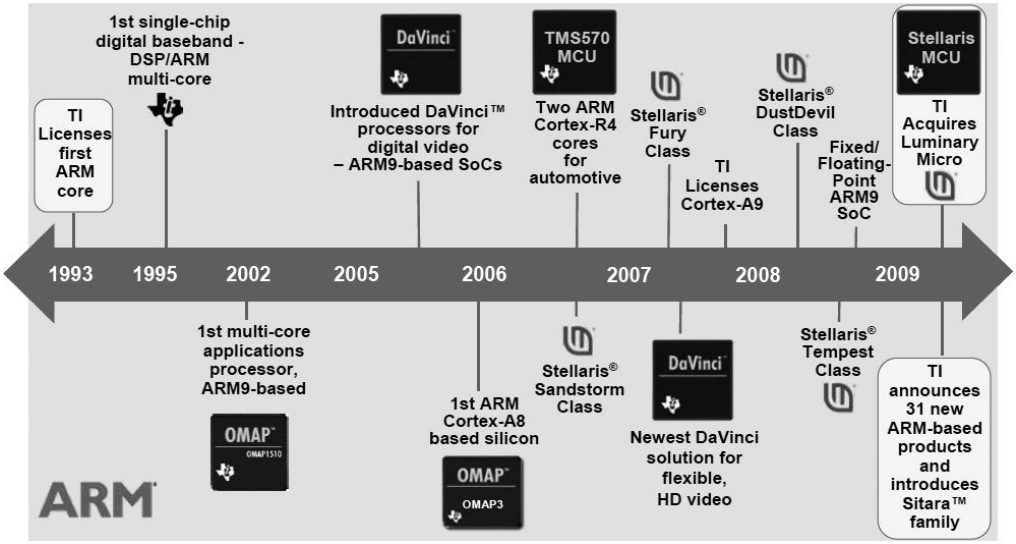
شرکت Texas Instrument Inc که در بسیاری از موارد با نام TI شناخته می‌شود، شرکتی آمریکا واقع در Dallas ایالت Texas آمریکا است. این شرکت به خاطر توسعه و تجاری سازی فناوری کامپیوتر و نیمه هادی بسیار شناخته شده است.

TI ، چهارمین تولید کننده‌ی نیمه هادی در دنیا بعد از شرکت‌های Intel، SAMSUNG و Toshiba، دومین تولید کننده‌ی تراشه‌های تلفن همراه بعد از Qualcomm و اولین تولید کننده‌ی تراشه‌های پردازنده سیگنال دیجیتال (DSP) و قطعات آنالوگ در بین بسیاری از تولید کنندگان مطرح نیمه هادی در سطح دنیا است. در بهار سال 1986 این شرکت نام اینترنتی TI.Com را برای خود برگزید و در سال 2010 ، جایگاه ۲۳۳ را در لیست Fortune 500 کسب نمود.

تاریخچه شرکت TI

در سال 1947 ، شرکت TI از شرکت مادری به نام GSI جهت تولید ترانزیستور (که اخیراً اختراع شده بود) شکل گرفت. آقای MC Dermott ، یکی از پایه گذاران شرکت GSI در سال 1930، نیز در هیئت مدیره شرکت تازه تأسیس حضور داشت. نام شرکت در ابتدا General Instrument Inc انتخاب شد. در همان سال، این نام را به علت مشابهت با نام شرکتی دیگر، به Texas Instrument تغییر دادند.

شرکت TI از تولید کنندگان حرفه‌ای میکروکنترولرهای ARM می‌باشد. تولیدات این شرکت طیف گسترده‌ای از محصولات بر مبنای ARM شامل می‌شود که عمدتاً بر پایه‌ی ARM9، ARM-CortexA استوار هستند. این شرکت در بعضی از تراشه‌ها از پردازنده‌های ARM در کنار پردازنده‌های سیگنال (DSP) استفاده کرده است.



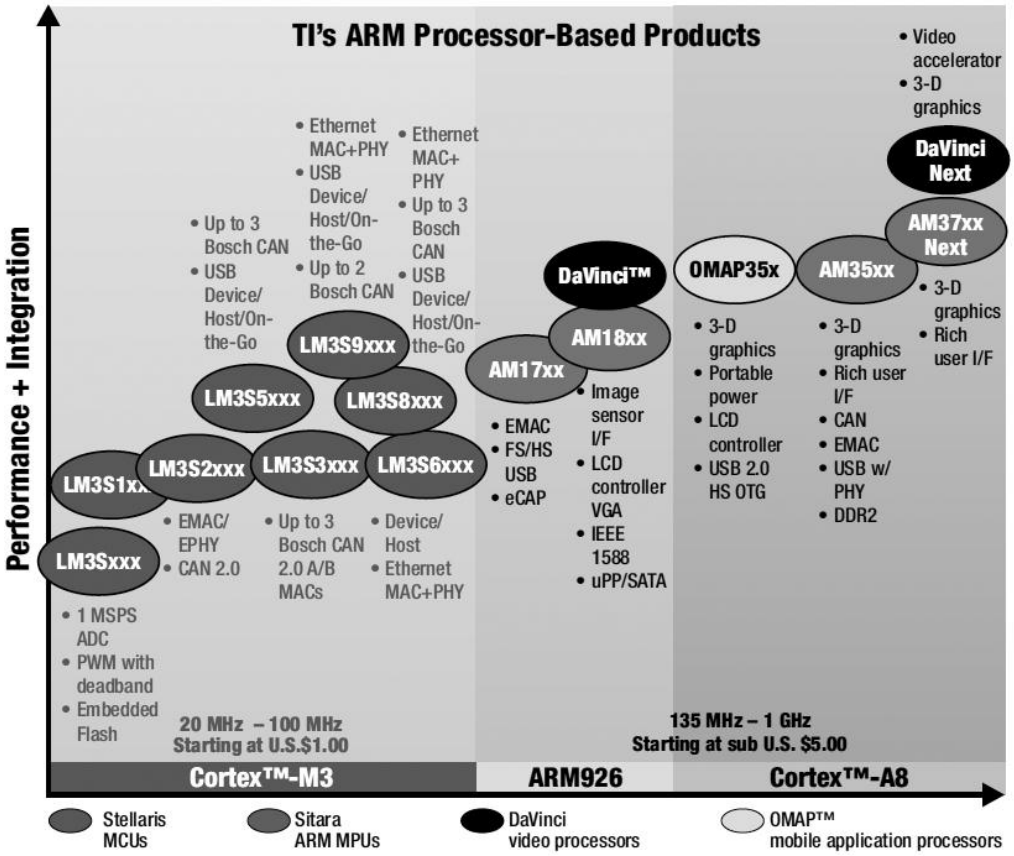
شکل ۵.۲۰ – گذری کلی بر تاریخچه‌ی تولیدات بر مبنای ARM

در شکل ۵.۲۰ ، گذری کلی بر تاریخچه‌ی تولیدات بر مبنای ARM این شرکت را می‌بینیم. شرکت TI در سال 2009، شرکت Luminary Micro را خرید و محصولات ARM این شرکت را نیز در زمره‌ی محصولات خود قرار داد.

هم‌اکنون تولیدات شرکت TI درزمینه‌ی میکروکنترولرهای ARM، شامل چهار گروه: DaVinci، OMAP، Sitara و Stellaris می‌شود.

میکروکنترولرهای Stellaris بسیار شناخته شده هستند و براساس ARM Cortex M3 ساخته شده‌اند. این میکروکنترولرها سابقه درخشان خود را مدیون شرکت پیشین (یعنی Luminary Micro) هستند. محصولات خانواده‌ی Sitara که به تازگی پا بر جهان گذاشته‌اند، راه‌حلی کارآمد را ارایه می‌دهند. پردازنده‌های معروف OMAP در تلفن‌های همراه بسیار کاربرد داشته‌اند.

پردازنده‌های Davinci که براساس ARM و یا ترکیبی از ARM و DSP ارایه شده‌اند، در کاربردهای ویدیویی و پردازش تصویر بسیار استفاده می‌شوند.



شکل ۵.۲۱ – انواع پردازنده‌های ARM

۵.۳.۱ خانواده‌ی پردازنده‌های Sitara

پردازنده‌های جدید Sitara شامل پردازنده‌های ARM9 و یا ARM Cortex A8 هستند. سرعت کاری آن‌ها نیز بین 395 MHz و 1 GHz است. خانواده‌ی Sitara با خانواده‌ی معروف OMAP دارای پایه‌ی نرم‌افزاری یکسان هستند که این به معنای آسانی انتقال کد، بین خانواده‌ها و استفاده آسان از نرم‌افزارهای از پیش نوشته شده (برای OMAP) است.

این پردازنده نیز همانند پردازنده‌های OMAP دارای مصرف توان بهینه‌ای هستند. از نقاط کلیدی پردازنده‌های Sitara همین مصرف توان کم‌شان است.

دوتا از اولین پردازنده‌های خانواده‌ی Sitara براساس ARM Cortex A8 ساخته شده‌اند. این دو عبارت‌اند از: AM3505 و AM3517. این دو پردازنده دارای گستره‌ای وسیع از ارتباطات جانبی برای اتصال به وسایل مختلف هستند. عده‌ای از مهندسان نیز برای گسترش و استفاده از سیستم‌عامل لینوکس بر روی آن‌ها، فعال هستند.

۵.۳.۱.۱ مشخصات کلیدی AM3505

- پردازنده سوپر اسکالر 500 MHz براساس ARM Cortex A8 که توانایی ارایه‌ی 1000 Dhryston MIPS در هر ثانیه را داراست؛
- کنترل کننده‌ی CAN داخلی که توانایی کنترل سنسورها و کنترل کننده‌ها را دارد؛
- ارایه پردازنده در نمونه صنعتی (محدوده‌ی دمایی بین 105° و 40°-)، به طراحان اجازه طراحی در محیط‌های صنعتی سخت را نیز می‌دهد؛
- مصرف توان زیر 1 W به معنای عدم نیاز به فن است. پس، سیستم طراحی شده می‌تواند در یک محیط بسته و بدون نیاز به فن و هیت سینک[۱] ساخته شود. یعنی، دستگاهی که هم ساکت است و هم گردوغبار، عملکرد آن را مختل نمی‌کند؛
- امکانات ارتباطی شامل USB OTG سرعت بالا به همراه PHY روی تراشه، واسط EMAC با سرعت 10/100 برای ارتباط شبکه و کنترل دستگاه‌های صنعتی، شامل کنترل از راه دور از طریق شبکه برای مانیتور و کنترل از راه دور دستگاه‌ها می‌باشد؛
- پشتیبانی از حافظه‌ی DDR2، هزینه‌ی کلی سیستم را کاهش می‌دهد؛
- پشتیبانی از ورودی/ خروجی‌های با سطح ولتاژ 3.3 V کار ارتباط با وسایل جانبی را بسیار راحت کرده است. نیازی به هیچ‌گونه تراشه‌ی انتقال سطح ولتاژ نیست؛

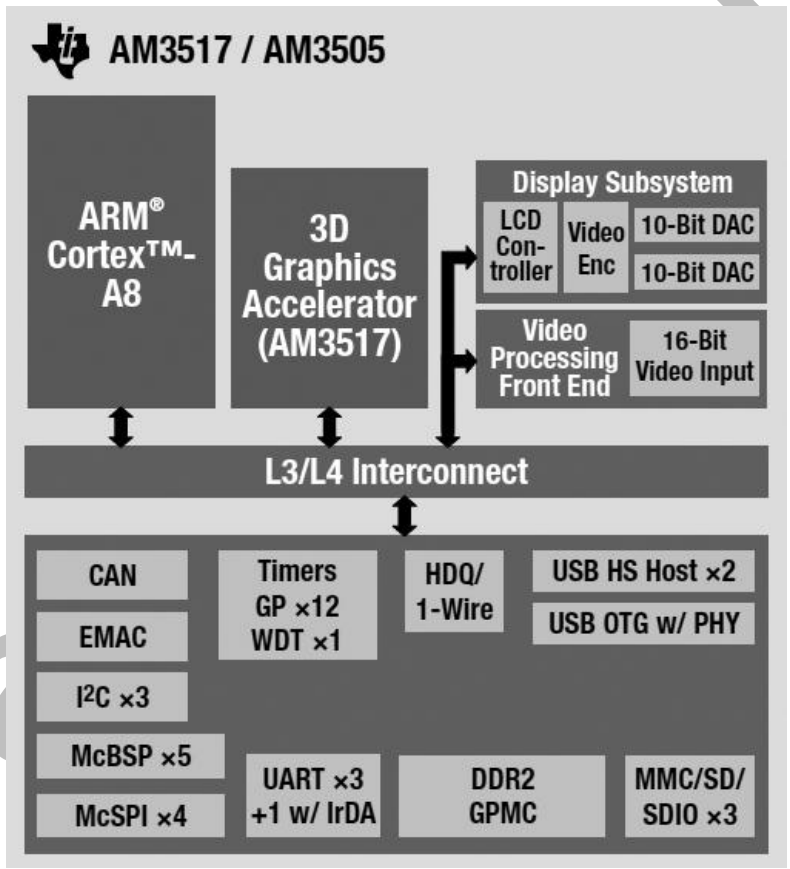
---

[۱] Heat Sink

- زیرسیستمی برای پشتیبانی از قابلیت‌های حرفه‌ای نمایشی که توانایی نمایش تصـویر در تصـویر، تبدیل فضای رنگ، چرخاندن و تغییر اندازه‌ی تصویر برای نمایشگرهای LCD را دارد.

۵.۳.۱.۲ مشخصات کلیدی AM3517

- این تراشه تمام مشخصات AM3505 را دارد و علاوه بر آن، یک موتور گرافیکی Power VR™ SGX را نیز داراست. این قابلیت اضافه به راحتی امکان افزودن یک واسط کاربر (HMI) بـا رابطـی فعال و زیبا را می‌دهد. ایـن شـتاب‌دهنـده، توانـایی پـردازش 10 Mpolygon/Sec را داشـته و از Open GL® ES2.0 پشتیبانی می‌کند.



شکل ۵.۲۲

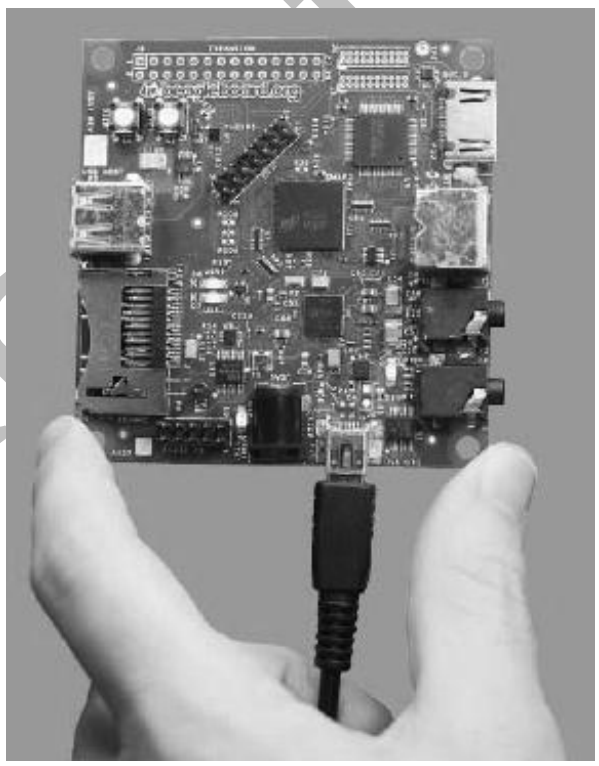
۵.۳.۲ پردازنده‌ی برنامه‌های کاربردی، OMAP

پردازنده‌های OMAP35x به عنوان اولین محصول تجاری که از ARM-Cortex A8 استفاده می‌کرد مطرح شده بود. این پردازنده تا چهار برابر کارآیی پردازنده‌های ARM9 کنونی را ارایه می‌داد و در حقیقت از یک پردازنده‌ی Cortex A8 به همراه یک پردازنده‌ی DSP به شماره‌ی TMS320C64x+ تشکیل شده بود. چهار پردازنده در این خانواده موجودند که عبارت‌اند از: OMAP3503، OMAP3515، OMAP3525 و OMAP3530.

مشخصات کلیدی

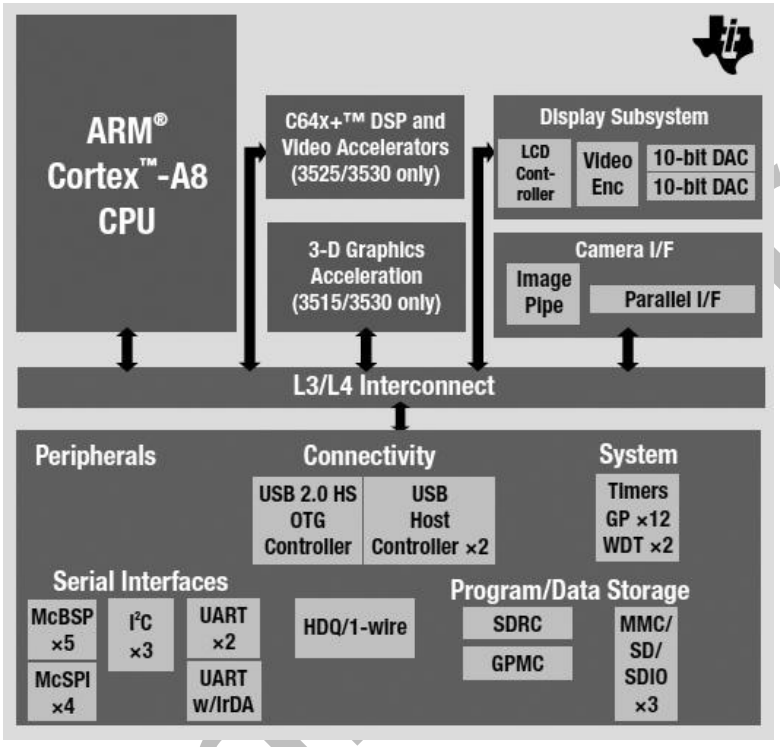
- کارآیی بهینه همانند لپ‌تاپ که دارای مصرف توان کم و مناسب برای دستگاه‌های دستی است؛
- استفاده از تکنولوژی Smart Reflex برای ذخیره سازی هرچه بیشتر توان؛
- امکان بهره‌برداری از موتورهای گرافیکی نرم‌افزاری همانند Open GL ES2.0.

بردی به نام Beagle Board ، یک طراحی باز است که تمامی مواد اولیه‌ی طراحی را در دسترس گذاشته است. این برد، با ابعاد 3-inch، ارزان قیمت و قابلیت تغذیه از طریق USB عرضه می‌شود. این برد، بسیار کاربردی شده و کاربران زیادی در سراسر دنیا از آن استفاده می‌کنند.



شکل ۵.۲۳ – Beagle Board

شمای کلی OMAP35x را در شکل ۵.۲۴ می‌بینیم.



شکل ۵.۲۴

بدین ترتیب ارایه‌ی اولین نمونه‌ها از پردازنده‌های OMAP و کسب موفقیت در بازار مصرف، شرکت TI پردازنده‌های OMAP را در دسته‌بندی‌های جدید قرار داد. دسته‌بندی کامل آن‌ها عبارت‌اند از:

۵.۳.۲.۱ مشخصات کلیدی OMAP4

- برای راه‌اندازی تلفن‌های هوشمند، تبلت‌ها[۱] و دیگر دستگاه‌های غنی از امکانات چندرسانه‌ای؛
- شتاب‌دهنده‌های سخت‌افزاری IVA3 برای استفاده از تصاویر Full HD 1080p، رمز کردن/ رمز گشایی چنداستانداردی تصاویر؛
- گرفتن تصاویر ویدیویی و ثابت SLR-Like تا رزولوشن حداکثر تا 20 Mega Pixels؛
- پردازنده‌ی دوهسته‌ای ARM Cortex A9 با سیستم چندپردازشی متقارن (SMP)؛

[۱] Tablet

- شتاب‌دهنده‌ی گرافیکی Power VR SGX540 با قابلیت راه‌اندازی بازی‌ها و واسط کاربری ۳ بعدی؛
- پلتفرم قابل حمل بسیار بهینه؛
- OMAP4430 با فرکانس حداکثری 1 GHz؛
- OMAP4440 با فرکانس حداکثری 1.5 GHz.

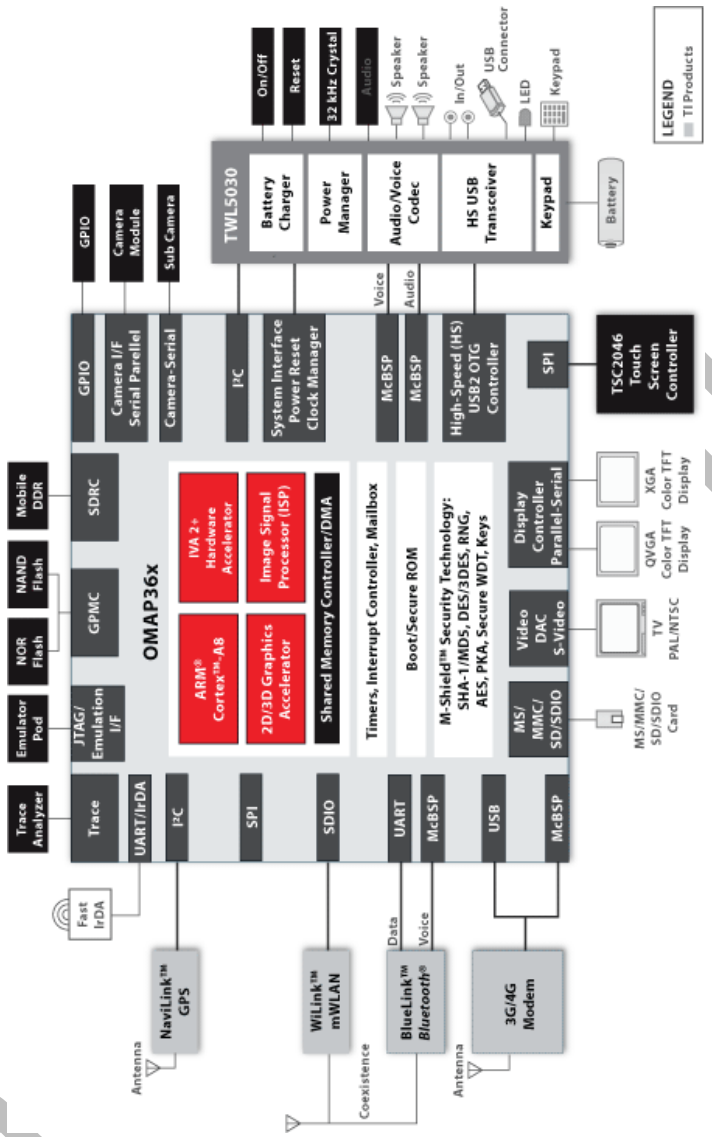
OMAP44x
ARM® Cortex™-A9 MPCore™
ARM® Cortex™-A9 MPCore™
IVA 3 Hardware accelerator
POWERVR™ SGX540 graphics accelerator
Image Signal Processor (ISP)
Shared memory controller/DMA
Timers, Interrupt controller, mailbox
Boot/secure ROM
M-Shield™ Security Technology: SHA-1/MD5, DES/3DES, RNG, AES, PKA, secure WDT, keys
GPIO
MIPI™ CSI-2
MIPI™ CSI-2
I²C
PDM
SPI
Touch screen controller
Display controller parallel-serial
WUXGA
HDMI™
TPD12S015
HD television
eMMC/MMC/SD
eMMC
MMC/SD card
REF/CLK
CDC3S04 clock driver
I²C
SDIO
UART
McBSP
USB
McBSP
HSI
Fast IrDA
NaviLink™ GPS
WiLink™ mWLAN
Coexistence
BlueLink™ Bluetooth®
3G/4G Modem
Trace
JTAG/ Emulation
FLASH controller
SDRAM controller
Digital MIC
Keypad
High-Speed USB 2 OTG
SIM
Trace analyzer
Emulator pod
NOR flash
NAND flash
LPDDR2
Keypad
USB
SIM card MIM card
GPIO
Camera
Sub camera
TWL6030
Power
Monitor
Charger
Main battery
TWL6040
Audio
Headset
Speakers
Vibrators
Amplifiers
Micro
32 kHz Crystal
In/Out
HF Speakers
Handset microphone

شکل ۵.۲۵

OMAP36x ۵.۳.۲.۲

معماری اثبات شده‌ی OMAP3 کارآیی بالایی به همراه قابلیت‌های سرگرم کننده‌ی زیاد برای نسل بعدی تلفن‌های همراه، در اختیار می‌گذارد. این خانواده در فرکانس بالاتری نسبت به پردازنده‌های قبلی OMAP کار کرده و نسبت به ARM11 تا حد ۳ برابر افزایش کارآیی را به همراه دارد.
واسط تصویری Power VR SGX ۲ و ۳ بعدی به کاربران و برنامه نویسان امکانات حرفه‌ای و بی-شماری را ارایه می‌دهد.
با کارکردهای حرفه‌ای چندرسانه‌ای، از پخش کامل و چنداستانداردی تصاویر تا رزولوشن حداکثری 720 pHD پشتیبانی می‌کند. این تراشه همچنین از نمایشگرهای عریض W×GA و نمایشگرهای با چند LCD نیز پشتیبانی می‌کند.
پردازنده‌ی سیگنال تصویر[۱] برای اتصال مستقیم سنسورهای با رزولوشن حداکثر 12 MP طراحی شده است.

[۱] Image Signal Processor-ISP


OMAP36x
ARM® Cortex™-A8
IVA 2+ Hardware Accelerator
2D/3D Graphics Accelerator
Image Signal Processor (ISP)
Shared Memory Controller/DMA
Timers, Interrupt Controller, Mailbox
Boot/Secure ROM
M-Shield™ Security Technology: SHA-1/MD5, DES/3DES, RNG, AES, PKA, Secure WDT, Keys
GPIO
Camera I/F Serial Parellel
Camera-Serial
I²C
System Interface Power Reset Clock Manager
McBSP
McBSP
High-Speed (HS) USB2 OTG Controller
SPI
SDRC
GPMC
JTAG/ Emulation I/F
Trace
UART/IrDA
I²C
SPI
SDIO
UART
McBSP
USB
McBSP
Display Controller Parallel-Serial
Video DAC S-Video
MS/ MMC/ SD/SDIO
Voice
Audio
TWL5030
Battery Charger
Power Manager
Audio/Voice Codec
HS USB Transceiver
Keypad
On/Off
Reset
32 kHz Crystal
Audio
Speaker
Speaker
In/Out
USB Connector
LED
Keypad
Battery
TSC2046 Touch Screen Controller
XGA Color TFT Display
QVGA Color TFT Display
TV PAL/NTSC
MS/MMC/ SD/SDIO Card
Mobile DDR
NAND Flash
NOR Flash
Emulator Pod
Trace Analyzer
Fast IrDA
NaviLink™ GPS
Antenna
WiLink™ mWLAN
Coexistence
BlueLink™ Bluetooth®
Data
Voice
3G/4G Modem
Antenna
GPIO
Camera Module
Sub Camera
LEGEND
TI Products

شکل ۵.۲۶

جدول TI-2

۵.۴ شرکت STMicroelectronics و میکروکنترولرهای ARM

شرکت STMicroelectronics ، یک شرکت ایتالیایی- فرانسوی فعال در زمینه‌ی تولیدات الکترونیک و نیمه هادی است. این شرکت، در سال ۱۹۸۷ با ادغام دو شرکت SGS Microelettronica ایتالیا و Thomson Semiconducteurs کشور فرانسه شکل گرفت. با گذشت چندین دهه از فعالیت شرکت ST و انجام فعالیت های مختلف با شرکت های تولید کننده نیمه هادی، اکنون این شرکت به یکی از بزرگ‌ترین تولید کننده های تراشه های سیلیکونی در زمینه های مختلفی چون الکترونیک، ارتباطات، ... مبدل گشته است. بر خلاف بسیاری از تولید کنندگان بدون کارخانه سازنده نیمه هادی[۱]، ST ویفر های سیلیکونی[۲] اش را خود تولید می کند. این شرکت، در سال ۲۰۰۶ دارای ۴ کارخانه تولید ویفر های ۸ اینچ (200 mm) و ۱ کارخانه تولید ویفر های ۱۲ اینچ (300 mm) بود و بیشتر محصولاتش نیز با پروسه های ساخت 0.18µm ، 0.13µm ، 90nm و 65nm است[۳].

محصولات این شرکت را می توان به چندین گروه تقسیم کرد:

- ارتباطات چند رسانه ای قابل حمل – محصولات چندرسانه ای، بی سیم، کد کننده ها و دیکود کننده های MPEG-2 و MPEG-4. ASIC های بیسیم، کارت‌های هوشمند و ...؛
- محصولات حافظه – تراشه های حافظه مستقل E^2PROM، Flash (NAND و NOR)، Serial Flash . هم اکنون حافظه های فلش NAND و NOR به شرکت Numonyx واگذار شدند؛
- محصولات صنایع خودرویی – تراشه های انالوگ و دیجیتال برای بازار محصولات خودرویی. محصولات سرگرمی، رادیویی و میکروکنترولرهای کنترل وسایل داخلی خودرو؛
- گروه آنالوگ، منبع تغذیه و میکروکنترولر – مدارات آنالوگ، منبع تغذیه و میکروکنترولرهای خانواده‌ی STM : STM8, STM32, STR9, STR7, ST10, ST9, µPSD, ST7, ST6 . تولیدات قطعات نیمه هادی گسسته؛

---

[۱] Fabless Semiconductor Companies

[۲] Silicon Wafer

[۳] این ابعاد، کمتری‌ن اندازه گی‌ت ترانزی‌ستورها در ی‌ک قطعه نی‌مه هادی است.

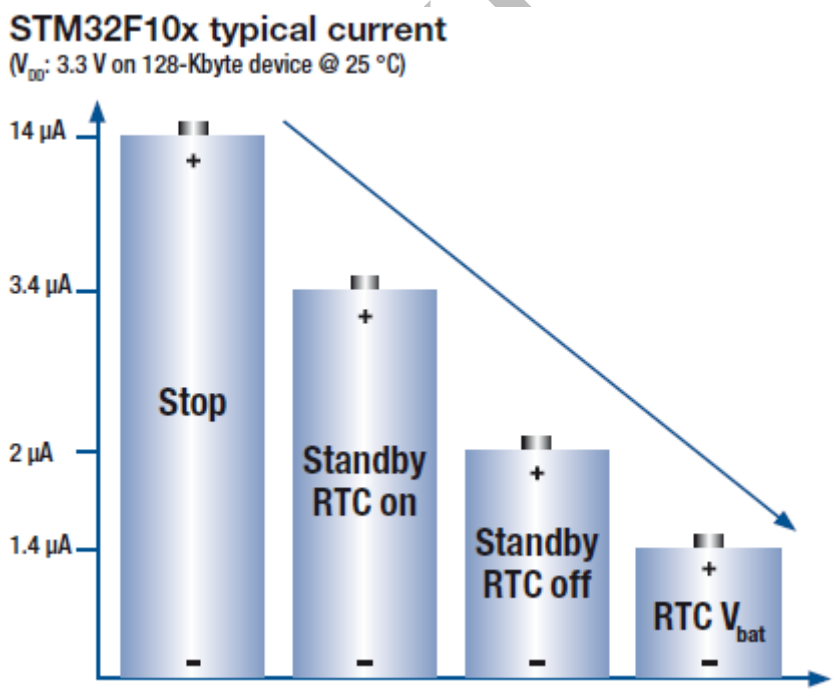
- وسایل جنبی رایانه.

میکروکنترولر های این شرکت به چندین گروه تقسیم می شوند که از هسته های ARM7 و ARM9 و ARM Cortex-M3 استفاده می کنند:

۱. خانواده‌ی STR7 ، هسته‌ی ARM7TDMI را با مجموعه ای فراگیر از وسایل جانبی و یک حافظه‌ی فلش 0.18µm ترکیب می کند و با اسامی STR710 ، STR730 و STR750 به بازار ارایه کرده است؛

۲. پلتفرم STR9 بر اساس هسته‌ی پردازنده‌ی ARM9E™ ساخته شده است. این خانواد، کدها را از روی حافظه‌ی فلش داخلی و تنها در یک سیکل اجرا می کند. از وسایل جانبی این پردازنده می‌توان به Ethernet ، CAN و USB اشاره کرد؛

۳. خانواده میکروکنترولرهای STM32L بر اساس هسته‌ی ARM Cortex-M3 و با استفاده از تکنولوژی‌های **مصرف توان بسیار کم**[۱] ساخته شده اند. این میکروکنترولرها با بهره گیری از یک معماری بسیار کارآمد به همراه تکنولوژی "مصرف توان بسیار کم" مورد استفاده در میکروکنترولر های STM8L یک سری از تراشه های بسیار کارامد و در عین حال بسیار کم مصرف را ارایه کرده اند.



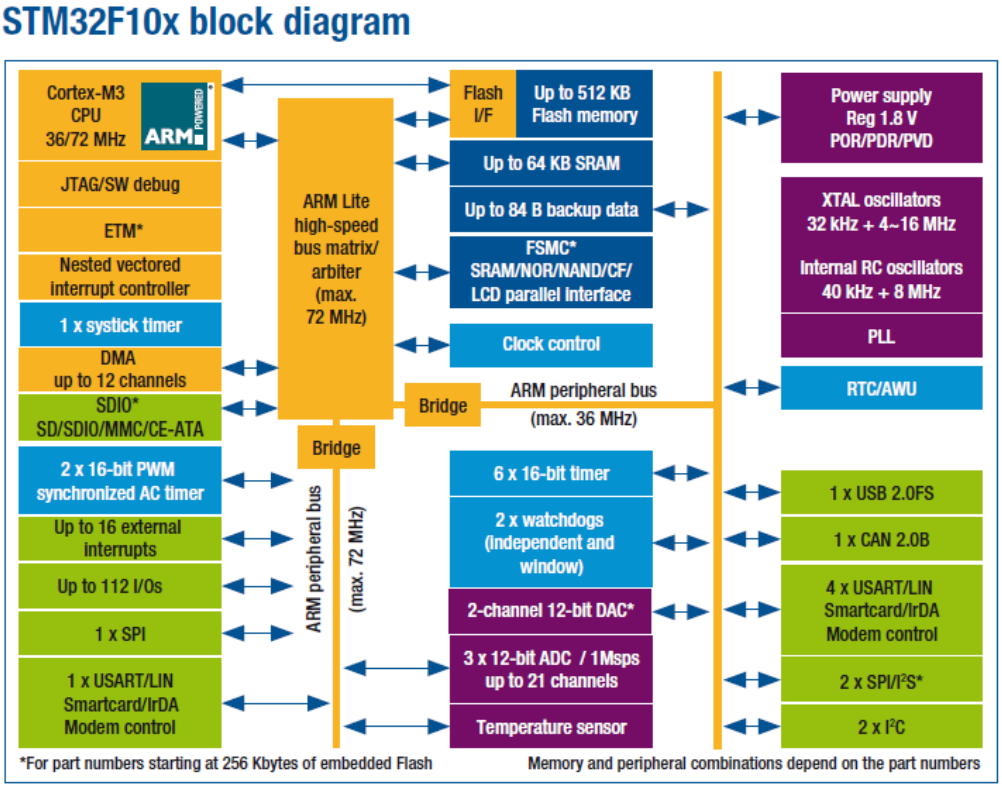
[۱] Ultra Low Power

شکل ۵.۲۷ – توان مصرفی در حالت‌های مختلفِ کاری

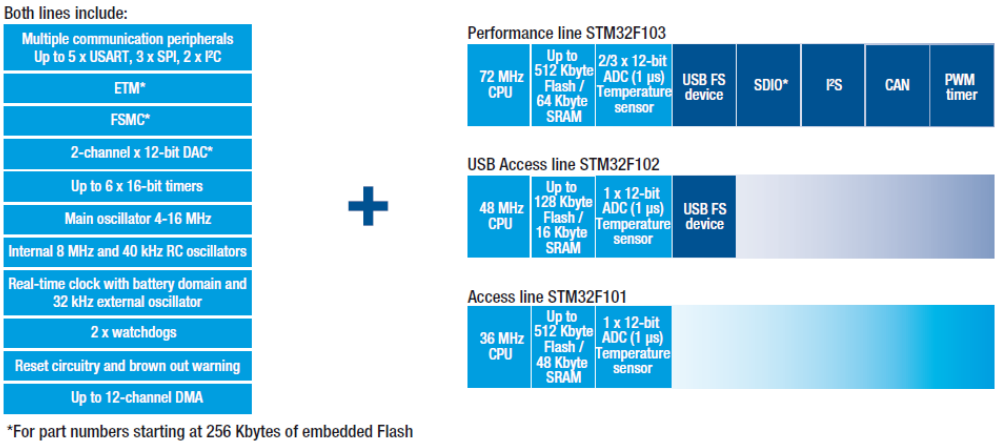
خانواده STM32L ، در دو سری STM32L151 و STM32L152 تولید می شوند. این خانواده دارای قابلیت های زیر هستند:

- دارای حالت کمترین توان مصرفی: 0.27µA
- توان مصرفی در حال اجرا: 230µA/MHz؛

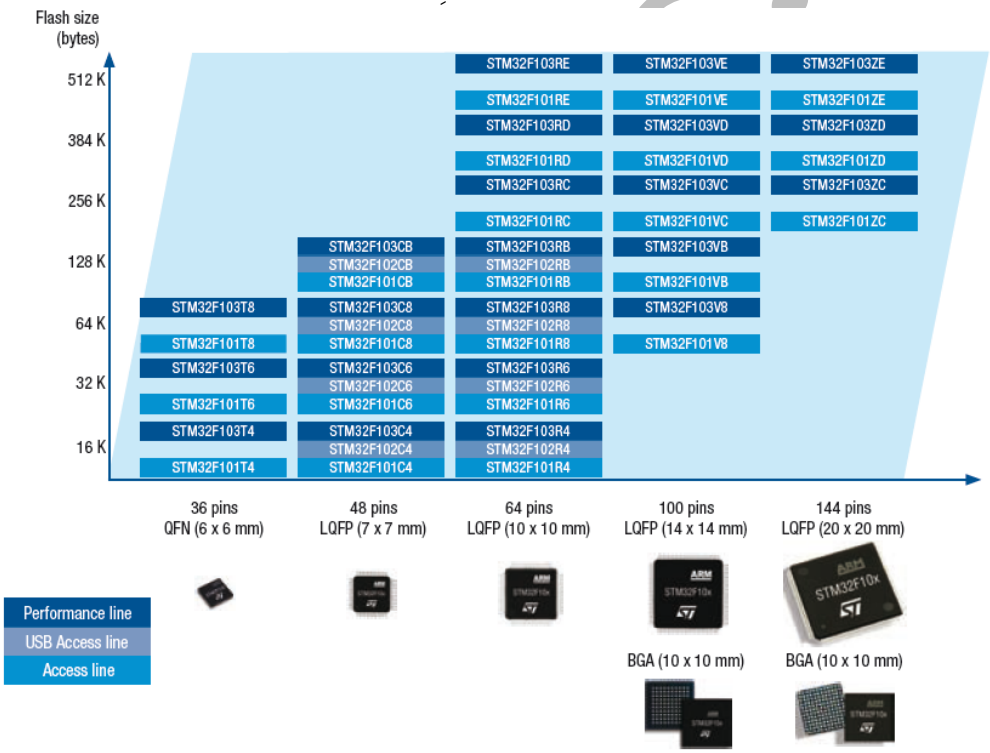
۴. میکروکنترولرهای STM32F بنیادگذار خانواده‌ی بزرگ STM32 بودند. سری میکروکنترولرهای ۳۲ بیتی STM32F با حافظه‌ی فلش داخلی از هسته‌ی قدرتمند ARM Cortex-M3 بهره می برند. این میکروکنترولرها کارایی بالا را با مصرف توان کم و ولتاژ پایین به همراه مجموعه ای گسترده از بهترین وسایل جانبی در هم آمیخته اند. علاوه بر آن، بیشترین مجتمع سازی در عین قیمت قابل رقابت و ارایه ابزار ها توسعه گوناگون، این میکروکنترولر ها را برای استفاده در وسایل گوناگون بسیار مناسب ساخته است.
میکروکنترولرهای STM32F1xx در پنج خط تولید به بازار فروش ارایه شده اند.
دیاگرام بلوکی این میکروکنترولرها را در شکل زیر میبینیم.



شکل ۵.۲۸ – دیاگرام بلوکی STM32F10x

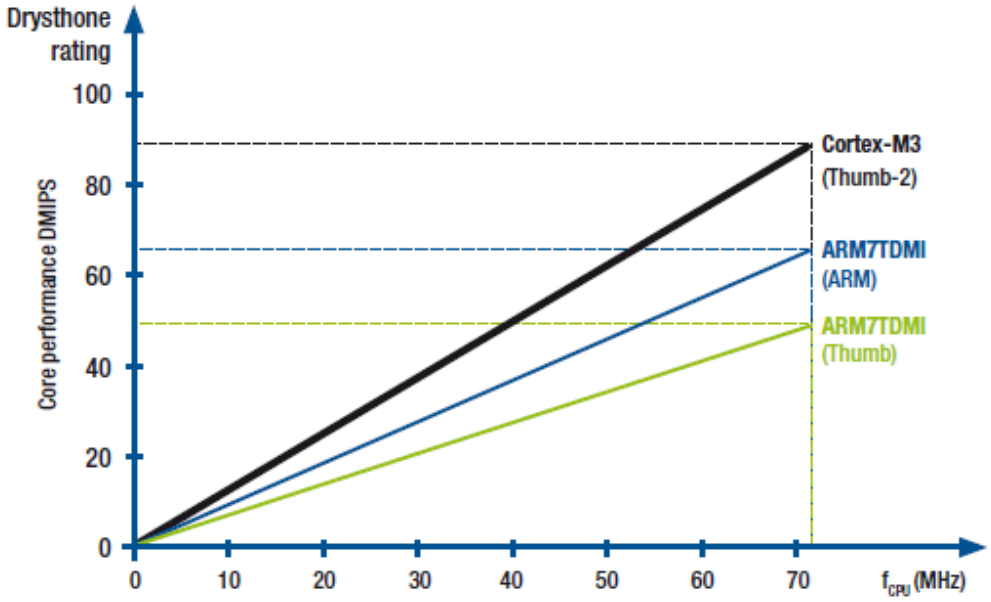


شکل ۵.۲۹ – سه خط تولیدیِ STM32F10x



شکل ۵.۳۰ – پردازنده‌های مختلف STM32F10x

در شکل زیر، کارایی هسته Cortex M3 مورد استفاده در میکروکنترولرهای STM32 ، با نمونه‌های ARM7TDMI در فرکانس‌های کاری مختلف، مقایسه شده است.



شکل ۵.۳۱ – نمودار کارآیی Cortex-M3

۵.۵ خلاصه

در این فصل، نگاه نسبتاً کاملی به خانواده‌های مختلف میکروکنترولرهای ARM داشتیم. امروزه میکروکنترولرهای ARM تولید شرکت‌های Atmel، NXP، ST، TI و ... از جمله معروف‌ترین و پرکاربردترین میکروکنترولرهای بسیار مجتمع و کاربردی بازار الکترونیک می‌باشند.
با پایان این فصل در انتخاب میکروکنترولر خود با مشخصات دلخواه‌تان مشکلی نخواهید داشت. در فصول بعد به بررسی وسایل داخلی این میکروکنترولرها خواهیم پرداخت. مثال‌های گوناگونی ارایه خواهند شد و کار با آن‌ها برایتان آسان خواهد گردید.

فصل ششم

وسایل جانبی میکروکنترولرهای ARM

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ مدارات داخلی کنترل کننده‌ی سیستم در میکروکنترولرهای AT91؛
- ✓ GPIO در میکروکنترولرهای AT91؛
- ✓ ریست و انواع آن در میکروکنترولرهای AT91؛
- ✓ GPIO و نوسان‌سازها در میکروکنترولرهای LPC؛
- ✓ پیکره‌بندی SPI، USART، USB و CAN در LPC ها؛
- ✓ بررسی حالات کم مصرف LPC؛
- ✓ کنترل‌کننده وقفه برداری یا VIC؛
- ✓ بررسی وسایل جانبی گوناگون در میکروکنترولرهای STM32 به همراه مثال.

در فصل قبل، با تعداد گسترده‌ای از میکروکنترولرهای مختلف آشنا شدید. شاید این فهرست بزرگ شما را کمی گیج کند. شاید گمان می‌کنید که کار با این تعداد پردازنده‌های کنترلی چگونه از عهده‌تان بر می-آید؟ البته شاید نیازی نباشد که با همه آن‌ها کار کنید و در همه‌شان متبحر شوید. ولی، همان‌طور که در مقدمه کتاب گفته شد، هدف این‌است که خواننده با نحوه کار و "اسلوب فراگیری" این میکروکنترلرها آشنا شود.

با این روش، خواننده احساس خوبی از کار کردن با کلیه میکروکنترولرهای ARM خواهد داشت و به راحتی می‌تواند به یادگیری میکروکنترولرهای گوناگون (از انواع ساده تا پیشرفته) ساخت تولیدکنندگان مختلف بپردازد. به همین دلیل در این فصل به بیان مختصری از بعضی مشخصات تراشه‌های معروف بسنده خواهیم کرد و از ورود به بسیاری از جزییات (شاید نامربوط)، خودداری خواهیم کرد.

در بعضی موارد نیز، با ارایه مثال به آموزش نحوه راه‌اندازی وسایل جانبی خواهیم پرداخت.

۶.۱ پیکره‌بندی میکروکنترولرهای ARM

در این قسمت با ارایه چندین نکته‌ی کلیدی و قوانین کلی، نحوه پیکره‌بندی و راه‌اندازی وسایل جانبی را فراخواهیم گرفت.

۶.۱.۱ دست‌رسی بیتی

در میکروکنترولرهای ARM دستکاری بیتی ثبات‌ها نسبت به قبل، آسان‌تر شده است. به عنوان مثال برای '0' کردن یک پایه‌ی GPIO ، باید بیت متناظر با شماره‌ی پایه را در ثبات مربوط به خودش، به '1' تنظیم کنیم. همینطور، برای '1' کردن‌اش نیز باید بیت متناظر را در ثباتی دیگر، به '1' تنظیم کرد.
بسیاری از ثبات‌ها به این ترتیب آرایش یافته‌اند. بدین معنا که یک ثبات ۳۲ بیتی برای تنظیم به '1' و ثبات ۳۲ بیتی دیگر برای تنظیم به '0' در نظر گرفته شده‌اند. شایان ذکر است که این نحوه‌ی آرایش، معمولا در قسمت‌های GPIO وجود دارند.

۶.۱.۲ کنترل توان مصرفی

در میکروکنترولرهای ARM خصوصیت دیگری برای کنترلِ میزانِ توانِ مصرفی تراشه گنجانده شده است. این خصوصیت، توانایی کنترل کلاک برای هر وسیله‌ی جانبی، به صورت جداگانه، می‌باشد. بدین معنا که ما می‌توانیم کلاک هر زیر-بخش را قطع کرده و یا آن را کم و زیاد کنیم.
همانطور که می‌دانیم، در مدارات CMOS، توان مصرفی با فرکانس سوییچ ترانزیستورها رابطه مستقیم دارد. پس در میکروکنترولرهای ARM که دارای طیف گسترده‌ای از وسایل جانبی هستند، نیاز قطعی به یک چنین مکانیزمی برای کنترل توان مصرفی داریم. بسیاری از وسایل جانبی همانند: CAN، USB، EMAC و ... بعد از وصل کردن تغذیه‌ی میکروکنترولر، خاموش هستند. برای روشن کردن این وسایل باید کلاک مربوطه را از طریق ثبات‌های مخصوص این کار، تنظیم کرد.

۶.۱.۳ ریست[۱] و کنترل‌کننده‌ی ریست

ریست در یک سیستم کامپیوتری، به معنای پاک کردن هرگونه حالت خطای پیش آمده در سیستم و بازگرداندن آن به حالت اولیه و طبیعی است. این فرآیند معمولاً کنترل شده و طبق اصول خاصی صورت می‌گیرد. ریست، معمولاً در شرایطی همچون آغاز به کار سیستم، بعد از رخداد خطا در برنامه، پس از رخداد خطای time-out (خطایی که در آن بعد از گذشتن زمانی مشخص، پردازنده به دستورالعمل پاسخ نمی‌دهد) اجرا می‌شود. ریست، می‌تواند به صورت دستی و یا به شیوه‌ی خودکار (حالتی که در سیستم‌های میکروکنترولری رایج است) اعمال شود.
ریست‌ها، به دو دسته‌ی نرم‌افزاری و سخت‌افزاری تقسیم می‌شوند. ریست‌های سخت‌افزاری، از طریق پایه‌ی مربوط و ریست‌های نرم‌افزاری توسط اجرای دستورالعمل ذی‌ربط اجرا می‌شوند.
ریست، در میکروکنترولرهای ARM ، عملیات کاملی را صورت می‌دهد. در هنگام آغاز به کار و روشن شدنِ میکروکنترولر، یک ریست Power-On رخ می‌دهد. بنابراین، نیازی به قطعات خارجی برای ریست

---

[۱] Reset

ابتدایی میکروکنترولر نیست (هرچند وجود این قطعات اطمینان بیشتری نسبت بـه درسـتی ریسـت مـی-دهند). همچنین، نوع ریست رخ داده توسط ثبات‌های مخصوص ثبت می‌شود.
سیستم کنترلگر ریست، در بعضی تراشه‌ها توانایی تولید ریست برای پردازنده‌های خارجی را نیز دارد.

۶.۱.۳.۱ ریست Brown-Out

نوعی ریست است که هنگامی ولتاژ تغذیه از حدی مشخص پایین باشد، فعـال مـی‌شـود. ایـن فرآینـد از اجرای ناصحیح دستورها و رخداد حالات ناخواسته در پردازنده جلوگیری می‌کند.

۶.۲ میکروکنترولرهای AT91

۶.۲.۱ کنترل کننده ریست در میکروکنترولرهای خانواده‌ی AT91SAM7/9

این واحد، شامل مدیریت NRST، مدیریت ریست Brown-out شمارنده‌ی آغاز به کـار و یـک مـدیریت حالت ریست می‌باشد. این بخش، با کلاک سرعت پایین (Show Clock) کار کرده و سیگنال‌های زیـر را تولید می‌کند:

- Proc_nreset: خط ریست پردازنده و همچنین زمان سنج watchdog؛
- Periph_nreset: مجموعه‌ی کامل زیرسیستم‌های داخلی میکرو را ریست می‌کند؛
- Nrst_out: پایه‌ی NRST را به عنوان ریست خارجی راه‌اندازی می‌کند؛
- Backup_nreset: تمام زیرسیستم‌هایی که با VDDBU تغذیه می‌شوند را ریست می‌کند (فقط بـر روی AT91SAM9).

سیگنال‌های ریست تولید شده توسط این بخش، ناشی از اثر حوادث خارجی و یا دستورهای نرم‌افزاری است. بخش مدیریت NRST نیز سیگنال NRST را برای ریست کردن دستگاه‌های جانبی، به صورت یک پالس با زمان مشخص فعال می‌سازد. شمارنده، وظیفه دارد تا هنگـام پایـدار شـدن نوسـانات کریسـتال نوسان ساز، صبر کند.
دیاگرام بلوکی واحد مدیریت NRST را در شکل ۶.۱ می‌بینیم. پایه‌ی NRST توسط کلاک سـرعت پـایین نمونه‌برداری می‌شود. وقتی سطح ولتاژ آن پایین باشد، یک **ریست کاربری** اعلام می‌شود. رخداد ایـن ریست می‌تواند فعال و یا غیرفعال باشد. در صورت غیرفعال بودن، این پایه هـیچ گونـه ریسـتی، ایجـاد نمی‌کند. طریقه‌ی انجام این عملیات را در توضیح ثبات‌های مربوطه می‌بینیم. کنترلگـر ریسـت، مـی‌توانـد برای ایجاد یک وقفه به جای ریست، پیکره‌بندی شود. مدت زمانی را که واحد کنترلگر ریست برای تولید ریست خارجی استفاده می‌کنـد، بـا اسـتفاده از ERSTL در RSTC_MR تنظـیم مـی‌شـود. مقـدار زمـان صرف شده برابر است با تعداد $2^{(ERSTL+1)}$ پالس از کلاک سرعت پایین.

زمان پالس تولید شده، بین25µS تا 60µS است. مقدار صفر برای ERSTL معادل 2 سیکل برای پـالس پایین‌رونده‌ی NRST می‌باشد. این مشخصات و تنظیمات اجازه می‌دهند که پالس مناسب برای هـر نـوع مدار خارجی فراهم شود.

۶.۲.۱.۱ مدیریت ریست Brownout (فقط در AT91SAM7 موجود است)

واحد آشکارسازی Brownout، پردازنده را از افتادن به حالتی ناخواسته هنگام کاهش ولتاژ تغذیـه‌اش، نجات می‌دهد. این کار در واحد کنترلگر ریست و با ایجاد یـک وقفـه صـورت مـی‌گیـرد. بـرای فعـال یـا غیرفعال سازی ایجاد این وقفه، از بیت BODIEN استفاده می‌کنیم.

۶.۲.۲ کنترل کننده‌ی ورودی/ خروجی موازی[۱]

برای پیکره‌بندی ابتدایی باید مراحل زیر صورت بگیرد:

۱. توان مورد نیاز: توسط PMC_PCER و یا PMC_DCDR کنترل می‌شود؛
۲. کلاک: برای خواندن ورودی و یا تولید وقفه از پایه‌های ورودی، باید کلاک این بخش فعال باشد.

[۱] Parallel Input/Output-PIO

۶.۳ میکروکنترلرهای LPC

۶.۳.۱ بلوک کنترل سیستم

این بلوک شامل چندین واحد کاربردی برای کنترل سیستم می‌باشد:

- ریست؛
- آشکارسازی Brown-Out؛
- ورودی‌های وقفه‌ی خارجی؛
- کنترل‌های متفرقه‌ی سیستمی.

۶.۳.۱.۱ تشریح پایه‌های کنترل سیستم

پایه‌های این بخش را در جدول زیر می‌بینیم.

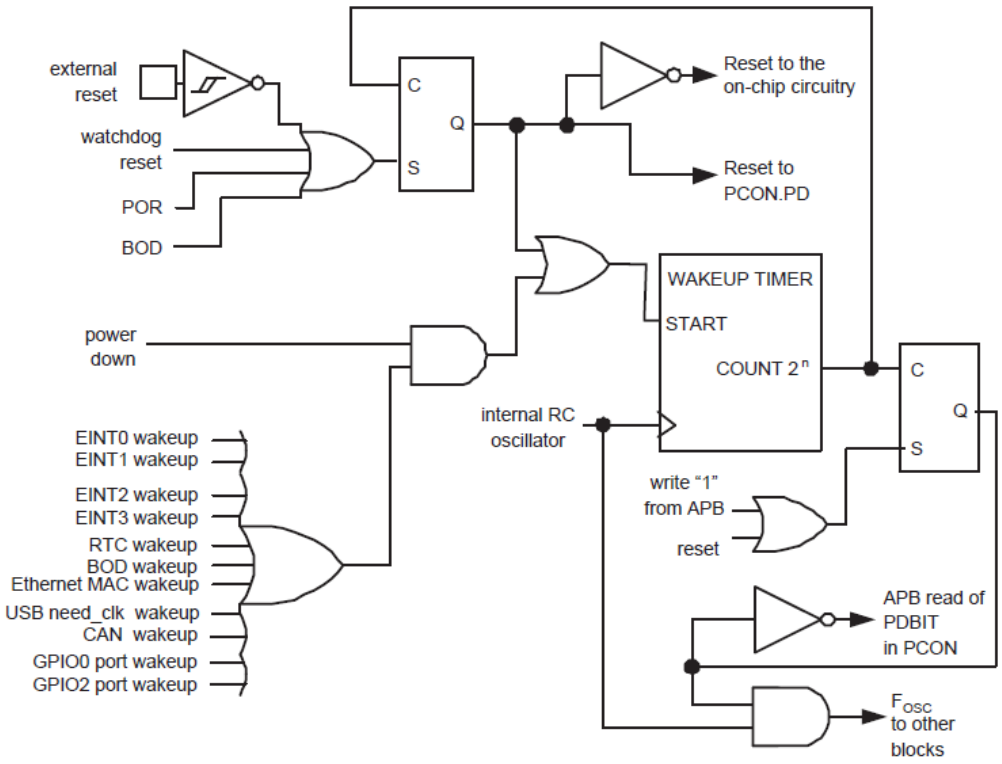
اسم پایه	وضعیت پایه	توضیح عملکرد
EINT0/EINT1 EINT2/EINT3	ورودی	ورودی وقفه‌های خارجی 0 تا 3. سطح فعال بالا/پایین و یا لبه بالا/پایین رونده می‌تواند در این پایه‌ها ایجاد وقفه کند. این پایه‌ها همچنین می‌توانند پردازنده را از حالت Idle و یا Power-down بیرون آورند.
#RESET	ورودی	یک ورودی با سطح پایین بر روی این پایه باعث بروز ریست شده، پورت‌ها و وسایل جانبی را به حالت پیش-فرض باز خواهد گرداند. بعد از ریست، پردازنده از آدرس 0x0000 0000 شروع به اجرای دستورات خواهد کرد.

جدول ۲۲

LPC23xx/24xx دارای ۴ پایه‌ی وقفه‌ی خارجی است که توانایی بیدار کردن پردازنده از حالت خواب را دارند. این کار توسط ثبات INTWAKE انجام می‌شود. این بخش، ۳ ثبات مرتبط دارد که کارآیی آن را تنظیم می‌کند. ثبات EXTINT شامل پرچم‌های وقفه است و ثبات‌های EXTMODE و EXTPOLAR پارامترهای حساسیت به سطح و لبه را تنظیم می‌کنند.

۶.۳.۱.۲ ریست

ریست، دارای ۴ منبع است: پایه‌ی $\overline{\text{RESET}}$، ریست Watchdog، ریست هنگام روشـن شـدن (POR) و ریست مدار آشکارساز Brown Out (BOD). پایه‌ی $\overline{\text{RESET}}$ یک پایه‌ی اشمیت تریگر شـده‌ی ورودی است. هنگامی که حالت ایجاد کننده‌ی ریست برداشته شود، پردازنده از آدرس "0x0000 0000"، کدها را اجرا می‌کند. این آدرس بردار ریست است که از بلوک بوت نگاشته شده اسـت. در ایـن نقطـه، تمـامی ثبات‌های پردازنده و وسایل جانبی به مقادیر از پیش تعیین شده‌ای، مقداردهی می‌شود.



شکل ۶.۱

۶.۳.۱.۳ تشریح ثبات‌های کنترل سیستم

ثبات‌های کنترلی بلوک فوق را در جدول زیر می‌بینیم.

نام	توضیح	نوع دستیابی[2]	مقدار اولیه[1]	آدرس
وقفه‌های خارجی				
EXTINT				
EXTMODE		R/W	0	0XE003 C004
EXTPOLAR		R/W	0	0XE003 C008
Reset				
RSID		R/W	0	0XE003 C010
ثبات‌های پیکره‌بندی AHB				
AHBCFG1		RO	0	0XE003 C018
AHBCFG2		RO	0	0XE003 C01C
ثبات‌های متفرقه Syscon				
SCS		RO	0	0XE004 0008

جدول ۲۳

۶.۳.۱.۴ پرچم‌های کنترلی متفرقه سیستم

بعضی از پرچم‌های مهم که در زیر سیستم‌های دیگر جای نمی‌گیرند، در این‌جا آمده‌اند.

بیت	نماد	مقدار	توضیح	مقدار اولیه
0	GPIOM		انتخاب حالت دستیابی به GPIO.	0
		0	دسترسی به پورت‌های 0 و 1 همانند LPC2000 و از طریق APB صورت می‌گیرد.	
		1	GPIO سریع برای پورت‌های 0 و 1 انتخاب می‌شود.	
1	EMC Reset Disable		ناتوان کننده ریست کنترل‌کننده حافظه‌ی خارجی.	
		0		0
		1		
2	EMC Burst Control		کنترل پیوسته‌ی حافظه‌ی خارجی.	0
		0	حالت پیوسته‌ی فعال.	
		1	حالت پیوسته‌ی غیر فعال.	
3	MCIPWR		کنترل پایه MCIPWR	0

[1] Reset Value : مقدار اولیه و پیش فرض بعد از آغاز به کار

[2] R/W : قابل خواندن و نوشتن ؛ RO : فقط قابل خواندن

	Active Level	0	پایه‌ی MCIPWR در سطح پایین است.	
		1	پایه‌ی MCIPWR در سطح بالا است.	
4	OSCRANGE		انتخاب محدوده‌ی نوسان‌ساز اصلی.	0
		0	محدوده‌ی فرکانسی نوسان‌ساز 1MHz تا 20MHz است.	
		1	محدوده‌ی فرکانسی نوسان‌ساز 15MHz تا 25MHz است.	
5	OSCEN		فعال‌سازی نوسان‌ساز اصلی.	
		0	نوسان‌ساز اصلی غیرفعال است.	
		1	نوسان‌ساز اصلی فعال است. درصورتی‌که مداری مناسب به پایه‌های XTAL1 و XTAL2 متصل شده باشد، شروع به کار می‌کند.	
6	OSCSTAT		وضعیت نوسان‌ساز اصلی.	**0**
		0	نوسان‌ساز اصلی برای استفاده به عنوان منبع کلاک آماده نیست.	
		1	نوسان‌ساز اصلی برای استفاده به عنوان منبع کلاک آماده است. نوسان‌ساز اصلی باید توسط OSCEN فعال شود.	
31:7	-	-	برای استفاده‌های بعدی.	NA

جدول ۳۸

۶.۳.۱.۵ آشکارساز Brown-Out

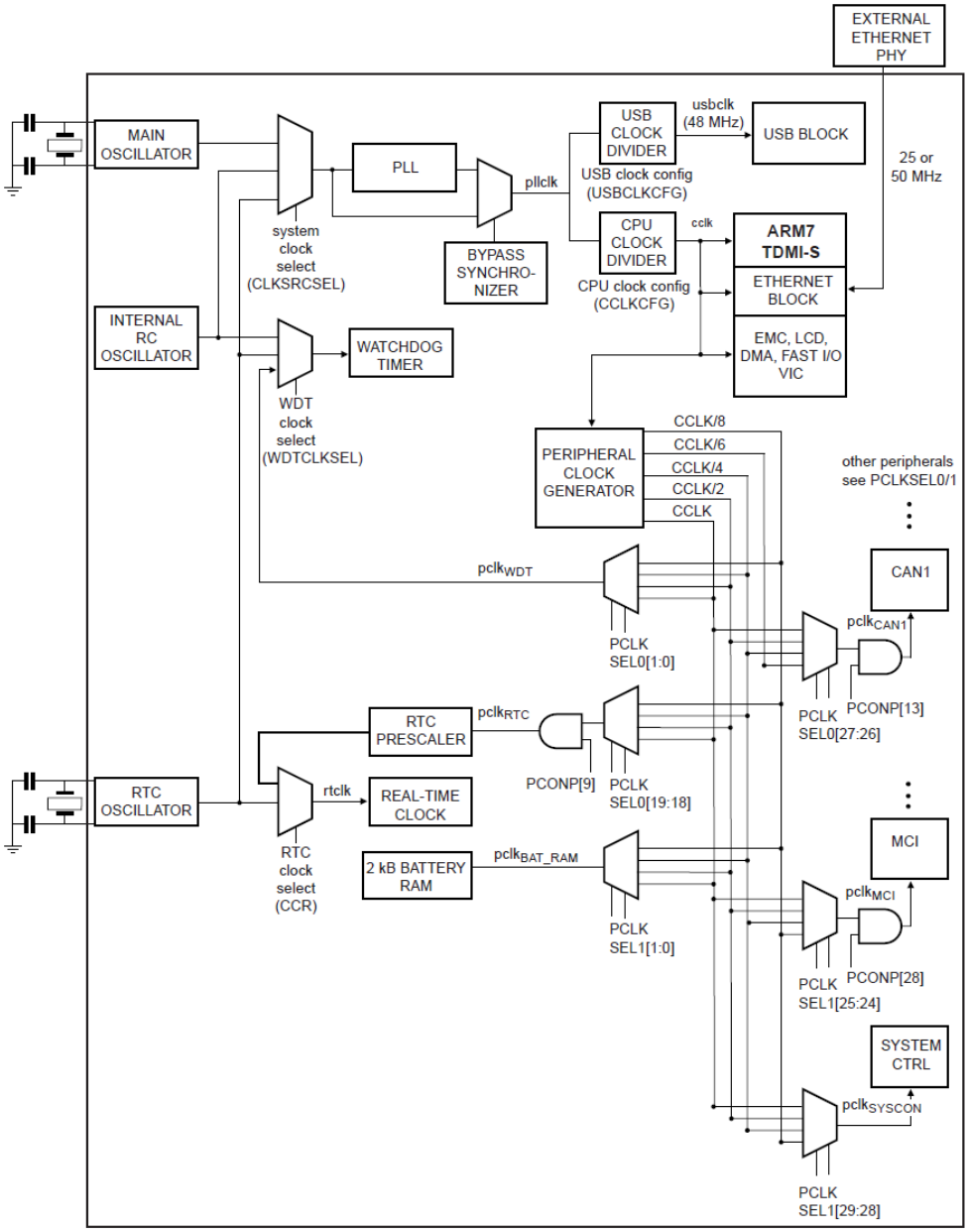
دو سطح مختلف ولتاژ، برای ایجاد این نوع ریست وجود دارد. اگر ولتاژ $VDD_{(DCDC)(3V3)}$ پایین‌تر از 2.95 بشود، آشکارساز BOD یک سیگنال وقفه برای واحد VIC صادر می‌کند.
چنانچه ولتاژ این پایه پایین‌تر از 2.65 بشود، سیگنال ریست فعال شده و از هرگونه دستکاری فلش، ممانعت به عمل می‌آید.

۶.۳.۲ خلاصه‌ی اتصالات کلاک و کنترل تغذیه

در این بخش به بررسی موارد زیر می‌پردازیم:

- نوسان‌سازها؛
- انتخاب منبع کلاک؛
- PLL؛
- تقسیم کننده‌های فرکانس کلاک؛
- کنترل تغذیه.

دیاگرام کلی بخش تولید کننده‌ی کلاک را در شکل ۶.۲ می‌بینیم.


EXTERNAL ETHERNET PHY
MAIN OSCILLATOR
PLL
USB CLOCK DIVIDER
usbclk (48 MHz)
USB BLOCK
25 or 50 MHz
pllclk
USB clock config (USBCLKCFG)
system clock select (CLKSRCSEL)
BYPASS SYNCHRO-NIZER
CPU CLOCK DIVIDER
cclk
ARM7 TDMI-S
ETHERNET BLOCK
CPU clock config (CCLKCFG)
INTERNAL RC OSCILLATOR
WATCHDOG TIMER
EMC, LCD, DMA, FAST I/O VIC
WDT clock select (WDTCLKSEL)
PERIPHERAL CLOCK GENERATOR
CCLK/8
CCLK/6
CCLK/4
CCLK/2
CCLK
other peripherals see PCLKSEL0/1
pclkWDT
CAN1
PCLK SEL0[1:0]
pclkCAN1
PCLK SEL0[27:26]
PCONP[13]
RTC PRESCALER
pclkRTC
PCONP[9]
PCLK SEL0[19:18]
RTC OSCILLATOR
rtclk
REAL-TIME CLOCK
RTC clock select (CCR)
2 kB BATTERY RAM
pclkBAT_RAM
PCLK SEL1[1:0]
MCI
pclkMCI
PCLK SEL1[25:24]
PCONP[28]
SYSTEM CTRL
pclkSYSCON
PCLK SEL1[29:28]

شکل ۶.۲

۶.۳.۲.۱ نوسان‌ساز

LPC2400 ، سه نوسان‌ساز مستقل دارد. نوسان‌سازها عبارت‌اند از: نوسان‌ساز اصلی، نوسان‌ساز RC داخلی و نوسان‌ساز RTC. هر یک از این نوسان‌سازها می‌توانند برای بیش از یک منظور استفاده شوند.

بعد از ریست، LPC2400 با نوسان‌ساز RC داخلی کار می‌کند تا زمانی که توسط نرم‌افزار نوسان‌ساز دیگری انتخاب شود. با استفاده از نوسان‌ساز RC داخلی، در ابتدای کار نیاز به کریستال خارجی نبوده و در ضمن کد مربوط به بوت‌لودر در فرکانس نوسان مشخصی، کار خواهد کرد.

نوسان‌ساز RC داخلی (IRC)

این نوسان‌ساز ممکن است به عنوان منبع کلاک برای زمان سنج Watchdog، یا به عنوان کلاک راه‌انداز PLL و CPU استفاده گردد. دقت IRC برای استفاده در بخش USB مناسب نمی‌باشد. همچنین اگر نرخ باور واسط CAN از 100 k bits بیشتر است، از این نوسان‌ساز استفاده نکنند. فرکانس IRC برابر 4 MHz است.

توجه

بعد از هرگونه ریست و یا اتصال تغذیه‌ی سیستم، LPC2400 از IRC به عنوان منبع کلاک استفاده می‌کند. این کار تا زمانی که توسط نرم‌افزار منبع کلاک دیگری انتخاب نشده باشد، ادامه دارد.

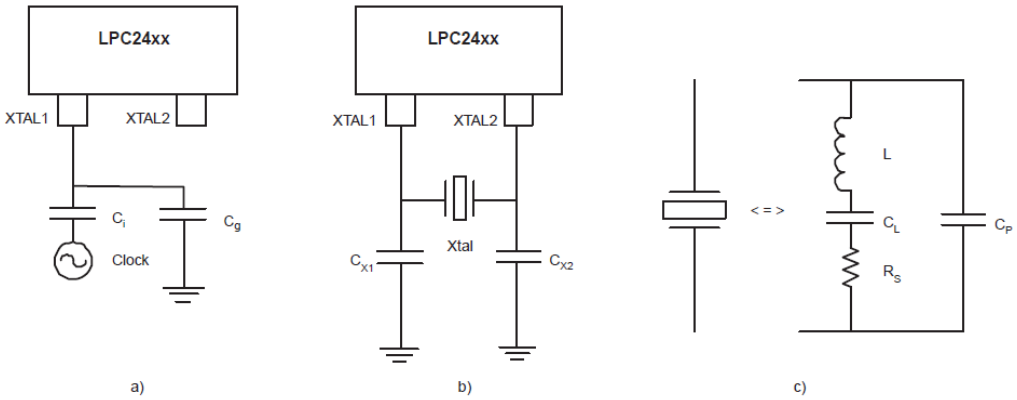
نوسان‌ساز اصلی

نوسان‌ساز اصلی، از دیگر منابع کلاک CPU است. فرکانس نوسان این نوسان‌ساز، می‌تواند بین 1 MHz تا 24 MHz باشد. با استفاده از PLL می‌توان این فرکانس را تا سقف حداکثر فرکانس کاری پردازنده افزایش داد.

این نوسان‌ساز می‌تواند در دو حالت فرمانبردار[۱] و حالت نوسان‌ساز، عمل کند.

روش کار را در شکل ۶.۳ می‌بینیم.

[۱] Slave



شکل ۶.۳

مقادیر توصیه شده برای خازن‌های کریستال در حالت نوسان‌سازی، در جدول ۴۰ و ۳۹ آمده‌اند.

Fundamental oscillation frequency F_{OSC}	Crystal load capacitance C_L	Maximum crystal series resistance R_S	External load capacitors C_{X1}, C_{X2}
1 MHz - 5 MHz	10 pF	< 300 Ω	18 pF, 18 pF
	20 pF	< 300 Ω	39 pF, 39 pF
	30 pF	< 300 Ω	57 pF, 57 pF
5 MHz - 10 MHz	10 pF	< 300 Ω	18 pF, 18 pF
	20 pF	< 200 Ω	39 pF, 39 pF
	30 pF	< 100 Ω	57 pF, 57 pF
10 MHz - 15 MHz	10 pF	< 160 Ω	18 pF, 18 pF
	20 pF	< 60 Ω	39 pF, 39 pF
15 MHz - 20 MHz	10 pF	< 80 Ω	18 pF, 18 pF

Fundamental oscillation frequency F_{OSC}	Crystal load capacitance C_L	Maximum crystal series resistance R_S	External load capacitors C_{X1}, C_{X2}
15 MHz - 20 MHz	10 pF	< 180 Ω	18 pF, 18 pF
	20 pF	< 100 Ω	39 pF, 39 pF
20 MHz - 25 MHz	10 pF	< 160 Ω	18 pF, 18 pF
	20 pF	< 80 Ω	39 pF, 39 pF

جدول‌های ۳۹ و ۴۰

نوسان‌ساز RTC

نوسان‌ساز RTC ، می‌تواند به عنوان منبع کلاک برای RTC و یا زمان سنج Watchdog، ورودی PLL و یا CPU استفاده شود.

۶.۳.۲.۲ مالتی پلکسر انتخاب منبع کلاک

برای راه‌اندازی PLL (بالتبع CPU و وسایل جانبی)، چندین منبع کلاک وجود دارند. آن‌ها عبارت‌اند از: نوسان‌سازی اصلی، RTC و IRC.

نکته

همواره قبل از تنظیم و اتصال PLL ، منبع کلاک آن را انتخاب کنید.

ثبات انتخاب منبع کلاک (PCLKSRCSEL)

ثبات PCLKSRCSEL شامل بیت‌هایی برای انتخاب منبع کلاک PLL است.

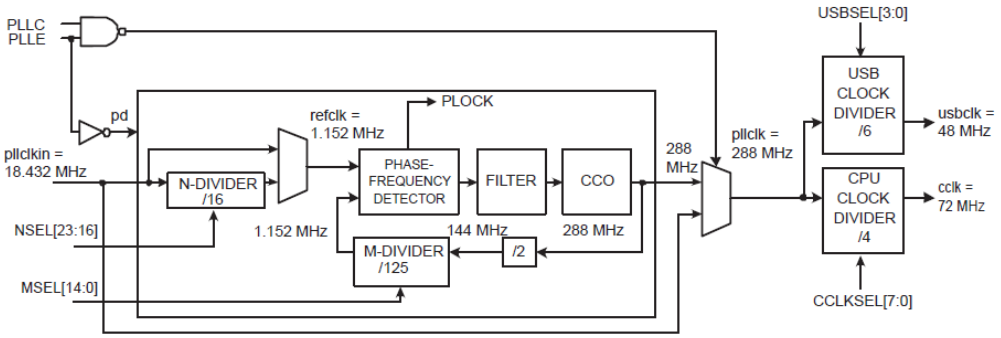
بیت	نماد	مقدار	توضیح	مقدار اولیه
1:0	CLKSRC		منبع کلاک PLL را همانند زیر انتخاب می‌کند.	0
		00	RC داخلی را به عنوان منبع کلاک PLL انتخاب می‌کند. (پیش فرض)	
		01	نوسان‌ساز اصلی را به عنوان منبع کلاک PLL انتخاب می‌کند.	
		10	نوسان‌ساز RTC را به عنوان منبع کلاک PLL انتخاب می‌کند.	
		11	برای استفاده‌های بعدی. نباید همگی این بیتها، 1 باشند.	
7:2	-	0	استفاده نمی‌شوند. همواره 0.	0

جدول ۴۲

۶.۳.۲.۳ حلقه‌ی فاز قفل شده (PLL)

PLL ، فرکانس‌های ورودی بین 32 KHz و 24 MHz را پذیرفته، آن را با ضرب در یک عدد، به فرکانس بالاتری تبدیل می‌کند. سپس، با تقسیم بر یک عدد دیگر، فرکانس لازم برای تغذیه CPU و USB را به دست می‌آورد.

شمای کلی PLL را در شکل ۶.۴ می‌بینیم.



شکل ۶.۴

۶.۳.۳ درگاه ورودی/ خروجی استاندارد[۱]

۶.۳.۳.۱ پیکره‌بندی ابتدایی GPIO

GPIO ها با استفاده از ثبات‌های زیر پیکره‌بندی می‌شوند:

۱- توان مورد نیاز: همواره فعال است؛

۲- کلاک: برای پورت‌های GPIO سریع بخش ۷.۱ از مبحث کلاک را ببینید. برای بقیه‌ی GPIO ها ثبات PCLKSEL1 را نگاه کنید؛

۳- پایه‌ها: پایه‌های GPIO و حالاتشان را توسط PINSEL0 تا PINSEL10 و PINMODE0 تا PINMODE10 انتخاب کنید؛

۴- پریدن از حالت خواب: از ثبات INTWAKE برای پیکره‌بندی پورت‌های 0 و 2 مربوط به GPIO برای این منظور استفاده کنید؛

۵- وقفه‌ها: وقفه‌های GPIO را در ثبات IO0/2IntEnR و یا IO0/2IntEnF فعال کنید. وقفه‌ها در VIC توسط ثبات VICIntEnable فعال می‌شوند.

---

[۱] General Purpose Input/ Output Port-GPIO

۶.۳.۳.۲ مشخصات پورت‌های I/O دیجیتال

- دسترسی به پورت‌های PORT0 و PORT1 ، یا از طریق گروهی از ثبات‌ها با مشخصات بهبود یافته و یا دسترسی عادی به پورت‌ها ممکن است. پورت‌های PORT2/3/4 فقط به عنوان پورت-های سریع در دسترس هستند؛
- کارکردهای GPIO شتاب یافته:
  - ثبات‌های GPIO بر روی گذرگاه محلی ARM قرار دارند. بنابراین، سریع‌ترین زمان-بندی برای دسترسی به IO ها فراهم آمده است؛
  - ثبات‌های ماسک، دسترسی به گروهی از بیت‌های یک پورت را به صورت یک مجموعه فراهم آورده‌اند؛
  - تمام ثبات‌های GPIO به شیوه‌ی بایت و دو بایت (Half Word) ، قابل آدرس‌دهی هستند؛
  - تمام مقدار یک پورت در یک دستورالعمل می‌تواند نوشته شود.
- ثبات‌های تنظیم به '1' و پاک کردن در سطح بیت، اجازه‌ی دستکاری دو تعداد بیت، توسط یک دستور را می‌دهند؛
- تنظیم جهت شارش داده برای هر بیت به صورت جداگانه؛
- تمام I/O ها هنگام ریست پیش‌فرض به صورت ورودی تنظیم شده‌اند؛
- با قطعات قدیمی که ثبات‌هایشان در آدرس‌های قبلی بر روی گذرگاه APB قرار داشتند، سازگاری کامل دارد.

۶.۳.۳.۳ پورت‌هایی که توانایی تولید وقفه دارند

- PORT0 و PORT2 برای هر پایه این پورت‌ها توانایی تولید وقفه دارند؛
- دو وقفه می‌تواند برای تولید وقفه در لبه‌ی پایین رونده، بالارونده و یا هردوی‌شان برنامه‌ریزی شود؛
- تشخیص وقفه آسنکرون هستند. یعنی برای تشخیص وقفه نیازی به کلاک نیست. (مثلاً در حال Power-down) با این مشخصه، نیازی به وقفه‌های حسّاس به سطح نمی‌باشد؛
- هر وقفه فعال شده می‌تواند قطعه را از حالت خواب دربیاورد؛
- وقفه‌های GPIO0 و GPIO2 مکان یکسانی با رویداد وقفه خارجی 3[۱] در VIC دارند.

---

[۱] Ext INT 3

۶.۳.۳.۴ پایه‌های GPIO

نام پایه	نوع	توضیح
P0[31:0] P1[31:0] P2[31:0] P3[31:0] P4[31:0]	ورودی / خروجی	ورودی/خروجی‌های همه منظوره. عموما با پایه‌های دیگر دارای اشتراک کارکردی هستند. پس ممکن است همگی آن‌ها در یک مدار در دسترس نباشند. نوع بسته‌بندی نیز در تعداد GPIO های در دسترس تاثیرگذاراست.

جدول GPIO_1

۶.۳.۳.۵ توضیح ثبات‌های GPIO

LPC2300 تا سقف ۵ پورت GPIO ۳۲ بیتی دارد. PORT0 و PORT1 توسط دو گروه ثبات که در جداول ۱۳۲_۱۰ و ۱۳۳_۱۰ آمده‌اند، کنترل می‌شوند. به غیر از این ثبات‌ها، این تراشه می‌تواند تا ۳ پورت ۳۲ بیتی دیگر نیز داشته باشد. (PORT2/3/4) مجموعه ثبات‌های IOSET، IOPIN، IODIR و IOCLR در جدول ۱۳۲ برای سازگاری با قطعات قدیمی با همان زمان‌بندی‌ها آورده شده‌اند.

جدول ۱۳۳ مجموعه‌ی خصوصیات بهبود یافته موجود بر روی تمامی پورت‌های LPC2300 را لیست می‌کند.

هنگام استفاده از PORT0/1، کاربر باید مشخص کند که دسترسی به پورت از طریق کدام مجموعه از ثبات‌های پایه و یا بهبود یافته صورت می‌گیرد. این دو مجموعه ثبات از هم مستقل هستند. به عنوان مثال، تغییر خروجی یک پایه از طریق ثبات سریع، به معنی مشاهده‌ی این تغییر در ثبات پایه‌ی منطبق-اش نیست. در ادامه‌ی متن، از GPIO های قدیمی به عنوان "کُند" و از GPIO های بهبود یافته با نام "سریع" یاد می‌شود.

ثبات	توضیح	دستیابی	مقدار اولیه	نام ثبات
IOPIN	مقدار فعلی پایه توسط این ثبات خوانده می‌شود. با نوشتن مقدار خاصی بر روی این ثبات، مقدار پایه متناظر به بیت نوشته شده تغییر می‌یابد.	R/W	NA	IO0PIN IO1PIN
IOSET	با نوشتن بیت‌های '1' درون این ثبات، سطح پایه موردنظر به '1' تنظیم می‌شود. نوشتن مقادیر '0' تاثیری بر خروجی ندارند.	R/W	0x0	IO0SET IO1SET
IODIR	این ثبات، به تنهایی جهت هریک از پایه ها را	R/W	0x0	IO0DIR IO1DIR

	مشخص می‌کند.			
IOCLR	با نوشتن بیت‌های '1' درون این ثبات، سطح پایه موردنظر به '0' تنظیم می‌شود. نوشتن مقادیر '0' تاثیری بر خروجی ندارند.	WO	0x0	IO0CLR IO1CLR

جدول ۱۳۲

ثبات	توضیح	دستیابی	مقدار اولیه	نام ثبات
FIOPIN	مقدار فعلی پایه توسط این ثبات خوانده می‌شود. با نوشتن مقدار خاصی بر روی این ثبات، مقدار پایه متناظر به بیت نوشته شده تغییر می‌یابد. (اگر پایه موردنظر به عنوان ADC استفاده نشود) مقدار خوانده شده، با مقدار معکوس شده FIOMASK ، AND می شود. نوشتن بر روی این ثبات، مقادیر متناظر را، اگر بیت‌های مربوط در FIOMASK 0 باشند، بر روی پایه ها قرار می‌دهد.	R/W	NA	FIO0PIN FIO1PIN FIO2PIN FIO3PIN FIO4PIN
FIOSET	با نوشتن بیت‌های '1' درون این ثبات، سطح پایه موردنظر به '1' تنظیم می‌شود. نوشتن مقادیر '0' تاثیری بر خروجی ندارند. تنها بیت‌های '0' شده در FIOMASK می‌توانند تغییر داده شوند.	R/W	0x0	FIO0SET FIO1SET FIO2SET FIO3SET FIO4SET
FIODIR	این ثبات، به تنهایی جهت هریک از پایه ها را مشخص می‌کند.	R/W	0x0	FIO0DIR FIO1DIR FIO2DIR FIO3DIR FIO4DIR
FIOMASK	ثبات پوشش پورت. تمامی عملیات نوشتن، خواندن و هرگونه تغییر پایه‌ها تنها وقتی صورت می‌گیرند که بیت مورد نظرشان در این ثبات، '0' باشد.	R/W	0x0	FIO0MASK FIO1MASK FIO2MASK FIO3MASK FIO4MASK
FIOCLR	با نوشتن بیت‌های '1' درون این ثبات، سطح پایه موردنظر به '0' تنظیم می‌شود. نوشتن مقادیر '0' تاثیری بر خروجی ندارند. تنها بیت‌های '0' شده در FIOMASK می‌توانند تغییر داده شوند.	WO	0x0	FIO0CLR FIO1CLR FIO2CLR FIO3CLR FIO4CLR

۶.۳.۳.۶ مثال‌هایی از نحوه‌ی استفاده‌ی GPIO ها

مثال ۶.۱

دسترسی ترتیبی به یک پایه، از طریق IOSET و IOCLR.

وضعیت هر پایه‌ی پورت از طریق نوشتن بر ثبات‌های IOSET و IOCLR تعیین می‌شود.

در مثال زیر

```
IO0DIR = 0x0000 0080       ;pin P0.7 configured as output
IO0CLR = 0x0000 0080       ;P0.7 goes LOW
IO0SET = 0x0000 0080       ;P0.7 goes HIGH
IO0CLR = 0x0000 0080       ;P0.7 goes LOW
```

پایه‌ی P0.7 ، به عنوان خروجی تنظیم شده است (نوشتن بر IO0DIR). سپس P0.7 به سطح منطقی 0 تنظیم می‌شود (از طریق نوشتن بر IO0CLR). از طریق IOSET، خروجی مورد نظر به سطح '1' تنظیم می‌شود.

مثال ۶.۲

خروجی 0 و 1 لحظه‌ای بر روی GPIO

کد زیر را درنظر بگیرید:

```
IO0PIN = (IO0PIN && 0xFFFF00FF) || 0x0000A500
```

این کار از طریق واسطه‌های سریع نیز به این صورت قابل انجام است.

راه‌حل ۱: استفاده از دستیابی ۳۲ بیتی به پورت‌های سریع.

```
FIO0MASK = 0xFFFF00FF;
FIO0PIN = 0x0000A500;
```

راه‌حل ۲: استفاده از دستیابی ۱۶ بیتی به پورت‌های سریع.

```
FIO0MASKL = 0x00FF;
FIO0PINL = 0xA500;
```

راه‌حل ۳: استفاده از دستیابی ۸ بیتی به پورت‌های سریع.

```
FIO0PIN1 = 0xA5;
```

۶.۳.۳.۷ نوشتن بر IOSET/IOCLR در مقابل IOPIN

نوشتن بر روی IOSET/IOCLR راه‌حلی آسان برای تغییر سطوح پایه‌ها به سطوح بالا/ پایین را فراهم می‌آورند. تنها بیت‌هایی که بر روی این ثبات‌ها تأثیر می‌گذارند، مقادیر '1' هستند. مقادیر '0' هیچ اثری ندارند. برای '1' کردن هر یک از پایه‌ها، صرفاً بر روی IOSET و برای'0' کردن پایه‌ها، فقط بر روی IOCLR، مقادیر '1' باید نوشت.

برای نوشتن مجموعه‌ای از صفر و یک‌ها به طور لحظه‌ای بر روی GPIO ، از ثبات IOPIN باید استفاده کرد. تمامی صفرها، پایه‌های خروجی را '0' و تمامی یک‌ها، پایه‌های خروجی را '1' می‌کند.

۶.۳.۳.۸ زمان‌بندی پایه‌های GPIO

استفاده از ثبات‌های سریع، امکان کنترل سریع‌تر پایه‌های GPIO را می‌دهد.

نکته

به عنوان یک قانون کلی، استفاده از ثبات‌های سریع، دستیابی به پورت‌ها را نسبت به استفاده از پورت‌های پایه تا 3.5 برابر سریع‌تر می‌کند. فرکانس قابل حصول نیز به همین ترتیب افزایش دارد. البته برای دستیابی به حداکثر سرعت، باید از کد اسمبلی (به جای استفاده از C) و در حالت کاری ARM استفاده کرد.

۶.۳.۴ ارتباط سریال UART

مشخصات:

- ۱۶ بایت حافظه‌ی FIFO ارسال و دریافت؛
- مکان قرارگیری ثبات‌ها با استاندارد صنعتی 550' سازگار هستند؛
- نقطه‌ی تحریک حافظه‌ی FIFO در تعداد بایت ۱، ۴، ۸ و ۱۴؛
- تولید کننده نرخ باود داخلی؛
- از یک حالت IrDA برای ارتباطات مادون قرمز استفاده می‌کند.

۶.۳.۴.۱ پایه‌های مربوط به UART

پایه‌های مربوط به UART را در جدول UART می‌بینیم.

۶.۳.۴.۲ پیکره‌بندی ابتدایی UART

با استفاده از ثبات‌های زیر، UART را پیکره‌بندی می‌کنیم:

۱. تغذیه: در ثبات PCONP بیت‌های PCUARTx (x نشان‌دهنده‌ی شماره‌ی UART است) را '1' کنید؛
۲. کلاک وسایل جانبی: بیت‌های PCUARTx را در ثبات‌های PCLKSEL011 به "x" تنظیم کنید؛
۳. نرخ باود: در ثبات‌های UxCLR بیت DLAB=1 قرار دهید. علاوه بر آن، نرخ باود کسری را در ثبات‌های تقسیم کننده‌ی کسری جدول 362-16 تنظیم کنید؛
۴. حافظه‌ی FIFO مربوط به UART: برای فعال ساختن FIFO، از بیت شماره‌ی صفر ثبات U0FCR استفاده کنید؛
۵. پایه‌ها: پایه‌های UART و حالت پایه‌ها را در ثبات‌های PINSELN و PINMODEN انتخاب کنید؛

توجه

دقت کنید که مقاومت‌های pull down بر روی پایه‌های دریافت UART ، نباید فعال باشند.

۶. وقفه‌ها: برای فعال سازی، بیت DLAB=0 قرار دهید. با این‌کار به UxIER دسترسی خواهید داشت. وقفه‌ها در VIC توسط ثبات VICIntEnable فعال می‌شوند.

۶.۳.۴.۳ کُد ارتباط با UART

Example: Left Shifting Bits.c Code COP_1

```
1  void UARTSend(BYTE *BufferPtr, DWORD Length )
2  {
3    while ( Length != 0 )
4    {
5      // THRE status, contain valid data
6      while ( !(UART0TxEmpty & 0x01) );
7      U0THR = *BufferPtr;
8      UART0TxEmpty = 0; // not empty in the THR until it shifts
9  out
10     BufferPtr++;
11     Length--;
12   }
13   return;
14 }
15
```

```
volatile DWORD UART0Status;
volatile BYTE UART0TxEmpty = 1;
volatile BYTE UART0Buffer[BUFSIZE];
volatile DWORD UART0Count = 0;

DWORD Fdiv;
PINSEL0 = 0x00000050;       // RxD0 and TxD0

U0LCR = 0x83;   // 8 bits, no Parity, 1 Stop bit
Fdiv = ( Fpclk / 16 ) / 115200 ;  //baud rate
U0DLM = Fdiv / 256;
U0DLL = Fdiv % 256;
U0LCR = 0x03;   // DLAB = 0
U0FCR = 0x07;   // Enable and reset TX and RX FIFO


if ( install_irq( UART0_INT, (void *)UART0Handler,
HIGHEST_PRIORITY ) == FALSE )
{
    return (FALSE);
}

U0IER = IER_RBR | IER_THRE | IER_RLS; //Enable UART0 interrupt

  while (1)
  {       // Loop forever
    if ( UART0Count != 0 )
    {
      U0IER = IER_THRE | IER_RLS;     // Disable RBR
      UARTSend( (BYTE *)UART0Buffer, UART0Count );
      UART0Count = 0;
      U0IER = IER_THRE | IER_RLS | IER_RBR; // Re-enable RBR
    }
  }
```

۶.۳.۵ کنترل کننده‌های CAN1/2

رابط CAN مجموعه‌ای از تعاریف و سیگنال‌های ارتباطی سریال کارآمد می‌باشد. کنترل کننده‌ی CAN برای پیاده سازی کامل پروتکل CAN براساس مشخصات CAN نسخه‌ی 2.0B است.
میکروکنترولرهایی که دارای واسط CAN داخلی هستند، برای ساخت شبکه‌های قدرت‌مند محلی جهت کنترل بی‌درنگ و ایمن استفاده می‌شوند. نتیجه‌ی استفاده از چنین واسطهایی، تعداد سیم‌های به شدت کاهش یافته، سیستم عیب‌یابی بهبود یافته و توانایی‌های نگهبانی و کنترل بهینه است.
کنترل کننده‌ی CAN توانایی پشتیبانی چندین گذرگاه CAN به طور هم‌زمان را دارد. هر یک از این قطعات می‌توانند به عنوان یک مسیریاب (router)، سوییچ (Switch) و یا Gate way در بین چندین گذرگاه CAN در کاربردهای مختلف استفاده شوند.
ماژول CAN شامل دو زیربخش است: کنترل کننده و یک فیلتر دریافت اطلاعات.

تمامی ثبات‌ها و حافظه‌ی RAM به صورت کلمات ۳۲ بیتی در دسترس هستند.

۶.۳.۵.۱ مشخصات عمومی CAN

- سازگار با مشخصات CAN نسخه‌ی 2.0B، ISO11898؛
- پشتیبانی از ساختار چندین فرمانده بر روی یک گذرگاه[۱]؛
- تقدم دستیابی به گذرگاه توسط مشخص کننده‌ی پیغام[۲] مشخص می‌شود (۱۱ یا ۲۹ بیت)؛
- زمان تأخیر تضمین‌شده برای پیغام‌های با تقدم بالا؛
- نرخ انتقال قابل تنظیم (تا سقف 1 Mbit/S)؛
- امکان ارسال پیغام‌های broadcast و multicast؛
- طول پیغام داده از ۰ تا ۸ بیت؛
- قابلیت رسیدگی به خطای قدرت‌مند؛
- کدکننده و دیکود کننده‌ی NRZ.

۶.۳.۵.۲ مشخصات کنترل کننده‌ی CAN

- دو کنترل کننده CAN و گذرگاه‌های مربوط؛
- پشتیبانی از مشخص کننده‌های (ID) ۱۱ و ۲۹ بیتی؛
- بافر دریافت دوگانه و بافر ارسال سه‌گانه؛
- شمارنده خطاهای دریافتی قابل برنامه‌ریزی.

۶.۳.۵.۳ مشخصات فیلتر دریافت

- الگوریتم جستجوی سخت‌افزاری با پشتیبانی از تعداد زیادی CAN ID؛
- فیلتر عمومی CAN، ID های ۱۱ و ۲۹ بیتی دریافت را برای تمامی گذرگاه‌ها تشخیص می‌دهد.

۶.۳.۵.۴ تشریح پایه‌های CAN

پایه‌های مربوط به CAN را در جدول CAN-1 می‌بینیم.

---

[۱] Multi-master

[۲] Message ID

۶.۳.۵.۵ پیکره‌بندی ابتدایی CAN

۱. تغذیه: بیت‌های PCAN1/2 را در ثبات PCONP به '1' تنظیم کنید؛
۲. کلاک وسایل جانبی: بیت‌های PCLK_CAN1/2 و بیت PCLK_ACF (مربوط به بخش فیلتر) را در ثبات PCLK_SEL0 بـه '1' تنظیم کنیـد. بیـت‌هـای PCLK_CAN1/2 و PCLK_ACF بایـد دارای مقدار یکسانی باشند؛

توجه

اگر نرخ کلاک CAN بیش از 100 Kbit/S باشد، نباید از IRC به عنوان منبع کلاک استفاده شود.

۳. پریدن از خواب: از ثبات INTWAKE برای این‌کـه رابـط CAN بتوانـد میکروکنترولـر را از حالـت خواب دربیاورد، استفاده کنید؛
۴. پایه‌ها: پایه‌های CAN1/2 و حالت آن‌ها را در ثبات‌های PINSELN و PINMODEN تنظیم کنید؛
۵. وقفه‌ها: وقفه‌های CAN با استفاده از ثبات‌های CAN1/2 IER فعال می‌شوند؛
۶. مقداردهی اولیه‌ی کنترل کننده‌ی CAN: ثبات CANMOD رابببینید.

۶.۳.۶ کنترل کننده‌ی USB Device

USB یک گذرگاه چهار- سیمه است که امکان ارتباط بین یک میزبان[۱] و یک یا چندین وسیله‌ی جانبی[۲] (تا سقف ۱۲۷) را می‌دهد. کنترل کننده‌ی میزبان، پهنای باند داده‌ی مربوط بـه دسـتگاه‌هـای متصـل‌شـده را ازطریق پروتکلی براساس توکن[۳] اختصاص می‌دهد. این گذرگاه، از اتصال "زنـده و بهنگـام"[۴] و پیکـره-بندی پویا پشتیبانی می‌کند. تمامی نقل و انتقالات توسط میزبان شروع می‌شود.

---

[۱] Host
[۲] Peripheral
[۳] Token
[۴] Hot Plugging

میزبان نقل و انتقالات را به صورت فریم‌های 1MS، زمان‌بندی می‌کند. هر فریم دارای یک علامت "شروع فریم" (SOF) و ترنزکشن[۱] است که داده‌ها از و یا به اندپوینت‌ها[۲] منتقل می‌کند. هر دستگاه می‌تواند دارای ۱۶ اندپوینت منطقی یا ۳۲ اندپوینت فیزیکی باشد.

۴ نوع انتقال داده برای اندپوینت‌ها تعریف شده است. انتقال‌های کنترلی[۳] برای پیکره‌بندی دستگاه تعریف شده‌اند. نقل و انتقالات وقفه‌ای[۴]، برای انتقال‌های دوره‌ای داده‌ها به کار می‌رود. انتقال از نوع توده‌ای[۵] هنگامی به کار می‌رود که نرخ انتقال مهم نیست و انتقال‌های هم‌زمان[۶]، موعد تحویل را ضمانت می‌کنند ولی هیچ‌گونه مکانیزم تصحیح خطایی ندارند.

برای اطلاعات بیشتر در مورد USB ، به سایت رسمی آن مراجعه کنید. کنترل کننده‌ی USB Device با سرعت کامل[۷] دارای نرخ انتقال داده‌ی 12 Mbit/S با کنترل کننده‌ی میزبان است.

اینترنت

سایت رسمی USB در آدرس http://www.usb.org قرار دارد.

جدول زیر، فهرستی از اختصارات مربوط به کنترل کننده‌ی USB را ارایه می‌دهد:

اختصار	توضیح
AHB	AHB Advanced High-performance bus
ATLE	ATLE Auto Transfer Length Extraction
ATX	ATX Analog Transceiver
DD	DD DMA Descriptor
DDP	DDP DMA Description Pointer
DMA	DMA Direct Memory Access
EOP	EOP End-Of-Packet
EP	EP Endpoint

[۱] Transaction

[۲] End Point

[۳] Control

[۴] Interrupt

[۵] Bulk

[۶] Isochronous

[۷] Full Speed

EP_RAM	EP_RAM Endpoint RAM
FS	FS Full Speed
LED	LED Light Emitting Diode
LS	LS Low Speed
MPS	MPS Maximum Packet Size
NAK	NAK Negative Acknowledge
PLL	PLL Phase Locked Loop
RAM	RAM Random Access Memory
SOF	SOF Start-Of-Frame
SIE	SIE Serial Interface Engine
SRAM	SRAM Synchronous RAM
UDCA	UDCA USB Device Communication Area
USB	USB Universal Serial Bus

جدول USB_1

۶.۳.۶.۱ مشخصات کنترل کننده USB

- کاملاً با مشخصات USB 2.0 با سرعت کامل سازگار است.
- پشتیبانی از ۳۲ اندپوینت فیزیکی (۱۶ اندپوینت منطقی)؛
- پشتیبانی از اندپوینت‌های کنترلی، وقفه‌ای، توده‌ای و هم‌زمان؛
- قابلیت مقیاس‌بندی اندپوینت‌ها در زمان اجرا؛
- قابلیت انتخاب حداکثر اندازه‌ی اندپوینت‌ها در زمان اجرا (تا سقف اجازه داده شده توسط USB)؛
- دارای مشخصاتی چون Soft Connect و Good Link است؛
- دارای انتقال‌های DMA بر روی تمام اندپوینت‌هایی است که کنترلی نیستند؛
- به راحتی می‌توان بین حالت‌های کنترل توسط CPU و یا DMA سوییچ کرد؛
- وجود بافر دوگانه برای اندپوینت‌های هم‌زمان و توده‌ای.

۶.۳.۶.۲ پیکره‌بندی اندپوینت‌ها

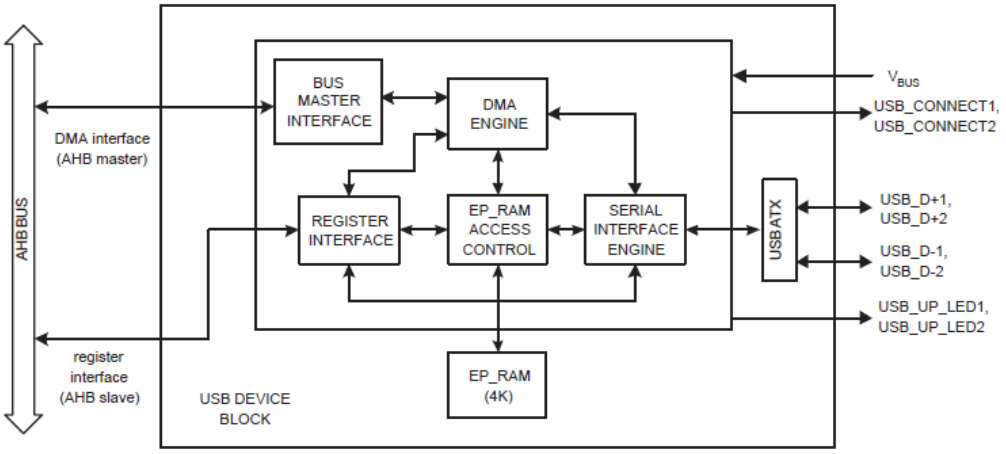
جدول USB-2 ، پیکره‌بندی‌های اندپوینت پشتیبانی شده را نشان می‌دهد. اندپوینت، توسط ثبات‌های مربوط به "تحقق اندپوینت"[۱]، پیکره‌بندی می‌شوند. برای اطلاعات بیشتر به بخش مربوط مراجعه کنید.

جدول USB_2

[۱] Endpoint Realization

۶.۳.۶.۳ توضیحات کاربردی کنترل‌کننده‌ی USB

ساختار داخلی کنترل کننده‌ی USB در شکل ۶.۵ نشان داده شده‌اند.



شکل ۶.۵

فرستنده_ گیرنده‌ی آنالوگ

کنترل کننده‌ی دستگاه USB ، دارای یک فرستنده- گیرنده‌ی آنالوگ تعبیه شده است. (ATX) واسط ATX وظیفه ارسال و دریافت اطلاعات را از/ به خطوط +D و -D بر عهده دارد.

موتور واسط سریال (SIE)

SIE لایه‌ی پروتکل USB را کاملاً پیاده سازی می‌کند. این قسمت کلاً سخت‌افزاری پیاده سازی شده است که سرعت بالایی را به همراه دارد. این قسمت وظیفه‌ی انتقال داده‌ها بافرهای اندپوینت در EP_RAM و گذرگاه USB را بر عهده دارد. وظیفه‌ی این بخش شامل: تشخیص الگوی هم‌زمانی، تبدیل سریال/ موازی، bit Stuffing/deStuffing، تولید و آزمایش CRC، تولید و تأیید PID، تشخیص آدرس، تولید آدرس و ارزیابی دست‌دهی[۱] است.

۶.۳.۶.۴ حافظه‌ی RAM اندپوینت (EP_RAM)

بافرهای اندپوینت، همانند FIFO واقع در SRAM پیاده سازی شده‌اند. SRAM اختصاص داده شده به این منظور را با نام EP_RAM می‌شناسیم. هر اندپوینتِ پیاده سازی شده دارای فضای اختصاصی

[۱] Handshake

خود بر روی EP-RAM است. فضای کلی مورد نیاز EP-RAM بستگی به تعداد کل اندپوینت‌های پیاده سازی شده، بیشینه اندازه‌ی هر بسته در اندپوینت و این‌که "آیا اندپوینت از بافر دوگانه پشتیبانی می‌کند یا نه"، دارد.
ثبات‌های مربوط به PLL را در جدول ۴۳ می‌بینیم.

۶.۳.۷ کنترل توان مصرفی

این میکروکنترولرها، از حالت‌های مختلفی برای کاهش توان مصرفی پشتیبانی می‌کنند. چهار حالت اصلی عبارت‌اند از: حالت بیکار[۱]، خواب[۲]، تغذیه خاموش[۳] و تغذیه خاموش عمیق[۴].
برای کاهش هرچه بیشتر توان مصرفی، می‌توان از فرکانس قسمت‌های مختلف را کاسته و یا آن‌ها را خاموش کرد. همچنین، می‌توان در مواقع ضروری توان مصرفی را قطع نموده و از اتصال باتری به RTC و RAM کوچک کنار آن برای نگه‌داری زمان و اطلاعات ضروری، استفاده کرد.

۶.۳.۷.۱ حالت بی‌کار

در حالت بی‌کاری، کلاک هسته قطع می‌شود. برای بازگشت به حالت عادی، نیاز به انجام کاری نیست و با اتصال کلاک مورد نظر، هسته به حالت عادی بازمی‌گردد.
در حالت بی‌کاری، اجرای دستورها متوقف شده و تا هنگامی که ریست یا وقفه‌ای رخ نداده باشد، این وضع ادامه دارد.

توجه

در حالت بی‌کاری، تمامی وسایل جانبی مشغول به کار هستند و می‌توانند وقفه‌ای (در صورت لزوم) تولید کنند. ولی، پردازنده، سیستم‌های حافظه و کنترل کننده‌هایشان و گذرگاه‌های داخلی قطع می‌باشند.

---

۱ Idle
۲ Sleep
۳ Power-Down
۴ Deep Power Down

۶.۳.۷.۲ حالت خواب

در این حالت، نوسان‌ساز اصلی قطع شده و تمامی کلاک‌ها از کار می‌ایستند و خروجی IRC قطع می‌شود. ولی IRC از کار نمی‌ایستد تا هنگام بیدار شدن از این حالت سریع عمل شود.
در این حالت، حافظه و محتویاتش، وضعیت‌های پردازنده و محتوای ثبات‌ها و حالات منطقی پایه‌ها نیز حفظ می‌شوند.

۶.۳.۷.۳ حالت تغذیه‌ی خاموش

این حالت، همانند حالت خواب است اما علاوه بر آن حافظه‌ی فلش را نیز خاموش می‌کند. این حالت، توان مصرفی را بیشتر کاهش می‌دهد، اما هنگام پریدن از خواب، زمان بیشتری باید منتظر بمانیم.

۶.۳.۷.۴ حالت تغذیه‌ی خاموش عمیق

این حالت، همانند حالت تغذیه خاموش است اما علاوه بر آن رگولاتورهای ولتاژ داخلی که وظیفه‌ی تأمین تغذیه‌ی مدارات داخلی را دارند نیز خاموش می‌شوند. این حالت بیشینه امکان ذخیره‌ی توان را قبل از قطع کامل تغذیه به ما می‌دهد. در این حالت، محتویات حافظه‌ها و ثبات‌ها حفظ نمی‌شوند. بازگشت از این حالت همانند ریست کردن میکروکنترولر است.

نکته

چون در حالت تغذیه‌ی خاموش عمیق، محتویات RAM حفظ نمی‌شوند، می‌توان از حافظه‌ی RAM باتری‌دار برای نگه‌داری اطلاعات مهم استفاده کرد. البته، در این حالت باید تغذیه‌ی V_{BAT} وصل شده باشد.

۶.۳.۷.۵ کنترل توان وسایل جانبی

با استفاده از این قابلیت، می‌توان تغذیه‌ی وسایل جانبی که استفاده نمی‌شوند را قطع نمود تا از مصرف توان اضافی جلوگیری کرد. این کار توسط ثبات PCONP انجام می‌شود.

Name	Description	Access	Reset value[1]	Address
PCON	Power Control Register. This register contains control bits that enable the two reduced power operating modes of the LPC2400. See Table 4–60.	R/W	0x00	0xE01F C0C0
INTWAKE	Interrupt Wakeup Register. Controls which interrupts will wake the LPC2400 from power-down mode. See Table 4–62	R/W	0x00	0xE01F C144
PCONP	Power Control for Peripherals Register. This register contains control bits that enable and disable individual peripheral functions, allowing elimination of power consumption by peripherals that are not needed.	R/W		0xE01F C0C4

جدول ۵۹

حالت‌های کاهش توان مصرفی توسط ثبات PCON کنترل می‌شود

Bit	Symbol	Description	Reset value
0	PM0 (IDL)	Power mode control bit 0. See Table 4–61 for details.	0
1	PM1 (PD)	Power mode control bit 1. See Table 4–61 for details.	0
2	BODPDM	Brown-Out Power-down Mode. When BODPDM is 1, the Brown-Out Detect circuitry will turn off when chip Power-down mode is entered, resulting in a further reduction in power usage. However, the possibility of using Brown-Out Detect as a wakeup source from Power-down mode will be lost. When 0, the Brown-Out Detect function remains active during Power-down mode. See Section 3–4 for details of Brown-Out detection.	0
3	BOGD	Brown-Out Global Disable. When BOGD is 1, the Brown-Out Detect circuitry is fully disabled at all times, and does not consume power. When 0, the Brown-Out Detect circuitry is enabled. See Section 3–4 for details of Brown-Out detection.	0
4	BORD	Brown-Out Reset Disable. When BORD is 1, the second stage of low voltage detection (2.6 V) will not cause a chip reset. When BORD is 0, the reset is enabled. The first stage of low voltage detection (2.9 V) Brown-Out interrupt is not affected. See Section 3–4 for details of Brown-Out detection.	0
6:3	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
7	PM2	Power mode control bit 2. See Table 4–61 for details.	0

جدول ۶۰

PM2, PM1, PM0	Description
000	Normal operation
001	Idle mode. Causes the processor clock to be stopped, while on-chip peripherals remain active. Any enabled interrupt from a peripheral or an external interrupt source will cause the processor to resume execution. See Section 4–3.4.1 for details.
101	Sleep mode. This mode is similar to Power-down mode (the oscillator and all on-chip clocks are stopped), but the flash memory is left in Standby mode. This allows a more rapid wakeup than Power-down mode because the flash reference voltage regulator start-up time is not needed. See Section 4–3.4.2 for details.
010	Power-down mode. Causes the oscillator and all on-chip clocks to be stopped. A wakeup condition from an external interrupt can cause the oscillator to re-start, the PD bit to be cleared, and the processor to resume execution. See Section 4–3.4.3 for details.
110	Deep power-down mode. This is the most extreme power saving mode. As in Power-down mode, Deep power-down mode causes the oscillator and all on-chip clocks to be stopped, but also turns off the on-chip DC-DC converter that supplies power to internal circuitry. See Section 4–3.4.4 for details.
Others	Reserved, not currently used.

جدول ۶۱

توسط ثبات INTWAKE می‌توان مشخص کرد که کدام وسیله‌ی جانبی می‌تواند پردازنده را از حالت تغذیه خاموش خارج کند.

Bit	Symbol	Description	Reset value
0	EXTWAKE0	When one, assertion of $\overline{\text{EINT0}}$ will wake up the processor from Power-down mode.	0
1	EXTWAKE1	When one, assertion of $\overline{\text{EINT1}}$ will wake up the processor from Power-down mode.	0
2	EXTWAKE2	When one, assertion of $\overline{\text{EINT2}}$ will wake up the processor from Power-down mode.	0
3	EXTWAKE3	When one, assertion of $\overline{\text{EINT3}}$ will wake up the processor from Power-down mode.	0
4	ETHWAKE	When one, assertion of the Wake-up on LAN interrupt (WakeupInt) of the Ethernet block will wake up the processor from Power-down mode.	0
5	USBWAKE	When one, activity on the USB bus will wake up the processor from Power-down mode. Any change of state on the USB data pins will cause a wakeup when this bit is set. For details on the relationship of USB to Power-down Mode and wakeup, see the relevant USB chapter(s).	0
6	CANWAKE	When one, activity of the CAN bus will wake up the processor from Power-down mode. Any change of state on the CAN receive pins will cause a wakeup when this bit is set.	0
7	GPIO0WAKE	When one, specified activity on GPIO pins on port 0 enabled for wakeup will wake up the processor from Power-down mode. For configuring the port 0 pins, see Section 10–6.6 .	0
8	GPIO2WAKE	When one, specified activity on GPIO pins on port 2 enabled for wakeup will wake up the processor from Power-down mode. For configuring the port 2 pins, see Section 10–6.6 .	0
13:9	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA
14	BODWAKE	When one, Brown-Out Detect interrupt will wake up the processor from Power-down mode. **Note:** since there is a delay before execution begins, there is no guarantee that execution will resume before $V_{DD(3V3)}$ has fallen below the lower BOD threshold, which prevents execution. If execution does resume, there is no guarantee of how long the processor will continue execution before the lower BOD threshold terminates execution. These issues depend on the slope of the decline of $V_{DD(DCDC)(3V3)}$. High decoupling capacitance (between $V_{DD(DCDC)(3V3)}$ and ground) in the vicinity of the LPC2400 will improve the likelihood that software will be able to do what needs to be done when power is in the process of being lost.	0
15	RTCWAKE	When one, assertion of an RTC interrupt will wake up the processor from Power-down mode.	0

جدول ۶۲

Bit	Symbol	Description	Reset value
0	-	Unused, always 0	0
1	PCTIM0	Timer/Counter 0 power/clock control bit.	1
2	PCTIM1	Timer/Counter 1 power/clock control bit.	1
3	PCUART0	UART0 power/clock control bit.	1
4	PCUART1	UART1 power/clock control bit.	1
5	PCPWM0	PWM0 power/clock control bit.	1
6	PCPWM1	PWM1 power/clock control bit.	1
7	PCI2C0	The I^2C0 interface power/clock control bit.	1
8	PCSPI	The SPI interface power/clock control bit.	1
9	PCRTC	The RTC power/clock control bit.	1
10	PCSSP1	The SSP1 interface power/clock control bit.	1
11	PCEMC	External Memory Controller	1
12	PCAD	A/D converter (ADC) power/clock control bit. **Note:** Clear the PDN bit in the AD0CR before clearing this bit, and set this bit before setting PDN.	0
13	PCCAN1	CAN Controller 1 power/clock control bit.	0
14	PCCAN2	CAN Controller 2 power/clock control bit.	0
18:15	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	
19	PCI2C1	The I^2C1 interface power/clock control bit.	1
20	PCLCD[1]	LCD controller power control bit.	0
21	PCSSP0	The SSP0 interface power/clock control bit.	1
22	PCTIM2	Timer 2 power/clock control bit.	0
23	PCTIM3	Timer 3 power/clock control bit.	0
24	PCUART2	UART 2 power/clock control bit.	0
25	PCUART3	UART 3 power/clock control bit.	0
26	PCI2C2	I^2S interface 2 power/clock control bit.	1
27	PCI2S	I^2S interface power/clock control bit.	0
28	PCSDC	SD card interface power/clock control bit.	0
29	PCGPDMA	GP DMA function power/clock control bit.	0
30	PCENET	Ethernet block power/clock control bit.	0
31	PCUSB	USB interface power/clock control bit.	0

جدول ۶۳

۶.۳.۸ کنترل کننده‌ی وقفه‌ی برداری

پردازنده‌ی ARM ، دارای دو ورودی وقفه با نام‌های IRQ و FIQ است. VIC به صورت کنترل شده و با برنامه‌ای مشخص، ورودی‌های مختلف از ۳۲ منبع وقفه را می‌گیرد و به دو پایه‌ی FIQ و یا IRQ متصل می‌کند. این روش برنامه‌ریزی، شامل اختصاص دادن حقوق تقدم به هر یک از منابع وقفه نیز می‌شود.

وقفه FIQ ، دارای بالاترین تقدم است. اگر بیش از یک وقفه برای اتصال FIQ اختصاص داده شده باشد، VIC همگی آن‌ها را با هم OR می‌شوند. سریع‌ترین حالت رسیدگی به FIQ وقتی است که تنها یک وقفه به FIQ اختصاص داده شده باشد. زیرا در این حالت نیازی به مشخص کردن منبع تولید وقفه نیست.

IRQ ها نیز دارای حقوق تقدم زیادی هستند. هر وقفه‌ای که به FIQ اختصاص نداشته یاشد، در گروه IRQ ها است. اگر چندین وقفه اختصاص یافته با یک سطح تقدم، به طور هم‌زمان فعال شوند، آن وقفه‌ای که دارای شماره‌ی کانال VIC کمتری است، اول اجرا می‌شود. (جدول 116-7)

۶.۳.۸.۱ ثبات‌های VIC

ثبات VIC Soft Int ، برای تولید وقفه‌ی نرم‌افزاری استفاده می‌شود. قبل از انجام هر کار دیگری، محتویات این ثبات با ۳۲ درخواست وقفه‌ی دیگر از وسایل جانبی، OR می‌شود. '1' کردن هر بیت از این ثبات، موجب ایجاد وقفه‌ی مورد نظر و هم‌شماره با **مکان بیت**، می‌شود. ثبات VIC Spft Int Clear فقط قابل نوشتن است و به نرم‌افزار اجازه می‌دهد بدون خواندن محتویات VIC Soft Int، بیت‌های خاصی را در آن صفر کند. نوشتن بیت '1' در هر یک از مکان‌ها، بیت مورد نظر در VIC Soft Int را پاک می‌کند.

ثبات VIC Row Int ، فقط خواندنی بوده و نشان‌دهنده‌ی وضعیت وقوع وقفه‌ها می‌باشد.

ثبات VIC Int Enable تعیین می‌کند که کدام‌یک از وقفه‌های ۳۲ گانه به سیگنال‌های FIQ و IRQ متصل شده‌اند. هنگام نوشتن بر روی این ثبات، '1' ها مشخص می‌کنند که کدام وقفه‌ها می‌توانند تولید شوند.

ثبات VIC Int En Clear ، می‌تواند برای ناتوان کردن وقفه‌ها در ثبات توانا کننده‌ی وقفه، مورد استفاده قرار بگیرد.

از ثبات انتخاب وقفه VIC Int Enable ، می‌توان برای طبقه‌بندی وقفه‌ها در دو گروه FIQ و IRQ استفاده کرد. بیت‌های '1' در این ثبات ۳۲ بیتی، وقفه‌ی مورد نظر را از نوع FIQ و بیت‌های '0' آن را از نوع IRQ قرار می‌دهد.

ثبات‌های VIC Vect Addr0-31 آدرس‌های سرویس‌دهنده‌های وقفه (ISR) را برای ۳۲ وقفه IRQ، در خود جای می‌دهد و ثبات‌های VIC Vect Priority0-31 تقدم وقفه‌های IRQ را مشخص می‌کنند. ۱۶ سطح مختلف برای وقفه‌های IRQ وجود دارند که متناظر با اعداد ۰ تا ۱۵ هستند. ۱۵ در آن متناظر

با پایین‌ترین سطح تقدم است. حالت پیش‌فرض این ثبات‌ها در هنگام شروع به کار، برابر با پایین‌ترین سطح تقدم می‌باشد. تنها ۴ بیت پایین این ثبات باارزش هستند.
هنگامی که یک وقفه‌ی IRQ رخ می‌دهد، ثبات VIC Address حاوی آدرس ISR مربوط به آن وقفه است.

۶.۳.۹ واسط SPI

واسط SPI یک ارتباط سریال دوطرفه‌ی کامل[۱] است. ارتباط SPI به صورت گذرگاه پیاده سازی می‌شود و بر روی گذرگاه دارای فرمانده[۲] و فرمانبردار[۳] است. در طی یک فرآیند انتقال داده، تنها **یک فرمانده** و یک فرمانبردار می‌توانند به طور هم‌زمان فعال باشند. در طی هر انتقال داده، همواره فرمانده ۸ تا ۱۶ بیت داده را به سمت فرمانبردار می‌فرستد و فرمانبردار همواره یک بایت داده به سمت فرمانده می‌فرستد.

---

[۱] Full Duplex
[۲] Master
[۳] Slave

۶.۳.۹.۱ مشخصات SPI

- ارتباط سنکرون، سریال و دوطرفه‌ی کامل؛
- فرمانده یا فرمانبردار در گذرگاه SPI؛
- حداکثر نرخ انتقال داده‌ها ۱/۸ نرخ کلاک ورودی است؛
- بین ۸ تا ۱۶ بیت در هر انتقال می‌تواند جابه‌جا شود.

۶.۳.۹.۲ پیکره‌بندی ابتدایی SPI

۱. تغذیه: بیت PCSPI را در ثبات PCONP به '1' تنظیم کنید؛

توجه

SPI به هنگام ریست روشن است؛

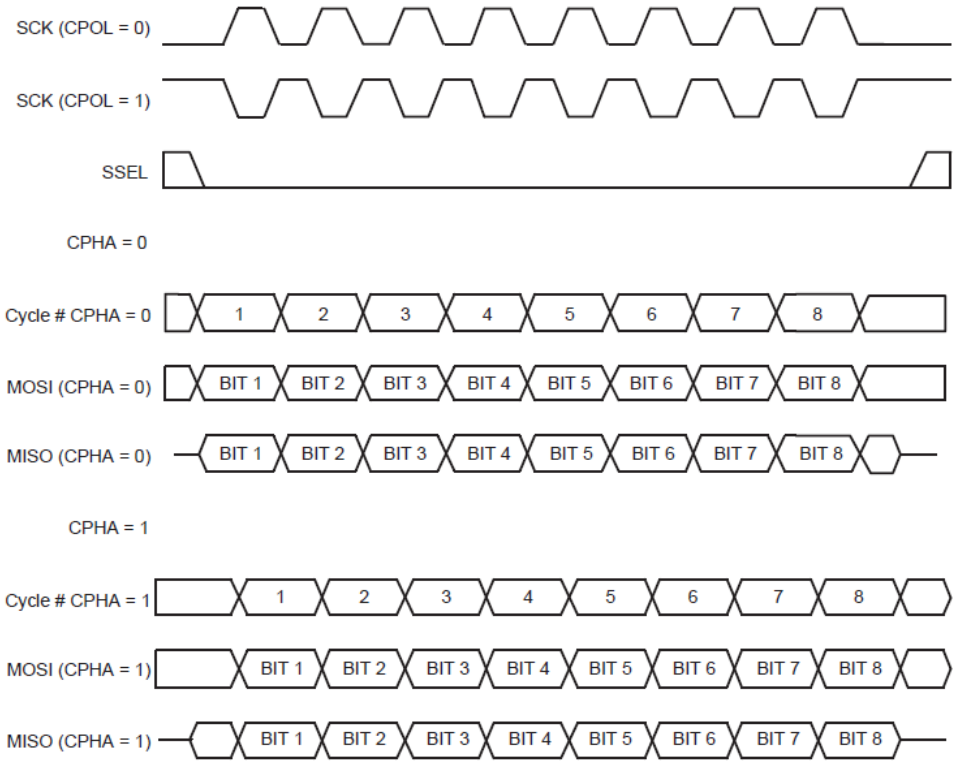
۲. کلاک: SPI_PCLK را در SEL0_PCLK انتخاب کنید؛
۳. پایه‌ها: پایه‌های SPI و حالت‌هایشان را در PINSEL0 تا PINSEL4 و PINMODE0 تا PINMODE4 انتخاب کنید؛
۴. وقفه‌ها: وقفه‌ها در ثبات S0SPINT فعال می‌شوند. وقفه‌ها در VIC توسط VIC Int Enable فعال می‌شوند.

توجه

در VIC، SPI و SSP0 دارای وقفه‌های مشترکی هستند.

۶.۳.۹.۳ انتقال داده در SPI

شکل ۶.۶، چهار حالت مختلف انتقال داده‌ای که در SPI استفاده می‌شود را به صورت یک نمودار زمانی نشان می‌دهد.



شکل ۶.۶

در این نمودار، ۸ بیت داده منتقل داده شده است. اولین چیزی که باید بدان توجه کنید این‌است که این نمودار به ۳ بخش افقی تقسیم می‌شود. اولین بخش آن سیگنال‌های SCK و SSEL را توضیح می‌دهند. در بخش دوم، سیگنال‌های MOSI و MISO هنگامی که CPHA=0 است و در بخش سوم، همین سیگنال‌ها هنگامی که CPHA=1 است به نمایش درآمده‌اند.

در بخش اول دقت کنید که SPI در هر دو حالت CPOL=0 و CPOL=1 درنظر گرفته شده است. رابطه‌ی بین داده و فاز کلاک در جدول ۴۵۹، خلاصه شده است. این جدول خلاصه‌ی حالات مختلف SPI را نشان می‌دهد.

CPOL and CPHA settings	First data driven	Other data driven	Data sampled
CPOL = 0, CPHA = 0	Prior to first SCK rising edge	SCK falling edge	SCK rising edge
CPOL = 0, CPHA = 1	First SCK rising edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 0	Prior to first SCK falling edge	SCK rising edge	SCK falling edge
CPOL = 1, CPHA = 1	First SCK falling edge	SCK falling edge	SCK rising edge

جدول ۴۵۹

۶.۳.۹.۴ عملکرد SPI در حالت فرمانده[۱]

در زیر، خواهیم دید که چگونه باید انتقال داده‌ها را در حالت فرمانده، مدیریت کرد. در این سلسله فرآیند فرض بر این‌است که هرگونه نقل و انتقال قبلی داده‌ها پایان یافته است.

۱. ثبات شمارنده کلاک SPI را به نرخ کلاک مورد نظر تنظیم کنید؛
۲. ثبات کنترلی SPI را با تنظیمات مورد نظر پر کنید؛
۳. داده‌ای که قرار است ارسال شود را در ثبات داده‌ی SPI بنویسید. به محض نوشتن، عملیات انتقال شروع می‌شود؛
۴. برای '1' شدن بیت SPIF در ثبات وضعیت منتظر بمانید. این بیت در آخرین سیکل انتقال داده‌ی SPI '1' می‌شود؛
۵. ثبات وضعیت SPI را بخوانید؛
۶. داده دریافت شده را از ثبات داده‌ی SPI بخوانید (اختیاری)؛
۷. اگر نیاز به داده بیشتری برای انتقال دارید، به مرحله‌ی ۳ بروید.

۶.۳.۹.۵ عملکرد SPI در حالت فرمانبردار[۲]

۱. ثبات کنترلی SPI را با تنظیمات مورد نظر پر کنید؛
۲. داده‌ای که قراراست در SPI نوشته شود را در ثبات داده بنویسید (اختیاری)؛
۳. منتظر '1' شدن بیت SPIF در ثبات وضعیت SPI بمانید؛
۴. ثبات وضعیت SPI را بخوانید؛
۵. داده‌ی دریافت شده از SPI را بخوانید (اختیاری)؛
۶. اگر داده بیشتری مورد نیاز است، به مرحله‌ی ۲ بروید.

پایه‌های SPI را در جدول ۴۶۰ می‌بینید.

---

[۱] Master
[۲] Slave

Pin Name	Type	Pin Description
SCK	Input/ Output	**Serial Clock.** The SPI is a clock signal used to synchronize the transfer of data across the SPI interface. The SPI is always driven by the master and received by the slave. The clock is programmable to be active high or active low. The SPI is only active during a data transfer. Any other time, it is either in its inactive state, or tri-stated.
SSEL	Input	**Slave Select.** The SPI slave select signal is an active low signal that indicates which slave is currently selected to participate in a data transfer. Each slave has its own unique slave select signal input. The SSEL must be low before data transactions begin and normally stays low for the duration of the transaction. If the SSEL signal goes high any time during a data transfer, the transfer is considered to be aborted. In this event, the slave returns to idle, and any data that was received is thrown away. There are no other indications of this exception. This signal is not directly driven by the master. It could be driven by a simple general purpose I/O under software control. On the LPC2400 (unlike earlier NXP ARM devices) the SSEL pin can be used for a different function when the SPI interface is only used in Master mode. For example, pin hosting the SSEL function can be configured as an output digital GPIO pin and used to select one of the SPI slaves.
MISO	Input/ Output	**Master In Slave Out.** The MISO signal is a unidirectional signal used to transfer serial data from the slave to the master. When a device is a slave, serial data is output on this signal. When a device is a master, serial data is input on this signal. When a slave device is not selected, the slave drives the signal high impedance.
MOSI	Input/ Output	**Master Out Slave In.** The MOSI signal is a unidirectional signal used to transfer serial data from the master to the slave. When a device is a master, serial data is output on this signal. When a device is a slave, serial data is input on this signal.

جدول ۴۶۰

ثبات‌های SPI را نیز در جدول ۴۶۱ می‌بینید.

۶.۴ میکروکنترولرهای STM32

این قسمت شامل مثال‌هایی است که از آن طریق، به آموزش نحوه‌ی کار با میکروکنترولرهای STM32 خواهیم پرداخت. این مثال ها بر روی برد آموزشی STM3210E-EVAL شرکت STMicroelectronics به راحتی قابل اجرا هستند. ولی استفاده از هر برد شخصی، با نقشه مشابه نیز نتایج یکسانی را در بر دارد.
اگر از محیط‌های برنامه نویسی IAR و یا Keil استفاده می کنید، فایل پروژه، فایل‌های برنامه به زبان C و ... به همراه تنظیمات لازم، در دایرکتوری مثال‌ها آورده شده است. در غیر این‌صورت، می توانید برای اجرای هر یک از مثال‌هایی که در ادامه می آیند، روند زیر را دنبال کنید:

- یک پروژه خالی ایجاد کرده و تمامی فایل‌های شروع به کار (یا Start-up) کامپایلر را تنظیم کنید؛
- محتویات دایرکتوری حاوی فایل‌های مثال را به همراه فایل‌های ضروری داده شده در انتهای هر مثال کامپایل کنید؛
- تمامی فایل‌ها را لینک کرده و سپس فایل باینری تولید شده را بر روی برد مورد نظر دانلود کنید؛
- اکنون برنامه را اجرا کنید.

مثال ۱۲.۱

در این مثال، هر یک از ۳ مبدل ADC ، به طور جداگانه استفاده می شوند. خروجی ۲ مبدل ADC1 و ADC3 به طور پیوسته و از طریق واسط DMA و خروجی ADC2 از طریق وقفهٔ "پایان عملیات تبدیل" به دست می آیند.

ADC1 ، برای تبدیل پیوسته مقادیر کانال ۱۴ پیکره بندی شده است. به هنگام وقوع "پایان عملیات تبدیل"، DMA1 به طور گردشی داده موجود در ADC1_DR را به درون متغیر ADC1_ConvertedValue کپی می کند.

ADC2 ، برای تبدیل پیوسته مقادیر کانال ۱۳ پیکره بندی شده است. به هنگام وقوع "پایان عملیات تبدیل"، یک وقفهٔ متناظر ایجاد شده و در درون ISR مربوط، داده موجود در ADC2_DR به درون متغیر ADC2_ConvertedValue کپی می شود.

ADC3 برای تبدیل پیوسته مقادیر کانال ۱۲ پیکره بندی شده است. به هنگام وقوع "پایان عملیات تبدیل"، DMA1 به طور گردشی داده موجود در ADC3_DR را به درون متغیر ADC3_ConvertedValue کپی می کند.

فرکانس کلاک ADC ها بر روی 14MHz تنظیم شده است. نتایج تبدیل هر سه ADC نیز از طریق سه متغیر ADCx_ConvertedValue (x شماره ADC متناظر است) در دسترس است. محدوده‌ی اندازه گیری ADC ها بین 0V تا 3.3V است. کانالهای ADC نیز مطابق زیر به پایه های میکروکنترولر متصل هستند :

کانال ADC	پایه
کانال ۱۲	PC.02
کانال ۱۳	PC.03
کانال ۱۴	PC.04

```
  /* System clocks configuration ----------------------*/
  RCC_Configuration();

  /* NVIC configuration -------------------------------*/
  NVIC_Configuration();

  /* GPIO configuration -------------------------------*/
  GPIO_Configuration();

  /* DMA1 channel1 configuration ----------------------*/
  DMA_DeInit(DMA1_Channel1);
  DMA_InitStructure.DMA_PeripheralBaseAddr = ADC1_DR_Address;
  DMA_InitStructure.DMA_MemoryBaseAddr =
(u32)&ADC1ConvertedValue;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
  DMA_InitStructure.DMA_BufferSize = 1;
  DMA_InitStructure.DMA_PeripheralInc =
DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
  DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
```

```
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
  DMA_Init(DMA1_Channel1, &DMA_InitStructure);
  /* Enable DMA1 channel1 */
  DMA_Cmd(DMA1_Channel1, ENABLE);

  /* DMA2 channel5 configuration ----------------------*/
  DMA_DeInit(DMA2_Channel5);
  DMA_InitStructure.DMA_PeripheralBaseAddr = ADC3_DR_Address;
  DMA_InitStructure.DMA_MemoryBaseAddr =
(u32)&ADC3ConvertedValue;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralSRC;
  DMA_InitStructure.DMA_BufferSize = 1;
  DMA_InitStructure.DMA_PeripheralInc =
DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
  DMA_InitStructure.DMA_PeripheralDataSize =
DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize =
DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  DMA_InitStructure.DMA_M2M = DMA_M2M_Disable;
  DMA_Init(DMA2_Channel5, &DMA_InitStructure);
  /* Enable DMA2 channel5 */
  DMA_Cmd(DMA2_Channel5, ENABLE);

  /* ADC1 configuration ------------------------------*/
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC1, &ADC_InitStructure);
  /* ADC1 regular channels configuration */
  ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 1,
ADC_SampleTime_28Cycles5);
  /* Enable ADC1 DMA */
  ADC_DMACmd(ADC1, ENABLE);

  /* ADC2 configuration ------------------------------*/
  ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_None;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfChannel = 1;
  ADC_Init(ADC2, &ADC_InitStructure);
  /* ADC2 regular channels configuration */
  ADC_RegularChannelConfig(ADC2, ADC_Channel_13, 1,
ADC_SampleTime_28Cycles5);
  /* Enable ADC2 EOC interupt */
  ADC_ITConfig(ADC2, ADC_IT_EOC, ENABLE);
```

```
77
78    /* ADC3 configuration ------------------------------------*/
79    ADC_InitStructure.ADC_Mode = ADC_Mode_Independent;
      ADC_InitStructure.ADC_ScanConvMode = DISABLE;
80    ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
81    ADC_InitStructure.ADC_ExternalTrigConv =
82  ADC_ExternalTrigConv_None;
83    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
84    ADC_InitStructure.ADC_NbrOfChannel = 1;
85    ADC_Init(ADC3, &ADC_InitStructure);
86    /* ADC3 regular channel14 configuration */
87    ADC_RegularChannelConfig(ADC3, ADC_Channel_12, 1,
88  ADC_SampleTime_28Cycles5);
89    /* Enable ADC3 DMA */
90    ADC_DMACmd(ADC3, ENABLE);
91
92    /* Enable ADC1 */
93    ADC_Cmd(ADC1, ENABLE);
94
95    /* Enable ADC1 reset calibaration register */
96    ADC_ResetCalibration(ADC1);
97    /* Check the end of ADC1 reset calibration register */
98    while(ADC_GetResetCalibrationStatus(ADC1));
99
100   /* Start ADC1 calibaration */
101   ADC_StartCalibration(ADC1);
102   /* Check the end of ADC1 calibration */
103   while(ADC_GetCalibrationStatus(ADC1));
104
105   /* Enable ADC2 */
106   ADC_Cmd(ADC2, ENABLE);
107
108   /* Enable ADC2 reset calibaration register */
109   ADC_ResetCalibration(ADC2);
110   /* Check the end of ADC2 reset calibration register */
111   while(ADC_GetResetCalibrationStatus(ADC2));
112
113   /* Start ADC2 calibaration */
114   ADC_StartCalibration(ADC2);
115   /* Check the end of ADC2 calibration */
116   while(ADC_GetCalibrationStatus(ADC2));
117
118   /* Enable ADC3 */
119   ADC_Cmd(ADC3, ENABLE);
120
121   /* Enable ADC3 reset calibaration register */
122   ADC_ResetCalibration(ADC3);
123   /* Check the end of ADC3 reset calibration register */
124   while(ADC_GetResetCalibrationStatus(ADC3));
125
126   /* Start ADC3 calibaration */
127   ADC_StartCalibration(ADC3);
128   /* Check the end of ADC3 calibration */
129   while(ADC_GetCalibrationStatus(ADC3));
130
```

```
131   /* Start ADC1 Software Conversion */
      ADC_SoftwareStartConvCmd(ADC1, ENABLE);
      /* Start ADC2 Software Conversion */
      ADC_SoftwareStartConvCmd(ADC2, ENABLE);
      /* Start ADC3 Software Conversion */
      ADC_SoftwareStartConvCmd(ADC3, ENABLE);

      while (1)
      {
      }
```

فایل‌های ضروری

stm32f10x_lib.c

stm32f10x_adc.c

stm32f10x_dma.c

stm32f10x_gpio.c

stm32f10x_rcc.c

stm32f10x_nvic.c

stm32f10x_flash.c

مثال ۱۲.۲

در این مثال، ADC1 و ADC2 در حالت تبدیل همزمان عادی به کار گرفته شده اند. ADC1 برای تبدیل پیوسته کانال‌های ۱۴ و ۱۷ در نظر گرفته شده اند. و ADC2 نیز کانال‌های ۱۱ و ۱۲ را تبدیل می کنند. در داخل میکروکنترولر نیز ارتباط Vref به کانال ۱۷ مربوط به ADC1 برقرار شده است.

با شروع نرم افزاری عملیات تبدیل، کانال ۱۴ از ADC1 و کانال ۱۱ از ADC2 به طور همزمان شروع به کار می کنند. سپس، نتایج ۳۲ بیتی حاصل در درون ADCx_DR ذخیره می شوند. DMA نیز این داده ها را به درون جدولی به نام ADC_DualConvertedValueTab انتقال خواهد داد. بلافاصله بعد از تبدیل‌های فوق، دو تبدیل دیگر از کانال ۱۷ بر روی ADC1 و کانال ۱۲ بر روی ADC2 صورت خواهند گرفت.

مجموعه‌ی عملیات فوق تا زمانی‌که تعداد مشخصی از انتقال داده در DMA پایان پذیرد، تکرار می شود. فرکانس کلاک ADC ها بر روی 14MHz تنظیم شده است.

فایل‌های ضروری

+ stm32f10x_lib.c

+ stm32f10x_adc.c

+ stm32f10x_dma.c

+ stm32f10x_gpio.c

+ stm32f10x_rcc.c

+ stm32f10x_nvic.c

+ stm32f10x_flash.c

مثال ۱۲.۳

این مثال، نشان می‌دهد که داده های کاربر را چگونه بر روی ثبات‌های داده‌ی پشتیبان [Backup Data Registers] می توان ذخیره کرد. از آن‌جایی‌که قسمت‌های سخت افزاری مربوط به بخش پشتیبان، با استفاده از پایه‌ی V_{BAT} تغذیه می شوند ، هنگام قطع VDD ، محتویات آن‌ها پاک نمی شوند.

برنامه زیر این‌گونه عمل می کند:

۱. هنگام شروع به کار سیستم، برنامه روی تراشه وضعیت تغذیه سیستم را بررسی می کند (که آیا تغذیه وصل شده است یا نه ؟) سپس اگر تغذیه وصل باشد ، محتویات ثبات‌های پشتیبان آزمایش می شوند:

أ. اگر یک باتری به پایه VBAT متصل شده باشد، محتویات ثبات‌های پشتیبان حفظ خواهند شد؛

ب. اگر باتری به پایه VBAT متصل نشده باشد، محتویات ثبات‌های پشتیبان پاک خواهند شد.

۲. بعد از اعمال ریست خارجی، محتویات ثبات‌های پشتیبان آزمایش نمی شوند.

چهار LED ، به عنوان نشان دهنده‌ی حالت‌های مختلف سیستم به پایه های GPIO (Pin6 (LD1), Pin7 (LD2), Pin8 (LD3) and Pin9 (LD4)) متصل شده اند. وضعیت سیستم به این صورت است :

۱. LD3 روشن و LD1 روشن : یک ریستِ "هنگامِ وصل تغذیه"، رخ داده است و محتویات ثبات‌های پشتیبان صحیح هستند؛
۲. LD3 روشن و LD2 روشن : یک ریستِ هنگامِ وصل تغذیه رخ داده است ولی محتویات ثبات-های پشتیبان صحیح نیستند و یا تا کنون برنامه ریزی نشده اند (مثلا در حالتی که برای اولین بار است که برنامه اجرا می شود)؛
۳. LD1 خاموش و LD2 خاموش و LD3 خاموش : ریستِ هنگامِ وصل تغذیه رخ نداده است؛
۴. LD4 روشن : برنامه در حال اجراست.

برای راه اندازی این مثال، باید یک باتری 3V سکه ای را در سوکت مربوط قرار دهید.

فایل‌های ضروری

+ stm32f10x_lib.c
+ stm32f10x_pwr.c
+ stm32f10x_bkp.c
+ stm32f10x_gpio.c
+ stm32f10x_rcc.c
+ stm32f10x_nvic.c
+ stm32f10x_flash.c

این مثال را بدون اتصال دیباگر به برد، اجرا کنید.

تغذیه برد را خاموش و سپس روشن کرده، ببینید که محتویات پشتیبانی شده تغییر نکرده است.

مثال ۱۲.۴

در این مثال، نحوه‌ی ارتباط با Can در حالت loopback را خواهیم دید. سلول CAN ، ابتدا یک فریم داده استاندارد را با سرعت 100Kbit/s ارسال و سپس دریافت می کند. فریم دریافتی برای اطمینان از صحت دریافت، آزمایش شده و سپس چند LED به نشانه موفقیت در عملیات دریافت، روشن می شوند. سپس، یک فریم بلند تر داده با سرعت 500Kbits/s ارسال می‌گردد. عملیات در یافت در ISR انجام می‌شود. سپس، چند LED به نشانه موفقیت در عملیات ارسال و دریافت، روشن خواهند شد.

فایل‌های ضروری

+ stm32f10x_lib.c

+ stm32f10x_can.c

+ stm32f10x_rcc.c

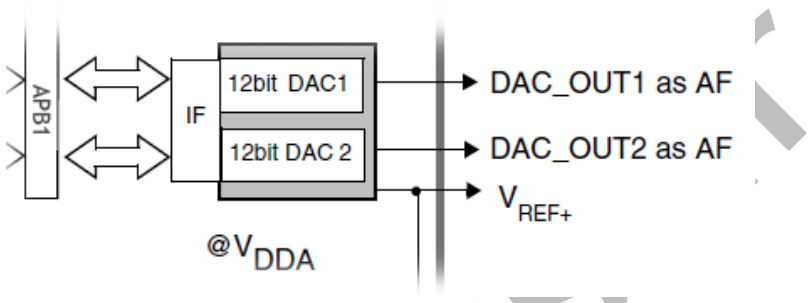
+ stm32f10x_gpio.c

+ stm32f10x_nvic.c

+ stm32f10x_flash.c

مثال ۱۲.۴

در این مثال، از دو DAC به صورت دوگانه و برای تولید شکل موج سینوسی با استفاده از DMA بهره گرفته ایم. پیکره بندی تمامی کانال‌های DAC این‌گونه‌اند که با سیگنال TIM8 TRGO تحریک شده[Trigger] و مولد موج مثلثی و یا نویز نیز نیستند. به علت این‌که در این‌جا از ثبات DAC_DHR12RD استفاده می کنیم، داده‌ی ۱۲ بیتی مرتب شده از راست[Right Aligned] انتخاب شده است.



شکل ۶.۷

کانال ۱۴ از DMA2 ، برای انتقال پیوسته محتویات یک بافر ۳۲ کلمه ای، به صورت کلمه به کلمه، به ثبات "DAC دوگانه" DAC_DHR12RD در نظر گرفته شده است.

محتویات بافر ۳۲ کلمه ای مذکور، برای تولید شکل موج سینوسی در هر دو کانال تنظیم شده است. در این‌جا فقط کانال ۲ از DMA استفاده می کند.

به محض این‌که TIM8 فعال شود، با هر رویداد TIM8 TRGO یک درخواست DMA صورت می گیرد و با این درخواست، یک مقدار عددی به ثبات DAC منتقل شده و سپس عملیات تبدیل شروع می شود. با اتصال پایه های PA.04 و PA.05 به اسیلوسکوپ، شکل موج‌های سینوسی را خواهید دید.

فایل‌های ضروری

+ stm32f10x_lib.c

+ stm32f10x_gpio.c

+ stm32f10x_rcc.c

+ stm32f10x_nvic.c

+ stm32f10x_flash.c

+ stm32f10x_dac.c

+ stm32f10x_dma.c

+ stm32f10x_tim.c

مثال ۱۲.۴

در این مثال، نحوه‌ی استفاده از یک کانال DAC را برای تولید شکل موج سیگنال به همراه نویز خواهیم دید. در این‌جا عملیات تبدیل کانال DAC1 با استفاده از نرم افزار شروع می شود. در این‌جا از داده ۱۲ بیتی مرتب شده از چپ (DAC_DHR12L1) استفاده شده است.

کانال DAC1 فعال شده، ثبات DHR12L1 مربوط به کانال ۱ مبدل، برای ایجاد یک خروجی معادل $V_{REF}/2$ تنظیم شده است. تحریک‌های نرم افزاری در یک حلقه نامتناهی، انجام می شوند. با هر بار تحریک، تبدیل را آغاز کرده و مقدار نویزی که باید به خروجی کانال ۱ اعمال کند را محاسبه می کند.

سیگنال خروجی به همراه نویز را می توان با اتصال پراب اسیلوسکوپ به پایه PA.04 دید.

فایل‌های ضروری

+ stm32f10x_lib.c

+ stm32f10x_gpio.c

+ stm32f10x_rcc.c

+ stm32f10x_nvic.c

+ stm32f10x_flash.c

+ stm32f10x_dac.c

مثال ۱۲.۴

در این مثال، از DMA برای انتقال یک مقدار عددی از یک وسیله جانبی میروکنترولر (ADC1) به وسیله‌ای دیگر (TIM1) استفاده شده است. در این‌جا کانال ۱۴ ADC به طور پیوسته کار کرده و کانال ۱ تایمر ۱ (TIM1_CH1) نیز برای تولید یک شکل موج PWM در خروجی اش پیکره بندی شده است.

کانال ۵، از DMA1 برای انتقال دایره‌ای آخرین مقدار تبدیل شده ADC به ثبات TIM1_CCR1 پیکره بندی گردیده است. سیگنال تحریک DMA نیز توسط رویداد به روز رسانی[Update event] TIM1 ساخته می شود. پس، با تغییر مقدار ولتاژ آنالوگ ورودی، سیکل کاری[Duty cycle] سیگنال PWM تولید شده تغییر خواهد کرد.

این تغییرات، با اتصال پایه TIM1_CH1 (PA.08) به اسیلوسکوپ و تغییر ولتاژ کانال ۱۴ ADC (به عنوان مثال با یک پتانسیومتر) قابل مشاهده هستند.

فایل‌های ضروری

+ stm32f10x_lib.c
+ stm32f10x_dma.c
+ stm32f10x_rcc.c
+ stm32f10x_gpio.c
+ stm32f10x_adc.c
+ stm32f10x_tim.c
+ stm32f10x_nvic.c
+ stm32f10x_flash.c

مثال ۱۲.۴

در این مثال، از کانال ۶ رابط DMA برای انتقال محتویات یک بافر ۳۲ کلمه ای داخل FLASH به درون SRAM ، استفاده کرده ایم.

دستور شروع عملیات انتقال، با استفاده از نرم افزار صادر می شود. انتقال **حافظه-به-حافظه** و افزایش خودکار آدرس‌های مبدا و مقصد، در بخش DMA فعال شده است. شروع کار با '1' کردن بیت فعال سازی DMA در کانال مربوط صورت می گیرد. با اتمام کار، وقفه‌ی "پایان عملیات انتقال" تولید شده و سپس، محتویات بافرهای مبدا و مقصد با یکدیگر مقایسه می شوند تا صحت عملیات انتقال تایید گردد.

فایل‌های ضروری

+ stm32f10x_lib.c

+ stm32f10x_dma.c

+ stm32f10x_rcc.c

+ stm32f10x_nvic.c

+ stm32f10x_flash.c

مثال ۱۲.۴

در این مثال، نحوه‌ی پیکره بندی وقفه های خارجی را خواهیم آموخت. در این‌جا پایه‌ی EXTI برای تولید وقفه در لبه پایین رونده، پیکره بندی شده است. با هر وقفه‌ی تولیدی، یک LED متصل به یکی از پایه های GPIO تغییر وضعیت می دهد.(از خاموش به روشن یا بر عکس)

در این مثال، LED1 به PF.06 و یک کلید فشاری به ورودی وقفه‌ی EXTI8 (پایه PG.08) متصل شده-اند.

بخش ۴

برنامه‌نویسی و نرم‌افزار

فصل هفتم

نرم‌افزار و محیطهای برنامه‌نویسی

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ محیط‌های برنامه نویسی ARM؛
- ✓ آشنایی با برنامه ریزی میکروکنترولرهای ARM؛
- ✓ آشنایی مقدماتی با زبان C.

۷.۱ پیدایش C/ C++

C++ تنها زبان مهمی است که هر برنامه نویس می‌تواند آن را فرابگیرد. این جمله، عبارتی قوی است و گزافه گویی نیست. C++ مرکز جاذبه ایست که تمامی برنامه نویسی‌های مدرن حول آن می‌چرخند. عبارات نحوی و دستوری آن و فلسفه‌ی طراحیش بیانگر ضرورت برنامه نویسی شئ‌گرا می‌باشند. علاوه بر این‌ها، C++ مسیر توسعه‌ای برای زبان‌های آینده فراهم می‌آورد. برای مثال هر دو زبان‌های برنامه نویسی Java و C# مستقیماً زاده‌ی C++ هستند.

C++ یک زبان فراگیر برای برنامه نویسی است. برنامه نویسان ایده‌های خود را از این طریق با یکدیگر به اشتراک می‌گذارند. برای این‌که در برنامه نویسی صلاحیت لازم را داشته باشید، نیاز است به اندازه‌ی کافی در C++ مسلط شده باشید.

۷.۱.۱ نقطه‌ی شروع C++

داستان C++ با C آغاز می‌شود. بدین معنا که C++ بر مبنای زبان برنامه نویسی C ساخته شد. C++ نمونه‌ی اصلاح شده و بهبود یافته زبان شناخته شده‌ی C بوده که مبانی برنامه نویسی شئ‌گرا[۱] را به آن اضافه می‌کند. علاوه بر آن، C++ یک سری توابع و کتابخانه‌های جدید را به زبان C اضافه کرده است. با این‌حال جوهره‌ی اصلی C++ از همان C به ارث رسیده است. پس برای بررسی C++ بایستی به ریشه و باید و نبایدهای C بپردازیم.

---

[۱] Object Oriented

۷.۱.۲ ایجاد زبان C

زبان C ، دنیای کامپیوتر را تکان داد. به تأثیر گذاری این زبان نباید ساده نگریست. زیرا یکی از زبان-هایی که دنیای برنامه نویسی را متحول ساخته و نگرش به آن را دگرگون کرد، همین زبان C بود. قبل از به وجود آمدن زبان C، زبان‌های برنامه نویسی به عنوان تمرین‌های دانشگاهی و یا در اداره‌ها کاربر داشت. اما C متفاوت بود و توسط برنامه نویسان باتجربه، نوشته شده و توسعه داده شد. مشخصات آن توسط افرادی که از آن استفاده می‌کردند، مورد بازبینی قرار گرفت. نتیجه‌ی این فرآیند زبانی بود که برنامه نویسان دوستش داشتند.

به‌راستی، C خیلی از دنبال کنندگانش که اشتیاق وافری بدان داشتند را به خود جذب کرد و در جوامع برنامه نویسی مقبولیت عام و سریعی پیدا کرد. به طور خلاصه، C زبانی است که توسط برنامه نویسان و برای برنامه نویسان، طراحی و ایجاد شد.

C ، برای اولین بار توسط Dennis Ritchie بر روی یک رایانه‌ی DEC PDP-11 که از سیستم‌عامل UNIX بهره می‌برد، ابداع و پیاده سازی شد. این زبان ادامه‌ی فرآیند توسعه‌ای بود که با زبان قدیمی‌تر BCPL آغاز می‌شد. BCPL که از زبان برنامه نویسی B تأثیر می‌گرفت، توسط Martin Richards به بهره‌برداری رسیده بود. زبان B ، نهایتاً منجر به ساخته شدن زبان برنامه نویسی C در دهه‌ی 1970 شد.

برای سال‌های طولانی، استاندارد مورد استفاده برای برنامه نویسی C همان نمونه‌ی غیر رسمی بود که توسط سازندگانش با نام *The C Programming Language* شناخته می‌شد. به دلیل نبود یک استاندارد برای این زبان، پیاده سازی‌های مختلفی از آن در دسترس عموم قرار داشت. در سال 1983 مؤسسه‌ی استاندارد گذاری ANSI فرآیند استاندارد سازی این زبان را آغاز کرده و نسخه‌های نهایی آن را در دسامبر سال 1989 استاندارد گذاری کرد. این نسخه از C با نام C89 معروف است و پایه‌ی شکل گیری C++ بود.

از دیدگاه تئوری یک "زبان سطح بالا" تلاش می‌کند به برنامه نویس هر آن‌چه را که می‌خواهد (به صورت پیش‌ساخته در ساختار زبان برنامه نویسی قرار دارد)، بدهد. "زبان سطح پایین" فقط دسترسی به کدها و دستورالعمل‌های ماشین را میسّر می‌سازد. یک "زبان سطح وسط" مجموعه‌ای از ابزارهای مختصر و مفید را در دسترس برنامه نویس گذاشته و به او امکان می‌دهد که ساختارهای پیشرفته‌ی مورد نیازش را خود، پیاده سازی کند. زبان سطح وسط به برنامه نویس، قدرتی ذاتی به همراه انعطاف‌پذیری را هدیه می‌دهد.

زبان C را به عنوان یک زبان سطح وسط می‌شناسند. C ، عناصر موجود در زبان‌های سطح بالا مانند Pascal، Modula-2 یا Visual Basic را با کارکردهای زبان اسمبلی در دسترس قرار می‌دهد. شما می‌توانید در C ، به دستکاری بیت‌ها، بایت‌ها و یا آدرس‌ها بپردازید. C ، در مخفی نگه‌داشتن مشخصات

سیستم مورد استفاده، کوششی ندارد. برای مثال، اندازه متغیر صحیح[1] به اندازه‌ی کلمات[2] در CPU مورد استفاده، بستگی دارد. بسیاری از توابع دستکاری عناصر مختلف همانند فایل‌ها، در زبان‌های سطح بالا توسط دستورهای ساده‌ای اجرا می‌شوند. این در حالیست که در C با استفاده از توابع موجود در کتابخانه‌ها به این عملیات دسترسی داریم. این خود به معنی افزایش انعطافی است که زبان مذکور به برنامه نویسش می‌دهد.

زبان C ، یک زبان ساخت یافته است[3]. از مشخصات اصلی زبان‌های ساخت یافته، استفاده از بلوک-هاست. بلوک[4] ها مجموعه‌ای از عبارات هستند که منطقاً به یکدیگر متصل شده‌اند.

یک زبان ساخت یافته، از زیربرنامه‌ها[5] و متغیرهای محلی پشتیبانی می‌کند. یک متغیر محلی، متغیری است که فقط در همان زیربرنامه‌ای که در آن تعریف شده، شناخته شده است.

زبان ساخت یافته، از چندین ساختار حلقه نیز همانند while و do_while و for پشتیبانی می‌کند. در این نوع زبان‌ها، استفاده از دستور goto برای پریدن به مکانی خاص، همانند آن‌چه در BASIC و یا FORTRAN وجود دارد، منع شده است.

فلسفه زبان C این است که واقعاً به شما اجازه‌ی انجام هر کاری حتی کارهای غیرمعمول و غیرمتعارف را نیز می‌دهد. C ، اجازه کنترل کامل ماشین و یا دستگاه مورد استفاده را می‌دهد. با این قدرت عملکردی، این شمایید که مسئول همه چیز هستید و باید از برنامه‌های پرخطر و یا رفتارهای برنامه نویسی غیرمعمول، اجتناب ورزید.

۷.۱.۳ نیاز به C++

درحالی که C یک زبان موفق و مفید برنامه نویسی کامپیوتری بود، پس به‌راستی چه نیازی به زبانی دیگر بود؟ جواب این سؤال در توانایی رسیدگی به پیچیدگی‌های نرم‌افزاری بود. C ، در پیاده سازی ساختارهای پیچیده‌ی نرم‌افزاری، مشکلات زیادی را برای برنامه نویس ایجاد می‌کرد. در این‌جا بود که C++ مسئولیت فوق را بر عهده گرفت.

C ، یک زبان ساخت یافته بود. با یک چنین زبانی، پیاده سازی زبان‌هایی با پیچیدگی متوسط ممکن شده بود. با این وجود، هنگامی که ابعاد یک پروژه به طرزی باور نکردنی بزرگ می‌شد، توانایی مدیریت صحیح آن از دست برنامه نویس خارج می‌شد. در اواخر دهه 1970 ، اکثر پروژه‌ها با یک چنین مشکلی

---

[1] Integer

[2] Word

[3] Structured

[4] Block

[5] Subroutine

روبه‌رو شده بودند. برای حل این مشکل، روش جدیدی به نام **برنامه نویسی شئ‌گرا**[1] راه خود را به دنیای برنامه نویسی باز کرد. C ، از روش OOP پشتیبانی نمی‌کرد. کلید کار در مدیریت برنامه‌ها توسط OOP بود. در این‌جا بود که C++ با ارایه مبانی شئ‌گرا و مدیریت پیچیدگی‌های نرم‌افزارهای حجیم، مشکلات عمده را از راه داشت.

۷.۲ برنامه نویسی شئ‌گرا[2]

برنامه نویسی شئ‌گرا از تمامی ایده‌های برنامه نویسی ساخت یافته، به همراه یک سری مشخصات خوب دیگر به‌منظور حل مشکلات پیچیدگی برنامه‌ها پشتیبانی می‌کند. هنگامی که سعی در حل مشکلات خود توسط OOP دارید، یک مسأله را به اجزای تشکیل دهنده‌اش تقسیم می‌کنید. هر جزء به یک **شئ** کامل تبدیل می‌شود که حاوی دستورالعمل‌ها و داده‌های مربوط به آن جزء است. تمامی زبان‌های برنامه نویسی سه چیز مشترک دارند: کپسوله کردن[3]، پولی مورفیزم[4] و وراثت[5].
هم اکنون نگاهی جزئی به هر یک از این سه جزء می‌اندازیم.

۷.۲.۱ کپسوله کردن

همان‌طور که می‌دانید، همه‌ی برنامه‌ها از دو جزء کد و داده تشکیل می‌شوند. **کدها**، جزئی از برنامه‌ها هستند که عملیات گوناگون را انجام می‌دهند و **داده‌ها** اطلاعاتی هستند که از آن عملیات تأثیر می‌پذیرند. در فرآیند کپسوله کردن، داده‌ها و کدهای مربوط به یک شئ، در یک محیط جداگانه و در کنار هم نگه‌داری شده و دسترسی‌های ناخواسته به آن‌ها ناممکن می‌شود.
دسترسی به اشیاء از طریق دستورهایی که خود آن‌ها در اختیار قرار داده‌اند، میسّر می‌گردد و هر برنامه‌ی خارجی، دسترسی به اطلاعات جزئی داخل آن‌ها را ندارد. دستورهای در دسترس بقیه اجزای برنامه را عمومی[6] و دستورهایی که فقط در دسترس خود شئ هستند را خصوصی[7] می‌نامند.

---

[1] OOP
[2] Object Oriented Programming-OOP
[3] encapsulation
[4] Polymorphism
[5] Inheritence
[6] Public
[7] Private

۷.۲.۲ پولی مورفیزم

پولی مورفیزم، مفهومی است که آن را با عبارت "**یک واسط، چندین روش**" می‌شناسند. بدین معنا که در پولی مورفیزم، شما می‌توانید برای یک سری عملیات مرتبط، یک واسط کلی و عمومی طراحی کنید. انتخاب واسط مشترک برای انجام یک کار خاص با استفاده از کامپایلر صورت گرفته و یا در اختیار برنامه نویس است.

۷.۲.۳ وراثت

فرآیندی است که در آن یک شئ، خصوصیات یک شئ دیگر را به دست می‌آورد. اهمیت این موضوع در طراحی یک سلسله مراتب و توانایی دسته‌بندی اشیاء گوناگون است. به عنوان مثال، یک سیب، جزئی از یک کلاس بزرگ‌تر به نام میوه می‌باشد که آن نیز عضوی از کلاس بزرگ‌تر خوردنی‌ها است. سیب مذکور، دارای بعضی از خواص کلاس میوه همین‌طور خواص کلاس خوردنی‌ها است. بدون استفاده از مفهوم وراثت، برای تعریف یک شئ بایستی تمامی مشخصات آن را برشمرد. در حالی‌که در وراثت می‌دانیم بسیاری از این خواص در کلاس‌های مرتبه‌ی بالاتر، از قبل تعریف شده‌اند.

۷.۳ عملگرهای بیتی در زبان C

چهار عملگر پرکاربرد در زبان C عبارت‌اند از: ~ ، & ، | و ^. این عملگرها برای دستکاری بیت‌های منفرد (1 یا 0) استفاده می‌شوند. دو عملگر مهم دیگر نیز وجود دارند که جهت عملیات شیفت استفاده می‌شوند << و >>. از آن‌جایی‌که حضور این عملگرها را در بسیاری از کدهای نوشته شده به زبان C می‌بینید، خوب است فهمی عمیق از روش کارشان پیدا کنیم.

۷.۳.۱ ~ عملگر منفی‌ساز بیتی

از ساده‌ترین عملگرهای بیتی که برای منفی‌سازی[۱]، مقادیر بیتی کاربرد دارد. بدین معنا که اگر متغیری با محتوای `0xf0f0` (`1111 0000 1111 0000`) داشته باشیم، با اعمال این عملگر، نتیجه‌ی `0xf0f0` (`0000 1111 0000 1111`) را خواهیم داشت. نمونه کد زیر نحوه‌ی کار این عملگر را نشان می‌دهد.

a	~a
0	1
1	0

[۱] Negation

۷.۳.۲ عملگر AND &

این عملگر، تنها وقتی مقدار بیتی '1' را برمی‌گرداند که هر دو ورودی آن '1' باشند. نحوه‌ی کار آن را در جدول COP-2 می‌بینیم.

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

۷.۳.۳ عملگر OR |

این عملگر، هنگامی که هر یک از ورودی‌ها و یا تمام آن‌ها '1' باشند، مقدار '1' را برمی‌گرداند. نحوه‌ی کار آن را درجدول COP-3 می‌بینیم.

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

۷.۳.۴ عملگر XOR ^

اگر دو بیت ورودی این عملگر با یکدیگر متفاوت باشند، این عملگر خروجی '1' را خواهد داشت. نحوه‌ی کارش را در جدول COP-4 می‌بینیم.

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

۷.۳.۵ شیفت بیت‌ها

عملگر شیفت بیتی از عملگرهای پرکاربرد است. نحوه‌ی کار آن را در نمونه کد زیر می‌بینیم.

```
unsigned int i    = 0xF0F0;             // 1111 0000 1111 0000
unsigned int left = 0xF0F0 | (1 << 10); // 1111 0100 1111 0000
```

در مثال بالا، یک بیت '1' را به میزان ۱۰ مکان به سمت چپ شیفت داده و سپس با استفاده از عملگر OR، آن را در مکان جدیدش جای می‌دهیم.

نکته

شیفت یک "تک بیت" و جای دادن آن در مکانی خاص از عملیاتی است که در سیستم‌های میکروکنترلی بسیار کاربرد دارد. عملکرد آن روش و یا خاموش کردن یک وسیله‌ی جانبی و یا پیکره‌بندی مشخصات مختلف است.

عملگر شیفت در صورت نیاز بر روی مجموعه‌ای از مقادیر نیز می‌تواند عمل کند. به عنوان مثال:

```
unsigned int orig   = 0xF0F0;           // 1111 0000 1111 0000
unsigned int insert = 0x000A;           // 0000 0000 0000 1010

unsigned int a = orig | (insert << 8); // 1111 1010 1111 0000
unsigned int b = orig | (insert << 6); // 1111 0010 1111 0000
```

در زیر، نمونه‌هایی ترکیبی از سایر عملگرهای بالا با نتایج حاصله را می‌بینیم.

```
unsigned int orig   = 0xF0F0;           // 1111 0000 1111 0000
unsigned int insert = 0x000A;           // 0000 0000 0000 1010

unsigned int OR  = orig | (insert << 6); // 1111 0010 1111 0000
unsigned int AND = orig & (insert << 6); // 0000 0000 1000 0000
unsigned int XOR = orig ^ (insert << 6); // 1111 0010 0111 0000
```

عملگر شیفت به چپ علاوه بر کاربرد ذکر شده، در خواندن "تک بیت" ها نیز کاربرد دارد. به مثال زیر توجه کنید.

```
unsigned long testValue = 0xFF00FF00;

// Check if the fifth bit is equal to 1 (hint: it isn't!)
if (testValue & (1 << 4))
{
  // Fifth bit is equal to 1
```

```
}
else
{
  // Fifth bit is equal to 0
}
```

در مثال بالا، ابتدا عدد '1'، ۴ مکان به چپ شیفت داده شده (**0000 0000 0001 0000**) و مقدار حاصل با عدد **test value**، AND می‌شود. اگر بیت پنجم متغیر مذکور '1' باشد، نتیجه‌ی AND حاصل مقداری غیرصفر است (معادل **true** برای عبارت **if**).

۷.۴ محیط‌های توسعه نرم‌افزاری

۷.۴.۱ IAR IAR Embedded Workbench برای ARM

IAR یک ابزار توسعه کامل و جامع و در عین حال ساده برای راه اندازی سیستم های تعبیه شده حرفه-ای بر اساس ARM می باشد. این برنامه قابلیت کامپایل برنامه های C/C++ و Assembly را داشته و از یک عیب یاب/شبیه ساز حرفه ای به نام CSPY برای این مقصود بهره می برد. از تمامی پردازنده‌های معروف ARM (ARM7, ARM9, ARM10, Xscale, ARM-Cortex) پشتیبانی کرده و به راحتی شما را وارد دنیای پردازنده های حرفه ای ARM می کند.

IAR ARM Product	Editor, Project manager, C/EC++ Compilers, Assembler, Linker, Librarian, Run-time Libraries	Code Limit	CSPY Debugger/ Simulator	MISRA-C Checker	Support/ Update	Licences
AT91EWARM	■	-	■	■	1 year/1 year	Node/Dongle/Floating
AT91EWARM-LE	■	-	-	-	1 year/1 year	Node/Dongle/Floating
AT91EWARM-BL	■	256K	■	■	1 year/1 year	Node/Dongle

۷.۴.۱.۱ فرآیند ساخت فایل‌های باینری[۱]

در این بخش، مروری بر فرآیند ساخت خواهیم داشت. چگونگی ارتباط ابزارهای گوناگون ساخت[۲] (کامپایلر، اسمبلر و لینکر) با یکدیگر که منجر به تبدیل فایل منبع به یک فایل باینری و اجرایی می‌شود را خواهیم آموخت.

۷.۴.۱.۲ فرآیند ترجمه[۳]

دو ابزار در محیط IAR وجود دارند که فایل‌های منبع را به فایل‌های واسطه‌ی شئ[۴] تبدیل می‌کنند: کامپایلر C/C++ و اسمبلر جابه‌جا پذیر[۵]. هر دوی این ابزار، فایل‌های شئ جابه‌جا پذیر را به فرمت استاندارد ELF (که شامل فرمت DWARF برای دیباگ نیز می‌شود) تهیه می‌کنند.

توجه

می‌توان از کامپایلر برای تبدیل فایل‌های C/C++ به فایل‌های اسمبلی نیز استفاده کرد. در صورت نیاز می‌توانید فایل‌های اسمبلی حاصل را تغییر داده و سپس فایل‌های شئ حاصل را تولید کنید.

فرآیند ترجمه را در زیر می‌بینیم:

---

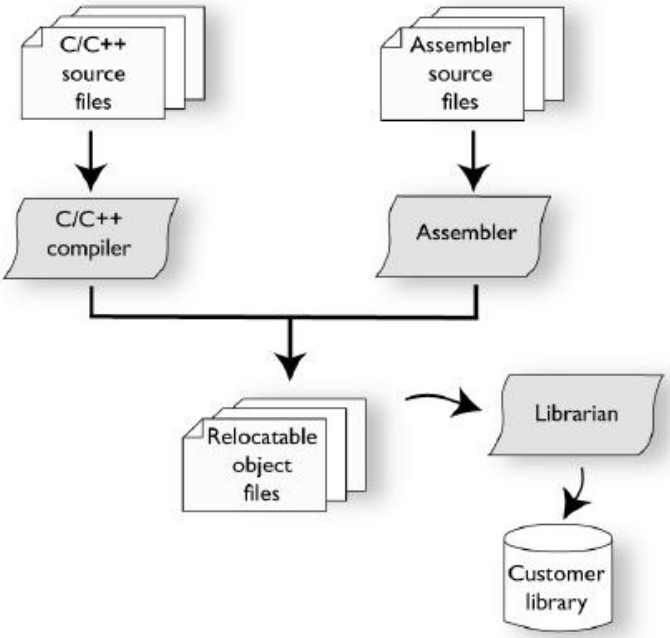
۱ Build

۲ Build Tools

۳ Translation

۴ Object

۵ Relocatable



شکل ۷.۱

بعد از عملیات ترجمه می‌توانید هر تعداد ماژول را به صورت یک کتابخانه بسته‌بندی کنید. اهمیت این کار در این است که شما باید از کتابخانه‌ها استفاده کنید. زیرا هر ماژول در کتابخانه به صورت مشروط لینک می‌شود. به عبارت دیگر، ماژول مورد نظر تنها اگر به صورت مستقیم یا غیر مستقیم توسط ماژول دیگری، در فایل شئ استفاده شده باشد، در فایل باینری حاصل قرار می‌گیرد.

۷.۴.۱.۳ فرآیند لینک[۱]

ماژول‌های جابه‌جا پذیر واقع در فایل‌های شئ و کتابخانه‌ها که توسط کامپایلر IAR و اسمبلر تولید شده‌اند، نمی‌توانند به خودی خود اجرا شوند. برای این‌که به یک فایل اجرایی تبدیل شوند، باید با یکدیگر **لینک** شوند.

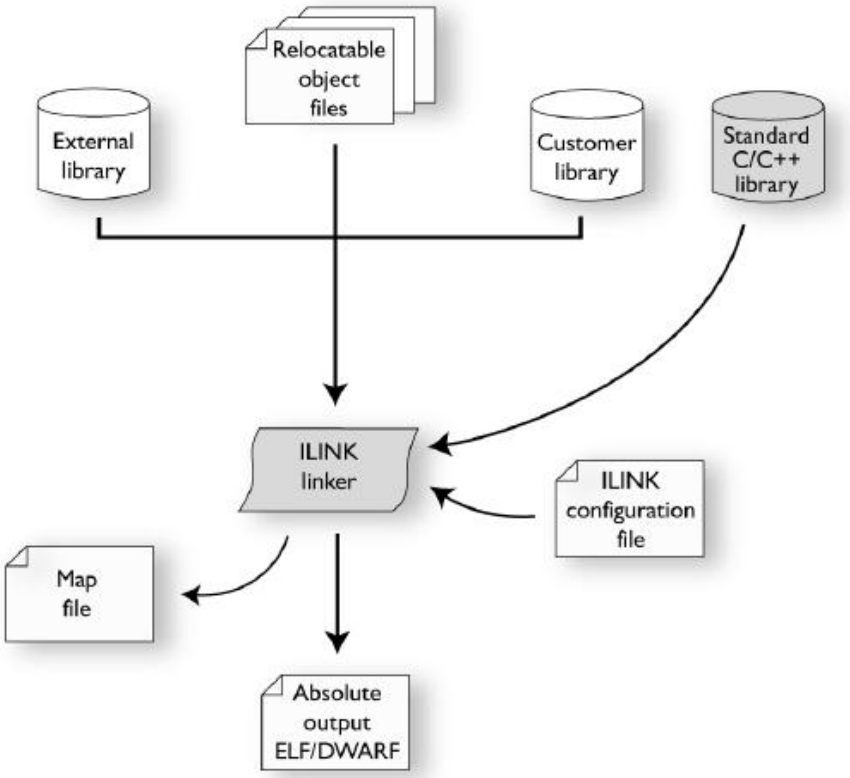
لینکر IAR LINK برای ساخت برنامه‌ی نهایی استفاده می‌شود. ILINK به اطلاعات ورودی زیر، نیاز دارد:

- تعدادی فایل‌های شئ و احتمالاً بعضی کتابخانه‌ها؛
- یک برچسب شروع برنامه؛
- فایل پیکره‌بندی لینکر که مکان قرارگیری کد و داده را در حافظه‌ی سیستم مورد نظر، شرح می‌دهد.

---

[۱] Link

در زیر شمایی کلی از فرآیند لینک را می‌بینیم:



شکل ۷.۲

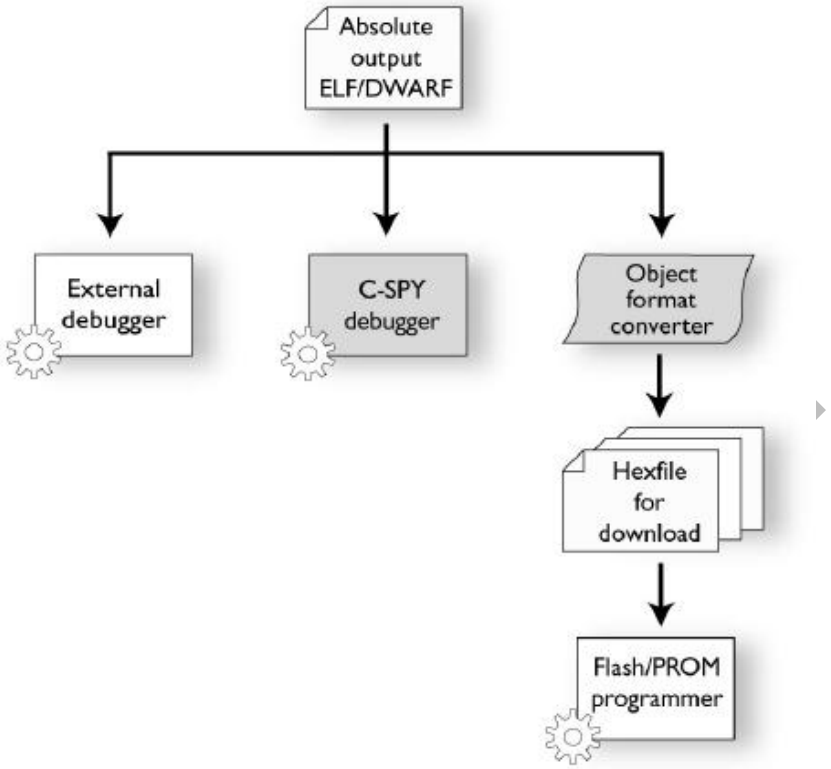
در طی فرآیند لینک، ممکن است ILINK پیغام‌های خطا و یـا ثبـت وقـایعی بـر روی Stdout و Stderr تولید کند.

۷.۴.۱.۴ بعد از لینک

لینکر ILINK یک فایل خروجی مطلق با فرمت ELF تولید می‌کند که حاوی فایل‌های اجرایی نیـز هسـت. فایل‌های اجرایی حاصل از فرآیند لینک را می‌توان برای مقاصد زیر استفاده کرد:

- بارگیری در درون دیباگر C-SPY یا هر دیباگر خارجی دیگر که توانایی خواندن فایـل‌هـای ELF و DWARF را دارد؛
- برنامه‌ریزی Flash/PROM ، ازطریق یک برنامه‌ریزی مناسب. قبل از انجام این عمـل، بایـد بایـت-های واقعی را به فرمت استاندارد s-record ۳۲ بیتی موتورولا یا Hex ۳۲ بیتی اینتل تبدیل کرد.

شکل زیر، روش استفاده از فایل‌های خروجی مطلق ELF/DWARF را نشان می‌دهد:



شکل ۷.۳

۷.۴.۱.۵ اجرای برنامه

این بخش، خلاصه‌ای از نحوه‌ی اجرای یک برنامه مخصوص سیستم، تعبیه شده که در سه مرحله توضیح می‌دهد:

- مرحله‌ی مقداردهی اولیه؛
- مرحله‌ی اجرایی؛
- مرحله‌ی پایانی.

مرحله‌ی مقداردهی اولیه

مقداردهی اولیه، هنگامی که یک برنامه شروع به کار می‌کند (هنگام ریست CPU) و قبل از ورود به تابع main، اجرا می‌شود.

برای سادگی، می‌توان مرحله‌ی مقداردهی اولیه را به تقسیمات زیر جداسازی کرد:

- مقداردهی اولیه‌ی سخت‌افزاری که برای مقداردهی اولیه‌ی اشاره‌گر پشته به کار می‌رود. مقداردهی اولیه‌ی سخت‌افزاری معمولاً در کد آغاز به کار سیستم یعنی startup.s انجام می‌شود.

در کنار تنظیمات فوق، این فایل تعدادی از کدهای مقداردهی اضافی سطح پایین را که شما به آن اضافه می‌کنید را نیز اجرا می‌کند.
به عنوان مثال، ریست و شروع به کار بقیه‌ی قسمت‌های سخت‌افزاری، تنظیم CPU و... از آن جمله‌اند. این اقدامات، نقطه‌ی شروع مرحله‌ی مقداردهی اولیه نرم‌افزاری C/C++ است؛

- مقداردهی اولیه سیستم C/C++ به صورت نرم‌افزاری

عموم وظایف این مرحله شامل حصول اطمینان از مقداردهی اولیه‌ی مناسب به نمادهای سراسری C/C++ قبل از شروع تابع main می‌باشد؛

- مقداردهی اولیه‌ی برنامه‌ی کاربردی

این مرحله کاملاً به برنامه‌ی شما بستگی دارد. معمولاً وظایفی چون تنظیم وقفه‌های مختلف، برپایی ارتباطات گوناگون و وسایل جانبی و... در این مرحله انجام می‌شوند.
در سیستمی که از حافظه‌ی فلش استفاده می‌کند، ثابت‌های نرم‌افزاری و توابع در این حافظه قرار می‌گیرند. تمام نمادهایی که در RAM قرار می‌گیرند نیز باید قبل از فراخوانی تابع main، مقداردهی شوند. لینکر قبل از این مراحل، RAM موجود را بین متغیرها، پشته، هیپ (Heap) و غیره تقسیم کرده است.

cstartup.s

```
1  /*
2       IAR startup file for AT91SAM7X microcontrollers.
3  */
4
5          MODULE  ?cstartup
6
7          ;; Forward declaration of sections.
8          SECTION IRQ_STACK:DATA:NOROOT(2)
9          SECTION CSTACK:DATA:NOROOT(3)
10
11 //------------------------------------------------------------------
12 ---------------
13 //         Headers
14 //------------------------------------------------------------------
15 ---------------
16
17 #define __ASSEMBLY__
18 #include "board.h"
19
20 //------------------------------------------------------------------
21 ---------------
22 //         Definitions
23 //------------------------------------------------------------------
24 ---------------
25
26 #define ARM_MODE_ABT     0x17
27 #define ARM_MODE_FIQ     0x11
28 #define ARM_MODE_IRQ     0x12
29 #define ARM_MODE_SVC     0x13
```

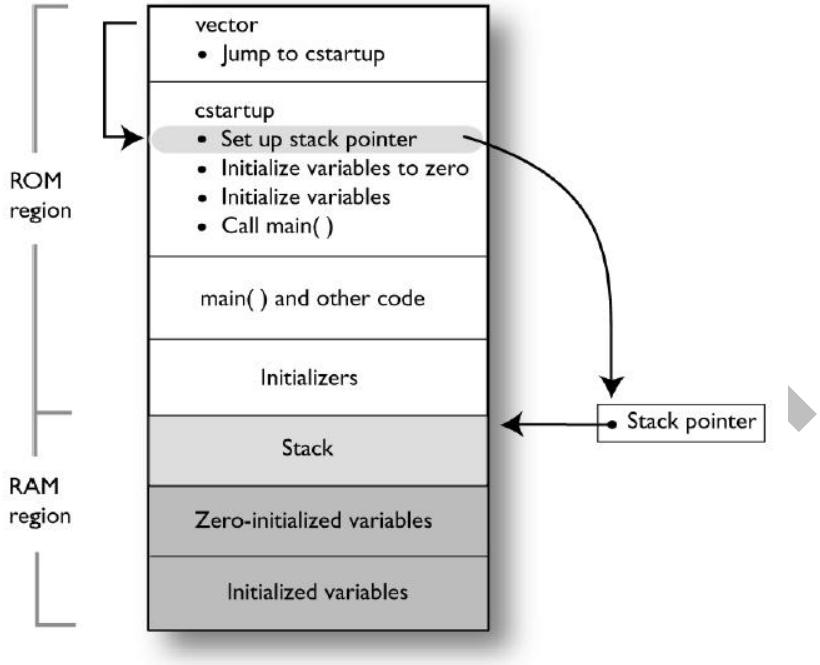
```
30 #define ARM_MODE_SYS     0x1F
31
32 #define I_BIT            0x80
33 #define F_BIT            0x40
34
35 //-------------------------------------------------------------------
36 ---------------
37 //         Startup routine
38 //-------------------------------------------------------------------
39 ---------------
40
41 /*
42    Exception vectors
43  */
44         SECTION .vectors:CODE:NOROOT(2)
45
46         PUBLIC  resetVector
47         PUBLIC  irqHandler
48
49         EXTERN  Undefined_Handler
50         EXTERN  SWI_Handler
51         EXTERN  Prefetch_Handler
52         EXTERN  Abort_Handler
53         EXTERN  FIQ_Handler
54
55         ARM
56
57 __iar_init$$done:      ; The interrupt vector is not needed
58                        ; until after copy initialization is done
59
60 resetVector:
61         ; All default exception handlers (except reset) are
62         ; defined as weak symbol definitions.
63         ; If a handler is defined by the application it will take
64 precedence.
65         LDR     pc, =resetHandler        ; Reset
66         LDR     pc, Undefined_Addr       ; Undefined instructions
67         LDR     pc, SWI_Addr      ; Software interrupt (SWI/SYS)
68         LDR     pc, Prefetch_Addr        ; Prefetch abort
69         LDR     pc, Abort_Addr           ; Data abort
70         B       .                        ; RESERVED
71         LDR     pc, =irqHandler          ; IRQ
72         LDR     pc, FIQ_Addr             ; FIQ
73
74 Undefined_Addr: DCD   Undefined_Handler
75 SWI_Addr:       DCD   SWI_Handler
76 Prefetch_Addr:  DCD   Prefetch_Handler
77 Abort_Addr:     DCD   Abort_Handler
78 FIQ_Addr:       DCD   FIQ_Handler
79
80 /*
81    Handles incoming interrupt requests by branching to the
82 corresponding
83    handler, as defined in the AIC. Supports interrupt nesting.
84  */
```

```
 85 irqHandler:
 86         /* Save interrupt context on the stack to allow nesting
 87 */
 88         SUB     lr, lr, #4
 89         STMFD   sp!, {lr}
 90         MRS     lr, SPSR
 91         STMFD   sp!, {r0, lr}
 92
 93         /* Write in the IVR to support Protect Mode */
 94         LDR     lr, =AT91C_BASE_AIC
 95         LDR     r0, [r14, #AIC_IVR]
 96         STR     lr, [r14, #AIC_IVR]
 97
 98         /* Branch to interrupt handler in Supervisor mode */
 99         MSR     CPSR_c, #ARM_MODE_SYS
100         STMFD   sp!, {r1-r3, r4, r12, lr}
101         MOV     lr, pc
102         BX      r0
103         LDMIA   sp!, {r1-r3, r4, r12, lr}
104         MSR     CPSR_c, #ARM_MODE_IRQ | I_BIT
105
106         /* Acknowledge interrupt */
107         LDR     lr, =AT91C_BASE_AIC
108         STR     lr, [r14, #AIC_EOICR]
109
110         /* Restore interrupt context and branch back to calling
111 code */
112         LDMIA   sp!, {r0, lr}
113         MSR     SPSR_cxsf, lr
114         LDMIA   sp!, {pc}^
115
116
117 /*
118    After a reset, execution starts here, the mode is ARM,
119 supervisor
120    with interrupts disabled.
121    Initializes the chip and branches to the main() function.
122  */
123         SECTION .cstartup:CODE:NOROOT(2)
124
125         PUBLIC  resetHandler
126         EXTERN  LowLevelInit
127         EXTERN  ?main
128         REQUIRE resetVector
129         ARM
130
131 resetHandler:
132
133         /* Set pc to actual code location (i.e. not in remap
134 zone) */
135         LDR     pc, =label
136
137         /* Perform low-level initialization of the chip using
138 LowLevelInit() */
139 label:
```

```
140         LDR     r0, =LowLevelInit
141         LDR     r4, =SFE(CSTACK)
142         MOV     sp, r4
143         MOV     lr, pc
144         BX      r0
145
146         /* Set up the interrupt stack pointer. */
147         MSR     cpsr_c, #ARM_MODE_IRQ | I_BIT | F_BIT   ; Change
    the mode
148         LDR     sp, =SFE(IRQ_STACK)
149
150         /* Set up the SYS stack pointer. */
151         MSR     cpsr_c, #ARM_MODE_SYS | F_BIT               ;
152 Change the mode
153         LDR     sp, =SFE(CSTACK)
154
155         /* Branch to main() */
156         LDR     r0, =?main
157         MOV     lr, pc
158         BX      r0
159
160         /* Loop indefinitely when program is finished */
161 loop4:
162         B       loop4
163
164         END
```

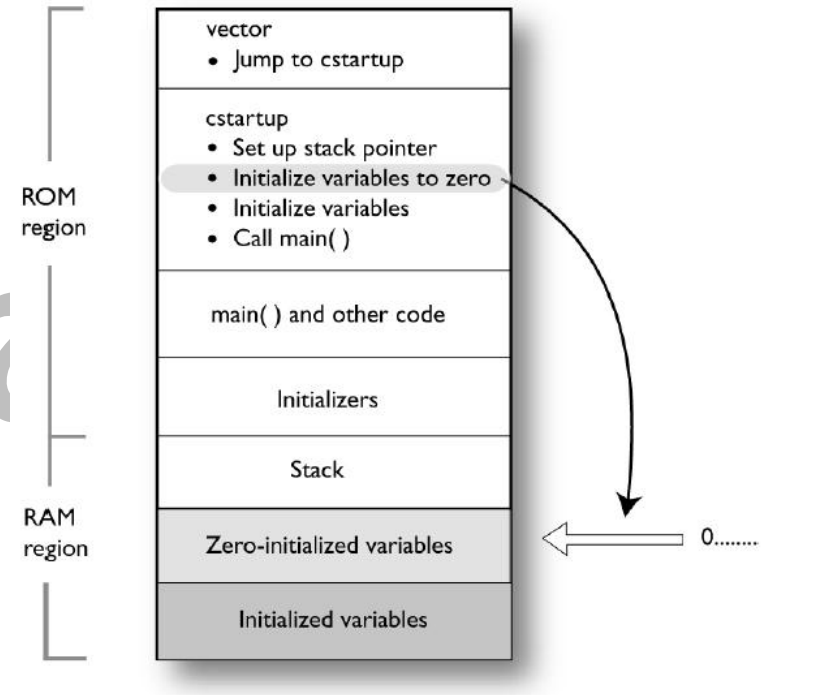
آن‌چه در زیر می‌بینیم، نمایشی تصویری از آن چیزی است که در مرحله‌ی مقداردهی اولیه می‌گذرد:

۱. هنگام شروع برنامه، کد آغاز به کار برنامه، مقداردهی اولیه‌ی سخت‌افزاری را انجام می‌دهد. به عنوان مثال، تنظیم مقدار اشاره‌گر پشته به یک آدرس خاص؛



شکل ۷.۴

۲. سپس، حافظه‌هایی که باید دارای مقدار اولیه‌ی صفر باشند، با صفر پر می‌شوند؛



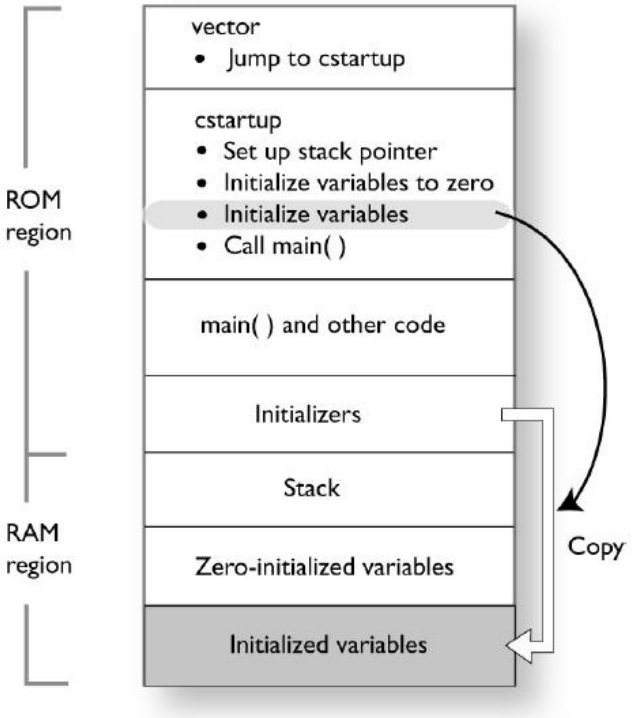
شکل ۷.۵

این داده‌ها را با نام zero initialized data می‌شناسند. به عنوان مثال کد زیر:

```
int i = 0;
```

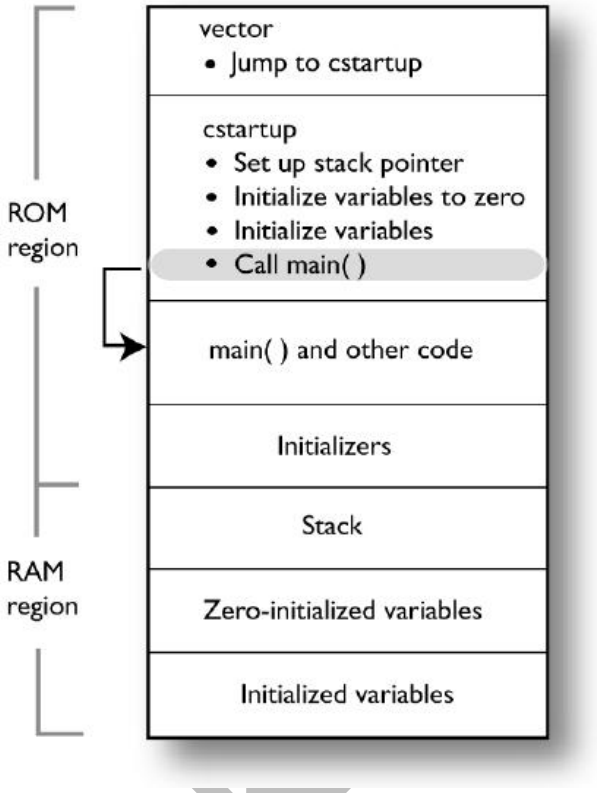
۳. برای داده‌های دارای مقدار اولیه، این مقدار اولیه از درون ROM و یا RAM کپی می‌شوند. به عنوان مثال:

```
int i = 6;
```



شکل ۷.۶

۴. نهایتاً تابع main فراخوانی می‌شود:



شکل ۷.۷

مرحله‌ی اجرایی

نرم‌افزار یک سیستم تعبیه شده عموماً به صورت یک حلقه پیاده سازی می‌شود. این حلقه، یا با وقفه کنترل می‌شود یا از سرکشی[۱]، برای ارتباط با خارج و یا وقایع داخلی استفاده می‌کند. برای سیستم که از وقفه استفاده می‌کند، عموماً وقفه‌ها در ابتدای تابع main مقداردهی اولیه می‌شوند.

مرحله‌ی پایانی

عموماً یک سیستم تعبیه شده نباید هرگز پایان یابد. اگر این طور نباشد، شما باید مکانیزم معینی برای این کار تعریف کرده باشید.

برای پایان دادن به تابع main به صورت مناسب، در انتهای آن از exit یا abort استفاده کرده و یا از return برای خروج از آن استفاده کنید. اگر از return استفاده کنید، exit نیز به صورت ضمنی اجرا خواهد شد.

---

[۱] Polling

البته، در صورتی که منطق برنامه صحیح نباشد، برنامه به روش ناصحیح و کنترل نشده‌ای پایان می‌یابد. این حالت را با نام خرابی[1] سیستم، می‌شناسند.

۷.۴.۲ Keil، برنامه توسعه نرم افزاری برای ARM7 ، ARM9 و ARM-Cortex

WinARM

این حالت را

RVDS

این حالت را

CodeWarrior

این حالت را

Rowley Crosswork

[1] Crash

۷.۴.۳ برپایی یک محیط توسعه براساس GCC

یکی از بزرگ‌ترین دغدغه‌های خیلی از کسانی که از ابزارهای GCC برای توسعه‌ی نرم‌افزاری ARM استفاده می‌کنند، نصب آسان و پیکره‌بندی صحیح ابزارهاست. این مسأله خصوصاً در محیط‌های غیرلینوکسی نمود بیشتری پیدا می‌کند. در این بخش، فرآیند برپایی یک چنین ابزارهای متن- بازی[۱] که ازGCC استفاده می‌کنند، مشتمل بر کامپایلر بین پلتفرمی[۲] و مجموعه‌ی ابزارها (Yagarto) ویک محیط توسعه‌ی گرافیکی (Code Lite) توضیح داده می‌شوند.

توجه

ابزارهای متن باز و رایگان مشابهی برای انجام عملیاتی مشابه موجود هستند. در این بخش، اصل بر سادگی و فهم آسان روش نصب و پیکره‌بندی این ابزارها بوده است.
ابزارهای ارایه شده در DVD همراه کتاب موجود هستند.

۷.۴.۳.۱ نصب Yagarto 4.5.0 برای ARM

Yagarto ، مجموعه‌ای از ابزارهای از پیش کامپایل شده برای ARM بوده که از هسته‌های رایج ARM همانند ARM7، ARM9 و Cortex M0/M3 پشتیبانی می‌کند. آخرین نسخه در دسترس این نرم‌افزار در هنگام نوشتن این متن نسخه‌ی 20100703 بوده که براساس GCC45.0 بنا شده است.
در هنگام نصب این برنامه، تنظیمات متغیرهای "Path" در ویندوز به نحو مناسبی صورت می‌پذیرد. بعد از نصب برنامه، فراخوانی ابزارهایی همانند "arm-none-eabi-gcc" به راحتی از هر خط فرمانی قابل اجراست. برای نصب صحیح برنامه، از ایجاد فضای خالی (Space) در مسیر نصب Yagarto جداً خودداری کنید.
بعد از انجام عملیات نصب، برای اطمینان از نصب صحیح برنامه، یک محیط خط فرمان را باز کرده و دستور زیر را در آن تایپ کنید. در صورتی که برنامه صحیح نصب شده باشد، اطلاعات مختلفی در مورد GCC می‌دهد.

```
arm-none-eabi-gcc  --version
```

[۱] Open Source
[۲] Cross-Compiler

۷.۴.۳.۲ نصب IDE متن باز Code Lite

در حالی‌که ساختن فایل‌های باینری از محیط خط فرمان Yagarto به‌راحتی امکان‌پذیر است، اما استفاده از IDE های قدرت‌مندی چون Code Lite ، بسیار دلپذیرتر و راحت‌تر است. Code Lite ، دو پیغام خطای تولیدی در برنامه را توسط پررنگ کردن خط مربوط، آشکار می‌کند. در این محیط، نیازی به تایپ دستورهای گوناگون و سردرگم کننده برای کامپایل فایل‌هایتان نخواهید داشت.
بعد از نصب صحیح Yagarto و ریست کردن کامپیوتر و نصب Code Lite ، می‌توانید وارد محیط این IDE شده و به ایجاد و ساخت پروژه‌های جدید بپردازید.
هم اکنون نمونه‌ای از یک پروژه‌ی باز شده در این محیط را می‌بینیم. برای کار در این محیط، نیاز به ایجاد یک فضای کاری[۱] جدید داریم. از منوی Work Space ، می‌توان فضایی جدید را ایجاد کرد. بعد از ایجاد فضای کاری جدید و اضافه کردن فایل‌های مختلف به پروژه‌ی مورد نظر با کلیک‌راست در پنجره‌ی Work Space و انتخاب 'Build' و 'Clean' به راحتی فایل‌های باینری مورد نظر ساخته می‌شوند.

توجه

در صورتی که هرگونه پیغامی در رابطه با ابزار make.exe و ایجاد خطا در آن مشاهده شد، از طریق منوی Settings> Build Settings و در بخش Build Systems قسمت Build Tool را به make.exe تغییر دهید.

Code Lite ، تعدادی مشخصات سودمند را نیز شامل می‌شود. امکان تکمیل خودکار کد، ویرایشگر متن عالی، گزارش خطای ساده در فرآیند ساخت و... از امکانات سودمند این IDE به شمار می‌روند. البته، این IDE و محیط عیب‌یابی آن مانند نرم‌افزارهای تجاری همانند IAR، Keil و... حرفه‌ای و کامل نیست. اما به عنوان یک ابزار متن باز، قابلیت‌های خوبی حتی در مواجهه با ابزاری چون Eclipse ارایه می‌کند.

[۱] Work Space

۷.۴.۴ سناریوی کامل برنامه نویسی و عیب یابی بر اساس ابزارهای GNU

در این با استفاده از ابزارهای رایگان GCC، eclipse، Yagarto و GDB Server به پیاده سازی یک سناریوی کامل از فرآیندهای برنامه نویسی، کامپایل و عیب یابی می پردازیم.

برای راه‌اندازی یک محیط توسعه‌ی نرم‌افزاری C/C++ براساس YAGARTO ، به اجزای زیر نیاز خواهید داشت:

۱) GDB Server؛

۲) مجموعه ابزار GNU ARM برای ویندوز؛

۳) محیط توسعه‌ی نرم‌افزاری مجتمع (یا IDE).

ابتدا، در مورد نصب J-Link GDB Server صحبت خواهیم کرد و در قسمت‌های بعد، درباره‌ی مجموعه‌ی ابزارهای GNU ARM و Eclipse توضیحاتی خواهیم داد.

J-Link GDB Server یک سرور راه دور برای GDB[۱] است. وظیفه‌ی سرور GDB ، تبدیل دستورهای GDB به دستورهای عیب‌یاب J-Link است. خوشبختانه، J-Link راه‌حلی کامل برای عیب‌یابی، برنامه-ریزی و ایجاد نقاط شکست متنوع بر روی تراشه را در اختیار می‌گذارد.

۷.۴.۴.۱ نصب برنامه

برای نصب برنامه نیاز به دو بسته‌ی زیر دارید:

۱) مجموعه نرم‌افزارهایی که به همراه J-Link ارایه می‌شوند؛

۲) ابزارهای YAGARTO (همانند make، sh، rm، cp و mkdir).

در مرحله‌ی اول، مجموعه‌ی نرم‌افزارهای J-Link موجود در CD را از حالت فشرده خارج نموده و نصب نمایید. در مرحله‌ی دوم، اگر به ابزارهایی همانند make، cp و... دسترسی ندارید، از فولدر مربوط به YAGARTO، فایل YAGARTO Tools را نصب کنید. نصب این برنامه، ساده بوده و با پذیرفتن تنظیمات پیش‌فرض، به سادگی به انتها می‌رسد.

بعد از نصب برنامه، به یک میز فرمان[۲] رفته و دستورهای زیر را تایپ کنید:

```
C:\make  -- version
```

اگر این دستور خروجی خاصی تولید نکرد، نصب "make-utils" درست انجام نشده و یا متغیرهای PATH ویندوز به درستی تنظیم نگردیده‌اند.

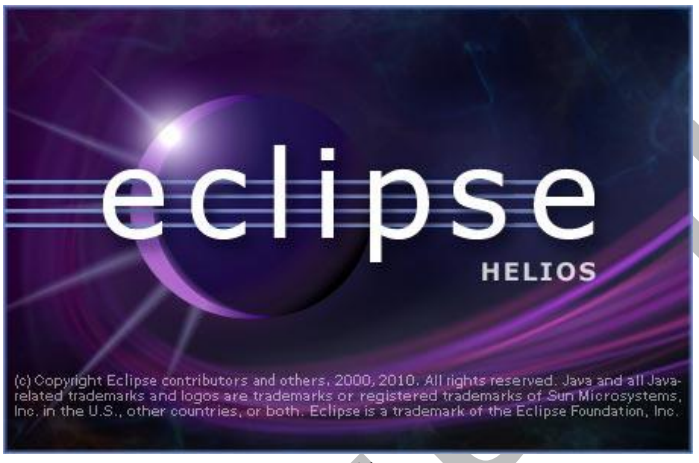
---

[۱] GNU Project Debugger

[۲] Command Prompt

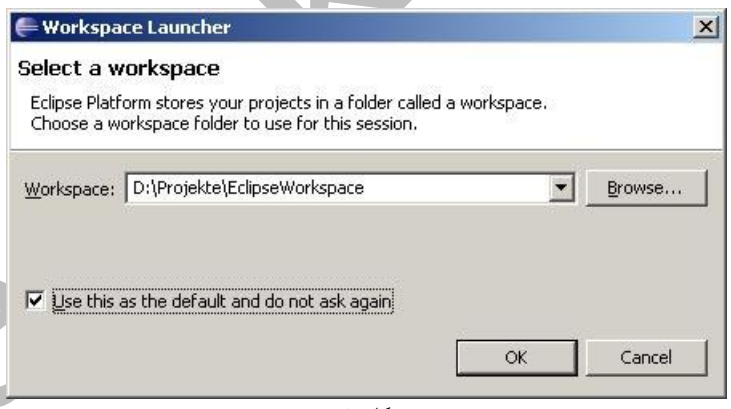
بر روی میان‌بر[1] ایجاد شده در مرحله‌ی قبل، کلیک کنید.

پس از شروع Eclipse ، پنجره‌ی زیر را خواهید دید.



شکل ۷.۸

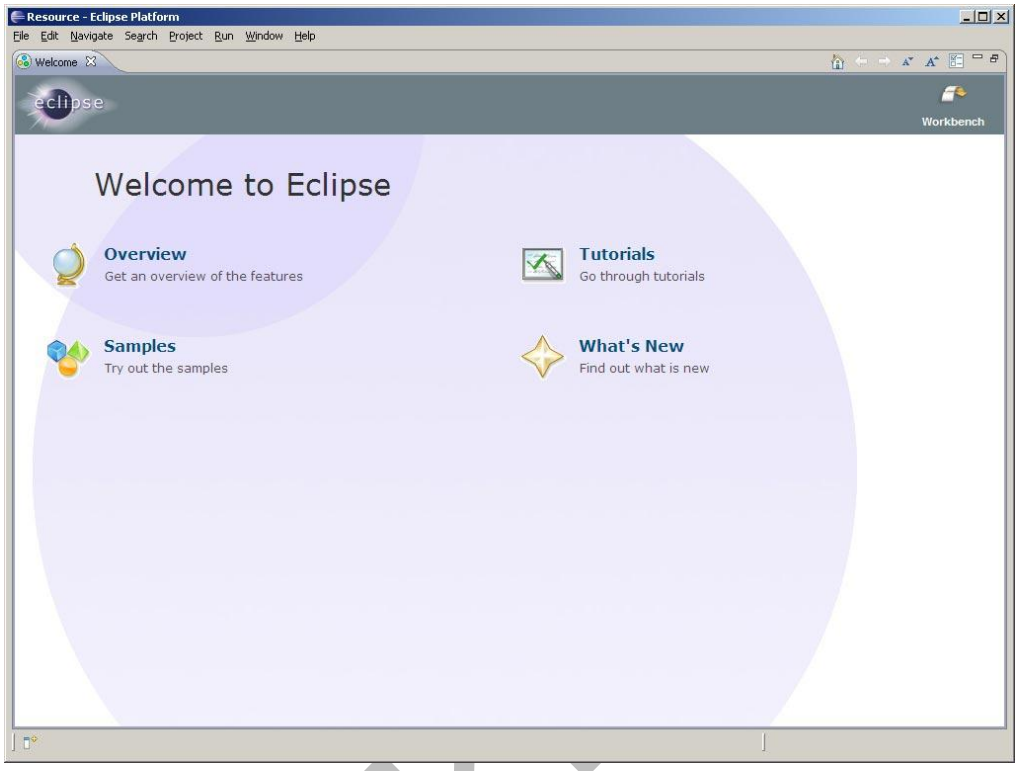
اگر در نصب JRE مشکلی داشته باشید، Eclipse درباره‌ی محل ذخیره سازی پروژه‌هایتان اطلاعاتی را سؤال می‌کند.



شکل ۷.۹

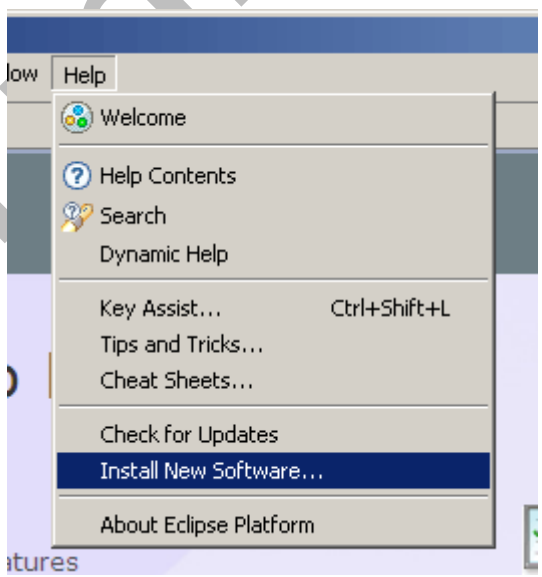
در این‌جا، آدرسی وارد کرده و "Ok" را می‌زنیم. سپس، پنجره‌ی "Welcome" را خواهید دید.

[1] Short cut



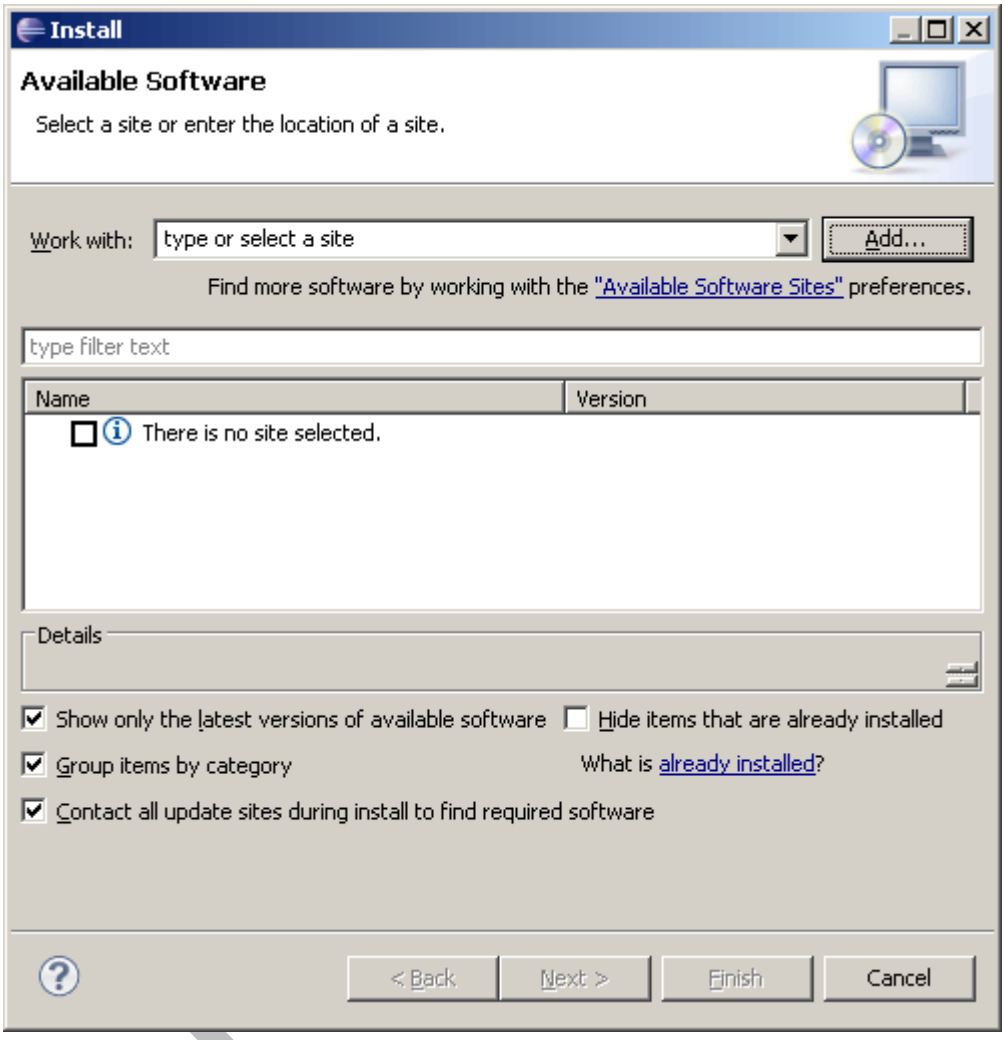
شکل ۷.۱۰

هم‌اکنون هنگام نصب ابزار توسعه‌ی C/C++ (CDT) است. از منوی Help>Install New Software...



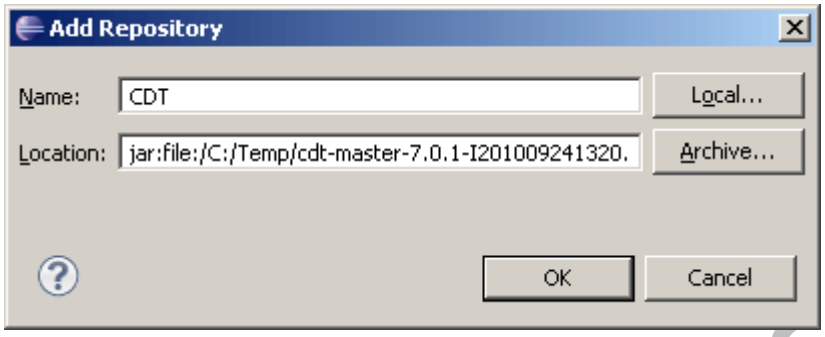
شکل ۷.۱۱

پنجره‌ی بعدی ظاهر می‌شود.



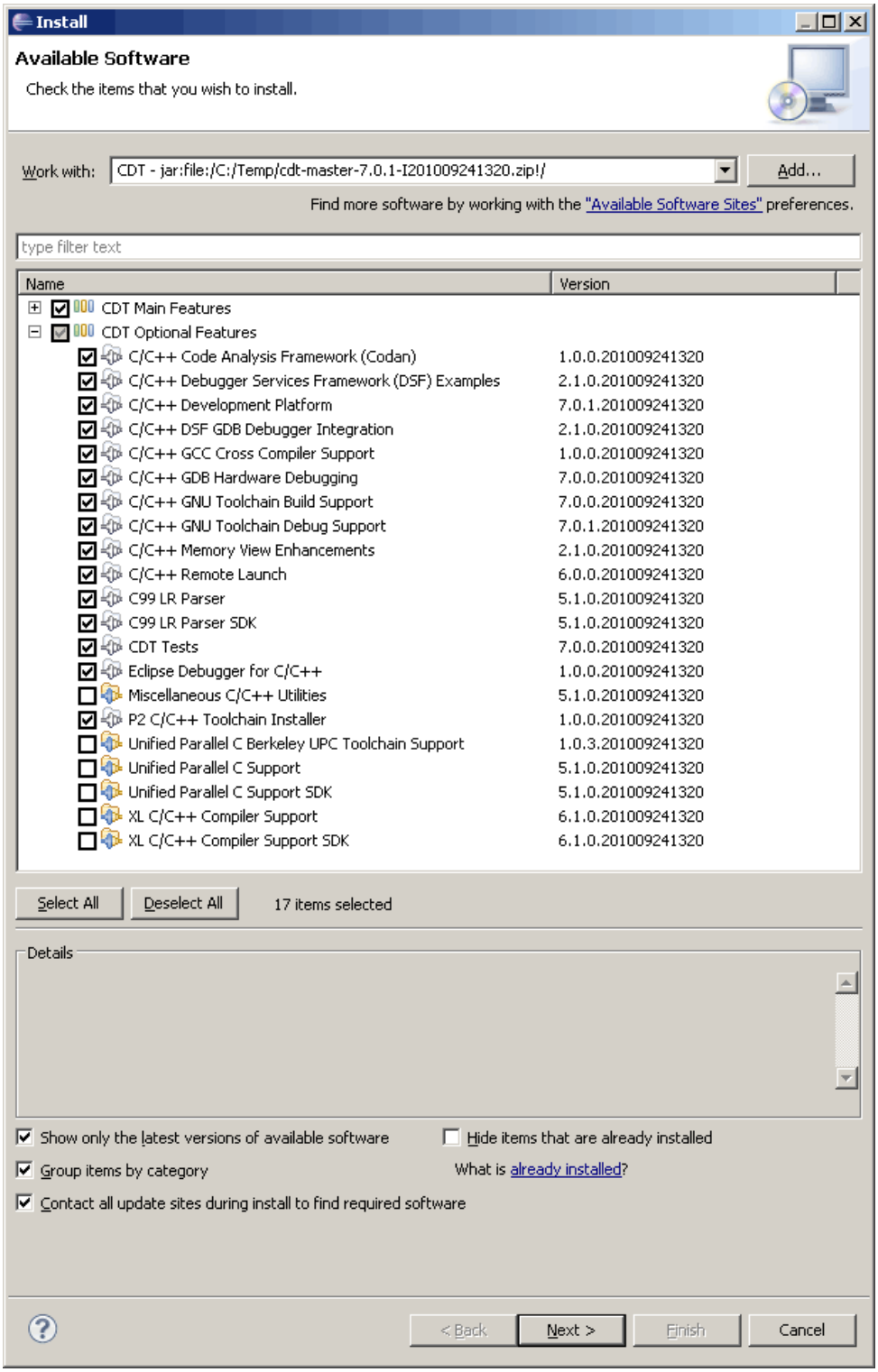
شکل ۷.۱۲

کلید "Add..." را بزنید و اطلاعات مورد نیاز را تکمیل کنید. در قسمت نام "CDT" و با استفاده از کلید "Archive..." به دنبال فایل فشرده شده (موجود در CD) بگردید. بعد از مشخص شدن آدرس فایل "Ok" را بزنید.



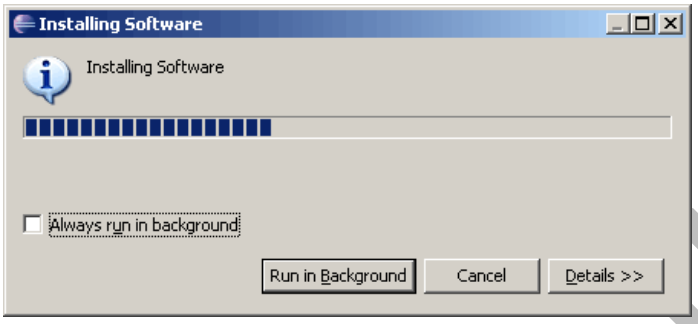
شکل ۷.۱۳

اکنون باید افزونه‌هایی که می‌خواهید نصب کنید را برگزینید. "CDT Main Features" را کلیک کرده و از قسمت "CDT Optional Features" افزونه‌هایی را که نمی‌خواهید، حذف کنید. نمونه‌ای از آن در زیر آمده است.


Install
Available Software
Check the items that you wish to install.
Work with:
CDT - jar:file:/C:/Temp/cdt-master-7.0.1-I201009241320.zip!/
Add...
Find more software by working with the "Available Software Sites" preferences.
type filter text
Name
Version
CDT Main Features
CDT Optional Features
C/C++ Code Analysis Framework (Codan) 1.0.0.201009241320
C/C++ Debugger Services Framework (DSF) Examples 2.1.0.201009241320
C/C++ Development Platform 7.0.1.201009241320
C/C++ DSF GDB Debugger Integration 2.1.0.201009241320
C/C++ GCC Cross Compiler Support 1.0.0.201009241320
C/C++ GDB Hardware Debugging 7.0.0.201009241320
C/C++ GNU Toolchain Build Support 7.0.0.201009241320
C/C++ GNU Toolchain Debug Support 7.0.1.201009241320
C/C++ Memory View Enhancements 2.1.0.201009241320
C/C++ Remote Launch 6.0.0.201009241320
C99 LR Parser 5.1.0.201009241320
C99 LR Parser SDK 5.1.0.201009241320
CDT Tests 7.0.0.201009241320
Eclipse Debugger for C/C++ 1.0.0.201009241320
Miscellaneous C/C++ Utilities 5.1.0.201009241320
P2 C/C++ Toolchain Installer 1.0.0.201009241320
Unified Parallel C Berkeley UPC Toolchain Support 1.0.3.201009241320
Unified Parallel C Support 5.1.0.201009241320
Unified Parallel C Support SDK 5.1.0.201009241320
XL C/C++ Compiler Support 6.1.0.201009241320
XL C/C++ Compiler Support SDK 6.1.0.201009241320
Select All
Deselect All
17 items selected
Details
Show only the latest versions of available software
Hide items that are already installed
Group items by category
What is already installed?
Contact all update sites during install to find required software
< Back
Next >
Finish
Cancel

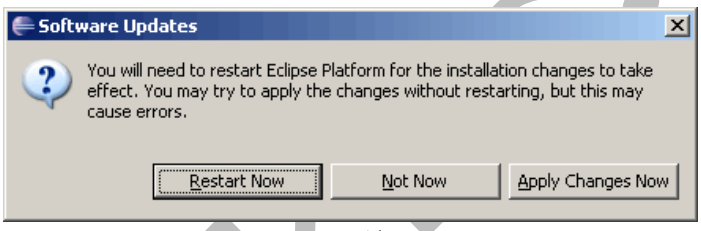
شکل ۷.۱۴

کلید "Next" را بزنید. مجدداً این کلید را بزنید. شما باید با مجوز نرم‌افزار موافقت کنید. این گزینه را انتخاب کرده و "Finish" را بزنید. در این‌جا اندکی زمان لازم است که نرم‌افزار جدید نصب شود.



شکل ۷.۱۵

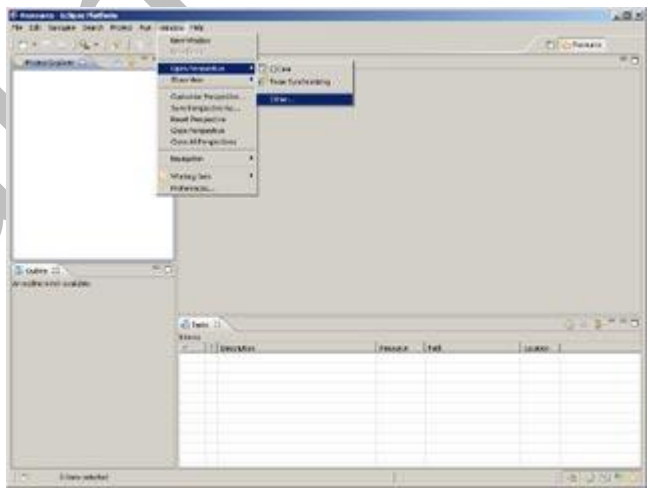
بعد از به‌روزرسانی Eclipse شما باید یک مرتبه آن را Restart کنید. کلید "Restart Now" را بزنید.



شکل ۷.۱۶

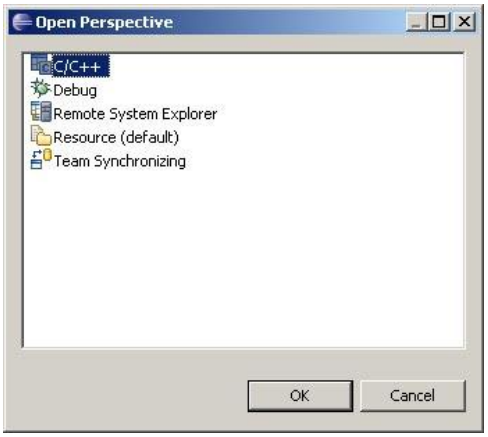
پس از شروع به کار مجدد Eclipse، پنجره‌ی خوش‌آمد گویی Eclipse را خواهید دید.

آیکون "Work bench" را در نوارابزار بالایی کلیک کنید. به مسیر زیر بروید " Window> Open Perspective> Other...". هم‌اکنون نمایی همانند زیر خواهید داشت.



شکل ۷.۱۷

با باز شدن پنجره‌ی زیر، "C/C++" را انتخاب کرده و "Ok" را بزنید.

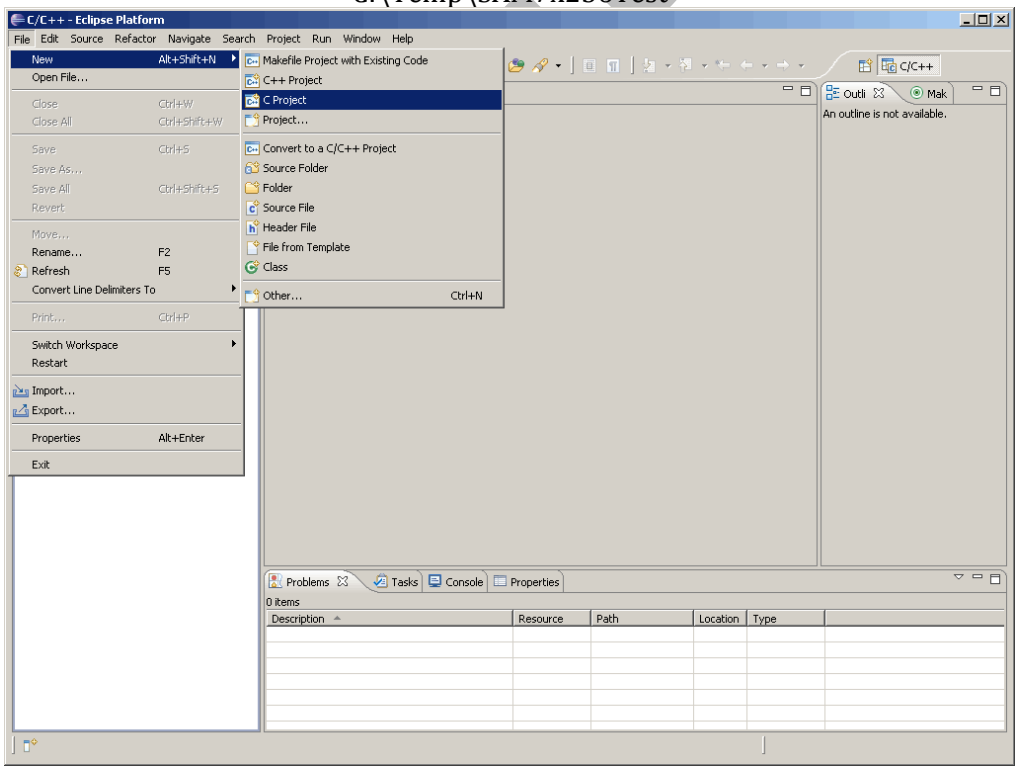


شکل ۷.۱۸

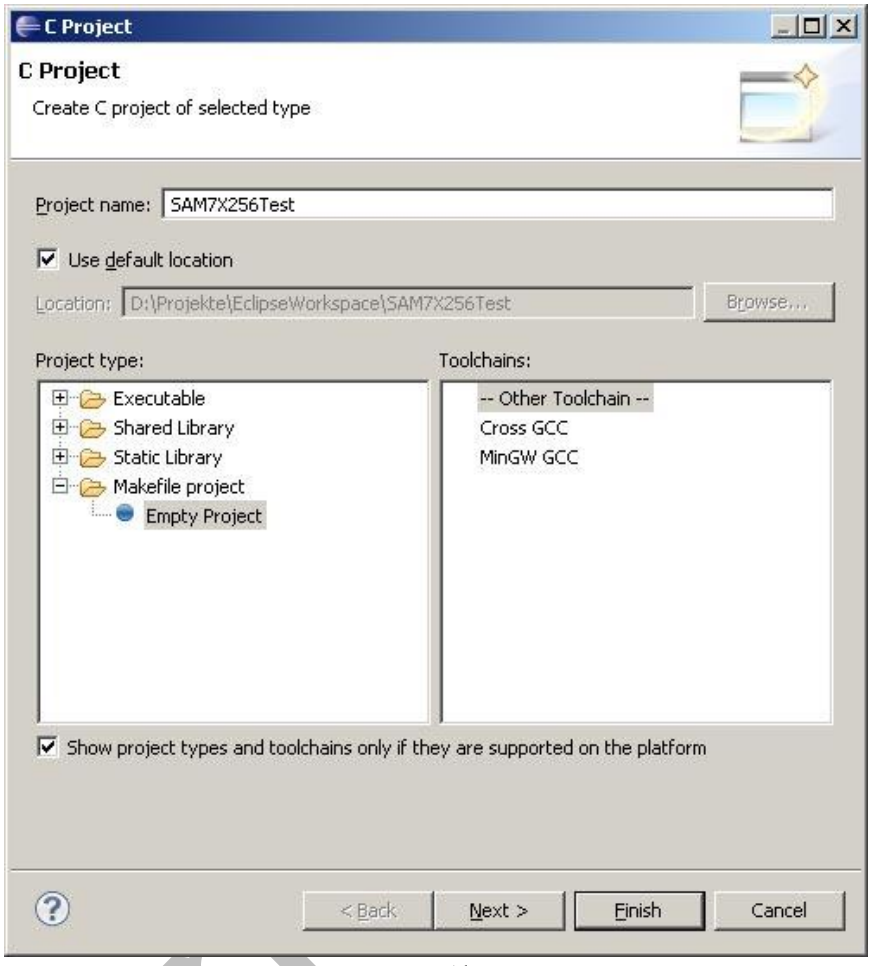
در این‌جا، با مثال کوچک کار را شروع می‌کنیم. مثال SAM7x256Test را در محلی از حالت فشرده خارج کرده و از طریق منوی "File> New> C Project" پروژه‌ای جدید را ایجاد کنید.

در این‌جا، فایل‌های مورد نظر در مسیر زیر قرار دارند.

C:\Temp\SAM7x256Test



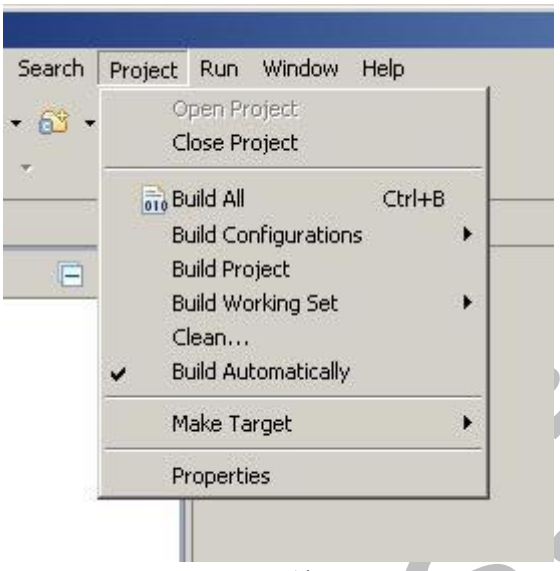
شکل ۷.۱۹



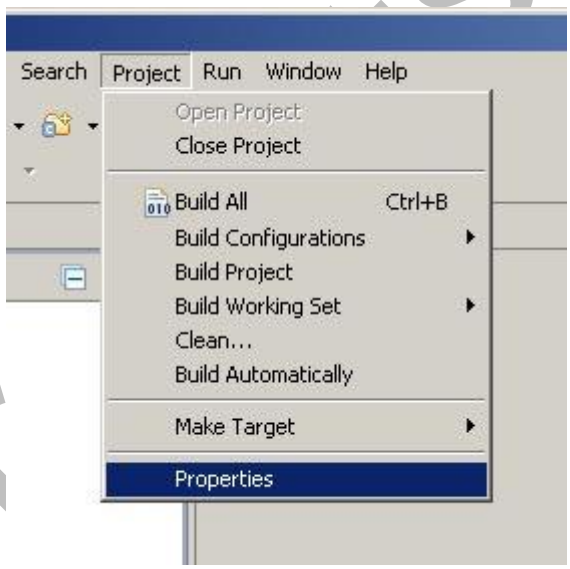
شکل ۷.۲۰

در پنجره‌ی باز شده نامی برای پروژه انتخاب کرده و از فهرست پایین قسمت‌های " Make File Project" و "Other Toolchain--" را انتخاب کنید. هم‌اکنون "Finish" بزنید.

در این‌جا، اولین پروژه Eclipse خود را ایجاد کرده‌اید. به منوی "Project" رفته و گزینه‌ی " Build Automatically" را غیرفعال کنید. مجدداً از طریق همان منو به قسمت "Properties" بروید.

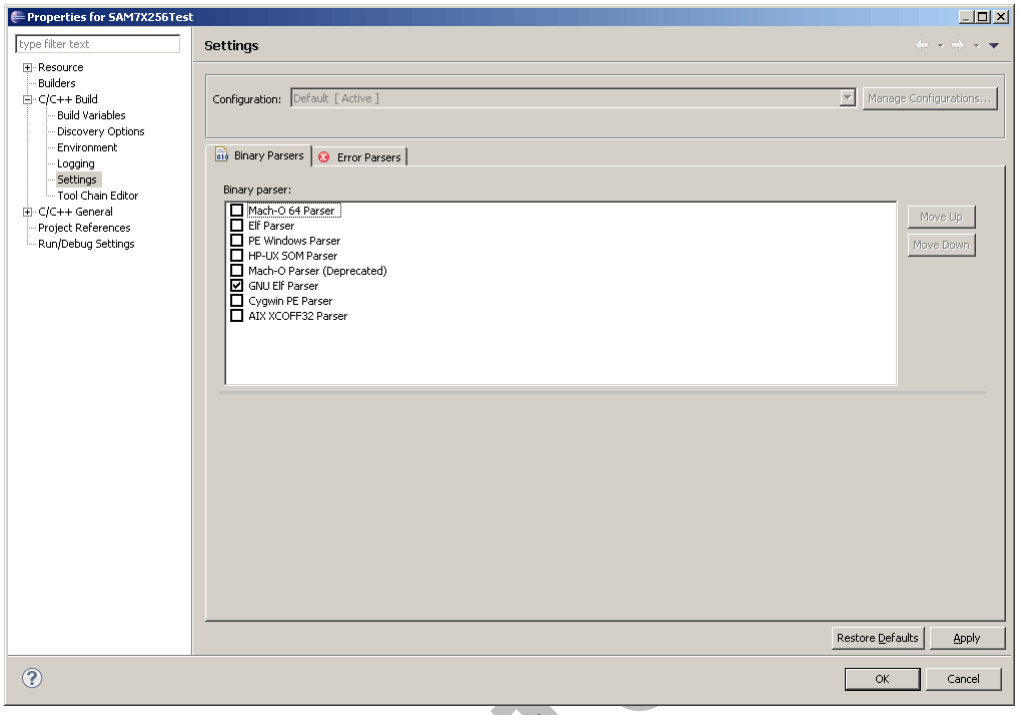


شکل ۷.۲۱



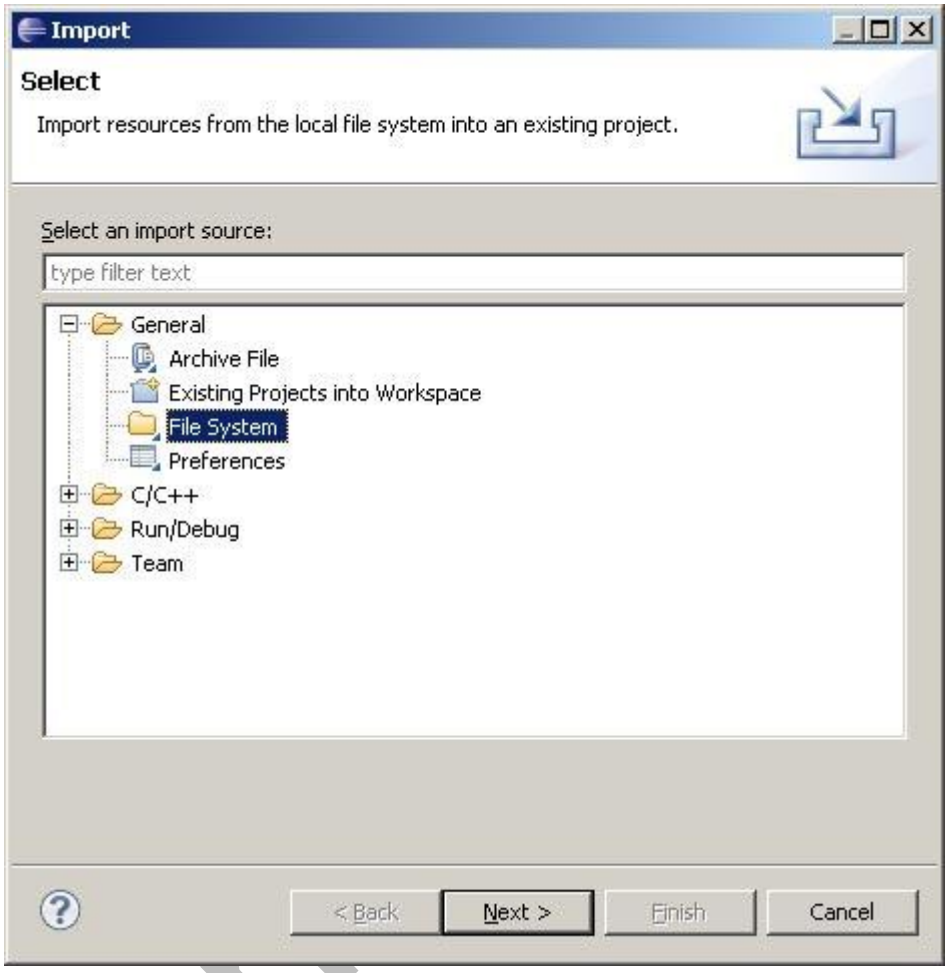
شکل ۷.۲۲

از پنجره‌ی باز شده به قسمت "C/C++" رفته و "GNU Elf Parser" را انتخاب کنید.



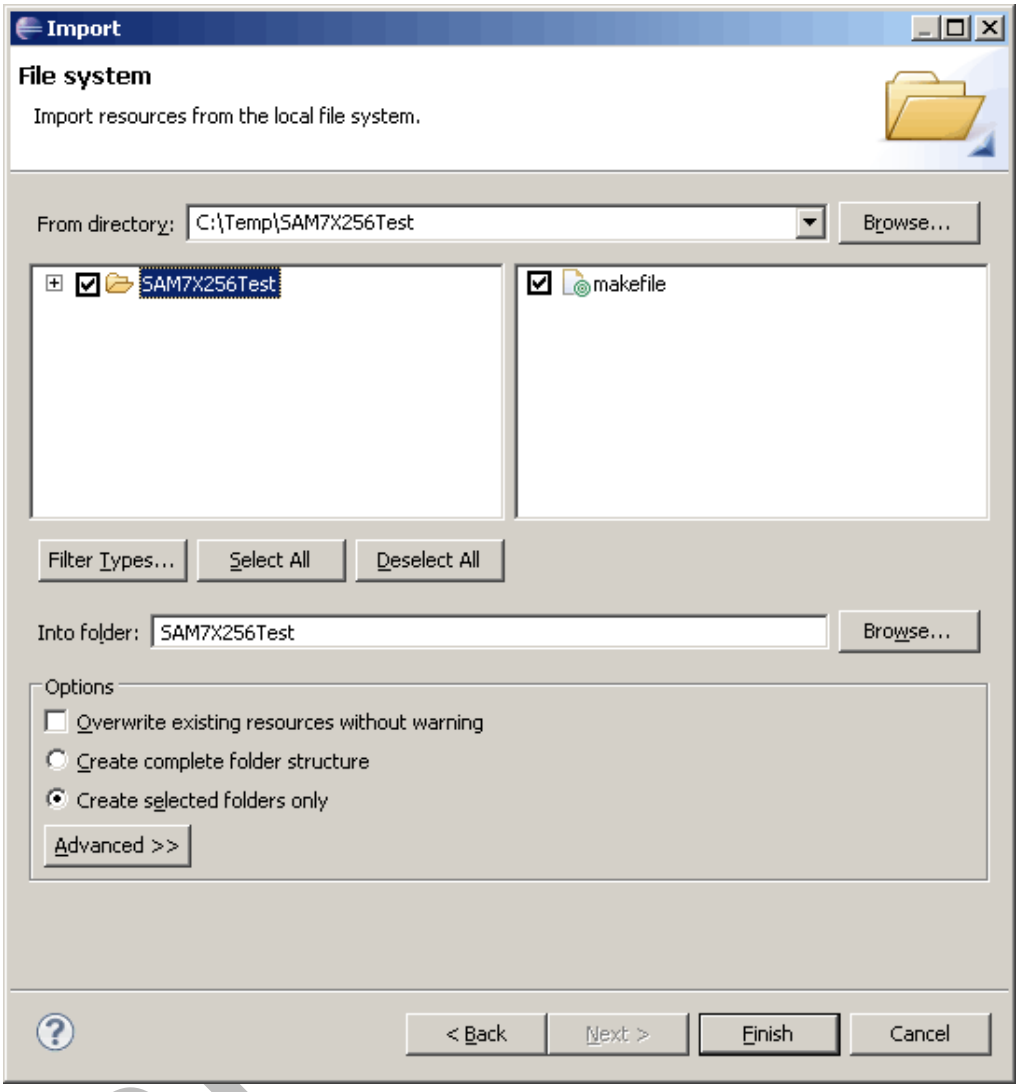
شکل ۷.۲۳

کلید "Ok" را بزنید. در این‌جا باید فایل‌های خود را به پروژه‌تان اضـافه کنیـد. بـه "File> Import..." رفته و از پنجره‌ی باز شده "File System" را انتخاب کنید. "Next" را بزنید.



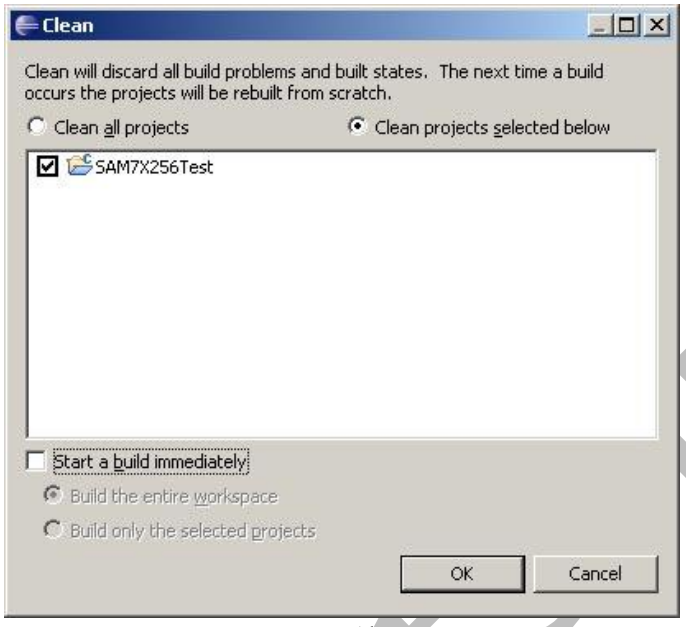
شکل ۷.۲۴

پنجره‌ی "Import" باز می‌شود. در قسمت "From Directory" آدرس محل فایل‌هایتان را انتخاب کنید. (در این‌جا از C:\temp\SAM7x256Test استفاده کرده‌ایم) تمام فایل‌ها را انتخاب کنید (با تیک‌زدن کلید مربوط) و "Finish" را بزنید.



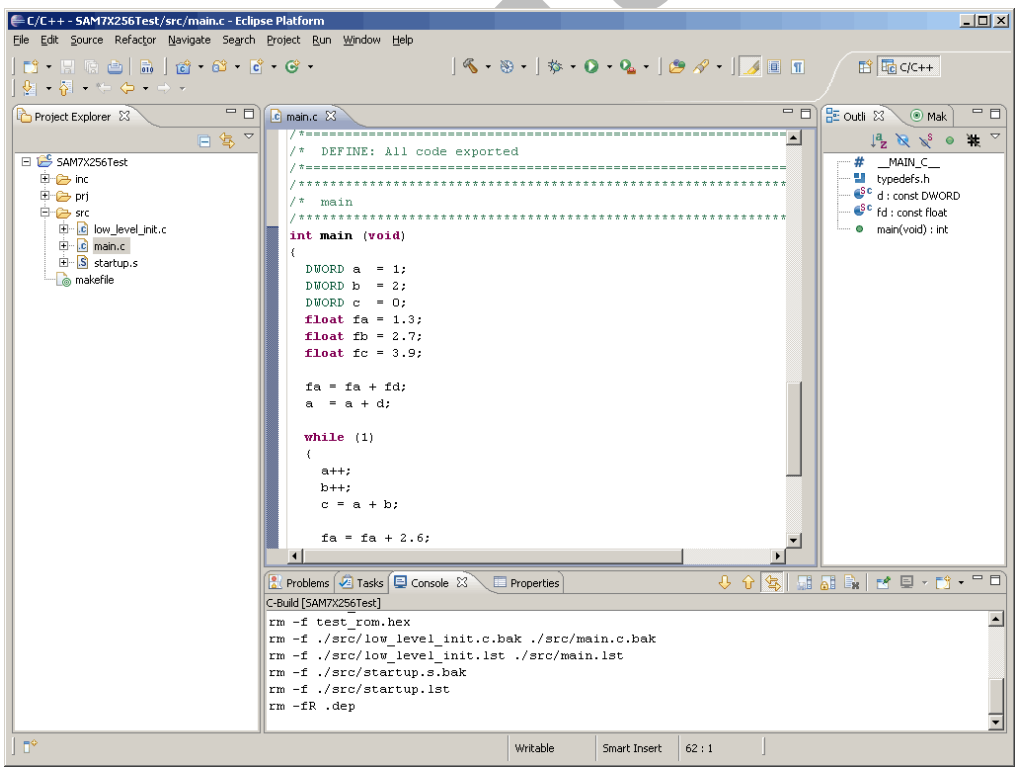
شکل ۷.۲۵

به منوی "Project> Clean..." رفته تا پروژه پاک‌سازی شود. تنظیمات را همانند پنجره زیر انتخاب کنید و "Ok" را بزنید.



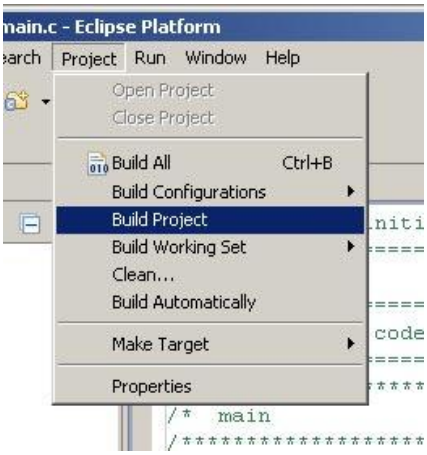
شکل ۷.۲۶

در این‌جا با انتخاب هر یک فایل‌ها از پنجره‌ی سمت چپ، آن فایل را در قسمت سمت راست خواهید دید.



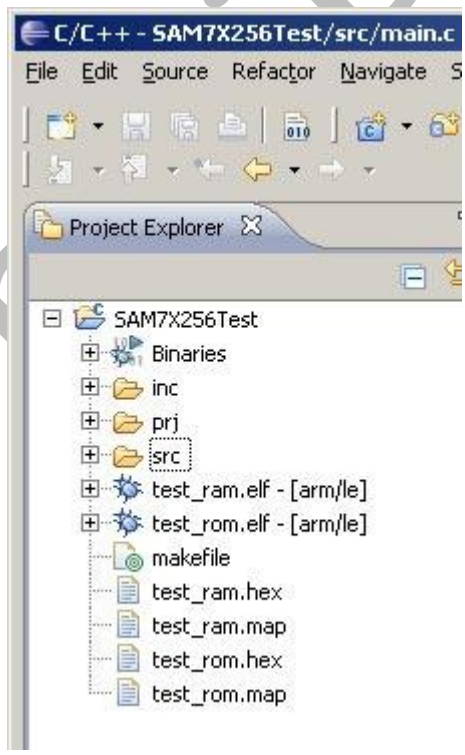
شکل ۷.۲۷

برای ساختن فایل‌های نهایی پروژه از طریق Project> Build Project اقدام کنید.



شکل ۷.۲۸

اگر در ساختن فایل با خطایی مواجه نشویم، یک فایل elf در خروجی تولید خواهد شد.

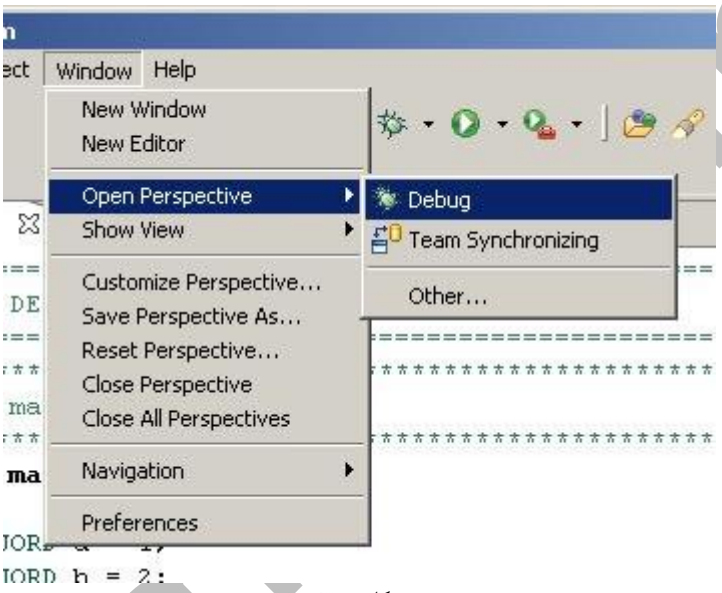


شکل ۷.۲۹

در این قسمت، عملیات نصب و تنظیم اولیه به پایان رسید. در قسمت بعد، شروع به نصب و پیکره‌بندی واسطه‌های عیب‌یاب[1] خواهیم نمود.

پیکره‌بندی عیب‌یاب

برای انجام مجموعه‌ی عملیات عیب‌یابی باید محیط مربوط را در آدرس "Debug Perspective" باز کنیم. بدین منظور، به آدرس "Window> Open Perspective Debug" بروید.



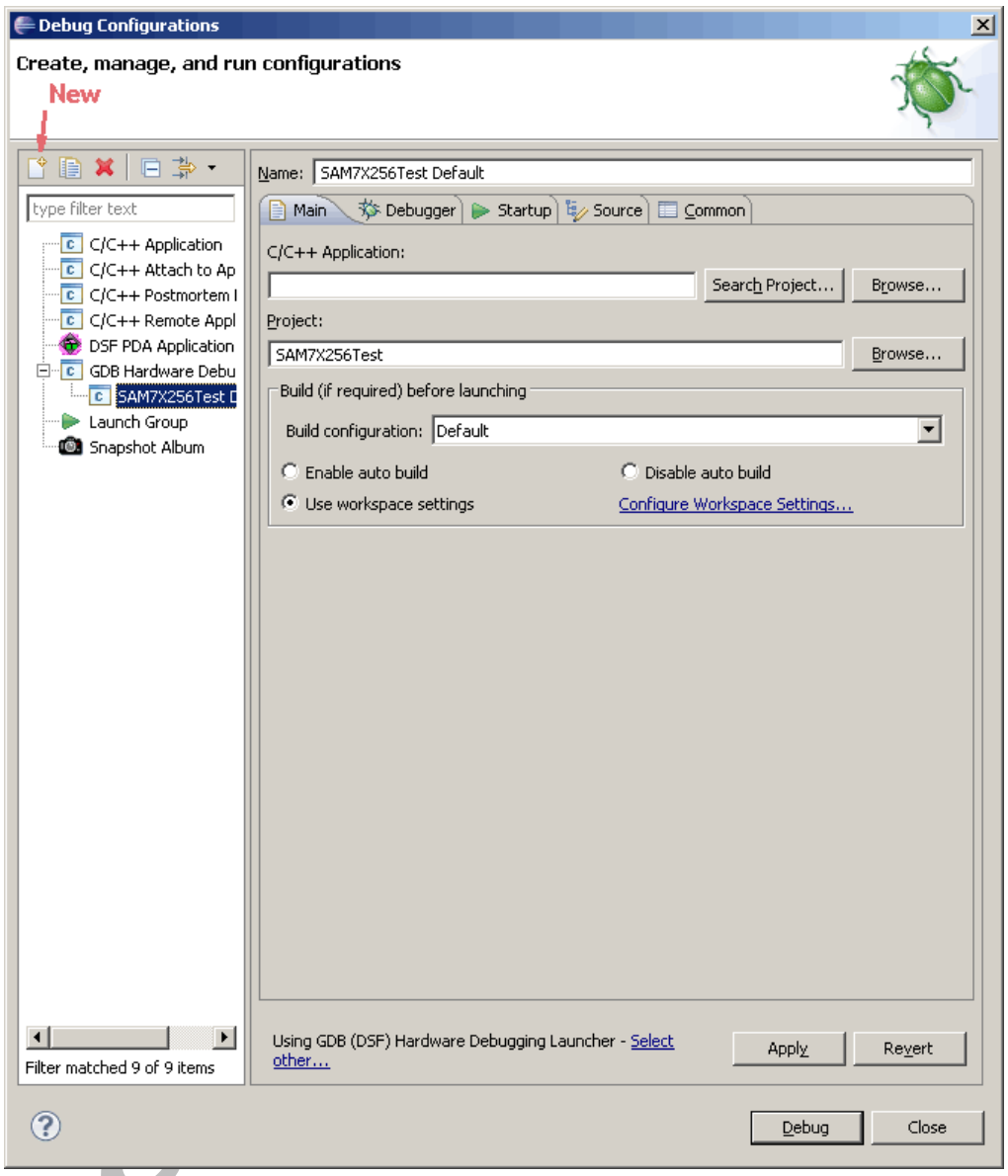
شکل ۷.۳۰

اینک برای پیکره‌بندی عیب‌یاب آیکون زیر را کلیک کرده و از طریق منوی بازشونده " Debug Configuration..." را انتخاب کنید.

[1] Debugger

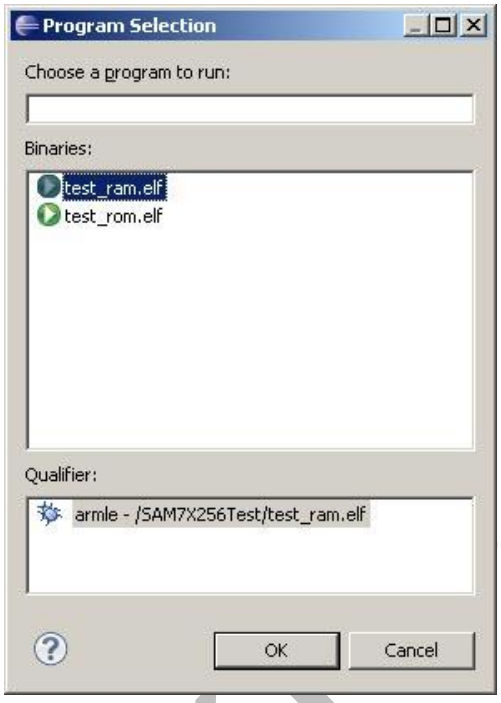
شکل ۷.۳۱

از پنجره‌ی باز شده "GDB Hardware Debugging" را انتخاب کرده و کلید "New" را بزنید. پنجره‌پیکره‌بندی به شکل زیر تغییر خواهد کرد.


Debug Configurations
Create, manage, and run configurations
New
type filter text
C/C++ Application
C/C++ Attach to Ap
C/C++ Postmortem I
C/C++ Remote Appl
DSF PDA Application
GDB Hardware Debu
SAM7X256Test D
Launch Group
Snapshot Album
Filter matched 9 of 9 items
Name: SAM7X256Test Default
Main
Debugger
Startup
Source
Common
C/C++ Application:
Search Project...
Browse...
Project:
SAM7X256Test
Browse...
Build (if required) before launching
Build configuration: Default
Enable auto build
Disable auto build
Use workspace settings
Configure Workspace Settings...
Using GDB (DSF) Hardware Debugging Launcher - Select other...
Apply
Revert
Debug
Close

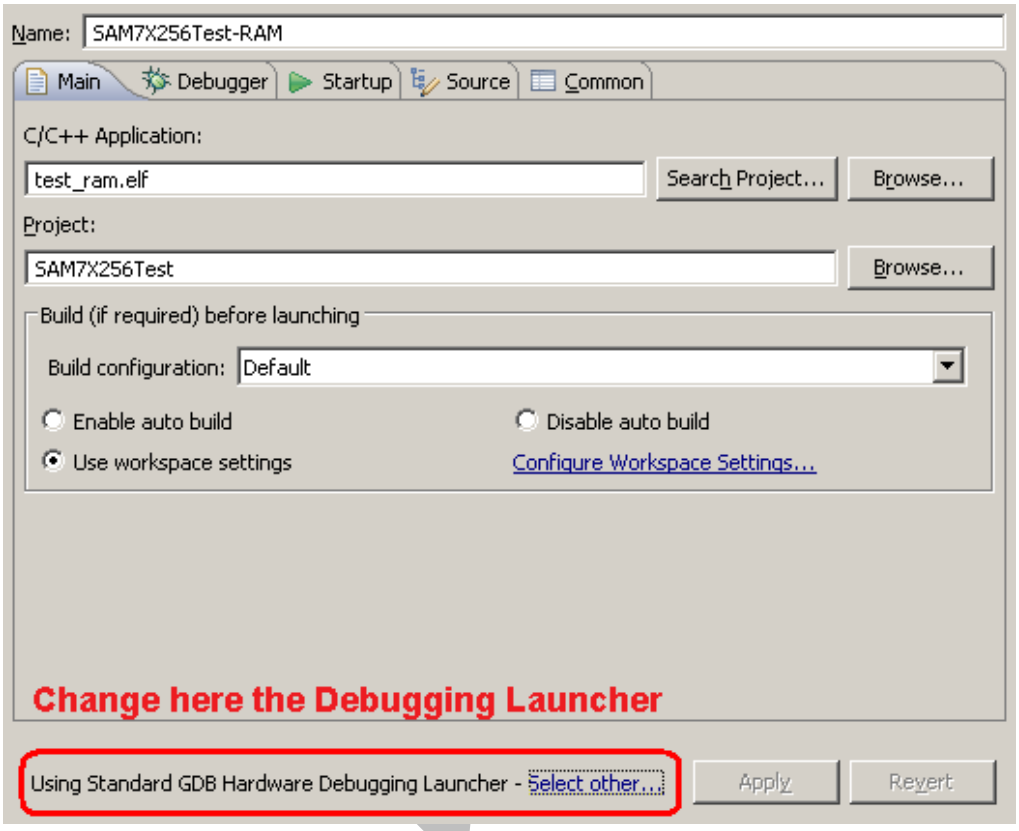
شكل ٧.٣٢

کلید "...Search Project" را زده و فایل test_ram.elf را انتخاب کنید.



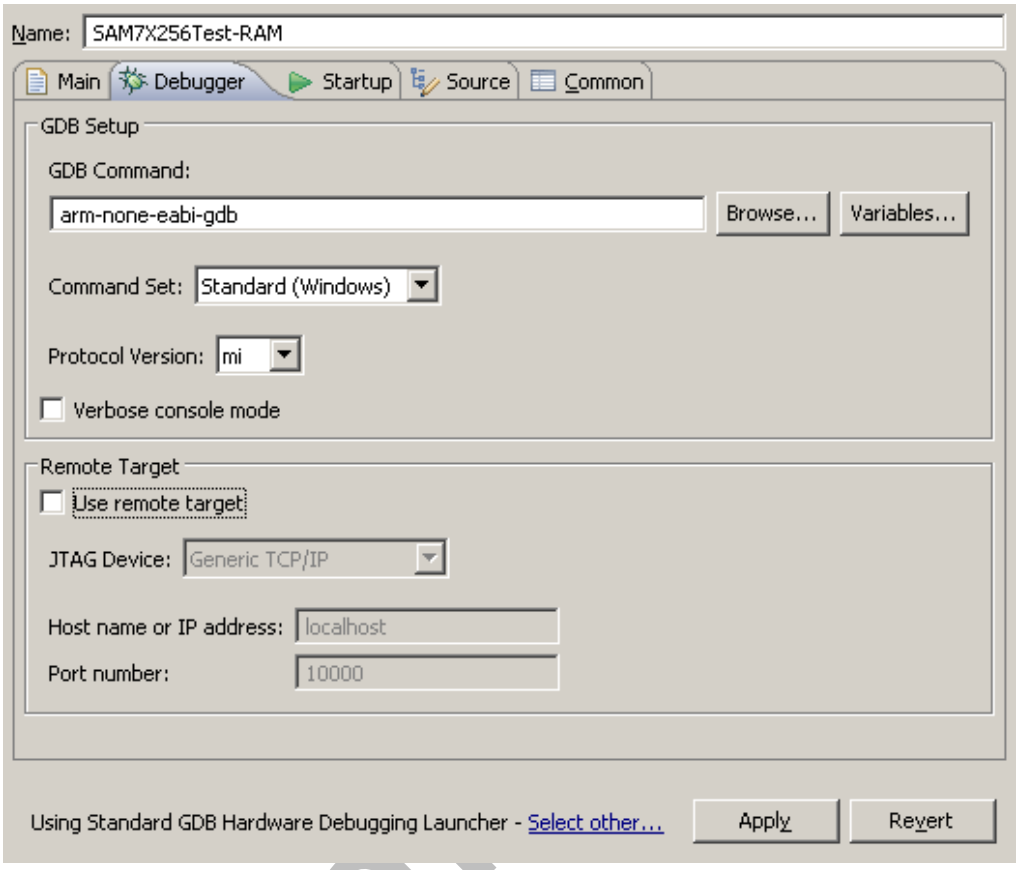
شکل ۷.۳۳

"Ok" را بزنید و نام پروژه را از "SAM7X256Test Default" به "SAM7X256Test-RAM" تغییر دهید. در پایین پنجره می‌توانید نوع عیب‌یاب را انتخاب کنید. در این‌جا، همانند آن اسمی که در شکل آمده، انتخاب شده است. پنجره‌ی پیکره‌بندی را هم‌اکنون در شکل زیر می‌بینید:



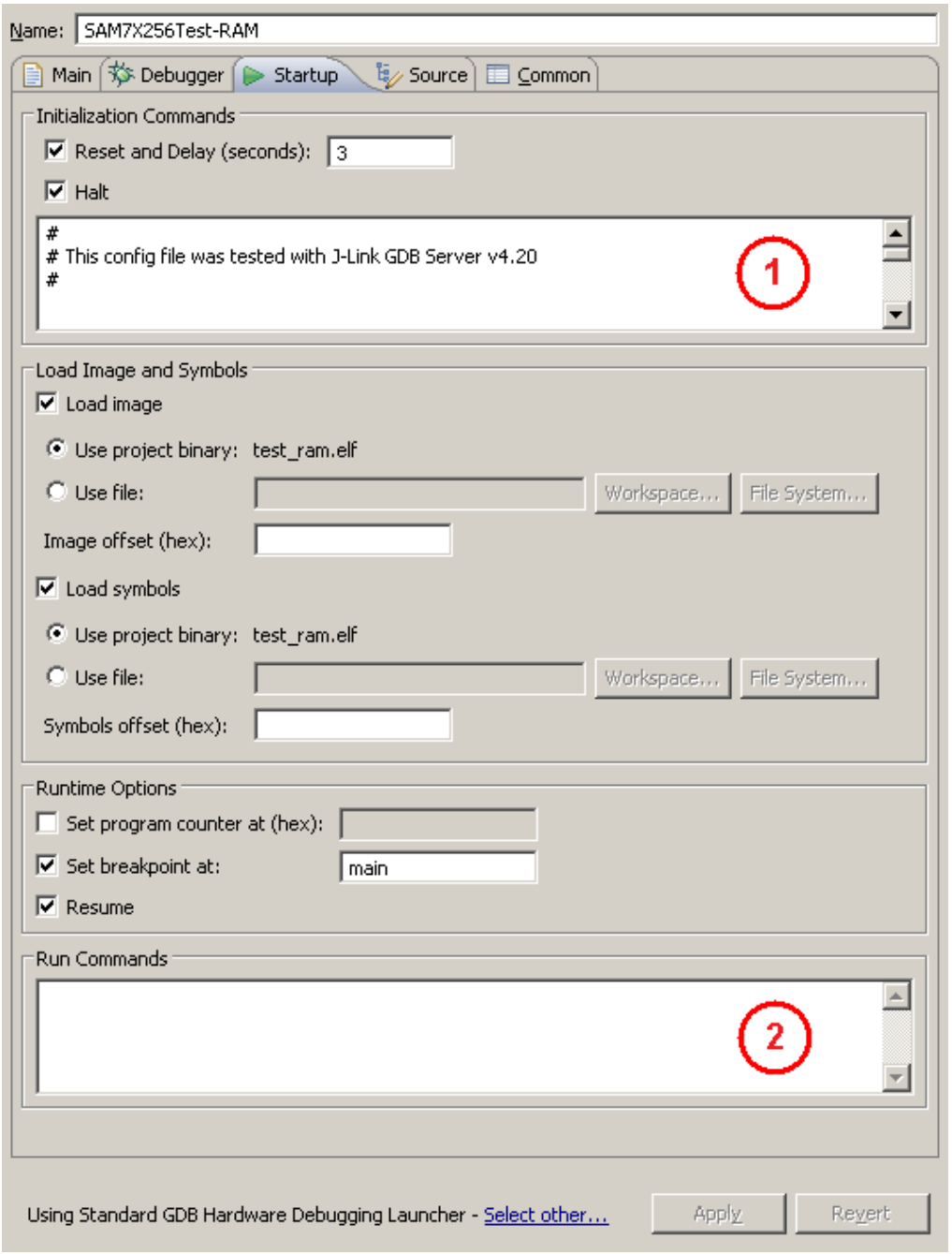
شکل ۷.۳۴

بـــه قســـمت "Debugger" رفتـــه و در قســـمت "GDB Command" از "arm-none-eabi-gdb" استفاده کنید. در اینجا **حتماً** علامت تیک "Use Remote Target" را بردارید.



شکل ۷.۳۵

به قسمت "Startup" رفته و تنظیمات زیر را انجام دهید.



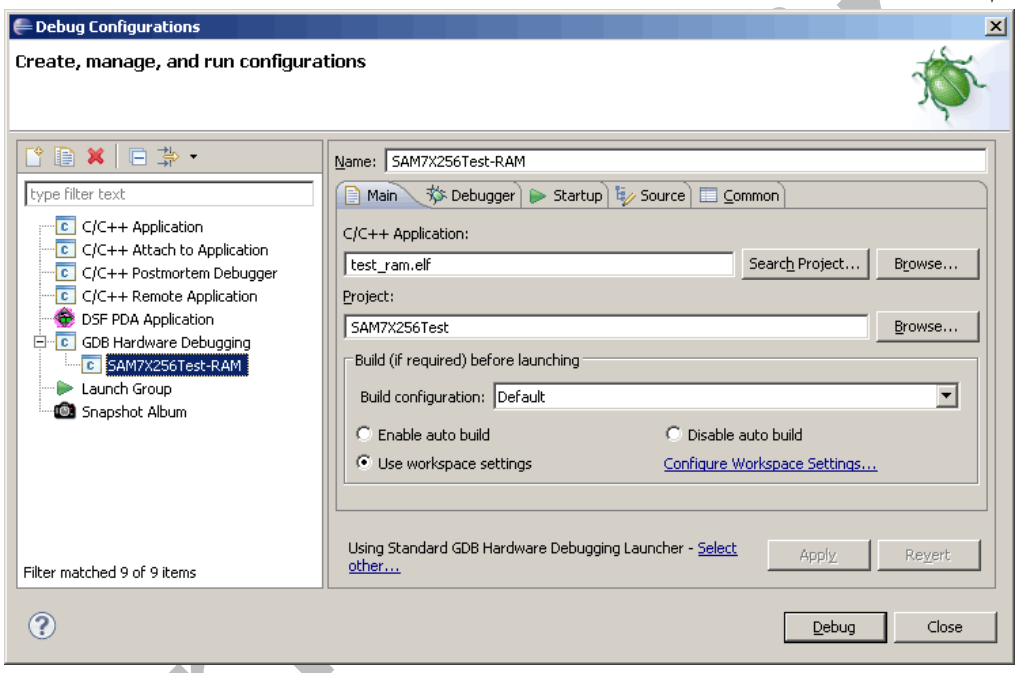
شکل ۷.۳۶

فایلی به نام sam7x256_ram_jlink.gdb ، جزیی از مثال ارایه شده است. محتویات فایل نمونه را در قسمت 1 در شکل بالا کپی کنید.

در این‌جا، عملیات پیکره‌بندی به پایان رسیده است. پنجره‌ی فوق را بسته و تغییرات را ذخیره کنید.

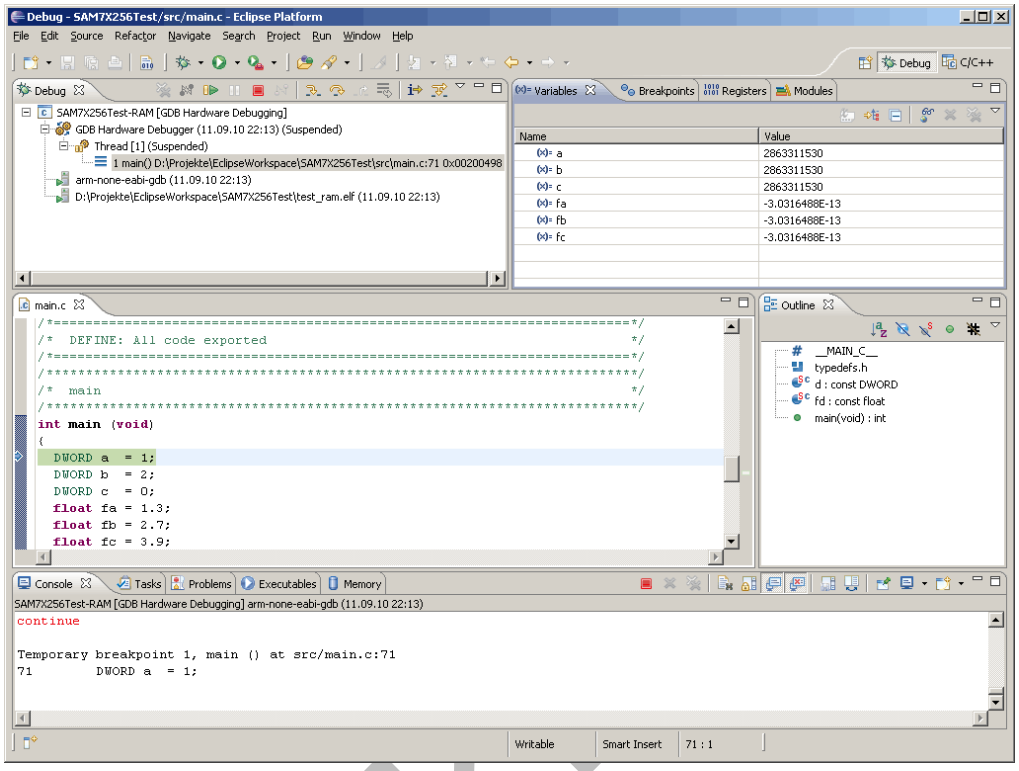
۷.۴.۴.۲ چگونه عیب‌یابی کنیم؟

برای عیب‌یابی توسط Eclipse از GDB استفاده خواهیم کرد. عیب‌یاب JLink را به رایانه‌ی شخصی و برد مورد آزمایش متصل کنید. تغذیه‌ی برد را روشن کرده و JLink-GDB-Server را نیز اجرا کنید. واسط عیب‌یابی، برای بار اول از طریق منوی پیکره‌بندی اجرا می‌شود. بر روی پیکره‌بندی جدید خود با نام "**SAM7×256Test–RAM**" رفته و کلید Debug را بزنید.



شکل ۷.۳۷

در این‌جا، محیط Eclipse به شکل زیر در خواهد آمد:



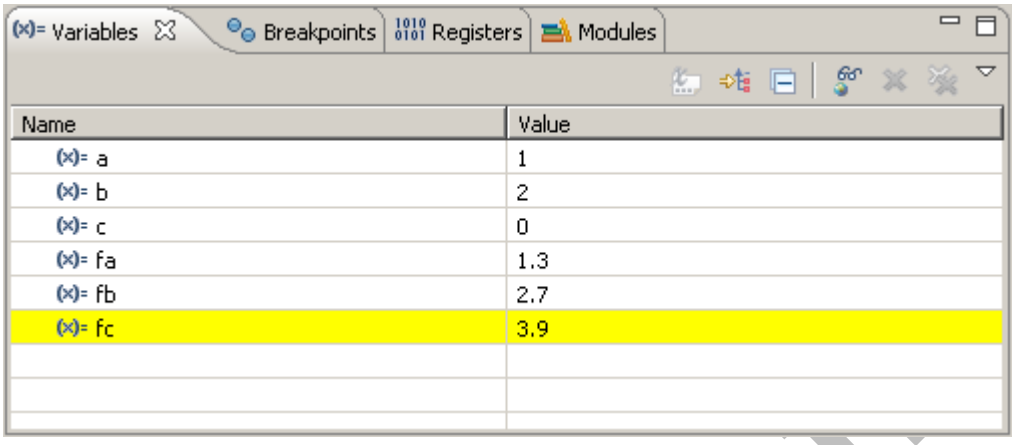
شکل ۷.۳۸

پنجره‌ای که در پایین می‌بینید، پنجره‌ی دستور "GDB Debugger" و پنجره‌ی بالا سمت چپ پنجره‌ی "Debug" است. خطی که در فایل کد منبع (main.c) پررنگ[۱] شده، خطی است که اجرا خواهد شد. پنجره‌ی بالا سمت راست متغیرها را نشان می‌دهد. می‌توانید از طریق دستکاری نمادها، "Break Points"، "Register" و "Modules" را نیز در این محیط بگنجانید. مقادیر متغیرها در اینجا اعداد عجیبی هستند. علت آن این‌است که این متغیرها مقداردهی اولیه نشده‌اند.

از طریق کلیدهای زیر در پنجره‌ی دیباگ می‌توانید بر روی فایل کد منبع خط به خط پیش رفته و نتیجه‌ی اجرا را ببینید:

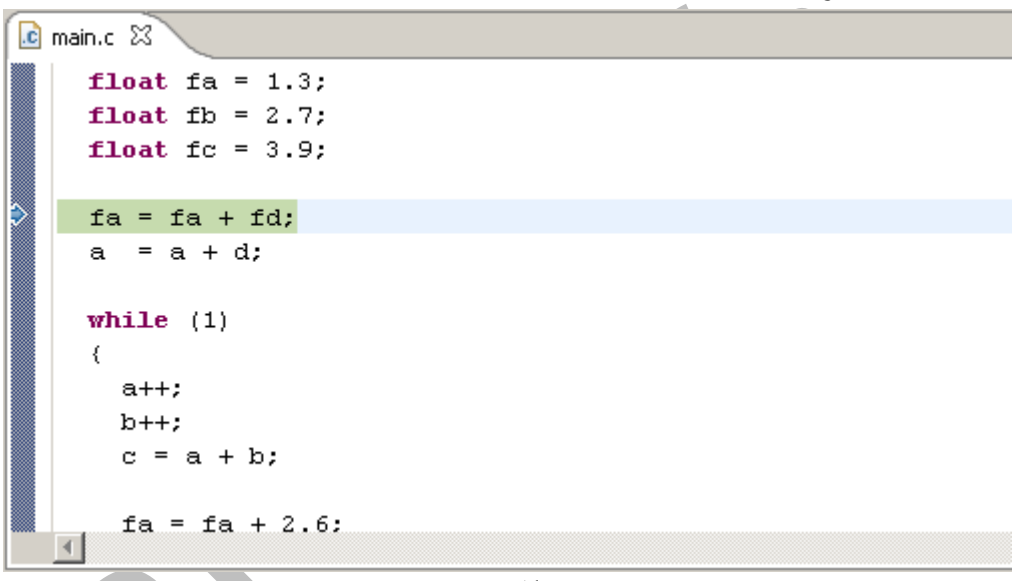
بعد از اجرای ۶ مرحله، پنجره متغیرها به شکل زیر در خواهند آمد:

[۱] Highlight



شکل ۷.۳۹

و پنجره‌ی فایل منبع نیز این‌گونه است:



شکل ۷.۴۰

۷.۴.۴.۳ تنظیم نقاط شکست[۱]

تنظیم نقاط شکست کار بسیار ساده‌ای است. برای این کار در قسمت خاکستری رنگ سمت چپ پنجره‌ی فایل منبع دوبار کلیک کنید. (Double Click) برای حذف آن مجدداً همان کار را تکرار کنید.

---

[۱] Break Point

```
while (1)
{
   a++;
   b++;
   c = a + b;

   fa = fa + 2.6;
   fb = fb + 1.67;
   fc = fa + fb;
}
```

شکل ۷.۴۱

با کلیک بر روی کلید "Resume"، برنامه تا رسیدن به خط حاوی نقطه‌ی شکست اجرا خواهد شد و بر روی این خط خواهد ایستاد.

```
while (1)
{
   a++;
   b++;
   c = a + b;

   fa = fa + 2.6;
   fb = fb + 1.67;
   fc = fa + fb;
}
```

شکل ۷.۴۲

مشکل: اگر در فرآیند ایستادن و یا اجرای عیب‌یاب با مشکلی برخوردید، Eclipse و GDB Server را ببندید، برد را ریست کرده و مجدداً آن‌ها را باز کنید.

فصل هشتم

برنامه‌ریزی و عیب یابی

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ Jlink چیست ؟
- ✓ روشهای برنامه‌ریزی حافظه فلش.

۸.۱ امولاتور Jlink

یک واسط سخت‌افزاری معروف، برای برنامه‌ریزی حافظه‌ی فلش و عیب‌یابی تعداد زیادی از پردازنده‌های ARM است. این واسط از طریق JTAG و SWD به این پردازنده‌ها متصل می‌شود. پردازنده‌های با تعداد پایه‌های بیشتر، از JTAG استفاده می‌کنند. ولی پردازنده‌هایی که تعداد پایه‌های کمتری دارند، فقط از ارتباط SWD پشتیبانی می‌کنند. SWD با ظهور پردازنده‌های Cortex، ارایه شد. این واسط از تعداد پایه‌های کمتری نسبت به JTAG استفاده می‌کنند.

این امولاتور، از پردازنده‌ی ۳۲ بیتی داخلی برای ارتباط با پردازنده‌های هدف استفاده می‌کند. هم‌اکنون تعداد زیادی از این امولاتور در سراسر دنیا فروخته شده و استفاده می‌شوند.

Jlink در تعداد زیادی از محیط‌های توسعه‌ی استاندارد همانند IAR، Keil، Rowley پشتیبانی شده است. با فروش بیش از 60,000 دستگاه از این امولاتور در سراسر دنیا، Jlink یکی از معروف‌ترین امولاتورهای ARM به حساب می‌آید.

۸.۱.۱ مشخصات JLink

- دانلود مستقیم کد بر روی حافظه‌ی فلش بسیاری از میکروکنترولرهای ARM؛
- رابط USB 2.0؛

- پردازنده‌های پشتیبانی شده: ARM7/9/11، Cortex-A5/A8، CortexM0/M1/M3/M4، Cortex-R4؛
- پشتیبانی از SWD؛
- پشتیبانی از SWV؛
- تشخیص خودکار هسته؛
- سرعت حداکثر JTAG تا 12 MHz؛
- سرعت دانلود کد تا حداکثر 720 Kbyte/Sec (برای ARM7 در فرکانس 50 MHz و سرعت JTAG 12 MHz)؛
- قرارگیری در ساختار IAR به صورت یکپارچه؛
- تغذیه‌ی Jlink از طریق USB انجام می‌شود؛
- پشتیبانی از کلاک تطبیق‌پذیر[1]؛
- تمامی سیگنال‌های JTAG و ولتاژ برد هدف اندازه گیری شده و نمایش داده می‌شوند؛
- رابط استاندارد JTAG ۲۰ پایه.

[1] Adaptive

جدول 1-JLink

با استفاده از JLink ، قادر خواهید بود برنامه‌ی در حال اجرا را متوقف ساخته و به صورت مرحله به مرحله اجرا کنید. در هر مرحله، محتویات ثبات‌ها و حافظه را خوانده و یا آن‌ها را تغییر دهید. همچنین با استفاده از نقاط شکست و یا نقاط نگهبان، روند اجرای برنامه را کنترل کنید.

۸.۲ ULink

شرکت Keil در اصل این مولاتور را برای نرم‌افزار uVision شرکتش طراحی نموده است. مولاتور Ulink ساخت شرکت Keil امکانات JLink را با محدودیت‌هایی مشخص در اختیار می‌گذارد. این مولاتور با محیط‌های توسعه‌ی کمتری سازگار است.

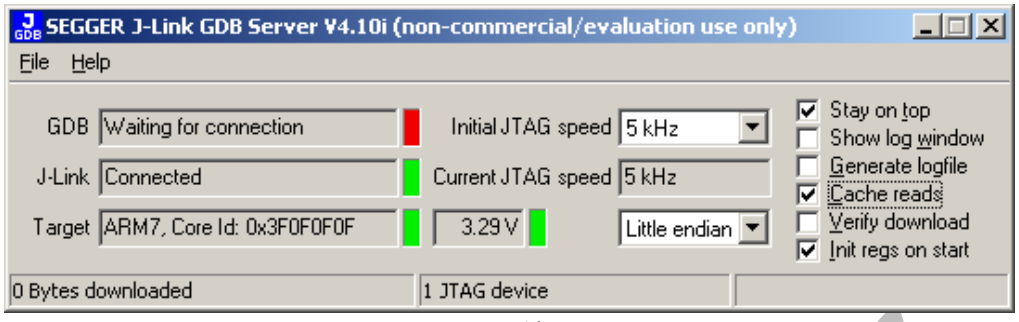
۸.۳ Wiggler

این مولاتور به پورت موازی کامپیوتر متصل شده و توانایی پروگرام کردن و عیب یابی بسیاری سیستم‌های ARM مبتنی را دارد. برای استفاده از این پرواگرامر ارزان قیمت (که نقشه مداری آن نیز به صورت رایگان موجود می‌باشد)، می‌توان از نرم‌افزار H-jtag (که آن هم به صورت رایگان در دسترس است) استفاده کرد.

۸.۴ سرور GDB

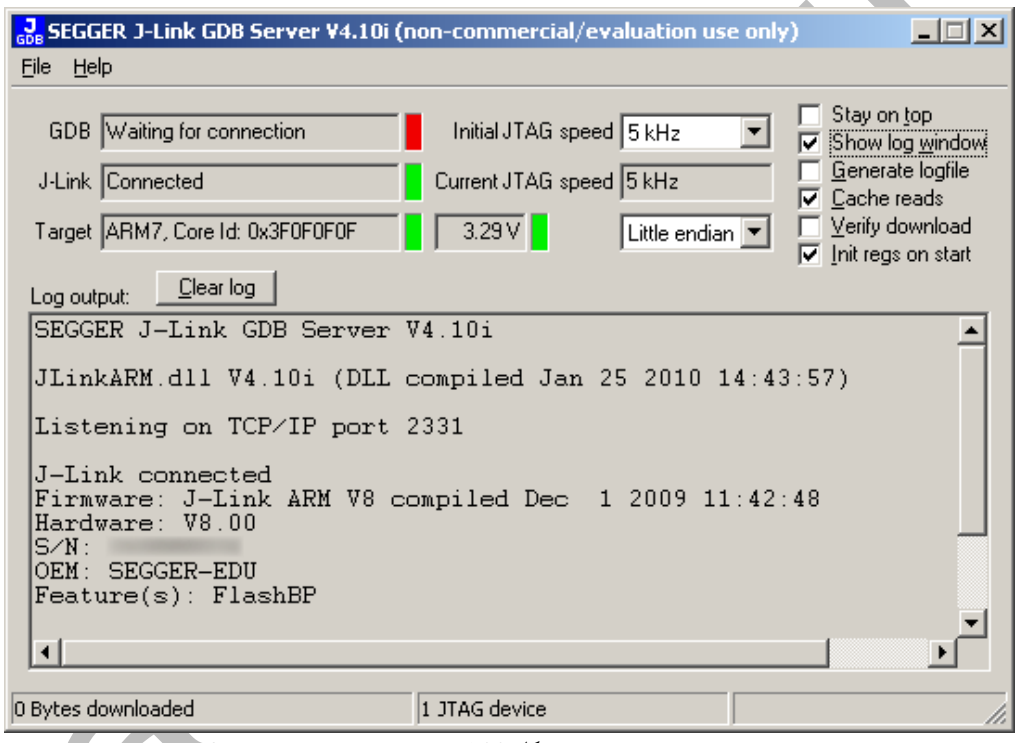
عیب‌یاب پروژه GNU[۱]، یک عیب‌یاب رایگان است که براساس اظهارات GPL آزاد شده است. این عیب-یاب توانایی اتصال به هرگونه عیب سخت‌افزاری که یک نرم‌افزار GDB Server به همراه خود دارد را داراست. به‌منظور عیب‌یابی با محیط توسعه‌ی Eclipse، در این‌جا از GDB به همراه یک GDB Server استفاده می‌کنیم. در این‌جا، باید نرم‌افزار JLink را نصب کرده باشید. اتصالات لازم برقرار باشند و برد مورد آزمایش روشن باشد. "JLink GDB Server" را نیز آغاز کنید. این برنامه از منوی Start و در شاخه‌ی Segger> J-Link ARM موجود است.

[۱] GNU Project Debugger-GDB



شکل ۷.۴۳

در سمت راست قسمت‌های مربوط را همانند زیر تنظیم کنید:



شکل ۷.۴۴

در این قسمت، تنظیمات عیب‌یاب را انجام داده‌ایم و به قسمت‌های بعد می‌پردازیم.

۸.۵ برنامه‌ریزی حافظه‌ی فلش میکروکنترولرها

برای برنامه‌ریزی میکروکنترولرهای ARM ، می‌توان از رابط JTAG استفاده کرد. JTAG یک پورت و مجموعه‌ای ازپروتوکل‌های استاندارد است که در استاندارد IEEE1194.1 به تفصیل آمده است. از طریق این پورت می‌توان تراشه‌های برنامه پذیر مختلف را برنامه ریزی و یا عیب یابی کرد. یک واسط استاندارد دیگر برای برنامه ریزی میکروکنترولرهای با هسته ARM Cortex ، استفاده از SWD [۱] است. در بعضی از میکروکنترولرهای ساده Cortex که دارای تعداد پایه اندکی هستند، از این رابط ۲-سیمه استفاده می شود.

یک راه استاندارد دیگر، برای برنامه‌ریزی این میکروکنترولرها نیز وجود دارد. این میکروکنترولرها می-توانند با استفاده از بوت‌لودر موجود بر روی ROM داخلی‌شان، یکی از وسایل جانبی خود را راه اندازی کرده (به عنوان مثال AT91SAM از USB و یا پورت سریال Debug ، STM32F و LPC ها از پورت USART استفاده می‌کنند) و از این طریق، به رایانه شخصی متصل شوند. با اتصال تراشه مورد-نظر به رایانه و اجرای نرم‌افزار های مربوط (SAM-BA برای AT91SAM ، Flash Magic برای LPC و Flash Loader برای STM32F) می‌توان تراشه مذکور را برنامه‌ریزی کرد. در بعضی از میکروکنترولرها با استفاده از نوعی بوت‌لودر می‌توان تراشه را به یک حافظه‌ی فلش USB تبدیل کرد. برای کار کردن با این حافظه‌ی USB ایجاد شده، فایل باینری موردنظر را از روی رایانه شخصی به راحتی به آن انتقال داده (از طریق یک Copy و Paste ساده و یا Drag کردن فایل موردنظر) و عملیات برنامه‌ریزی انجام می‌شود. همان‌طور که احتمالا حدس می‌زنید، انواع مختلف دیگری از بوت‌لوردربرای ارتباط از طریق CAN ، EMAC (شبکه اترنت) و ... نیز وجود دارند.

۸.۵.۱ Flash Magic

این برنامه، اختصاصا برای برنامه‌ریزی میکروکنترولرهای شرکت NXP طراحی شده است. از طریق این برنامه می‌توانید محتوای فلش بسیاری از میکروکنترولرهای ARM7/9 و ARM Cortex این شرکت را برنامه ریزی یا پاک کرده و یا بخوانید و یا قفل کنید. تمامی عملیات در این برنامه به صورت سکتور به سکتور قابل انجام است.

[۱] Serial Wire Debug - SWD

AT91 SAM-BA ۸.۵.۲

این برنامه اختصاصا برای برنامه‌ریزی میکروکنترولرهای شرکت Atmel طراحی شده است. از طریق این برنامه می‌توانید محتوای فلش بسیاری از میکروکنترولرهای ARM7/9 و ARM Cortex این شرکت را برنامه ریزی یا پاک کرده و یا بخوانید و یا قفل کنید. همچنین، امکان دسترسی به تراشه‌های SDRAM ، Nand Flash ، Data Flash و .. متصل به تراشه اصلی نیز وجود دارد. در این برنامه دسترسی کاملی به محتویات حافظه‌ها خواهید داشت. بدین ترتیب، توانایی تغییر تمامی بایت‌های حافظه را نیز خواهید داشت.

ST Flash Loader ۸.۵.۳

H-Jtag ۸.۵.۴

بخش ۵

مدارهای عملی و پروژه‌های کاربردی

فصل نهم

ارتباط با واسط‌های ورودی و خروجی

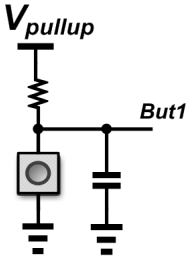
اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ اتصال ورودی‌های استاندارد همانند صفحه کلید؛
- ✓ راه‌اندازی خروجی‌هایی همانند رله، LED و ...؛
- ✓ راه‌اندازی رله ها و نمایشگرها؛
- ✓ ارتباط با USB و سنسورهای دما.

۹.۱ اتصال کلید به عنوان ورودی

برای اتصال انواع کلید به ورودی های GPIO میکروکنترولر،معمولا از مدارهایی مشابه زیر استفاده می شود:



شکل ۹.۱

خازن در این مدار، نقش فیلتر را بر عهده داشته و تا حد زیادی از نویز جلوگیری می کند. همچنین، هنگام فشار کلید، از ایجاد پالس،های اضافی نیز جلوگیری می کند. نقش مقاومت، حفظ سطح ولتاژ اولیه پایه کلید در یک سطح معین (در این‌جا V_{pullup}) است. با فشار کلید، سطح ولتاژ تغییر می کند و یک پالس بالا رونده یا پایین رونده به میکروکنترولر اعمال می شود.

۹.۲ ارتباط با انواع نمایشگرها

انواع مختلفی از نمایشگر وجود دارند که برای نشان دادن خروجی‌های میکروکنترولر و ارتباط با کاربر طراحی شده‌اند. LED ها ساده‌ترین انواع نشان دهنده‌ها هستند که به راحتی می‌توان آن‌ها را به خروجی‌های میکروکنترولر متصل کرد. جریان مصرفی LED ها، معمولاً بین 5mA تا 30mA می‌باشد. برای راه‌اندازی آن‌ها اگر به جریان کمی نیاز باشد (10mA و یا کمتر)، می‌توان از پایه‌های GPIO استفاده کرد. اگر جریان آن‌ها از 10mA بیشتر و از 20mA کمتر باشد، از پایه‌های GPIO سریع (که آن‌ها را با نام GPIO با جریان بالا نیز می‌شناسند. تعداد این پایه‌ها عموماً زیاد نبوده و به ۴ عدد می‌رسند)، می‌توان استفاده کرد.

به‌رغم مطالب گفته شده، بهتر است از بافرهای LVTTL و یا HCMOS برای خروجی میکروکنترولر استفاده کرد. به عنوان مثال، تراشه‌های 74HC244 و یا 74VCX244 پیشنهادهای مناسبی برای این منظور هستند.

سون سگمنت[۱] ها نیز، ترکیبی از ۸ نشان دهنده‌ی LED در یک بسته‌بندی، برای نشان دادن اعداد و بعضی حروف هستند. همچنین، با آن‌ها همانند LED ها رفتار می‌کنیم.

نشان دهنده‌ی کریستال مایع (LCD)، از انواع رایج نشان دهنده هستند که امروزه بسیار رایج می‌باشند. LCD ها به چندین دسته تقسیم می‌شوند که دو نوع پرکاربرد آن را با نام‌های کاراکتری و گرافیکی می‌شناسند. LCD ها انواعی شبیه به سون سگمنت‌ها نیز دارند که در بعضی دستگاه‌های خاص کاربرد دارند. LCD های کاراکتری از طریق رابط ۴ یا ۸ پایه‌ی داده، با میکروکنترولرها ارتباط دارند. میکروکنترولر توسط دستورهایی که از این طریق به LCD می‌دهد، بر روی آن اعداد، حروف و یا علایم مورد نظر را چاپ می‌کند. بیشتر LCD های کاراکتری از یک استاندارد یکسان برای قالب‌بندی دستورها، ترتیب و نام پایه‌ها استفاده می‌کنند. به همین علت، راه‌اندازی آن‌ها به سهولت از روی نمونه کدهای نوشته شده‌ی استاندارد، امکان‌پذیر است.

LCD های گرافیکی، بر مبنای نمایشگر پیکسلی[۲] کار می‌کنند. بدین منظور که برای نوشتن و یا ترسیم اجسام و یا خطوط بر روی آن‌ها، می‌بایستی یک یا چند پیکسل (که توسط نرم‌افزار انتخاب می‌شوند) خاص را روشن کرد. مشابه LCD های کاراکتری، برای راه‌اندازی این نوع نمایشگرها نیز به یک رابط داده و خطوط اضافی برای کلاک و پایه‌های نوشتن/ خواندن و داده/ دستور نیاز داریم. نمونه‌ای از این واسط‌ها را در شکل۹.۲ می‌بینیم.

راه‌اندازی LCD ها توسط میکروکنترولر، با خروجی‌های بافر نشده امکان‌پذیر است. تنها مسأله‌ی مهم سطوح ولتاژ (که می‌بایستی با سطوح 3.3V یا کمتر میکروکنترولر سازگار باشد)، رابط بین LCD و

---

[۱] Seven Segment

[۲] Pixel-Based

میکروکنترولر است. در صورت لزوم می‌توان از یک بافر، برای تطبیق سطوح ولتاژ و همچنین جلوگیری از جریان‌کشی بی‌مورد از میکروکنترولر استفاده کرد.

LCD های گرافیکی، دارای تراشه‌های راه‌انداز متنوعی هستند. بعضی از این تراشه‌ها بر روی خود پنل[۱] LCD موجود هستند (راه‌اندازی این نوع LCD ها عموماً ساده‌تر هستند). از انواع رایج این نوع LCD ها می‌توان نمایشگر با تراشه‌های راه‌انداز T6963، KS107/108 و بسیاری دیگر که عمدتاً ساخت کارخانه‌های Toshiba و یا Epson هستند.

راه‌اندازی LCD های بدون تراشه‌ی راه‌انداز، عموماً مشکل‌ترند. تراشه‌های راه‌انداز بسیاری از عملیات سطح پایین را بر عهده می‌گیرند و میکروکنترولر را از این حیث آزاد می‌گذارند. میکروکنترولر تنها از طریق واسط داده/ دستور و ارسال دستورهای لازم، به تراشه‌ی راه‌انداز، پیکسل مورد نظر را روشن و یا خاموش می‌کند. بر روی اکثر پنل‌های دارای تراشه‌ی راه‌انداز، حافظه‌های SRAM به عنوان حافظه‌ی گرافیکی، موجود است.

LCD های گرافیکی در انواع تک‌رنگ[۲] و رنگی[۳] یافت می‌شوند. LCD های رنگی دارای تراشه‌ی راه‌انداز، سهم کمی از بازار فروش را به خود اختصاص داده‌اند. در مقابل آن، LCD های رنگی بدون تراشه‌ی راه‌انداز هستند که بسیار پرکاربردتراند. این LCD ها با بسیاری از پردازنده‌های ARM دارای راه‌انداز داخلی، سازگار هستند. تعداد پایه‌های این مدل نمایشگر، زیادتر هستند. نمونه‌ای از آن‌ها به همراه نحوه‌ی ارتباط آن‌ها را در شکل ۹.۲ می‌بینیم.

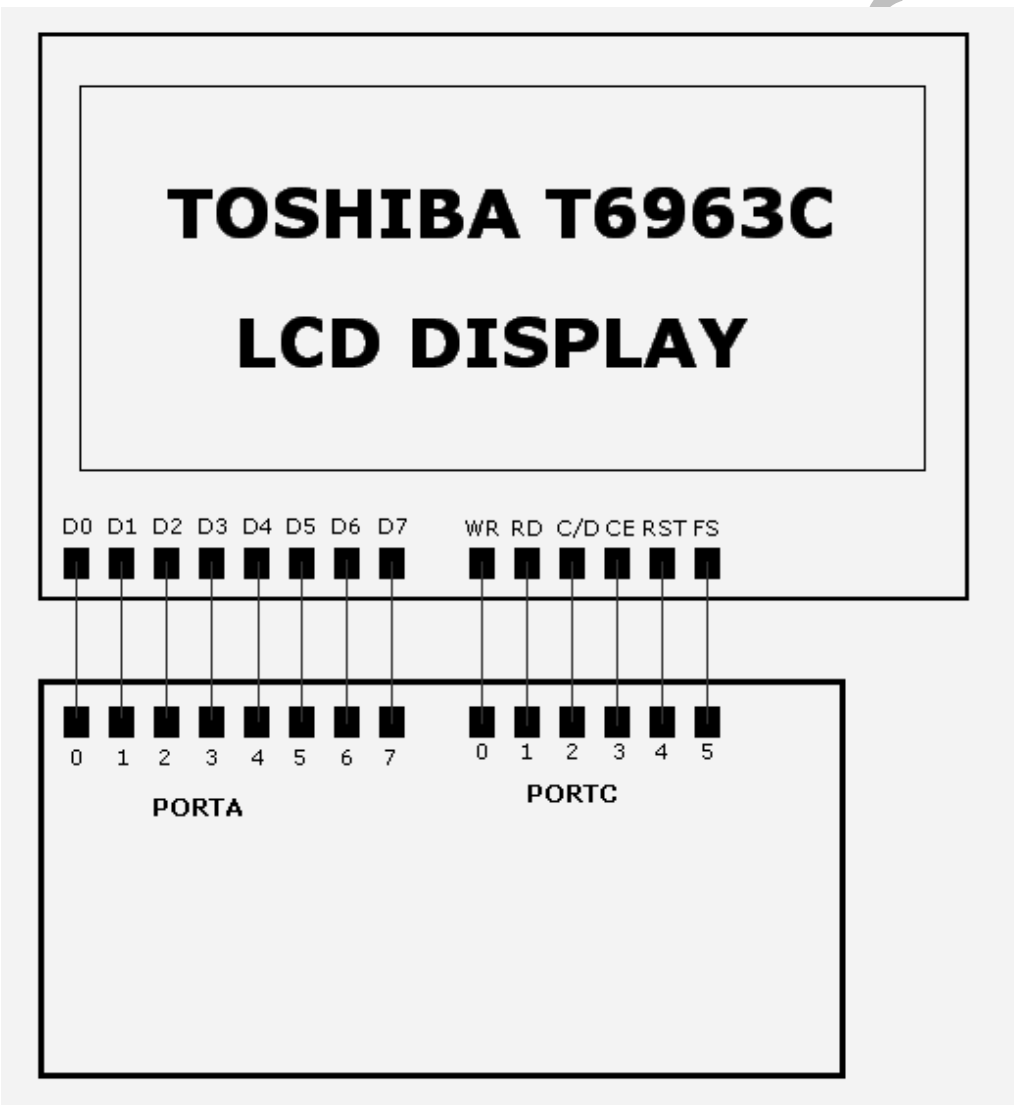
---

[۱] Panel

[۲] Monochrome

[۳] عموماً از نوع TFT


radzio.dxp.pl
TOSHIBA T6963C
AT91SAM7 & arm-gcc


TOSHIBA T6963C
LCD DISPLAY
D0 D1 D2 D3 D4 D5 D6 D7
WR RD C/D CE RST FS
0 1 2 3 4 5 6 7
0 1 2 3 4 5
PORTA
PORTC

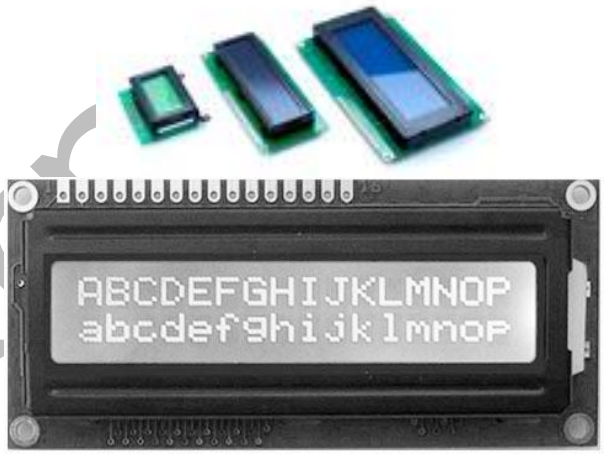
شکل ۹.۲

۹.۲.۱ نمایشگرهای LCD

با پیشرفت تکنولوژی، کاربران دستگاه‌های الکترونیکی انتظار بیشتری از واسطه‌ها کاربری و نمایشگرها دارند. این‌جا مشکلی برای طراحان سخت‌افزار مطرح می‌شود و آن ارزان بودن محصول نهایی است. کاربران علاوه بر انتظار واسط‌های گرافیکی جذاب‌تر، به دستگاه‌های ارزان‌تر اشتیاق بیشتری نشان می‌دهند. این در حالیست که در بسیاری از دستگاه‌ها، نمایش‌دهنده‌های گرافیکی قیمتی بیش از سایر عناصر مداری دارند. به‌منظور ارزیابی انواع مختلفی از نمایش‌دهنده‌ها به همراه قیمت‌های نمونه‌شان، انواع رایج این نمایش‌دهنده‌ها را در زیر بررسی کرده‌ایم.

۹.۲.۲ نمایشگرهای کاراکتری تک‌رنگ

راحت‌ترین، کاربردی‌ترین و ارزان‌ترین نوع نمایشگر با توانایی نمایش "متن" که عموماً براساس تراشه‌ی راه‌انداز HD 448 70 ساخته شده و توانایی نمایش اعداد، حروف انگلیسی و بعضی علامات و حروف خاص را دارند. این نمایشگرها در ابعاد و رنگ‌های مختلفی ساخته می‌شوند. بیشترین ابعاد ساخته شده‌ی آن‌ها برحسب کاراکتر، 2×16، 2×20، 4×16، 4×20 و... هستند. قیمت نمونه‌ی آن‌ها حوالی ۳ تا ۵ دلار است. برای راه‌اندازی، نیاز به ۷ پایه‌ی GPIO (۴ پایه داده و ۳ پایه‌ی RS و Rw و EN) دارند که برای بسیاری از کاربردهای میکروکنترولری با تعداد پایه‌های کم مناسب می‌باشند.



شکل ۹.۳

۹.۲.۳ نمایشگرهای گرافیکی تک‌رنگ

در این نوع نمایشگر، طراح توانایی کنترل هر یک از پیکسل‌های صفحه نمایش را دارد. ابعاد مختلفی از این نمایشگر موجودند. به عنوان مثال، 64×128، 128×240 و 240×320 و... . بسیاری از این نمایشگرها در مبنای تراشه‌ی راه‌انداز KS0108 ساخت شرکت سامسونگ و یا T6963 ساخت شرکت توشیبا بنا شده‌اند. در این نوع نمایشگرها، استفاده از فونت‌های خاص و ترسیم اشکال و دایره و... امکان‌پذیر شده است. برای راه‌اندازی آن، به حداقل ۱۱ پایه نیاز است. قیمت آن‌ها نیز بسته به ابعاد صفحه نمایش بین ۱۳ تا ۴۰ دلار است. بر روی این نمایشگرها عموماً از یک حافظه‌ی صفحه نمایش[۱] برای نگه‌داری اطلاعات صفحه نمایش استفاده می‌شود.

شکل ۹.۴

۹.۲.۴ نمایشگرهای گرافیکی رنگی

اساس کار این نمایشگرهای رنگی ۱۶ بیت یا ۲۴ بیتی، همانند نمایشگرهای رایانه‌های شخصی است که معمولاً کاربران خود را، تحت تأثیر زیبایی خود قرار می‌دهند. متأسفانه، کار با این نمایشگرها ساده نبوده و در بیشتر حالات به یک کنترل کننده‌ی LCD و حافظه‌ی صفحه نمایش زیاد در خارج از برد LCD نیاز است. این نمایشگرها، به ۴۰ پایه برای راه‌اندازی نیاز داشته و پردازنده‌هایی قدرت‌مند،

[۱] Frame buffer

توانایی راه‌اندازی آن‌ها را دارند. قیمت‌های این نمایشگرها بسیار متفاوت است (از ۲۵ دلار شروع شده و به بیش از ۲۰۰ دلار نیز می‌رسد). بعضی میکروکنترولرهای جدیدتر با هسته‌ی ARM7 دارای راه‌انداز داخلی LCD ۲۴ بیتی هستند. به عنوان مثال، LPC2470 و LPC2478 از آن جمله‌اند. بیشتر میکروکنترولرهای ARM9 نیز دارای یک چنین راه‌انداز داخلی هستند.

شکل ۹.۵

۹.۲.۵ ارتباط با صفحات لمسی[۱]

صفحات لمسی، بهترین راه استفاده از ورودی‌های کاربری پیچیده با کمترین هزینه و تعداد پایه‌های اندک می‌باشد. با استفاده از تنها ۴ سیم ارتباطی، امکاناتی از قبیل کلیک، حرکات اشاره‌ای[۲]، دنبال کردن حرکت انگشت و دیگر حرکات جذاب دیگر در اختیار طراح قرار می‌گیرد.

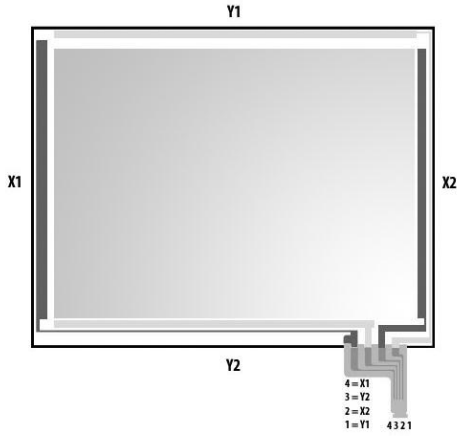
اساس کار صفحات لمسی مقاومتی براساس پتانسیومترهای مقاومتی ساده بنا شده‌اند. با اعمال یک ولتاژ DC (مثلاً 3.3V) و صفر بین دو پایه‌ی کناری آن، مقدار خوانده شده بر روی پایه‌ی وسط نمایانگر مکان اشاره شده بر روی صفحه است. مقدار مذکور با استفاده از ADC میکروکنترولر خوانده شده وبرای محاسبه‌ی مکان، توسط نرم‌افزار استفاده می‌شود.

تنها تفاوت این صفحه با پتانسیومتر در همین است که به جای استفاده از محور چرخنده برای تغییر مقدار مقاومت، از نقطه‌ی لمس شده برای تغییر مقدار مقاومت استفاده می‌شود.

در مورد صفحات لمسی مقاومتی، در محور X و Y برای اندازه‌گیری وجود دارند. اما، تنها ۴ پایه‌ی X1، X2، Y1 و Y2 موجوداند. ایده کار ساده است، ابتدا به دیاگرام زیر که شمایی کلی از این صفحات را نشان می‌دهد، توجه کنید.

---

[۱] Touch Screen
[۲] Gesture



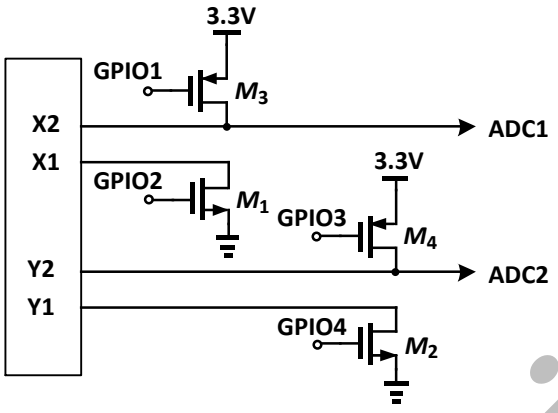
شکل ۹.۶

برای خواندن اطلاعات هر محور، ابتدا یک ولتاژ 3.3V و GND را بین دو پایه‌ی محور دیگر متصل کرده، سپس اطلاعات محور مورد نظر را به صورت ولتاژ و با استفاده از ADC می‌خوانیم. اطلاعات محور دیگر را نیز به همین صورت و با جابه‌جایی اتصالات ارایه شده می‌خوانیم. برای این منظور به دو شکل ۹.۷ و ۹.۸ و نیز جدول Touch_1 که اتصالات مورد نیاز را تشریح می‌کنند، توجه کنید.

جدول Touch_1 – اطلاعات پایه های صفحه لمسی مقاومتی

Y2	X2	Y1	X1	
—	GND	ADC	3.3V	برای اندازه‌گیری در محور X
GND	—	3.3V	ADC	برای اندازه‌گیری در محور Y

دقت کنید که اطلاعات دو محور مورد نظر را به طور همزمان نمی‌توان خواند. در عمل، ابتدا اطلاعات یک محور خوانده شده، سپس به سرعت اتصالات عوض شده و اطلاعات محور دیگر نیز خوانده می‌شوند. برای تعویض اتصالات از روش‌های الکترونیکی همانند روش ارایه شده در شکل ۹.۷ استفاده می‌کنیم.



شکل ۹.۷

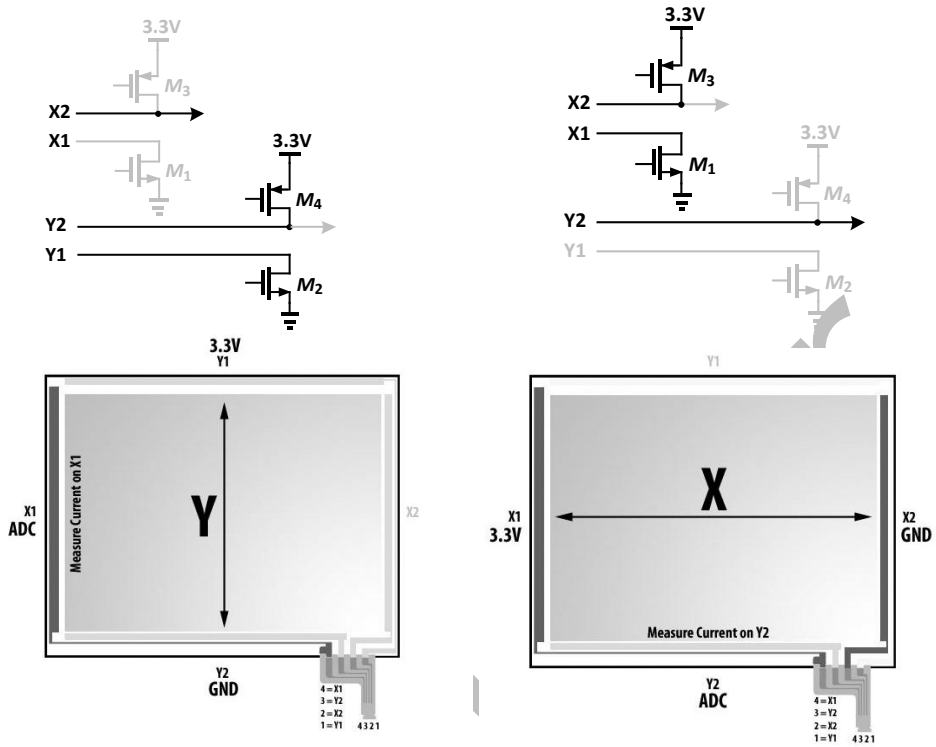
در این شکل، ماسفت[۱] ها از انواع با مقاومت $r_{ds,on}$ کم انتخاب شده‌اند.

اگر بدانیم که صفحه‌ی لمسی مقاومت بالایی داشته و از پایانه‌های 3.3V جریان زیادی نمی‌کشد، می‌توانیم مستقیماً با استفاده از پایه‌های GPIO نیز عملیات سوئیچینگ مزبور را صورت دهیم.

نمونه‌ای از کد عملیاتی فوق با استفاده از میکروکنترولر LPC2148، در زیر آمده است.

در شکل ۹.۸، نحوه‌ی خواندن هر یک از محورها در مراحل جداگانه آمده است. ابتدا M1 و M3 روشن شده و ADC2 خوانده می‌شود، سپس M2 و M4 روشن شده و مقدار ADC1 را می‌خوانیم (به ترتیب اشکال 2-Touch و 3-Touch). در هر یک از حالت‌ها، ماسفت‌هایی که اسم برده نشده‌اند، خاموش می‌باشند. دقت کنید که NMOS ها با سطح منطقی 1 و PMOS ها با سطح منطقی 0 روشن می‌شوند.

[۱] Mosfet



شکل ۹.۸

touch.c

```
1  //             X1      Y2      X2      Y1
2  //            -----   -----   -----   -----
3  // LPC2148    P0.22   P0.21   P0.28   P0.29
4
5  #include "touchscreen.h"
6
7  // Initialise touch screen
8  void tsInit(void)
9  {
10   // Enable power for ADC1
11   SCB_PCONP |= SCB_PCONP_PCAD1;
12
13   // Make sure P0.28 and 0.29 are set to GPIO (just in case)
14   // They are used to toggle between GND and 'floating'
15   PCB_PINSEL1 &= ~PCB_PINSEL1_P028_MASK;
16   PCB_PINSEL1 |= PCB_PINSEL1_P028_GPIO;
17   PCB_PINSEL1 &= ~PCB_PINSEL1_P029_MASK;
18   PCB_PINSEL1 |= PCB_PINSEL1_P029_GPIO;
19
20   // Initialise ADC converters (using 12MHz PCLK)
21   AD1_CR = AD_CR_CLKS10                    // 10-bit precision
22          | AD_CR_PDN                   // Exit power-down mode
```

```
          | ((3 - 1) << AD_CR_CLKDIVSHIFT) // 4MHz Clock
          | AD_CR_SEL6 | AD_CR_SEL7;       // Use channel 6 and 7
}

// Read the current X position using ADC1.6
unsigned int tsReadX(void)
{
  // Set P0.21 as AD1.6
  PCB_PINSEL1 &= ~PCB_PINSEL1_P021_MASK;
  PCB_PINSEL1 |= PCB_PINSEL1_P021_AD16;

  // Provide 3.3v with P0.22
  PCB_PINSEL1 &= ~PCB_PINSEL1_P022_MASK;
  PCB_PINSEL1 |= PCB_PINSEL1_P022_GPIO;
  GPIO0_IODIR |= (1 << 22);   // Set 0.22 as output
  GPIO0_IOSET |= (1 << 22);   // Set 0.22 high (provides 3.3v)

  // Provide GND with P0.28
  GPIO0_IODIR |= (1 << 28);   // Set 0.28 as output
  GPIO0_IOCLR |= (1 << 28);   // Set 0.28 low (provides GND)

  // Set 0.29 'floating' (by setting it to input)
  GPIO0_IODIR &= ~(1 << 29);   // Set 0.29 as input

  // Start AD conversion
  AD1_CR &= ~(AD_CR_START_MASK | AD_CR_SELMASK);
  AD1_CR |=  (AD_CR_START_NONE | AD_CR_SEL6);
  AD1_CR |=   AD_CR_START_NOW;

  // Wait for the conversion to complete
  while (!(AD1_DR6 & AD_DR_DONE))
    ;

  // Return the processed results
  return ((AD1_DR6 & AD_DR_RESULTMASK)
          >> AD_DR_RESULTSHIFT);
}

// Read the current Y position using AD1.7
unsigned int tsReadY(void)
{
  // Set P0.22 as AD1.7
  PCB_PINSEL1 &= ~PCB_PINSEL1_P022_MASK;
  PCB_PINSEL1 |= PCB_PINSEL1_P022_AD17;

  // Provide 3.3v with P0.21
  PCB_PINSEL1 &= ~PCB_PINSEL1_P021_MASK;
  PCB_PINSEL1 |= PCB_PINSEL1_P021_GPIO;
  GPIO0_IODIR |= (1 << 21);   // Set pin to output
  GPIO0_IOSET |= (1 << 21);   // Set pin high (providing 3.3V)

  // Provide GND with P0.29
  GPIO0_IODIR |= (1 << 29);   // Set 0.29 as output
  GPIO0_IOCLR |= (1 << 29);   // Set 0.29 low (provides GND)

```

```
78    // Set 0.28 'floating' (by setting it to input)
79    GPIO0_IODIR &= ~(1 << 28);    // Set 0.28 as input
80
81    // Start AD conversion
82    AD1_CR &= ~(AD_CR_START_MASK | AD_CR_SELMASK);
83    AD1_CR |=  (AD_CR_START_NONE | AD_CR_SEL7);
84    AD1_CR |=   AD_CR_START_NOW;
85
86    // Wait for the conversion to complete
87    while (!(AD1_DR7 & AD_DR_DONE))
88      ;
89
90    // Return the processed results
91    return ((AD1_DR7 & AD_DR_RESULTMASK)
92            >> AD_DR_RESULTSHIFT);
93  }
94
```

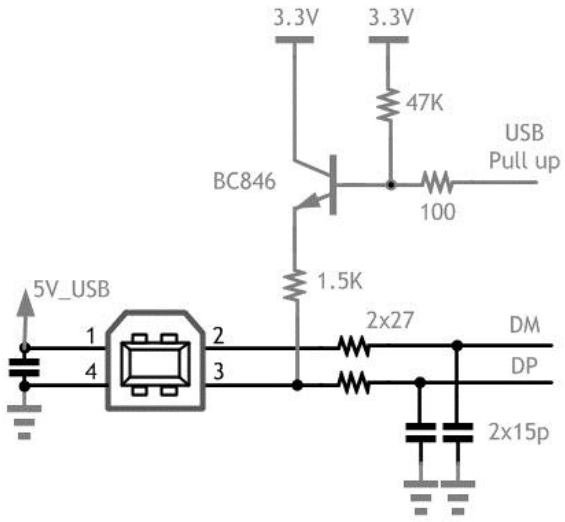
۹.۳ ارتباط با USB

USB مجموعه‌ای از مشخصات برای برقراری ارتباطات بین دستگاه‌ها و یک کنترل-کننده میزبان است. این استاندارد در شرکت اینتل و توسط یکی از مهندسان هندی آن به نام Ajay Bhatt توسعه یافت. این پورت، برای اتصال موس، کیبورد، دوربین‌های دیجیتال، پرینتر، درایوهای هارد خارجی و ... به کامپیوتر، استفاده می‌شود. USB در بسیاری از این دستگاهها، به تنها راه ارتباطی تبدیل شده‌اند. برخلاف بسیاری از استانداردهای ارتباطی قدیمی‌تر همانند RS-232 و پورت موازی، این پورت توانایی تامین تغذیه به دستگاه‌های متصل به آن را دارد.

توسعه تجاری این پورت در سال ۱۹۹۴ توسط شرکت‌های Compaq، DEC، IBM، Intel و تعدادی دیگر انجام گرفت. این پورت در ۴ نسخه معروف 1.0 با سرعت 1.5Mbits/S، 1.1 با سرعت 12Mbits/S، 2.0 با سرعت 480Mbits/S و اخیرا نسخه 3.0 با سرعت حداکثر تا 5Gbit/S ارایه شده است.

بسیاری از میکروکنترولرهای ARM دارای راه‌انداز USB داخلی هستند که بسیاری از عملیات ارتباطی را به صورت سخت‌افزاری انجام می‌دهند. این کنترل‌کننده‌ها شامل کنترل‌کننده‌های Device، Host و یا OTG هستند. این راه‌اندازها از سرعت 1.5Mbps و 12Mbps و به ندرت از سرعت 480Mbps پشتیبانی می‌کنند.

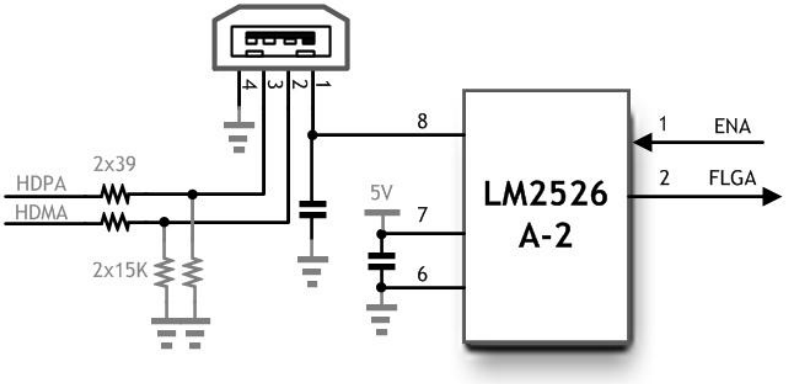
برای ارتباط میکروکنترولرهای ARM با USB Device عموما از مدار ساده زیر استفاده می‌کنیم:



شکل ۹.۹ – USB Device

خط USB Pull up برای وصل نمودن مقاومت 1.5K (که برای تشخیص اتصال دستگاه توسط میزبان ضروری است)، توسط نرم‌افزار استفاده می‌شود. با "1" نمودن آن، مقاومت 1.5K به صورت Pull Up بر روی خط +D قرار می‌گیرد.

برای ارتباط میکروکنترولرهای ARM با USB Host از مدار زیر استفاده می‌کنیم:



شکل ۹.۱۰ – USB Host

LM2526 در این مدار، نقش محدودکننده جریان را بر عهده دارد. در این کاربردها، مدار میزبان به دستگاه مورد نظر تغذیه ۵ ولتی با محدودیت جریان 500mA را اعمال می‌کند. پایه‌های ENA و FLGA به ترتیب برای فعال‌سازی اتصال تغذیه و آگاهی از وضعیت محدودیت جریان استفاده می‌شوند.

مثال‌های مختلفی از اتصال USB به کامپیوتر در DVD برنامه آمده است.

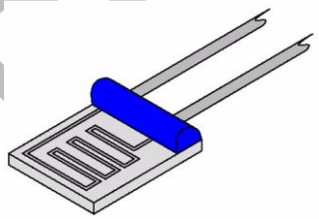
۹.۴ مدارهای آنالوگ و سنسورها

برای ارتباط مستقیم با مدارهای آنالوگ و یا سنسورها، باید از ADC داخلی میکروکنترولر استفاده کرد. اگر سنسور موردنظر دارای مدار دیجیتال تبدیل کننده باشد، می‌توان از واسط ارایه شده آن استفاده کرد. به عنوان مثال بعضی سنسورها دارای رابط‌هایی چون I2C، SPI و ... هستند. گاهی ناگزیر هستیم برای دستیابی به دقت بیشتر، از ADC های خارجی ۱۶ بیت یا بیشتر، استفاده کنیم.

۹.۴.۱ سنسورهای RTD

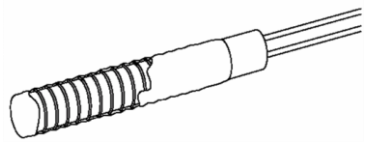
سنسورهای مقاومتی حرارتی RTD[۱]، نسبت تغییرات دقیقی از **مقاومت نسبت به حرارت** را از خود به نمایش می‌گذارند. جنس ماده تشکیل دهنده‌ی آن‌ها اکثرا از پلاتینیوم بوده و جایگزین خوبی برای حرارت‌سنج‌های ترموکوپلی در دماهای زیر 700°C به حساب می‌آیند.
این سنسورهای در ۴ نوع پایه ساخته می‌شوند:

- مقاومت‌های کربنی. بسیار پر کاربرد بوده و ارزان هستند. این قطعات در دماهای خیلی پایین از دقت خوبی برخوردار بوده و از اثر هیسترزیس Strain gauge در آن‌ها خبری نیست؛
- حرارت سنج‌های لایه‌ای. لایه ای از پلاتینیوم بر روی یک زیر بنا دارند. این لایه بسیار نازک بوده و ضخامتش حدود 1µm است. مزیت این گروه، قیمت پایین و پاسخ‌گویی سریعش می‌باشد.



شکل ۹.۱۱

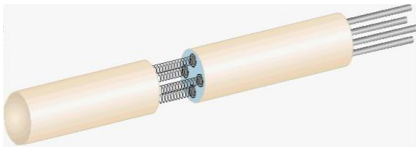
- حرارت سنج‌های سیم پیچی شده. دارای دقت بالاتر در محدوده دمایی وسیع‌تری هستند.



شکل ۹.۱۲

[۱] Resistive Thermal Device

- اجزای دارای کویل. جای انواع سیم پیچی شده را در صنعت گرفته‌اند. این طرح، از افزایش اندازه‌ی سیم‌پیچ ها در مکان خود استفاده می‌کنند و در دراز مدت، دارای استحکام بیشتری هستند.



شکل ۹.۱۳

بیشتر قطعاتی که در صنعت استفاده می‌شوند، دارای مقاومت 100Ω در دمای $0°C$ می‌باشند و با نام سنسورهای PT100 (PT نمادی برای پلاتینیوم است) شناخته می‌شوند. حساسیت نمونه‌های استاندارد این سنسورها، $0.385\ \Omega\ /°C$ است. با این وجود، سنسورهایی با حساسیت‌های $0.375\ \Omega\ /°C$ و $0.392\ \Omega\ /°C$ و ... نیز وجود دارند.

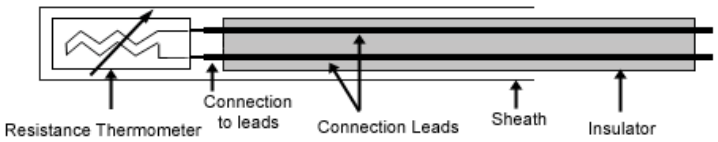
سنسورهای RTD در صنایع مختلف، جایگزین خوبی برای ترموکوپل‌ها بوده و دارای دقت بیشتر، پایداری و تکرارپذیری بالاتری هستند. بر خلاف ترموکوپل‌ها، RTD ها از تغییرات مقدار مقاومت نسبت به حرارت استفاده می‌کنند. برای راه‌اندازی آن‌ها نیاز به منبع ولتاژ بوده و در حالت ایده‌آل، نسبت مقاومت به حرارت، در آن‌ها خطی است.

۹.۴.۱.۱ چه موقع از RTD و یا ترموکوپل استفاده کنیم؟

نفاوت‌های دو راه‌حل صنعتی اندازه‌گیری دما را در زیر بررسی می‌کنیم:

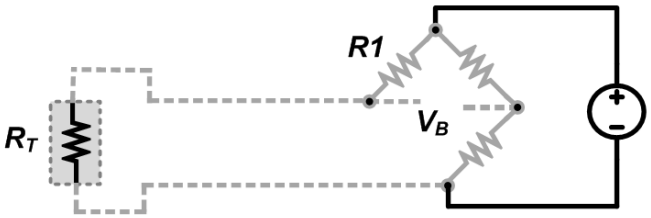
- اگر دما قابل اندازه‌گیری فرآیند مورد استفاده بین $-200°C$ تا $500°C$ باشد، استفاده از یک RTD صنعتی، راه‌حلی مناسب است. در مقابل، استفاده از ترموکوپل‌ها برای دماهای تا $2000°C$ مناسب است. پس، تنها راه‌حل اندازه‌گیری دمای تماسی برای دماهای بیش از $500°C$ هستند؛
- اگر زمان پاسخگویی کمتر از یک ثانیه مورد نیاز باشد، استفاده از ترموکوپل انتخابی شایسته است؛
- اندازه‌ی سنسور مورد استفاده نیز در بعضی موارد تعیین‌کننده است. قطر RTD ها عموماً بین $3.17mm$ تا $6.35mm$ هستند. اما قطر غلاف یک ترموکوپل می‌تواند کمتر از $1.6mm$ هم باشد؛
- تلرانس اندازه‌گیری ترموکوپل $2°C$ بوده و در کاربردهایی که نیاز به تکرارپذیری زیاد دارند، استفاده نمی‌شوند. در عوض، RTD ها دارای دقت بالاتری بوده و بدون تغییر مشخصات، تا چندین سال کار می‌کنند.

ساختار داخلی نمونه‌ای از این سنسورها را دراین‌جا می‌بینیم.

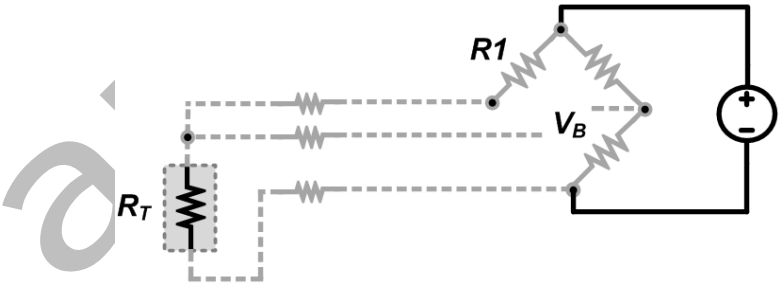


شکل ۹.۱۴

برای سیم‌بندی و اندازه گیری RTD ها، از اتصالاتی مشابه **پل وتستون** استفاده می‌شود. در اشکال زیر روش کار را می‌بینیم. در روش اول، ساده‌ترین مدل آمده است که در آن اثر مقاومت سیم‌ها مزاحم است. استفاده از شکل دوم برای اندازه‌گیری ۳- سیمه روشی بهتر و دقیق‌تر است. در این شکل مقاومت سیم‌ها نیز مدل شده‌اند. در این مدراها R1 را برابر مقدار اهمی اولیه RTD در پایین‌ترین دمای فرآیند (مثلا دمای محیط $27°C$) انتخاب می‌کنیم. در این دما، مقدار V_B برابر صفر ولت است.



شکل ۹.۱۵ – اندازه‌گیری ۲- سیمه



شکل ۹.۱۶ – اندازه‌گیری ۳- سیمه

فصل دهم

ارتباط با حافظه ها

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ انواع تراشه‌های فرّار و غیرفرّار ذخیره کننده داده کدام‌اند؟
- ✓ چگونه میکروکنترولرها را به تراشه‌های RAM استاتیک و دینامیک متصل کنیم؟
- ✓ طریقه ارتباط میکروکنترولرها با تراشه‌های Flash و یا E2PROM رایج کدام است؟
- ✓ کدام حافظه را برای مدار خود انتخاب کنیم ؟

میکروکنترولرهای ARM ، اغلب دارای مقادیر کافی حافظه‌ی فلش و RAM استاتیک داخلی برای ذخیره برنامه‌ها و داده‌های سیستم‌های تعبیه شده‌ی معمولی هستند. این تراشه‌ها عموما دارای E^2PROM داخلی نیستند. مواردی که نیاز به ذخیره‌سازی بر روی یک حافظه‌ی غیرفرّار داریم، مجبور به استفاده از تراشه‌های Flash و یا E2PROM خارجی هستیم (برای این‌که تعداد کمتری از پایه‌های میکروکنترولر اشغال شوند، این تراشه‌ها عموما دارای رابطهای سریال هستند). در مواردی که نیاز به حافظه‌های RAM بیشتری داریم نیز باید از تراشه‌های خارجی استفاده کنیم. ارتباط با تراشه های حافظه‌ی خارجی در مواردی که میکروکنترولر دارای کنترل کننده‌ی حافظه‌ی خارجی هستند، کار آسانی است. به عنوان مثال،LPC2478 دارای کنترل‌کننده‌ی SDRAM خارجی است و بعضی خانواده‌های STM32F103 دارای FSMC برای کنترل و اتصال انواع حافظه‌ی خارجی هستند.

میکروکنترولرهای ARM9، ARM11 و ARM Cortex-A ، دارای توان پردازشی بالاتری بوده و با سرعت بالاتری کار می‌کنند. این موضوع به معنای اعلام نیاز به واسط حافظه‌ی سریع است. این میکروکنترولرها عموما حافظه فلش داخلی ندارند (دلیل آن به وضوح سرعت پایین حافظه‌های فلش است). نبود حافظه‌ی RAM داخلی نیز چندان تعجب‌آور نیست. هر چند، اغلب دارای مقادیر کمی RAM و یا حافظه‌ی نهان هستند. در بعضی موارد نادر، حضور فلش نیز دیده می‌شود (همانند AT91SAM9Xxxx) که البته استقبال چندانی از آن‌ها به‌عمل نیامده است.

با استناد به دلایل گفته شده، اتصال حافظه های خارجی سریع و حجیم به این میکروکنترولرها مرسوم شده است. حافظه‌های SDRAM، DDR RAM سریع، از جمله تراشه‌های RAM مرسوم و حافظه‌های NAND Flash حجیم (اما کند) از جمله تراشه‌های Flash رایجی هستند که در این مدارها استفاده می‌شوند. NAND ها برای نگهداری فایل‌های image سیستم عامل، سیستم فایل‌ها (مثلا سیستم فایل

jffs2 در Linux که دارای حجم 10Mbyte یا بیشتر است)، استفاده می‌شود. برای کار با این حافظه های حجیمِ کُند، میکروکنترولر و بوت لودر روی آن، کد داخل NAND را بر روی SDRAM کپی کرده و سپس استفاده می‌کنند.

حافظه‌های RAM ، به دو دسته کلی SRAM و DRAM تقسیم می‌شوند. DRAM ها دارای اندازه‌های بزرگ‌تر بوده و از SRAM ها کندتر هستند. کار کردن با آن‌ها نیز از SRAM های استاندارد سخت‌تر است. DRAM ها برای افزایش سرعت از تکنولوژی خط لوله بهره می‌برند. نمونه‌ی آن را در SDRAM ها و DDR RAM ها می‌بینیم.

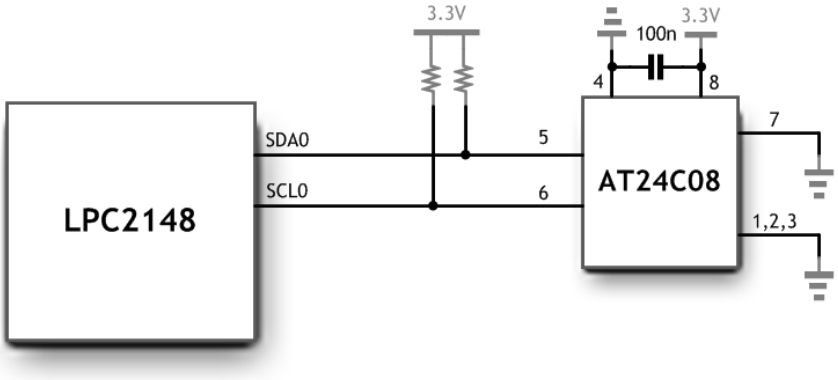
۱۰.۱ تراشه‌های E2PROM و Flash

تراشه‌های خانواده‌ی AT24Cxx از جمله‌ی E2PROM های پر کاربرد با واسط I2C هستند. شمای پایه‌های آن به همراه توضیح کارکرد پایه‌ها را در شکل زیر می‌بینید.

کارکرد	اسم پایه
ورودی‌های آدرس	A0 to A2
داده سریال	SDA
ورودی سریال کلاک	SCL
محافظت از نوشتن	WP
بدون اتصال	NC

شکل ۱۰.۱

برای راه‌اندازی این تراشه، پایه‌های A0 تا A2 را برای انتخاب آدرس تراشه استفاده می‌کنیم. مثلا آن‌ها را در این‌جا به زمین وصل می‌کنیم (آدرس = 0). پایه WP نیز باید به زمین وصل باشد.



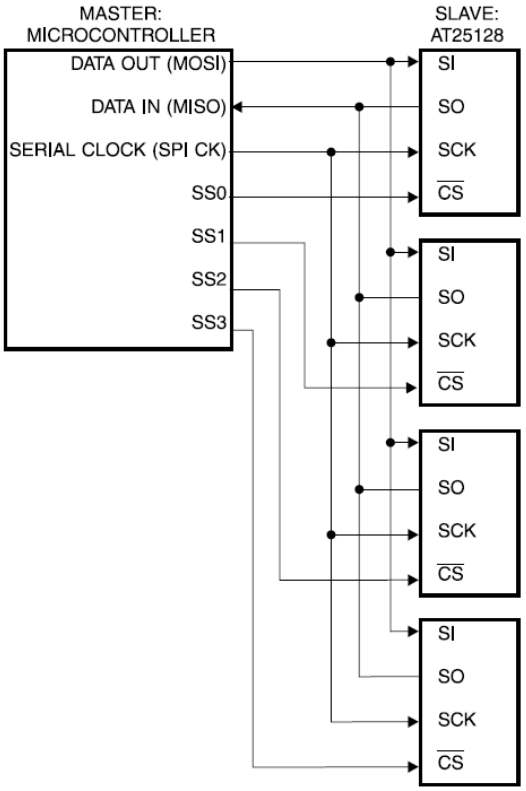
شکل ۱۰.۲

در کنار حافظه‌های نام‌برده، حافظه‌های خانواده‌ی AT25Cxx قرار دارند که دارای رابط SPI هستند.

شکل ۱۰.۳

کارکرد	اسم پایه
انتخاب تراشه	#CS
ورودی داده سریال	SI
ورودی سریال کلاک	SCK
خروجی داده سریال	SO
محافظت از نوشتن	WP
تعلیق ورودی سریال	#Hold

ارتباط با این تراشه‌ها، همانند سایر ارتباطات SPI استاندارد صورت می‌گیرد. می‌توان چندین قطعه‌ی مختلف را به یک میکروکنترولر و یک واسط SPI ، متصل نمود.



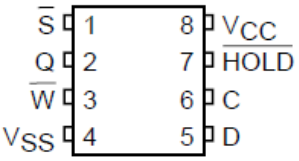
شکل ۱۰.۴

انواع دیگر E2PROM پرکاربرد، حافظه‌های خانواده‌ی 93Cxx هستند.

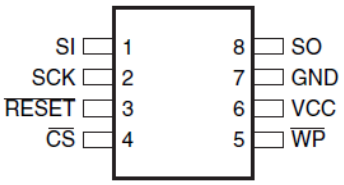
شکل ۱۰.۵

این حافظه‌ها نیز دارای رابطی ۳- سیمه مشابه SPI هستند (رابط Microwire که زیر مجموعه از SPI است). ORG برای سازمان‌دهی آرایه داخلی حافظه، استفاده می‌شود.

حافظه های فلش و دیتافلش[1] سریال در مدارهای شامل ARM ، به فراوانی دیده می‌شوند. تراشه‌های AT45DBxxx شرکت Atmel و M25P64 شرکت ST و ... از جمله حافظه‌های با حجم متوسط و با رابط کاربری SPI هستند.



شکل ۱۰.۶ – M25P64



شکل ۱۰.۷ – AT45DB161

نمونه کدهای ارتباطی را می توانید در DVD همراه پیدا کنید.

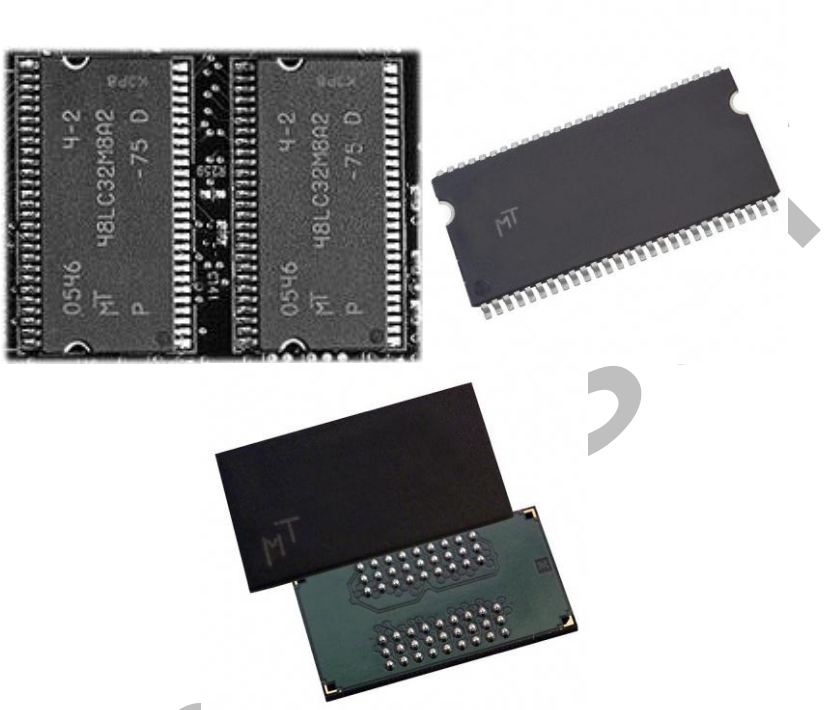
۱۰.۲ حافظه‌های SDRAM

SDRAM ، از انواع حافظه‌های پویا[2] بوده که با گذرگاه سیستم سنکرون هستند. DRAM های معمولی آسنکرون هستند یعنی به محض این‌که هر یک از ورودی‌های کنترلی تغییر کنند، حافظه‌ی مذکور پاسخ می‌دهد. SDRAM به انتظار کلاک سنکرون کننده می‌ماند و به محض رسیدن لبه‌ی این سیگنال کلاک، به ورودی‌های کنترلی پاسخ می‌دهد. این کلاک برای راه‌اندازی یک ماشین حالت داخلی[3] که دستورالعمل‌های ورودی به حافظه را در خط لوله قرار می‌دهد، استفاده می‌شود. این روش یک الگوی اجرایی بسیار پیچیده‌تری از یک DRAM آسنکرون ساده و با سرعت بیشتری به دست می‌دهد.
خط لوله در این‌جا دستورهای جدید را قبل از اتمام پردازش دستورهای قبلی می‌پذیرد. مثلاً، در دستور نوشتن خط لوله‌ای[4]، این دستور می‌تواند بلافاصله با دستور دیگری دنبال شود. این‌کار نیازی به ایجاد حالات انتظار[5] برای اتمام عملیات نوشتن داده بر روی آرایه‌ی حافظه نمی‌باشد. عملیات خواندن نیز به

---

[1] Data Flash
[2] Dynamic
[3] Finite State Machine-FSM
[4] Pipe Line Write
[5] Wait State

همین صورت دنبال می‌شود. بعد از گذر چند سیکل کلاک ماشین، داده مورد نظر بر روی خط داده آماده، خواندن است. این پارامتر را به عنوان Latency در حافظه‌های SDRAM می‌شناسیم.
SDRAM ها کاربرد فراوانی در رایانه‌ها دارند. نسل‌های بعدی SDRAM ها را با نام‌های DDR، سپس DDR2 و DDR3 می‌شناسیم.



شکل ۱۰.۸

۱۰.۲.۱ سیگنال‌های کنترلی SDRAM

تمامی سیگنال‌های زیر، با لایه‌ی بالارونده سیگنال کلاک هم‌زمان می‌شوند. تعداد ۶ سیگنال کنترلی وجود دارند که اکثراً از نوع باحالت فعال پایین[۱] هستند. این سیگنال‌ها عبارت‌اند از:

- Clock Enable-CKE. اگر این سیگنال در حالت low باشد، همانند این‌است که تراشه سیگنال کلاکی دریافت نمی‌کند. هیچ دستورالعملی اجرا نشده و سیگنال‌های دیگر نادیده گرفته می‌شوند؛
- Chip Select -/CS. هنگامی که این سیگنال High باشد، تراشه تمامی سیگنال‌ها (به جز CKE) را نادیده می‌گیرد و همانند این‌است که یک دستور NOP دریافت شده است؛

[۱] Active Low

- Data Mask-DQM. هنگام High بودن این سیگنال از انجام عملیات ورودی- خروجی داده ممانعت به عمل می‌آید. در هنگام نوشتن داده، داده‌ی واقعی بر روی DRAM نوشته نمی‌شود. اگر دو سیکل کاری قبل از عملیات خواندن، فعال شود، داده خوانده شده از تراشه خارج نمی‌شود. برای حافظه‌ها با پهنای گذرگاه داده X16 یا بیشتر، به ازای هر ۸ بیت یک سیگنال DQM داریم؛
- /RAS-Row Address Strobe. این سیگنال به همراه /CAS و /WE یکی از ۸ دستورالعمل را انتخاب می‌کنند؛
- /CAS-Column Address Strobe. این سیگنال به همراه /RAS و /WE یکی از ۸ دستورالعمل را انتخاب می‌کنند؛
- /WE-Write Enable.به همراه /CAS و /RAS ، یکی از ۸ دستورالعمل را انتخاب می‌کنند. این سیگنال دستورهای شبه نوشتن را از دستورهای شبه خواندنی جدا می‌کند.

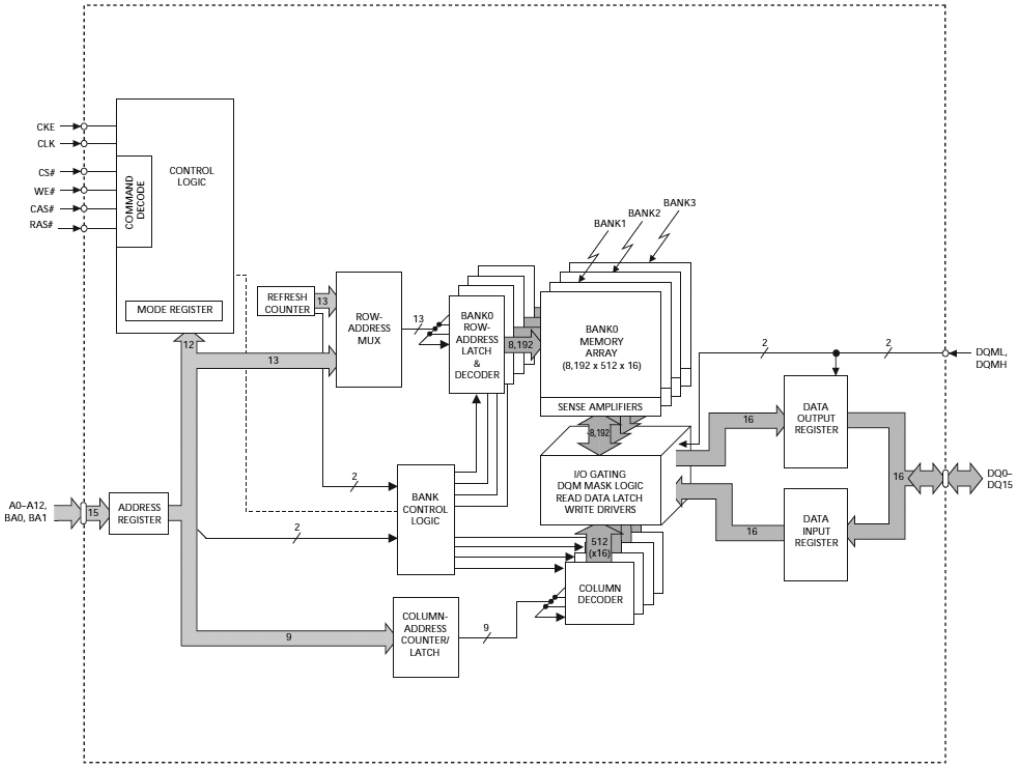
حافظه‌های SDRAM دارای ۲ تا ۴ بانک داده مستقل داخلی هستند. انتخاب کننده‌های بانک (یعنی BAC و BA1) مشخص می‌کنند که دستور ورودی به کدام بانک داده هدایت شود.

بسیاری از دستورها از آدرس ورودی برای مشخص کردن مکان داده‌ها استفاده می‌کنند. بعضی دستورها از آدرس استفاده نمی‌کنند و یا از آدرس ستون و یا سیگنال A10 استفاده می‌کنند. مجموعه‌ی دستورها SDRAM را در جدول 1-SDRAM می‌بینیم.

دستور	An	A10	BAn	/WE	/CAS	/RAS	/CS
ممانعت از اجرای دستور (**NOP**)	x	x	x	x	x	x	H
NOP	x	x	x	H	H	H	L
Burst Terminate – عملیات خواندن و یا نوشتن در حال کار را باز می ایستاند.	x	x	x	L	H	H	L
Read – از سطر فعال به طور پی‌وسته داده ها را می خواند.	column	L	bank	H	L	H	L
Read with auto precharge – همانند سطر بالا. در انتهای خواندن، حافظه را **precharge** می کند.	column	H	bank	H	L	H	L
Write – بر روی سطر فعال به طور پی‌وسته داده ها را می نویسد.	column	L	bank	L	L	H	L
Write with auto precharge – همانند	column	H	bank	L	L	H	L

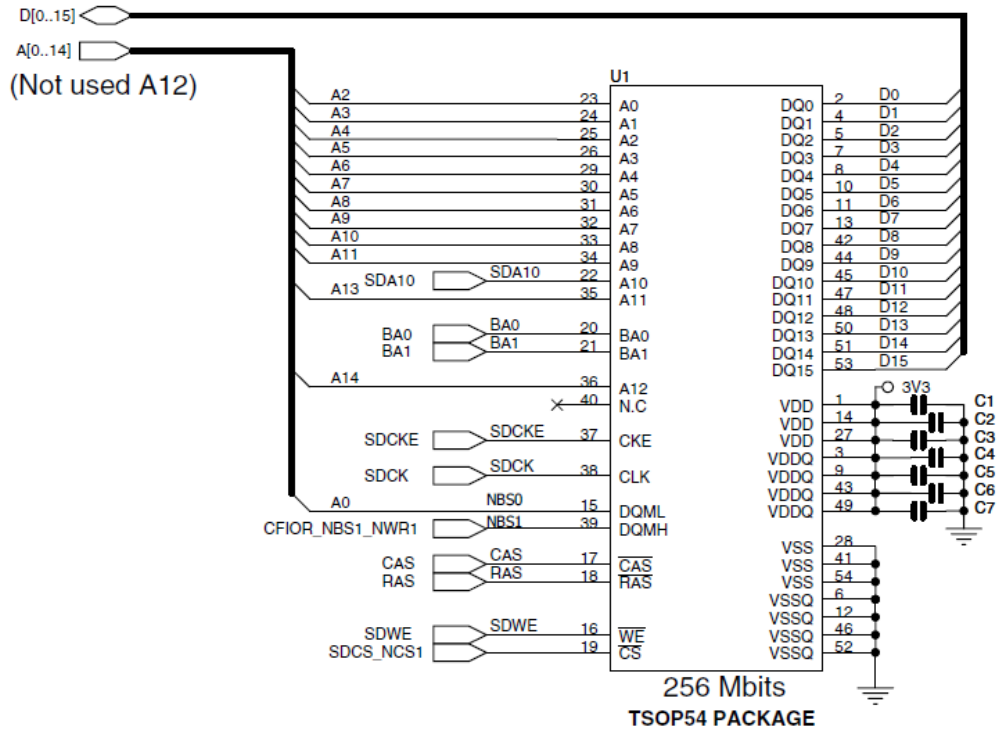
سطر بالا. در انتهای نوشتن، حافظه را **precharge** می کند.							
Active (Activate) : یک سطر را برای خواندن و یا نوشتن باز می کند.	row		bank	H	H	L	L
Precharge : سطر فعلی بانک انتخاب شده را غیر فعال می کند.	x	L	bank	L	H	L	L
Precharge all : سطر فعلی تمامی بانکها را غیر فعال می کند.	x	H	x	L	H	L	L
Auto refresh : یک سطر از هر بانک را توسط یک شمارنده داخلی **refresh** می کند.	x	x	x	H	L	L	L
Load Mode Register : **A0** تا **A9** برای پیکره بندی **DRAM** به داخل تراشه بارگیری می شوند.	mode		0 0	L	L	L	L

در شکل ۱۰.۹ نمای داخلی یک حافظه‌ی 16M×16 SDRAM را می‌بینیم. پهنای داده این حافظه ۱۶ بیت است.

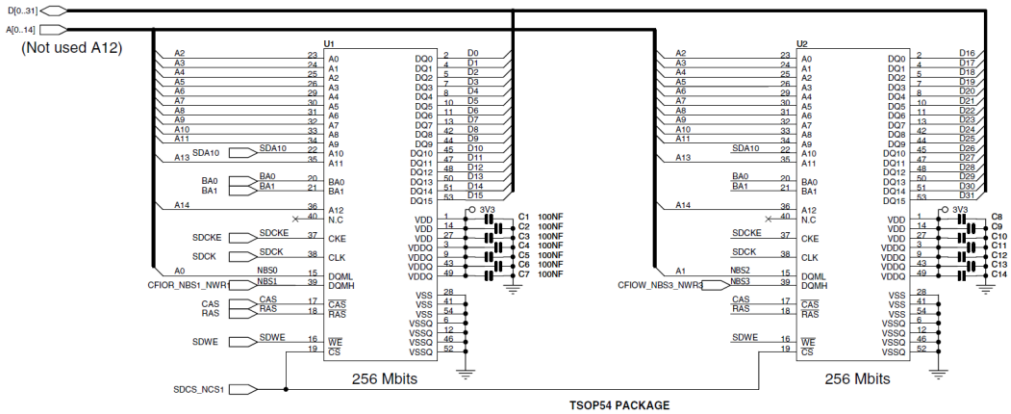


شکل ۱۰.۹

در شکل‌های ۱۰.۱۰ و ۱۰.۱۱، به ترتیب روشهای اتصال حافظه‌ها به یک میکروکنترولر ARM9 (در اینجا AT91SAM9263) در حالت‌های ۱۶ بیت و ۳۲ بیتی هستند.



شکل ۱۰.۱۰



شکل ۱۰.۱۱

برای پیکره‌بندی و انجام تنظیمات اولیه‌ی SDRAM ، از کد نمونه‌ی زیر می‌توان استفاده کرد.

```
1   void BOARD_ConfigureSdram48MHz(unsigned char busWidth)
```

```
{
    volatile unsigned int i;
    static const Pin pinsSdram = PINS_SDRAM;
    volatile unsigned int *pSdram = (unsigned int *)
AT91C_EBI_SDRAM;
    unsigned short sdrc_dbw = 0;
    unsigned int tmp = 0;

    switch (busWidth) {
        case 16:
            sdrc_dbw = AT91C_SDRAMC_DBW_16_BITS;
            break;

        case 32:
        default:
            sdrc_dbw = AT91C_SDRAMC_DBW_32_BITS;
            break;

    }

    // Enable corresponding PIOs
    PIO_Configure(&pinsSdram, 1);

    // Enable EBI chip select for the SDRAM
    tmp = READ(AT91C_BASE_MATRIX, MATRIX_EBI) |
AT91C_MATRIX_CS1A_SDRAMC;
    AT91C_BASE_MATRIX->MATRIX_EBI = tmp;


    // CFG Control Register
    AT91C_BASE_SDRAMC->SDRAMC_CR = AT91C_SDRAMC_NC_9
                                   | AT91C_SDRAMC_NR_13
                                   | AT91C_SDRAMC_CAS_2
                                   |
AT91C_SDRAMC_NB_4_BANKS
                                   | sdrc_dbw
                                   | AT91C_SDRAMC_TWR_1
                                   | AT91C_SDRAMC_TRC_4
                                   | AT91C_SDRAMC_TRP_1
                                   | AT91C_SDRAMC_TRCD_1
                                   | AT91C_SDRAMC_TRAS_2
                                   | AT91C_SDRAMC_TXSR_3);

    for (i = 0; i < 1000; i++);

    AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_NOP_CMD; //
Perform NOP
    pSdram[0] = 0x00000000;

    AT91C_BASE_SDRAMC->SDRAMC_MR =
AT91C_SDRAMC_MODE_PRCGALL_CMD; // Set PRCHG AL
    pSdram[0] = 0x00000000;                    // Perform PRCHG

    for (i = 0; i < 10000; i++);

```

```
57      AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_RFSH_CMD;
58  // Set 1st CBR
59      pSdram[1] = 0x00000001;                     // Perform CBR
60
61      AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_RFSH_CMD;
62  // Set 2 CBR
63      pSdram[2] = 0x00000002;       // Perform CBR
64
65      AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_RFSH_CMD;
66  // Set 3 CBR
67      pSdram[3] = 0x00000003;                     // Perform CBR
68
69      AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_RFSH_CMD;
70  // Set 4 CBR
71      pSdram[4] = 0x00000004;       // Perform CBR
72
73      AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_RFSH_CMD;
74  // Set 5 CBR
75      pSdram[5] = 0x00000005;       // Perform CBR
76
77      AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_RFSH_CMD;
78  // Set 6 CBR
79      pSdram[6] = 0x00000006;                     // Perform CBR
80
81      AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_RFSH_CMD;
82  // Set 7 CBR
83      pSdram[7] = 0x00000007;                     // Perform CBR
84
85      AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_RFSH_CMD;
86  // Set 8 CBR
87      pSdram[8] = 0x00000008;       // Perform CBR

        AT91C_BASE_SDRAMC->SDRAMC_MR = AT91C_SDRAMC_MODE_LMR_CMD;
    // Set LMR operation
        pSdram[9] = 0xcafedede;       // Perform LMR burst=1, lat=2

        AT91C_BASE_SDRAMC->SDRAMC_TR = (48000000 * 7) / 1000000);
    // Set Refresh Timer

        AT91C_BASE_SDRAMC->SDRAMC_MR =
    AT91C_SDRAMC_MODE_NORMAL_CMD);  // Set Normal mode
        pSdram[0] = 0x00000000;       // Perform Normal mode
    }
```

کد SDRAM-1

Bus Width ، پهنای گذرگاه است که می‌تواند ۱۶ و یا ۳۲ باشد.

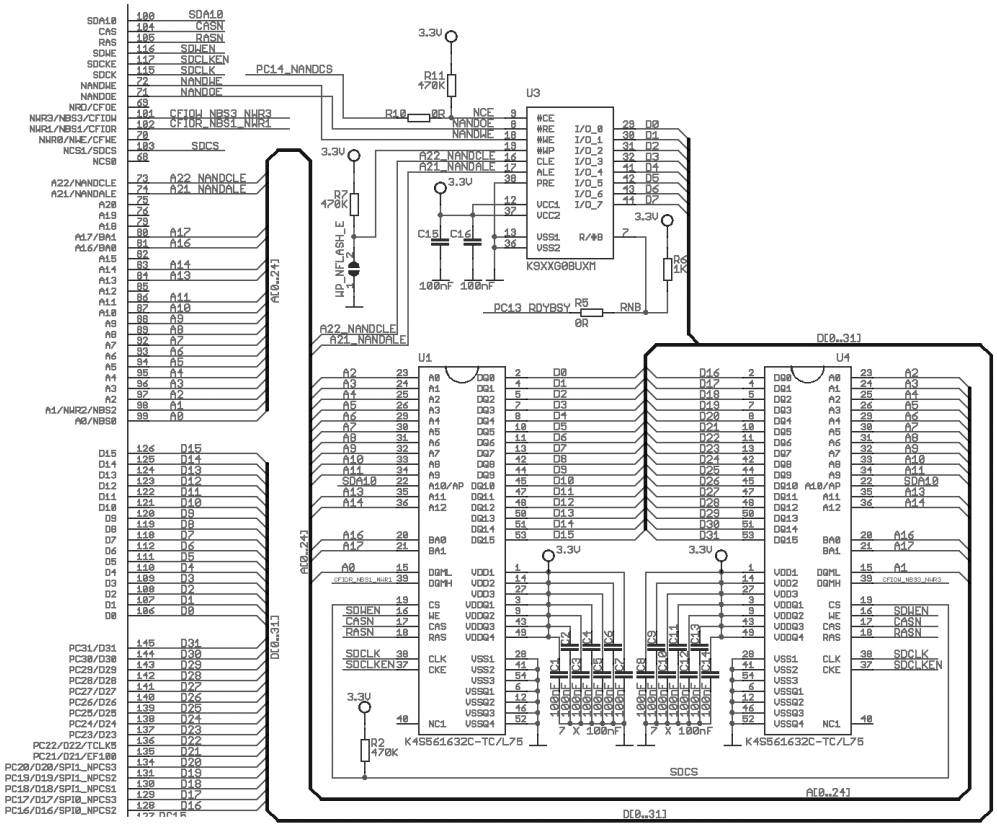
ازمشهورترین تولید کنندگان این تراشه‌ها، شرکت‌های Micro، SAMSUNG، Hynix و... هستند که طیف وسیعی ازتولیدات با پهنای گذرگاه متفاوت وحجم‌های مختلف داده‌ها را ارایه می‌دهند.

LPC2478 از معدود میکروکنترولرهـای ARM7 بـوده کـه دارای کنتـرل کننـده‌ی SDRAM داخلـی است. بیشتر میکروکنترولرهای ARM9 ، از این نوع کنترل کننده بهره می‌برند. برای آشنایی با نحـوه‌ی نام‌گذاری حافظه‌های SDRAM ، به برگه‌ی اطلاعات آن‌ها مراجعه کنید.

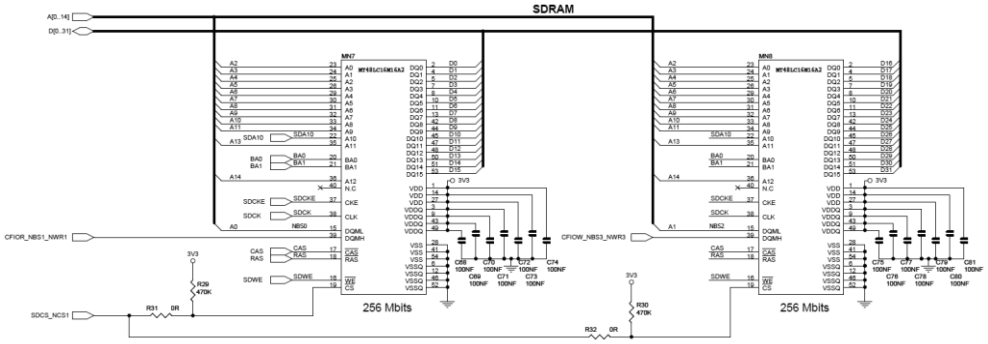
مراجعه

در DVD همراه کتاب، تعدادی از برگه‌های داده SDRAM های گوناگون آورده شده است.

شیوه‌ی اتصال ایـن حافظـه و حافظـه NAND در مـدارات کـاربردی زیـر آمـده‌انـد. در ایـن اشـکال از پردازنده‌های مختلف و حافظه‌های SDRAM و NAND سازندگان متفاوتی استفاده شده است.



شکل ۱۰.۱۲



شکل ۱۰.۱۳

فصل یازدهم
کارت‌های حافظه

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ آشنایی با انواع کارت حافظه.

۱۱.۱ انواع کارت حافظه

کارت حافظه (آن را با نام کارت حافظه‌ی فلش می‌شناسیم)، قطعه‌ای الکترونیکی برای ذخیره داده است. اولین کارت‌های حافظه توسط شرکت در دهه 1980 اختراع شد. این وسیله را به عنوان حافظه‌ای می‌شناسند که با قطع کردن ولتاژ تغذیه‌اش اطلاعات ذخیره شده بر روی خود را از دست نمی‌دهد. توانایی حفظ اطلاعات هنگام قطع شدن منبع تغذیه، کلیدی‌ترین نکته در مورد این حافظه‌هاست.

به عنوان مثال، دوربین‌های دیجیتال: عکس‌های خود را بر روی کارت‌های حافظه ذخیره می‌کنند. این عکس‌ها بعد از ذخیره سازی بر روی کارت و با قطع تغذیه پاک نمی‌شوند.

کارت‌های حافظه به فراوانی دستگاه‌های الکترونیکی و صنعتی استفاده می‌شوند. امروزه کاربردهایی از کارت حافظه را در دستگاه‌هایی همچون: کامپیوترهای شخصی، دوربین‌های دیجیتالی، تلفن‌های همراه، دوربین‌های فیلمبرداری، نوت‌بوک‌ها، دستگاه‌های مکان‌یابی GPS، PDA ها و پخش کننده‌های موسیقی دستی می‌بینیم.

در دستگاه‌های صنعتی همچون تجهیزات شبکه، سیستم‌های نظامی، دستگاه‌های مخابراتی، محصولات پزشکی و... نیز حضور فعال این حافظه‌ها مشاهده می‌شود.

کارت‌های حافظه براساس دو تکنولوژی پایه NAND و یا NOR ساخته می‌شوند.
فناوری NOR توانایی دسترسی به حافظه با سرعت بالا و با دسترسی تصادفی را به ما می‌دهد. در حافظه‌های NOR ، امکان دستیابی به **یک بایت** را نیز داریم. حافظه‌های NAND بعد از NOR اختراع شدند. امکان دسترسی به حافظه در تکنولوژی NAND به صورت ترتیبی و دسترسی به **صفحه** (Page) مجموعه‌ای از چندین بایت پشت سرهم. معمولاً ۲۵۶، ۵۱۲ و... بایت)، صورت می‌گیرد. برخلاف NOR ها در حافظه‌های NAND دسترسی به **یک بایت** را نداریم. کارت‌های حافظه‌ی گوناگونی در بازار تولیدات قطعات پرمصرف الکترونیکی موجودند. از آن جمله می‌توان به لیست زیر اشاره کرد:

- کارت Smart Media (SM)؛
- کارت Multi Media (MMC)؛
- کارت Compact Flash (CF)؛
- کارت Memory Stick (MS)؛
- کارت Micro Drive؛
- کارت Secure Digital (SD) .

مختصری از کارت‌های نام‌برده به همراه مشخصات آن‌ها را در ادامه بررسی می‌کنیم.

۱۱.۱.۱ کارت Smart Media

این کارت در سال 1995 توسط شرکت Toshiba ساخته شد. نام دیگری برای این کارت، "کارت دیسک فلاپی حالت جامد" (SSFDC) بود. این کارت، نازک‌ترین کارت از بین کارت‌های نام‌برده است. زیرا، محفظه‌ی آن پلاستیک و داخل آن فقط شامل یک تراشه‌ی فلش است. نمونه‌ای از این کارت را در شکل ۱۱.۱ می‌بینیم. ابعاد این کارت 45×37×0.76 mm و وزن آن تنها 1.8 gr است. کارت دارای کانکتور تک ردیفی ۲۲ پایه است.
دوربین‌های دیجیتال Olympus و Fuji عمده مصرف کنندگان این کارت‌ها بودند. در سال 2001 نزدیک 50% سهم بازار کارت‌های حافظه در اختیار این حافظه‌ها بود.

شکل ۱۱.۱

ظرفیت این کارت‌ها حداکثر 128 Mb و سرعت انتقال اطلاعات آن‌ها تقریباً 2 MB/S بود. با افزایش رزولوشن دوربین‌های دیجیتال و نیاز طرفیت حافظه‌ی مورد نیاز بیشتر، این کارت‌ها مقبولیت خود را از دست دادند. این حافظه‌ها در دونوع ولتاژ تغذیه 5 V و 3.3 V ساخته می‌شوند و از یک ID منحصربه‌فرد نیز برای محافظت اطلاعات داخل آن استفاده می‌شد. با کنار گذاشتن این محصول دو شرکت مذکور از کارت دیگر به نام xD استفاده کردند.

۱۱.۱.۲ کارت MMC

MMC ها، اولین بار در دهه‌ی 1970 توسط دو شرکت Intgemix و San Disk ساخته شدند. در ابتدا دستگاه‌های تلفن همراه و امروزه بسیاری از دستگاه‌ها از این کارت‌ها استفاده می‌کنند. کارت‌های MMC با کارت‌های SD سازگار هستند. بدین معنا که یک کارت MMC را می‌توان در سوکت یک کارت SD قرار داد. ابعاد MMC ها، 24×32×1.4 mm بوده و ۸ پایه دارد.

شکل ۱۱.۲

ولتاژ تغذیه این کارت، 3.3 V و سرعت انتقال داده در آن تقریباً 2.5 MB/S است. امروزه، کارت‌های MMC بهبود یافته و مشخصات بهتری را ارایه می‌کنند. ولتاژ تغذیه آن‌ها می‌توان بین 1.8 V و 3.3 V باشد و حداکثر ظرفیت آن‌ها به 4 GB نیز می‌رسد. نمونه‌ای از این کارت در شکل ۱۱.۲ آمده است.

۱۱.۱.۳ کارت‌های Compact Flash (CF)

CF ها، از دیگر محصولات San Disk بودند که در سال 1994 متولد شدند. این کارت‌ها، پرظرفیت‌ترین کارت‌ها با ظرفیت بین 2 MB تا 100 GB هستند. مورد استفاده‌ی این حافظه‌ها در دوربین‌های دیجیتال گران قیمت است. این کارت‌ها در دو نوع I و II (Type II و Type I) ساخته می‌شوند. نوع I از نوع II ضخامت کمتری داشته و جریان مصرفی آن در حدود 70 mA، در مقابل 500 mA نوع II می‌باشد. ولتاژ کاری آن‌ها 3.3 V و یا 5 V است. مزایای کارت‌های CF عبارت‌اند از:

- نسبت به کارت‌های دیگر در برابر ضربه و شرایط فیزیکی سخت مقاوم‌تر هستند؛
- عموماً ظرفیت‌های ذخیره سازی بیشتری دارند؛
- سرعت انتقال داده بالاتری دارند؛
- CF ها از استاندارد IDE/ATA پشتیبانی می‌کنند. پس می‌توانند جایگزین مناسبی برای هارددیسک‌ها باشند.

از معایب CF ، می‌توان به ابعاد بزرگ‌تر در مقایسه با کارت‌های دیگر و نداشتن محافظت سخت‌افزاری نوشتن بر روی حافظه (Write Protect)، اشاره کرد.
نمونه‌ای از این کارت در شکل ۱۱.۳ آمده است.

شکل ۱۱.۳

۱۱.۱.۴ کارت‌های Memory Stick (MS)

این کارت، اولین بار توسط شرکت Sony در سال 1998 متولد شد. اگرچه، اولین کارت MS دارای 128 MB حافظه‌ی فلش بود، اما امروزه بزرگ‌ترین حافظه‌ی MS به بیش از 16 GB نیز می‌رسد. شکل ۱۱.۴ نمونه‌ای از این کارت را نشان می‌دهد.
MS اولیه، دیگر تولید نمی‌شود و اکنون جای خود را به MS Pro، MS Duo و MS Pro-HG داده است. MS Pro در سال 2003 بر اثر همکاری مشترکی بین دو شرکت Sony و San Disk تولید شد. این کارت فضای ذخیره سازی بیشتر و سرعت انتقال داده بیشتری داشت. MS Duo حاصل تلاش برای کاهش اندازه و افزایش سرعت کارت‌های اولیه بود. MS Pro-HG نیز در سال 2006 براساس همکاری Sony و San Disk شکل گرفت. سرعت انتقال داده‌ها در این کارت، 60 MB/S است که از همه‌ی نسل‌های قبلی خود سریع‌تر بوده و تقریباً سه برابر سرعت کارت‌های MS Pro را دارد.

Speed Rating	Speed (MB/s)
6x	0.9
32x	4.8
40x	6.0
66x	10.0
100x	15.0
133x	20.0
150x	22.5
200x	30.0
266x	40.0
300x	45.0

شکل ۱۱.۴

۱۱.۱.۵ کارت‌های Micro drive

Micro drive ، اساساً یک دیسک سخت است که در بسته‌بندی کارت CF نوع II جای داده شده است. هرچند اندازه‌ی این کارت‌ها با اندازه‌ی کارت‌های CF یکسان است، اما مصرف توان آن به مراتب بیشتر از کارت‌های CF است. همین مسأله، کاربرد این کارت‌ها را در دستگاه‌های کم‌مصرف، محدود ساخته است. اندازه‌ی این کارت‌ها معمولاً از 8 GB بیشتر است. نمونه‌ای از این کارت در شکل ۱۱.۵ نشان داده شده است.

ابعاد یک کارت حافظه‌ی Micro drive ، برابر 42.8×36.4×5.0 mm است و وزن تقریبی آن به 16 gr می‌رسد. اولین Micro drive های تولیدی، ساخت IBM بودند و در سال 1999 ارایه شدند. این دو حافظه دارای حجم‌های 170 MB و 340 MB بودند.

با فاصله‌ی کمی، شرکت Hitachi در سال 2003 نمونه‌های بیش 2 GB آن را ارایه کرد. نمونه‌های با اندازه 8 GB و بیشتر نیز در سال 2008 با همکاری Hitachi و Seagate ساخته شدند.

مزیت کارت‌های Micro drive در این‌است که تعداد سیکل‌های نوشتن در این حافظه‌ها از انواع دیگر فراتر است. علاوه بر آن، اگر تغذیه‌ی کارت هنگام نوشتن بر روی کارت قطع شود، این مسأله در

کارت‌های Micro drive مشکل کمتری ایجاد می‌کند. یکی از مشکلات این کارت‌ها صدمه‌پذیر بودنشان است. اگر از ارتفاع 1 m یا بیشتر بر زمین بیافتند، قطعاً آسیب خواهند دید. سرعت انتقال داده در این حافظه‌ها 5 MB/S است که در مقایسه با بسیاری از کارت‌های امروزی، کندتر عمل می‌کنند. این کارت‌ها برای کار در ارتفاع زیاد نیز ساخته نشده‌اند.

شکل ۱۱.۵

شکل ۱۱.۶

۱۱.۱.۶ کارت‌های xD

xD ، مخفف **ex**treme **D**igital بوده و مورد استفاده اصلی این کارت در دوربین‌های دیجیتال، ضبط کننده‌های صدای دیجیتال و پخش کننده‌های MP3 است. کارت‌های xD در سال 2002 توسط دو شرکت Olympus و Fujifilm ارایه شدند. ولی، ساخت و تولید این محصول توسط شرکت‌های Toshiba و SAMSUNG ادامه یافت. شکل ۱۱.۶، نمونه‌ای از این کارت‌ها را نشان می‌دهد. کارت‌های xD در سه نوع ارایه می‌شوند: Type M، Type H و Type M+.

Type M ، در سال 2005 و با ظرفیت حداکثری 2 GB ارایه شد. سرعت خواند و نوشتن این کارت به ترتیب برابر 4 و 2.5 MB/S است.

کارت‌های xD نوع Type H در سال 2005 متولد شد که دارای سرعت انتقال داده‌ی بیشتری بود. سرعت خواندن و نوشتن این کارت‌ها 5 و 4 MB/S بود. تولید این کارت‌ها به علت هزینه‌ی بالایشان هم‌اکنون متوقف شده است. حداکثر اندازه‌ی این کارت‌ها، 2 GB بودند.

کارت‌های Type M+ ، در سال 2008 تولید شدند. این کارت سریع‌ترین کارت xD در بین سایر کارت‌ها بودند. سرعت نوشتن و خواندن آن به ترتیب برابر 6 و 3.75 MB/S بود. حداکثر اندازه‌ی آن‌ها نیز به 2 GB می‌رسید.

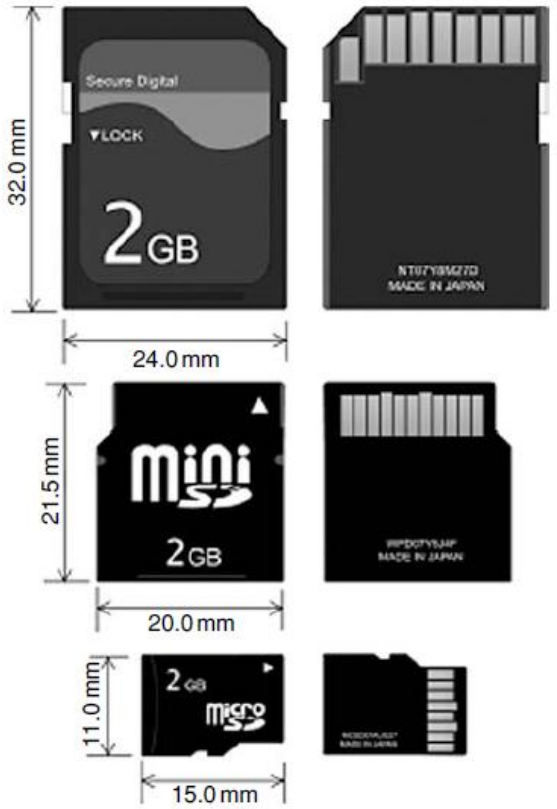
مزیت این کارت‌ها نسبت به کارت‌های SM، MMC و MS، سرعت بالاترشان است. اندازه‌ی کوچک این کارت‌ها، استفاده از آن‌ها را در بسیاری از دستگاه‌های کم‌مصرف مناسب ساخته است. از معایب آن‌ها، اندازه‌ی بزرگ‌ترشان نسبت به بعضی انواع دیگر کارت، قیمت بیشتر و از همه مهم‌تر، تعلق این کارت به شرکت‌های Olympus و Fujifilm است. بدین معنا که هیچ سند، برگه اطلاعات و... در مورد آن‌ها به صورت رایگان روی اینترنت وجود ندارد.

۱۱.۱.۷ کارت‌های Secure Digital (SD)

کارت‌های SD ، امروزه جزو پرکاربردترین کارت‌های حافظه هستند. کارت SD در اصل توسط شرکت‌های Matsushita، San Disk و Toshiba در سال 2000 توسعه داده شدند. کارت‌های SD کاربردهای فراوانی در دستگاه‌هایی چون دوربین‌های دیجیتال، تلفن‌های همراه، PDA ها، کامپیوترهای دستی، ضبط کننده‌های فیلم و... پیدا کرده‌اند. کارت‌های SD استاندارد دارای اندازه‌های 4 MB تا 4 GB هستند. اخیراً نوعی بهبود یافته از این کارت‌ها با نام SDHC تولید شده‌اند که دارای حجم ذخیره سازی 4 تا 32 GB هستند. همچنین طبق اخبار به دست رسیده، نوعی دیگر این کارت‌ها که eXtended Capacity (SDXC) نام گرفته‌اند به ظرفیت‌های ذخیره سازی تا 2 TB نیز خواهند رسید.

کارت‌های SD براساس MMC ها ساخته شدند اما دارای تفاوت‌هایی نیز هستند:

- کارت‌های SD از کارت‌های MMC ضخیم‌تر بوده و در سوکت‌های مخصوص (فقط) MMC جای نمی‌گیرند؛
- کارت‌های SD برخلاف کارت‌های MMC، دارای شکل نامتقارنی هستند. بنابراین نمی‌توان آن‌ها را به طور معکوس در جایشان قرار داد؛
- ساختار ثبات‌های داخلی این دو کارت نیز یکسان نیستند.



شکل ۱۱.۷

۱۱.۱.۷.۱ کارت‌های SD استاندارد

کارت‌های SD در سه اندازه‌ی مختلف ساخته می‌شوند، SD معمولی، mini SD و micro SD.
شکل ۱۱.۷ انواع مختلف این کارت‌ها را نشان می‌دهد.
کارت‌های SD معمولی .gr 2 بوده و دارای ابعاد 24.0×32.0×2.1 mm می‌باشند. یک کلید کوچک برای محافظت از نوشتن و یا پاک کردن تصادفی بر روی این کارت قرار دارد. سرعت انتقال داده در آن بین 15-20 MB/S است و با ولتاژ بین 2.7 V-3.6 V کار می‌کنند. این کارت دارای ۹ پایه است.
Mini SD ،در سال 2003 ساخته شد. ابعاد این کارت 20.0×21.5×1.4 mm و وزن آن .gr 1 است. کلید محافظت ازنوشتن در این کارت وجود ندارد. ولتاژ کاری آن 2.7-3.6 V بوده و دارای ۱۱ پایه است. سرعت انتقال داده‌ها در این کارت، تقریباً برابر 15 MB/S است و در ظرفیت‌های 16 MB تا 8 GB نیز موجودند.

خصیصه	SD	miniSD	microSD

11 mm	20 mm	24 mm	عرض
15 mm	21.5 mm	32 mm	طول
1 mm	1.4 mm	2.1 mm	ضخامت
0.5 g	1 g	2 g	وزن
2.7V – 3.6V	2.7V – 3.6V	2.7V – 3.6V	ولتاژ عملکرد
8	11	9	تعداد پایه

شکل ۱۱.۸

کارت‌های micro SD ، در سال 2008 متولد شده دارای ابعاد 11.0×15.0×10.0 mm و وزن 0.5 gr. است. سایر مشخصات این کارت‌ها مشابه mini SD بوده با این تفاوت که دارای ۸ پایه هستند و در اندازه‌های 64 MB تا بیش از 4 GB یافت می‌شوند.

در جدول MC-2 ، سایر مشخصات این سه نوع کارت مقایسه شده‌اند. هر یک از کارت‌های کوچک‌تر می‌توانند با استفاده از یک آداپتور، به جای کارت SD معمولی، استفاده شوند (شکل ۱۱.۸).

کارت‌های SD ، معمولی در حجم‌های تا 2 GB یافته شده و عموماً با یک فایل سیستم FAT 16 که از قبل بر روی آن‌ها قرار داده شده است، به فروش می‌رسند.

فصل دوازدهم
سیستم فایل

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ فایل سیستم FAT چیست؟
- ✓ روش ارتباط MMC و SD با میکروکنترولر؛
- ✓ کتابخانه‌های رایگان EFSL و FatFS برای راه‌اندازی FAT؛
- ✓ فایل‌های سیستم فلش چه هستند؟
- ✓ JFFS2، UBIFS و YAFFS.

۱۲.۱ درباره‌ی FAT

سیستم فایل FAT[۱] (که با نام‌های FAT12، FAT16 و FAT32 نیز شناخته می‌شود)، توسط Bill Gates و Marc Mc Donald توسعه داده شدند. از ابتدایی‌ترین و شناخته شده‌ترین سیستم‌های فایل مورد استفاده در بیشتر سیستم‌های عامل و کارت‌های حافظه است.

FAT ، برای مدیریت بهینه‌ی دیسک‌ها ساخته شد. نام‌گذاری FAT به خاطر استفاده از جداولی بود که اطلاعاتی را درباره‌ی نواحی مورد استفاده‌ی فایل‌ها و مکان قرارگیری‌شان بر روی دیسک بود. برای محدود کردن اندازه‌ی جدول، فضای دیسک به فایل‌هایی در سکتورهای به هم پیوسته به نام کلاستر (Cluster)، ذخیره می‌شوند.

با گذشت زمان و سیر تکاملی دیسک‌ها، تعداد بیشینه‌ی کلاسترها و تعداد بیت‌های مشخص کننده‌ی کلاسترها نیز افزایش یافته است. شماره‌ی مشخص کننده‌ی FAT، تعداد بیت‌های هر یک از عناصر جدول‌های FAT هستند.

۱۲.۱.۱ راه‌اندازی فایل سیستم FAT

EFSL و FATFS ، دو کتابخانه‌ی معروف توسعه‌ی سیستم فایل FAT هستند که مخصوص سیستم-های تعبیه شده طراحی شده‌اند. در این بخش، نحوه‌ی استفاده از این کتابخانه‌ها را توضیح می‌دهیم.

---

[۱] File Allocation Table

مجموعه‌ای از توابع API ساده، برای ارتباط با حافظه‌ی MMC/SD از طریق واسط SPI نیز فراهم آمده‌اند.

۱۲.۲ ارتباط با SD/MMC

کارت‌های SD و MMC، دو کارت حافظه با پایه‌ی فلش هستند که اختصاصاً برای این‌است، نیاز به حجم بالا و... مورد نیاز برای دستگاه‌های قابل حمل ساخته شده‌اند. ارتباط با کارت SD از طریق ارتباط حرفه‌ای ۹ پایه شامل: Clock، Command، ۴خط Data و ۳ پایه Power برقرار می‌شود. میزبانی که از کارت‌های SD پشتیبانی می‌کند، عموماً کارت‌های MMC را نیز راه‌اندازی می‌کند. این کارت‌ها در اندازه‌های کوچک‌تر، با نام‌های RS-MMC، mini SD، micro SD و... نیز ساخته شده‌اند که دارای همان کارکردها می‌باشند.

توجه

تفاوت اصلی بین کارت‌های SD و MMC در فرآیند راه‌اندازی اولیه‌شان است.

۱۲.۲.۲ واسط SD/MMC

واسط SD/MMC یک رابط ساده بوده که بدون توجه به نوع پردازنده‌ی مورد استفاده، می‌توان آن را در هر سیستمی به کار بست. کارت‌های SD/MMC برای سازگاری با کنترل کننده‌های قدیمی، از یک پروتکل استاندارد براساس SPI نیز پشتیبانی می‌کنند.
پایه‌های کارت SD/MMC ، در جدول MMC-1 آمده‌اند.

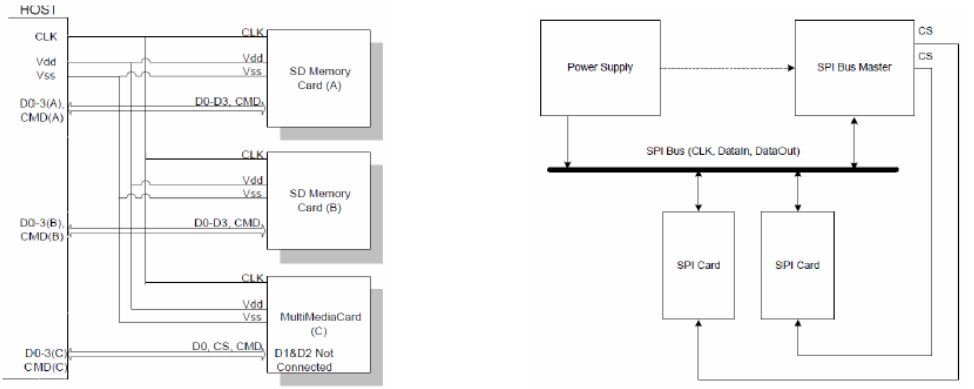
Table 1. SDC/MMC pin assignment

Pin No.	Name	Type	Description
SDC/MMC Bus Mode[1]			
1	CD/DAT3	I/O, PP	Card detect/Data line [Bit 3]
2	CMD	I/O, PP	Command/Response
3	Vss1	S	Supply voltage ground
4	Vdd	S	Supply voltage
5	CLK	I	Clock
6	Vss2	S	Supply voltage ground
7	DAT0	I/O, PP	Data line [Bit 0]
8	DAT1	I/O, PP	Data line [Bit 1]
9	DAT2	I/O, PP	Data line [Bit 2]
SPI Bus Mode			
1	CS	I	Chip Select (active low)
2	DataIn	I	Host-to-card Commands and Data
3	Vss1	S	Supply voltage ground
4	Vdd	S	Supply voltage
5	CLK	I	Clock
6	Vss2	S	Supply voltage ground
7	DataOut	O	Card-to-host Data and Status
8	RSV	---	Reserved
9	RSV	---	Reserved

[1] For MMC, only one data line, DAT0, is used.

جدول MMC_1

در شکل ۱۲.۱ ، نحوه‌ی ارتباط کارت‌های مختلف بر روی یک گذرگاه را می‌بینیم.



a. SDC/MMC bus mode

b. SPI bus mode

شکل ۱۲.۱

برای سادگی پیاده سازی رابط کارت مذکور، از ارتباط SPI و یکی از میکروکنترولرهای NXP استفاده می‌کنیم.

۱۲.۲.۳ ساختار گذرگاه SPI

SPI ، یک پروتکل ثانویه برای ارتباط با کارت‌های MMC/SD است. هر یک از این کارت‌های مذکور توسط سیگنال CS در درون گذرگاه کارت‌ها انتخاب شده و دستورهای لازم برای راه‌اندازی آن از طریق فرمانده[۱] ارسال می‌شود. در مرحله‌ی بعد، کارت حافظه به عنوان فرمانبردار[۲] به اجرای دستورهای مورد نظر می‌پردازد.

سیگنال CS ، باید در تمام طول ارتباط (انتقال دستورها، داده و پاسخ‌ها)، فعال باشد. حالت فعال این سیگنال، سطح منطقی "0" است.

۱۲.۲.۴ پروتکل گذرگاه SPI

میکروکنترولر، از طریق مجموعه‌ای از دستورها، پاسخ‌ها و بلوک‌های داده با کارت MMC/SD ارتباط برقرار می‌کند. دستورهای ارسالی به کارت حافظه، زیرمجموعه‌ای از دستورهای استاندارد پروتکل MMC/SD است. فرمانده تمامی ارتباطات بین میزبان و کارت‌ها را کنترل می‌کند. بعد از فعال کردن

---

[۱] Master

[۲] Slave

سیگنال CS ، ارتباط با کارت مذکور آغاز می‌شود. رفتار پاسخ‌دهی کارت‌ها در حالت SPI، از سه جنبه با حالت MMC/SD تفاوت می‌کنند:

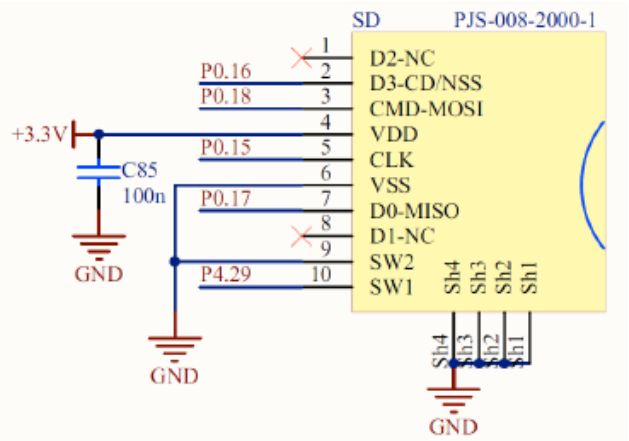
- تنها کارت انتخاب شده به دستور پاسخ می‌دهد؛
- ساختار پاسخ ارسالی به صورت ۸ و یا ۱۶ بیتی است؛
- هنگامی که کارت با یک خطا در داده ارسالی مواجه شود، با یک پاسخ خطا جواب می‌دهد. در حالی که در حالت MMC/SD، پاسخ حاصل به صورت Time-Out است.

۱۲.۲.۵ انتخاب حالت کاری

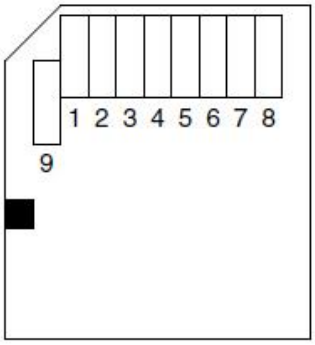
کارت‌های MMC/SD به هنگام اتصال تغذیه در حالت MMC/SD قرار دارند. با فعال کردن پایه‌ی CS و ارسال هم‌زمان یک دستور ریست (CMD0)، کارت به حالت SPI وارد می‌شود.

۱۲.۲.۶ ارتباط از طریق SPI

در این‌جا، از میکروکنترولرهای خانواده‌ی LPC1700 استفاده کرده‌ایم که دارای تنها یک رابط SPI هستند. SSP نیز همان کارآیی SPI را با سرعت انتقال داده‌ی بالاتری ارایه می‌دهد (نرخ کلاک ۴ برابر). برای مثال، اگر PCLk برابر 100 MHz انتخاب شده باشد، بیشینه‌ی نرخ انتقال SPI، 12.5 Mbit/Sec خواهد بود. در حالی که بیشینه‌ی نرخ انتقال در SSP برابر 50 Mbit/Sec است. ارتباط پایه‌های SSP0 میکروکنترولر LPC1768 و یک کارت MMC در شکل ۱۲.۲ نشان داده شده است.



شکل ۱۲.۲



شکل ۱۲.۳

Table 2. Micro SD card pin assignment

Pin No.	Name	Type	Description
SDC Bus Mode			
1	DAT2	I/O, PP	Data line [Bit 2]
2	CD/DAT3	I/O, PP	Card detect/Data line [Bit 3]
3	CMD	I/O, PP	Command/Response
4	Vdd	S	Supply voltage (2.7v / 3.6v)
5	CLK	I	Clock
6	Vss	S	Supply voltage ground
7	DAT0	I/O, PP	Data line [Bit 0]
8	DAT1	I/O, PP	Data line [Bit 1]
SPI Bus Mode			
1	RSV	---	Reserved
2	CS	I	Chip Select (active low)
3	DataIn	I	Host-to-card Commands and Data
4	Vdd	S	Supply voltage
5	CLK	I	Clock
6	Vss	S	Supply voltage ground
7	DataOut	O	Card-to-host Data and Status
8	RSV	---	Reserved

۱۲.۲.۷ توابع راه‌اندازی SPI

تعداد ۸ تابع API برای راه‌اندازی SPI ارایه شده‌اند:

- SPI_Init
  این تابع SSP0، PCONP، GPIO و ثبات‌های کنترلی را پیکره‌بندی می‌کند؛
- SPI_DeInit
  برای پاک کردن اطلاعات مربوط به پیکره‌بندی SSP0 و خاموش کردن آن استفاده می‌شود؛
- SPI_Select
  این تابع، سیگنال CS را به حالت فعال درمی‌آورد؛
- SPI_DeSelect
  عکس عملیات SPI_Select؛
- SPI_Release
  گذرگاه SPI را آزاد می‌کند؛
- SPI_SetSpeed
  سرعت ارتباط از طریق گذرگاه SPI را تنظیم می‌کند؛

نکته

عموماً در مرحله‌ی مقداردهی اولیه MMC/SD، سرعت ارتباط به 400 KHz محدود است ولی، بعد از آن وارد مرحله‌ی تبادل اطلاعات اصلی، سرعت حداکثر MMC برابر 20 MHz و SD برابر 25 MHz می‌باشد.

- SPI_Send Byte
  یک بایت داده را بر روی گذرگاه SPI می‌فرستد؛
- SPI_Recv Byte
  یک بایت داده را از روی گذرگاه SPI می‌خواند.

۱۲.۲.۸ توابع راه‌اندازی MMC/SD

برای دسترسی به کارت، ۴ تابع تعریف شده‌اند:

- SD_Init

مقداردهی اولیه‌ی MMC/SD را بر عهده دارد؛

- SD_ReadCfg

این تابع، اطلاعات پیکره‌بندی داخلی کارت‌ها را از درونشان می‌خواند. اطلاعاتی همانند OCR، CID و CSD را خوانده و همچنین به محاسبه‌ی اطلاعاتی چون تعداد سکتورها[۱]، اندازه‌ی سکتور و... می‌پردازد؛

- SD_Read Sector

تعداد مشخصی از سکتورها را از کارت می‌خواند. در این‌جا، اندازه‌ی سکتور برابر 512 بایت فرض شده است؛

- SD_Write Sector

تعداد معینی از سکتورها را بر روی کارت می‌نویسد.

۱۲.۳ معرفی EFSL و Fat FS

۱۲.۳.۱ EFSL

پروژه‌ی EFSL[۲]، برای ایجاد کتابخانه‌ای جهت سیستم فایل درست شده است. هدف این پروژه، سیستم‌های تعبیه شده بود. در حال حاضر، EFSL از تمامی اعضای خانواده‌ی FAT پشتیبانی می‌کند. EFSL کدهایش را به زبان C استاندارد منتشر ساخته است. بنابراین، کدها در هر کامپایلر C استانداری می‌توانند استفاده شوند.

اضافه نمودن کدهای استاندارد به سخت‌افزار دلخواهتان، کار آسانی است. فقط، کدهای مربوط به نوشتن و خواندن سکتورهای 512 بایتی را به برنامه اضافه کرده سپس، کتابخانه‌های ارایه شده بقیه‌ی کار را بر عهده می‌گیرند.

---

[۱] Sector

[۲] Embedded File System Library

Fat FS ۱۲.۳.۲

Fat FS ، یک ماژول سیستم فایل عمومی FAT است که برای سیستم‌های تعبیه شده‌ی کوچک درست شده است. Fat FS ، با زبان ANSI C کاملاً سازگار بوده و کاملاً از لایه‌ی I/O دیسک جدا می‌باشد. این به معنای استقلال کامل از معماری سخت‌افزاری است. پیاده سازی این ماژول در میکروکنترولرهای مختلف، نیاز به هیچ گونه تغییری ندارد.

Fat FS مشخصات زیر را دارند:

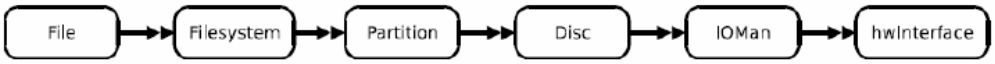
- سیستم فایل FAT 12/16/32 سازگار با ویندوز؛
- نابسته به پلتفرم مورد استفاده، پورت آسان (پورت در این‌جا به معنای فرآیندی است که در آن یک قطعه کد را برای اجرا بر روی یک سخت‌افزار خاص، مهیا می‌کنند)؛
- فضای کد مصرفی کم؛
- امکانات پیکره‌بندی مختلف:

۱) پشتیبانی از چندین پارتیشن و درایو؛
۲) چندین صفحه کد OEM، شامل DBCS؛
۳) پشتیبانی از اسم فایل‌های طولانی (LFN) در حالت‌های OEM و یا Unicode؛
۴) پشتیبانی از RTOS؛
۵) پشتیبانی از اندازه‌های مختلف سکتور؛

کدهای Fat FS رایگان بوده و می‌توانید آن‌ها را تغییر دهید و یا برای مقاصد تجاری و غیرتجاری استفاده کنید.

۱۲.۳.۳ پورت EFSL بر روی LPC1768

ساختار داخلی EFSL در شکل ۱۲.۴ نشان داده شده است.



شکل ۱۲.۴

EFSL ، یک مدل خطی ساده را دنبال می‌کند. شئ Partition مسئولیت ترجمه‌ی آدرس‌های نسبی پارتیشن به آدرس‌های دیسک LBA را بر عهده دارد. شئ Disk ، جدول پارتیشن را در خود نگه می‌دارد و ارتباط مستقیمی با مدیریت حافظه‌ی نهان (IOMan) دارد. IOMan بررسی می‌کند که آیا سکتور مورد نظر باید از دیسک (و یا حافظه) خوانده شده و یا بر روی آن نوشته شود. هنگام رسیدن درخواست نوشتن و یا خواندن، از hardware استفاده می‌شود.

واسط سخت‌افزاری سه وظیفه دارد:

۱. مقداردهی اولیه‌ی سخت‌افزار؛
۲. خواندن سکتورها از دیسک؛
۳. نوشتن سکتورها بر روی دیسک.

تمامی درخواست‌ها براساس سکتور بوده و اندازه‌ی سکتورها ۵۱۲ بایتی هستند. پورت کردن EFSL بر روی LPC1768 ، کار آسانی است. فقط باید کدهای مربوط به نوشتن و خواندن یک سکتور ۵۱۲ بایتی را نوشته و بقیه کار را EFSL انجام خواهد داد.

در این بخش، نحوه‌ی پورت کردن کد فوق را به صورت مرحله به مرحله خواهیم آموخت.

۱۲.۳.۳.۱ تعریف یک نام اندپوینت

برای انجام تعاریف (#define)، به این نام نیاز خواهیم داشت. در این‌جا، از نام HW_ENDPOINT_LPC17xx_SD استفاده کرده‌ایم. برای تغییر این نام، به فایل config.h رفته و به شرح زیر عمل کنید:

```
/* Hardware target
   ---------------

Here you will define for what hardware-endpoint EFSL should be
compiled.
Look in interfaces.h to see what systems are supported, and add your
own there if you need to write your own driver. Then, define the name
you selected for your hardware there here. Make sure that you only
select one device!
```

```
*/
	/*#define HW_ENDPOINT_LINUX*/
	/*#define HW_ENDPOINT_ATMEGA128_SD*/
	//#define HW_ENDPOINT_LPC2000_SD
	/* defines the interface for LPC213x (0=SPI0 1=SPI1) */
	// #define HW_ENDPOINT_LPC2000_SPINUM  (0)
	//#define HW_ENDPOINT_LPC2000_SPINUM  (1)
	/*#define HW_ENDPOINT_DSP_TI6713_SD*/
	/* define the interface for LPC17xx SSP0 */
	#define HW_ENDPOINT_LPC17xx_SD
```

۱۲.۳.۳.۲ تعیین اندازه‌ی انواع داده‌ای صحیح

فایل inc/types.h را باز کرده و چند خط کد زیر را در فایل مورد نظر کپی کنید (ممکن است از قبل نیز تعدادی از این متغیرها موجود باشند):

```
	typedef char eint8;
	typedef signed char esint8;
	typedef unsigned char euint8;
	typedef short eint16;
	typedef signed short esint16;
	typedef unsigned short euint16;
	typedef int eint32;
	typedef signed int esint32;
	typedef unsigned int euint32;
```

۱۲.۳.۳.۳ اندپوینت را به interface.h اضافه کنید

نام اندپوینت را در فایل inc/interface.h همانند زیر اضافه کنید:

```
#if defined(HW_ENDPOINT_LINUX) || defined(HW_ENDPOINT_LINUX64)
	#include "interfaces/linuxfile.h"
#elif defined(HW_ENDPOINT_ATMEGA128_SD)
	#include "interfaces/atmega128.h"
#elif defined(HW_ENDPOINT_DSP_TI6713_SD)
	#include "interfaces/dsp67xx.h"
#elif defined(HW_ENDPOINT_LPC2000_SD)
	#include "interfaces/lpc2000_spi.h"
#elif defined(HW_ENDPOINT_LPC17xx_SD)
	#include "interfaces/if_lpc17xx.h"
#else
	#error "NO INTERFACE DEFINED - see interface.h"
#endif
```

۱۲.۳.۳.۳ پیکره‌بندی EFSL

فایل پیکره‌بندی، efsl\conf\config.h\ رفتار کتابخانه را معین می‌کند. در این فایل، پیکره‌بندی تنظیمات گوناگونی وجود دارند که بسیاری از آن‌ها نیاز به تغییر ندارند. تنظیماتی که نیاز به تغییر دارند، در جدول ۳ آمده‌اند.

Table 3. Configurations of EFSL in this project

Item	Configuration	Description
Hardware target	#define HW_ENDPOINT_LPC17xx_SD	Access SDC/MMC via LPC17xx SSP0
Memory	/* #define BYTE_ALIGNMENT */[1]	Specify that the MCU can not access memory byte oriented
Cache	#define IOMAN_NUMBUFFER 6 #define IOMAN_NUMITERATIONS 3 #define IOMAN_DO_MEMALLOC	6x512 byte (3 KB) RAM used for cache
Cluster pre-allocation	#define CLUSTER_PREALLOC_FILE 2 #define CLUSTER_PREALLOC_DIRECTORY 0	The number of clusters pre-allocated when writing files.
Item	**Configuration**	**Description**
Endianess	#define LITTLE_ENDIAN	All FAT structures are stored in Intel little endian order
Date and Time support	/*#define DATE_TIME_SUPPORT*/	Disable date and time support
Error reporting support	#define FULL_ERROR_SUPPORT	Enable error recording for all object
List options	#define LIST_MAXLENFILENAME 12	Configure what kind of data you will get from directory listing requests
Debugging	/* #define DEBUG */	Disable debugging behavior

جدول ۳

۱۲.۳.۳.۳ ایجاد فایل منبع

فایل‌های هدر، در آدرس inc\interfaces و فایل‌های منبع در آدرس src\interfaces قرار دارند. فایل LPC17xx_spi.c(h) ، شامل توابعی برای ارتباط با SSP0 بوده و فایل LPC17xx_sd.c(h) حاوی توابع ارتباط با SD/MMC از طریق SSP0 است.

۱۲.۳.۳.۳ پیاده سازی توابع سطح پایین

hwinterface

این ساختار، توضیح دهنده‌ی سخت‌افزاری است. EFSL ، بـه بعضـی از ورودی‌هـای آن نیـاز دارد. در سیستم‌های تعبیه شده بهتر است که این ساختار را تا حد ممکن کوچک نگه داریم.

```
/*************************************************************\
             hwInterface
              ----------
* long       sectorCount         Number of sectors on the file.
\*************************************************************/
struct  hwInterface{
       euint32      sectorCount;
};
typedef struct hwInterface hwInterface;
```

If_init Interface

این تابع، فقط یک‌بار فراخوانی خواهد شد. وقتی سخت‌افزار توسـط esf_init() مقـداردهی اولیـه مـی-شود، این قسمت از کد سخت‌افزار را به یک حالت آماده به کار می‌رساند.

توصیه می‌شود که مقدار Sector Count واقع در hwinterface را با عددی مناسب پـر کنیـد (امـا ضروری نیست).

```
esint8 if_initInterface(hwInterface* file, eint8* opts)
{
       SDCFG SDCfg;

       if (LPC17xx_SD_Init() == false)
              return (-1);
       if    (LPC17xx_SD_ReadCfg(&SDCfg) == false)
              return (-2);

       file->sectorCount = SDCfg.sectorcnt;

       return 0;
}
```

If_read Buf

وظیفه‌ی این تابع، خواندن یک سکتور از دیسک و ذخیره کردن آن در یک بافر است.

توجه

در انتخاب اندازه‌ی بافر، دقت کنید. کوچک بودن اندازه‌ی بافر، باعث وقوع حالت‌های پیش-بینی نشده‌ای می‌شود.

```
/*
	read a sector from the disc and store it in a user supplied
buffer.
	note that there is no support for sectors that are not 512
bytes large
*/

esint8 if_readBuf(hwInterface* file,euint32 address,euint8* buf)
{
	if (LPC17xx_SD_ReadSector (address, buf, 1) == true)
		return 0;
	else
		return (-1);
}
```

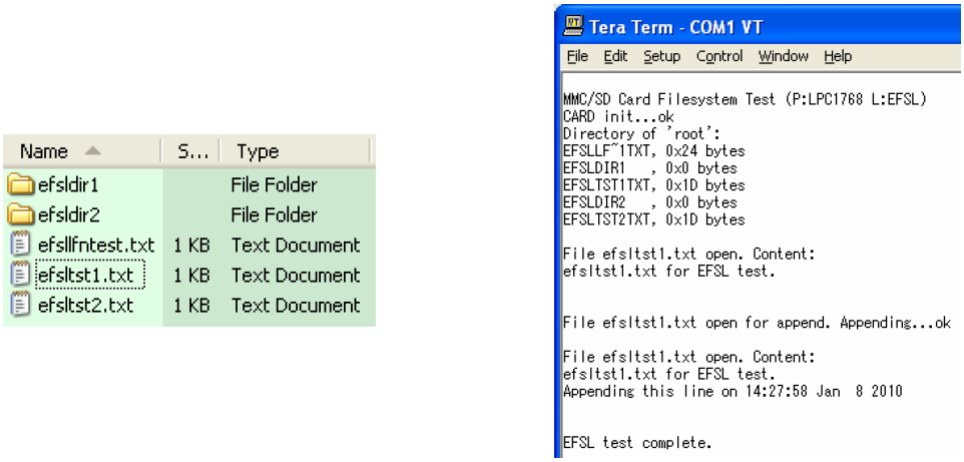
If_write Buf

همانند نمونه‌ی خواندن از دیسک است.

```
/*
	write a sector.

	note that there is no support for sectors that are not 512
bytes large.
*/

esint8 if_writeBuf(hwInterface* file,euint32 address,euint8* buf)
{
	if ( LPC17xx_SD_WriteSector(address, buf, 1) == true)
		return 0;
	else
		return (-1);
}
```

۱۲.۳.۳.۳ اجرای کد

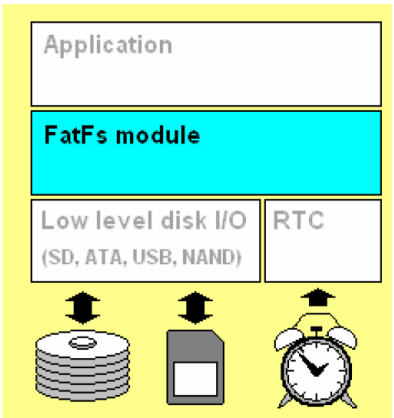
با قرار دادن فایل‌ها در کنار یکدیگر و ایجاد یک پروژه در محیط Keil، برنامه را اجرا کرده و خروجی‌های زیر را می‌بینیم.



شکل ۱۲.۵

۱۲.۳.۴ پورت کردن FAT FS روی LPC1768

ساختار FAT FS ، در شکل ۱۲.۶ نشان داده شده است. توابع زیادی در فایل ff.c تعریف شده‌اند که کارکردهای مفیدی ارایه می‌دهند. به عنوان مثال: f_open، f_close، f_read، f_write و... . مادامی-که کامپایلر C با استاندارد ANSI سازگار باشد، هیچ وابستگی به سخت‌افزار در این ماژول وجود ندارد. یک ماژول I/O ، برای نوشتن/ خواندن از دیسک فیزیکی اختصاص یافته است. یک ماژول RTC ، برای گرفتن زمان فعلی درنظر گرفته شده است.



شکل ۱۲.۶

دو ماژول I/O ، بر روی دیسک و زمان سنج RTC، کاملاً از FAT FS جدا هستند. این کدها باید توسط کاربر آماده شوند (فرآیند پورت کردن برای FAT FS) برای پورت کردن FAT FS بر روی LPC1768 مراحل زیر را دنبال می‌کنیم.

۱۲.۳.۳.۳ اندازه‌ی اعداد صحیح

در ماژول FAT FS ، فرض بر این‌است که اندازه‌ی long/short/char به ترتیب ۳۲/۱۶/۸ بیت و اندازه‌ی Int ،۱۶ و یا ۳۲ بیتی است. این اندازه‌ها در فایل integer.h تعریف شده است. هرگونه تداخل با مقادیر از پیش تعریف شده، باید برطرف شود.

```
/* These types must be 16-bit, 32-bit or larger integer */
typedef int               INT;
typedef unsigned int      UINT;

/* These types must be 8-bit integer */
typedef signed char       CHAR;
typedef unsigned char     UCHAR;
typedef unsigned char     BYTE;

/* These types must be 16-bit integer */
typedef short             SHORT;
typedef unsigned short    USHORT;
typedef unsigned short    WORD;
typedef unsigned short    WCHAR;

/* These types must be 32-bit integer */
typedef long              LONG;
typedef unsigned long     ULONG;
typedef unsigned long     DWORD;
```

۱۲.۳.۳.۳ پیکره‌بندی ماژول FAT FS

تمامی پیکره‌بندی‌ها و توضیحات مفصل‌شان در فایل ffcong.h آمده است. تنظیماتی که در این‌جا استفاده شده‌اند را در جدول ۴ می‌بینیم.

Item	Configuration	Description
Function and Buffer Configurations	#define _FS_TINY 0	Use the sector buffer in the individual file data transfer.
	#define _FS_READONLY 0	Enable both read and write functions.
	#define _FS_MINIMIZE 0	Enable full function.
	#define _USE_STRFUNC 0	Disable string functions.
	#define _USE_MKFS 1	Enable f_mkfs function
	#define _USE_FORWARD 0	Disable f_forward function
Locale and Namespace Configurations	#define _CODE_PAGE 850	OEM code page "Multilingual Latin 1" will be used on the target system.
	#define _USE_LFN 1	Enable LFN
	#define _MAX_LFN 255	Maximum LFN length to handle
	#define _LFN_UNICODE 0	Disable Unicode.
	#define _FS_RPATH 1	Enable the relative path feature and f_chdir and f_chdrive function are available.
Physical Drive Configurations	#define _DRIVES 1	Only 1 physical driver is allowed.
	#define _MAX_SS 512	Maximum sector size to be handled
	#define _MULTI_PARTITION 0	Each volume is bound to the same physical drive number and can mount only first primary partition.
System Configurations	#define _WORD_ACCESS 0	Enable the Byte-by-byte access
	#define _FS_REENTRANT 0	Disable reentrancy.

جدول ۴

_USE_LFN

ماژول FAT FS ، از اسامی بلند فایل‌ها (LFN) نیز پشتیبانی می‌کند.

برای فعال سازی LFN، مقدار USE_LFN_ را به مقدار ۱ یا ۲ تنظیم کنید. سپس، در صورت نیاز توابع تبدیل کدهای یونیکد[۱] (یعنی ff_convert و ff_wtoupper) را به پروژه اضافه کنید.

در جدول ۵، اندازه‌ی کد واقع در حافظه‌ی فلش را برای کدهای مختلف یونیکد می‌بینیم.

Code Page	افزایش اندازه ROM (بر حسب Byte)
SBSC	2796

[۱] Unicode

CP932 (Japanese Shift-JIS)	61656
CP936 (Simplified Chinese GBK)	176856
CP949 (Korean)	138912
CP950 (Traditional Chinese Big5)	110544

جدول ۵

۱۲.۳.۳.۳ پیاده سازی توابع سطح پایین

از آن‌جایی که ماژول FAT FS کاملاً از عملیات ورودی/ خروجی دیسک و RTC جدا است، این توابع باید توسط کاربر فراهم شوند.

disk_initialize

این شرکت یک درایو فیزیکی را مقداردهی اولیه می‌کند. این تابع، از فرآیند نصب پارتیشن در ماژول FAT FS فراخوانی می‌شود. هنگامی که ماژول FAT FS فعال باشد، این تابع نباید توسط برنامه‌ی کاربر فراخوانی شود. اگر می‌خواهید مجدداً سیستم فایل را مقداردهی اولیه کنید، تابع f_mount را فراخوانی کنید.

```
/*-----------------------------------------------------------------------*/
/* Initialize Disk Drive                                                 */
/*-----------------------------------------------------------------------*/
DSTATUS disk_initialize (
	BYTE drv		/* Physical drive number (0) */
)
{
	if (drv) return STA_NOINIT;		// Supports only single drive
//	if (Stat & STA_NODISK) return Stat; // No card in the socket

	if (LPC17xx_SD_Init() && LPC17xx_SD_ReadCfg(&SDCfg))
		Stat &= ~STA_NOINIT;

	return Stat;
}
```

disk_status

این تابع، وضعیت فعلی دیسک را برمی‌گرداند. این وضعیت، شامل پرچم‌های زیر است:

- STA_NOINIT: نشان دهنده‌ی این‌است که دیسک مقداردهی اولیه نشده است؛

- STA_NODISK: نشان دهنده‌ی این‌است که هیچ واسطه‌ی فیزیکی در درون درایو موجود نیست؛
- STA_PROTECTED: نشان دهنده‌ی این‌است که واسطه‌ی فیزیکی دیسک در برابر نوشتن محافظت شده است.

```
/*-----------------------------------------------------------------------*/
/* Get Disk Status                                                       */
/*-----------------------------------------------------------------------*/
DSTATUS disk_status (
	BYTE drv		/* Physical drive number (0) */
)
{
	if (drv) return STA_NOINIT;		// Supports only single drive

	return Stat;
}
```

disk_read

یک یا چند سکتور را از دیسک می‌خواند.

```
/*-----------------------------------------------------------------------*/
/* Read Sector(s)                                                        */
/*-----------------------------------------------------------------------*/
DRESULT disk_read (
	BYTE drv,			/* Physical drive number (0) */
	BYTE *buff,			/* Pointer to the data buffer to
store read data */
	DWORD sector,		/* Start sector number (LBA) */
	BYTE count			/* Sector count (1..255) */
)
{
	if (drv || !count) return RES_PARERR;
	if (Stat & STA_NOINIT) return RES_NOTRDY;

	if (LPC17xx_SD_ReadSector (sector, buff, count) == true)
		return RES_OK;
	else
		return RES_ERROR;
}
```

disk_write

یک یا چند سکتور را بر روی دیسک می‌نویسد.

```
DRESULT disk_write (
	BYTE drv,			/* Physical drive number (0) */
```

```
        const BYTE *buff,    /* Pointer to the data to be written */
        DWORD sector,        /* Start sector number (LBA) */
        BYTE count                  /* Sector count (1..255) */
)
{
        if (drv || !count) return RES_PARERR;
        if (Stat & STA_NOINIT) return RES_NOTRDY;
//      if (Stat & STA_PROTECT) return RES_WRPRT;

        if ( LPC17xx_SD_WriteSector(sector, buff, count) == true)
                return RES_OK;
        else
                return        RES_ERROR;

}
```

disk_ioctl

این تابع یک سری تنظیمات مرتبط با دیسک (به جز نوشتن و خواندن را انجام می‌دهد).

Command	Description
Device independent	
CTRL_SYNC	Ensures that the disk drive has finished pending write process. When the disk I/O module has a write back cache, flush the dirty sector immediately. This command is not required in read-only configuration
GET_SECTOR_SIZE	Returns sector size of the drive into the WORD variable pointed by Buffer. This command is not required in single sector size configuration, _MAX_SS is 512.
GET_SECTOR_COUNT	Returns total sectors on the drive into the DWORD variable pointed by Buffer. This command is used in only f_mkfs function.
GET_BLOCK_SIZE	Returns erase block size of the memory array in unit of sector into the DWORD variable pointed by Buffer. This command is used in only f_mkfs function.
Device dependent	
MMC_GET_TYPE	Get card type flags (1 byte)
MMC_GET_CSD	Receive CSD as a data block (16 bytes)
MMC_GET_CID	Receive CID as a data block (16 bytes)
MMC_GET_OCR	Receive OCR as an R3 response (4 bytes)
MMC_GET_SDSTAT	Receive SD status as a data block (64 bytes)

جدول ۶

get_fattime

زمان فعلی را بازمی‌گرداند.

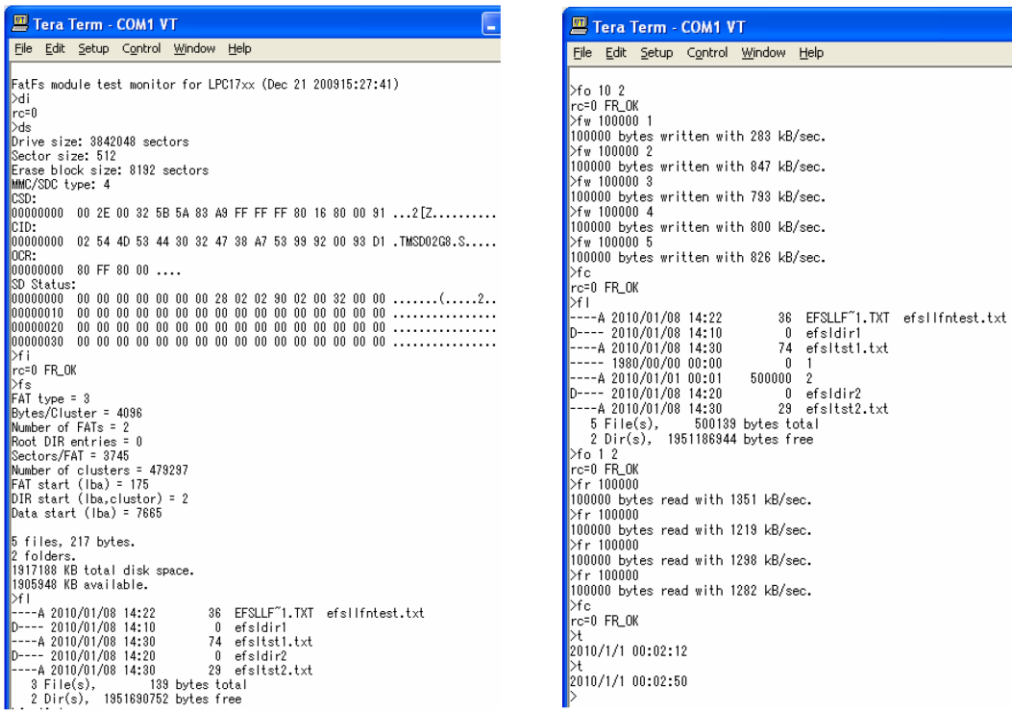
```
/*-----------------------------------------------------------*/
/* User Provided RTC Function for FatFs module               */
/*-----------------------------------------------------------*/
/* This is a real time clock service to be called from       */
/* FatFs module. Any valid time must be returned even if     */
/* the system does not support an RTC.                       */
/* This function is not required in read-only cfg.           */
DWORD get_fattime ()
{
	RTCTime rtc;

	/* Get local time */
	rtc_gettime(&rtc);

	/* Pack date and time into a DWORD variable */
	return	((DWORD)(rtc.RTC_Year - 1980) << 25)
			| ((DWORD)rtc.RTC_Mon << 21)
			| ((DWORD)rtc.RTC_Mday << 16)
			| ((DWORD)rtc.RTC_Hour << 11)
			| ((DWORD)rtc.RTC_Min << 5)
			| ((DWORD)rtc.RTC_Sec >> 1);
}
```

نمونه‌ای از پروژه‌ی نوشته شده با نرم‌افزار KEIL برای راه‌اندازی ماژول فوق در DVD همراه کتاب موجود است. با راه‌اندازی برد و اتصال آن از طریق پورت سریال به یک برنامه‌ی ترمینال، می‌توانید دستورهای لازم را به برد بدهید و خروجی‌های آن را مشاهده کنید. جدول ۷، دستورهای پشتیبانی شده را نشان می‌دهد.

Command	Description
Disk functions	
Di	Initialize the disk
Ds	Show disk status
dd [<lba>]	Dump a specific sector
File functions	
fi	Force initialize the logical drive
fs	Show logical drive status
fl [<path>]	Directory listing
fo <mode> <file>	Open a file
fc	Close a file
fe	Seek file pointers
fd <len>	Read and dump file from current fp
fr <len>	Read file
fw <len> <val>	Write file
fn <old_name> <new_name>	Change file/dir name
fu <name>	Unlink a file or dir
fv	Truncate file
fk <name>	Create a directory
fa <atrr> <mask> <name>	Change file/dir attribute
ft <year> <month> <day> <hour> <min> <sec> <name>	Change timestamp
fx <src_name> <dst_name>	Copy file
fg <path>	Change current directory
fj <drive#>	Change current drive
fm <partition rule> <cluster size>	Create file system
fz [<rw size>]	Change R/W length for fr/fw/fx command
Time functions	
t [<year> <mon> <mday> <hour> <min> <sec>]	Get or set the current date and time

جدول ۷

نمونه‌ای از خروجی برنامه را در شکل ۱۲.۷ می‌بینید.



a. Card and file system information test

b. Card R/W and RTC test

شکل ۱۲.۷

YAFFS ۱۲.۴

YAFFS[۱]، توسط Charles Manning و برای شرکت Aleph One طراحی و نوشته شد. YAFFS1 اولین نسخه این سیستم فایل بود و بر روی تراشه‌های Nand ی که دارای صفحات ۵۱۲ بایتی به علاوه نواحی یدکی[۲] ۱۶ بایتی (OOB یا همان Out Of Band) کار می‌کرد.

تراشه‌های Nand جدیدتر، از صفحات بزرگ‌تر ۲۰۴۸ بایتی به همراه نواحی یدکی ۶۴ بایتی با ملزومات نوشتن سخت و محکم‌تری بهره می‌برند. هر صفحه واقع در بلوک پاک کردن (با اندازه 128Kbyte) ،

[۱] Yet Another Flash File System

[۲] Spare Area

باید به طور ترتیبی و فقط یک بار نوشته شود. YAFFS2 ، برای تطبیق با این تراشه‌های جدید طراحی شده است.

طرح ابتدایی YAFFS2 ، بر اساس کد نوشته شده برای YAFFS1 بوده است. تفاوت اصلی این نسخه در ثابت نبودن اندازه بر روی ۵۱۲ بایت است. همچنین، یک شماره ترتیب بلوک در هر صفحه نوشته شده قرار داده می‌شود.

YAFFS ، یک سیستم فایل با ساختار Log است که تقدم عملکردش بر اساس نگهداری یکپارچگی داده است. هدف ثانویه این سیستم فایل کارایی بالاست به نحوی که این سیستم فایل از بسیاری از سیستم-های مشابهش کاراتر است. YAFFS ، یک سیستم فایل با قابلیت پورت کردن آسان است و در بسیاری از سیستم عامل‌های معروف همانند Linux، WinCE، pSOS، eCos، ThreadX و بسیاری دیگر از سیستم‌های عامل خاص نیز استفاده شده است. گونه‌ای از این سیستم فایل به نام YAFFS/Direct در اماکنی که سیستم های عامل تعبیه شده وجود ندارند، به کار گرفته می‌شود.

JFFS2 ۱۲.۴

JFFS2 ، یک سیستم‌فایل با ساختار Log است که برای استفاده در قطعات حافظه‌ی فلش طراحی شده است. JFFS2 از نسخه‌ی 2.4.10 لینوکس در آن گنجانده شده است. JFSS2 برای Open Firmware، eCos RTOS و بوت‌لودر Red Boot نیز در دسترس است.

سیستم‌فایل جدیدی با نام LogFS قصد جانشینی این سیستم‌فایل را دارد. ولی کاربرد آن در قطعات با حافظه‌ی بالا (بین 64 MB و 512 MB) است.

UBIFS ۱۲.۴

UBIFS [۱] یکی از جانشینان بعدی JFFS2 و رقیب LogFS است. مورد استفاده این سیستم فایل، در حافظه‌های فلش خام (و بدون قالب‌بندی) خاص است. طراحی و توسعه این سیستم فایل در اوایل سال 2007 شروع شد و اولین نسخه پایدار آزاد شده آن، برای Linux نسخه 2.6.27 در اکتبر سال 2008 بوده است.

توجه کنید که UBIFS ، از لحاظ طبقه بندی بر روی UBI [۲] کار می‌کند که آن‌ها نیز بر روی MTD ها کار می‌کنند. UBIFS در مقایسه با JFFS2 ، بر روی حافظه‌های فلش بزرگ‌تر، بهتر عمل می‌کنند.

---

[۱] Unsorted Block Image File System - UBIFS

[۲] Unsorted Block Image

UBIFS سیستم فایل ریشه و اصلی گوشی تلفن همراه هوشمند N900 شرکت نوکیا است. این سیستم فایل، از فشرده سازی بهنگام داده‌ها (از طریق zlib و یا LZO) نیز پشتیبانی می‌کند.

فصل سیزدهم

راه‌اندازی انواع موتورهای الکتریکی

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ راه‌اندازی انواع موتورهای پله‌ای، DC و AC.

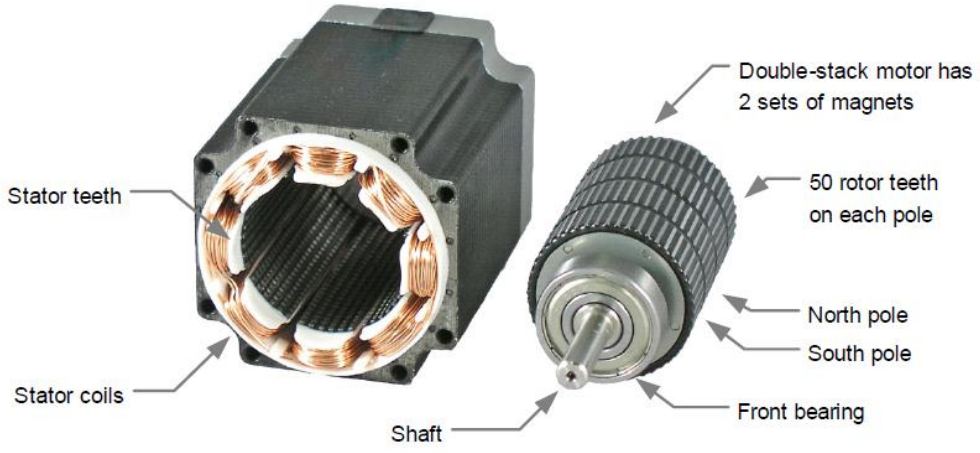
موتورهای الکتریکی، یکی از مهم‌ترین اجزای سیستم‌های الکترونیکی هستند. از سیستم‌های صنعتی و دستگاه‌های هواساز گرفته تا اسباب‌بازی‌های متحرک، روبات‌ها و... همگی از موتورها به عنوان عامل محرک بهره می‌برند. انواع مختلفی از موتورها وجود دارند که هر یک قابلیت یا مزیت خاصی را ارایه می‌کنند. از موتورهای پرکاربرد می‌توان به موتورهای پله‌ای[۱]، موتورهای AC القایی[۲]، موتورهای DC به همراه جاروبک[۳]، موتورهای DC بدون جاروبک[۴] و سروو موتورها اشاره نمود. هر یک از موتورهای نام برده، در جای خود کاربرد دارند و یکی را به جای دیگری نمی‌توان استفاده کرد. در ادامه به بررسی مشخصات و توانمندی‌های این موتورها می‌پردازیم. سپس با ارایه مثال، هر یک از موتورهای فوق را با میکروکنترولرهای توانمند ARM راه‌اندازی می‌کنیم. نقشه‌ی شماتیک و کد برنامه‌ی راه‌اندازی نیز ارایه شده‌اند. خواهیم دید که میکروکنترولرهای ARM ، چقدر در این زمینه توانا هستند.

۱۳.۱ موتورهای پله‌ای

موتورهای پله‌ای، موتورهای سنکرون DC هستند که با هربار تحریک، سیم‌پیچ‌های آن به **اندازه‌ی معینی** می‌چرخند. این اندازه‌ی معین یا همان پله‌ی چرخش، در انواع موتورهای پله‌ای رایج به میزان $0.9°$، $1.8°$، $7.5°$ و یا $15°$ هستند. به عنوان مثال، موتوری که در هر پله $1.8°$ می‌چرخد، در یک گردش کاملِ محورِ موتور به میزان 200 ($360°/1.8°=200$) پله می‌چرخد. نمونه‌ای از ساختار داخلی یک موتور پله‌ای را در شکل ۱۳.۱ می‌بینیم.

---

[۱] Stepper Motor
[۲] AC Induction Motor
[۳] Brushed DC Motor
[۴] Brushless DC Motor-BLDC



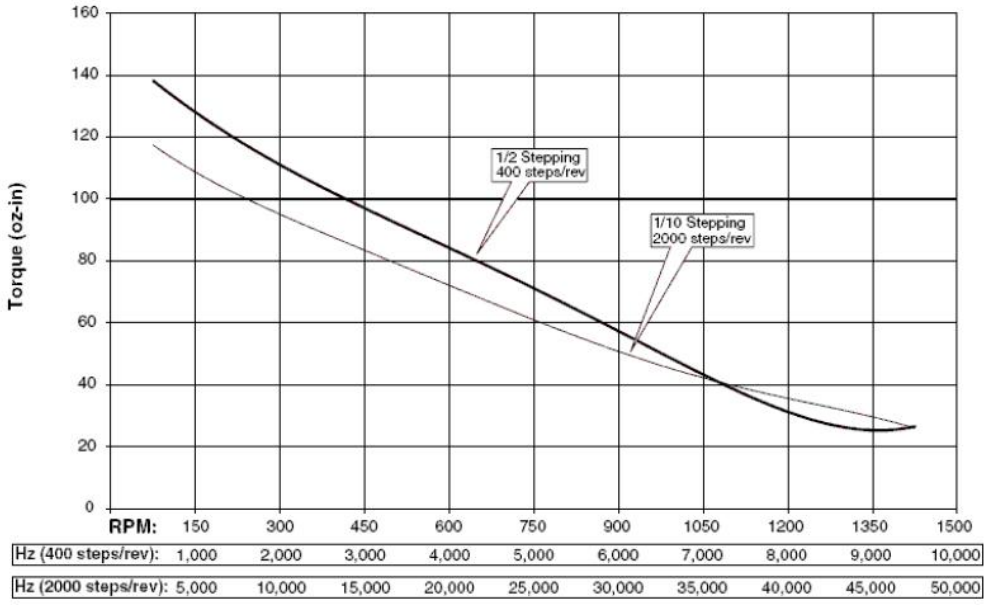
شکل ۱۳.۱

موتورهای پله‌ای در دو نوع **با آهن‌ربای دایم**[۱] و **بدون آهن‌ربای دایم**، ساخته می‌شوند. رایج‌ترین نوع موتور پله‌ای که امروز ساخته می‌شود، ترکیبی از مشخصات آهن‌ربای دایم و بدون آهن‌ربای دایم را با هم داراست. این موتورها را با نام "موتورهای هیبریدی یا مرکب" می‌شناسند.

قسمت چرخنده‌ی یک موتور پله‌ای هیبریدی (یا همان Rotor) از ماده‌ی مغناطیسی مناسبی ساخته شده که دارای ۲۰۰ دندانه است. این ماده‌ی مغناطیسی یک آهن‌ربای دایم قوی‌تر را نیز دربر گرفته است. وقتی یکی از فازهای استاتور (سیم‌پیچ ثابت موتور) تحریک می‌شود، قطب مخالف در دندانه‌ی بعدیِ روی روتور، جذب آن می‌شود و حرکت به میزان **یک پله** را موجب می‌شود.

نمودار ـ سرعت گشتاور موتور که از نمودارهای مهم هر موتور است. این نمودار برای موتورهای پله‌ای کمی از حالت خطی، انحراف دارد.

[۱] Permanent Magnet

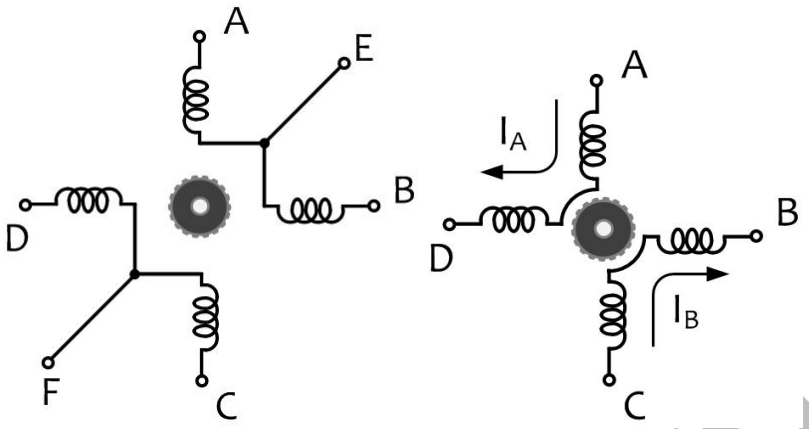


شکل ۱۳.۲

با افزایش سرعت، میزان گشتاور موتور به اندازه‌ی قابل ملاحظه‌ای کاهش می‌یابد. میزان گشتاور موتورهای پله‌ای همواره در حالت سکون، وقتی که محور موتور هیچ‌گونه حرکتی ندارد، سنجیده می‌شود. این گشتاور با نام **گشتاور ایستا** یا **گشتاور نگه‌دارنده**[۱] شناخته می‌شود.

موتورهای پله‌ای در دو نوع تک‌قطبی و دوقطبی به فروش می‌رسند. این تقسیم‌بندی، نحوه‌ی کارکردن با موتور را مشخص می‌کند. موتورهای تک‌قطبی دارای ۶ سیم برای راه‌اندازی هستند. این در حالی‌است که موتورهای دوقطبی دارای ۴ سیم هستند. در موتورهای تک‌قطبی گشتاور تولیدی نسبت به موتورهای دوقطبی کمتر است. دارای دو سیم‌پیچ با سر وسط بوده (در واقع ۴ سیم‌پیچ) که جهت جریان برای تحریک آن‌ها همواره در یک جهت است. در مقابل موتورهای دوقطبی دارای دو سیم‌پیچ هستند که برای چرخش، لزوماً جهت جریان بایستی عکس شود. برای روشن‌تر شدن موضوع، به شکل ۱۳.۳ توجه نمایید.

[۱] Holding Torque



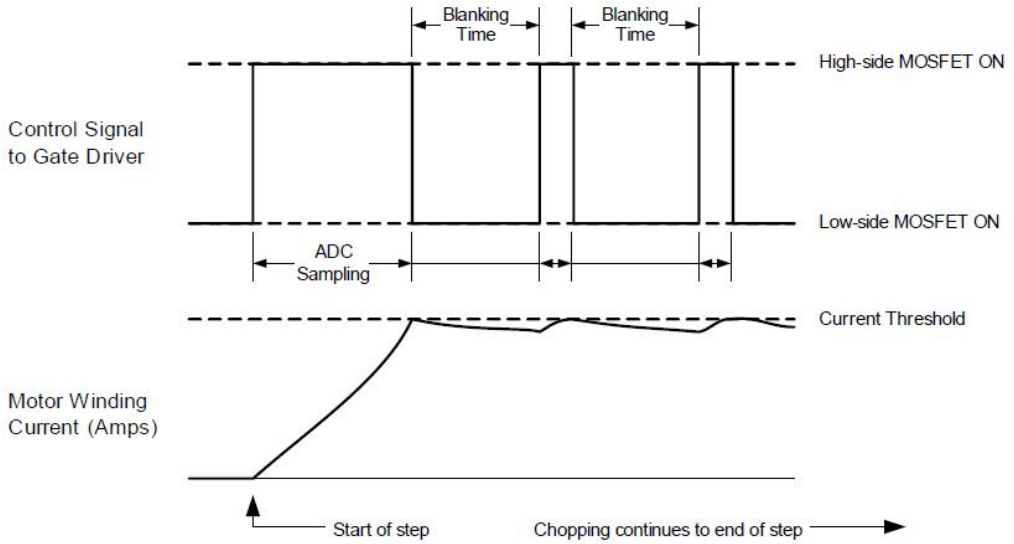
شکل ۱۳.۳– سمت چپ: موتور پله ای تک قطبی، سمت راست موتور پله ای دو قطبی

موتورهای پله‌ای، برای بازدهی بالا طراحی نشده‌اند. در عوض، برای قرار گرفتن در یک موقعیت خاص در مدار حلقه باز (بدون فیدبک) بسیار مناسب هستند. طراحی این موتورها به گونه‌ای‌است که در دماهای بالا به راحتی کار می‌کنند. با این حال، مقدار جریان موتور بایستی به گونه‌ای تنظیم شود که دمای محفظه‌ی موتور از $100°C$ بیشتر نشود. در غیر این‌صورت، موتور صدمه خواهد دید.

۱۳.۱.۱ کنترل موتور پله‌ای به روش Chopper

یک موتور پله‌ای، می‌تواند در ولتاژ اسمی تعیین شده‌ی خود کار کند. ولی، در این ولتاژ، جریان سیم-پیچ‌های موتور به آرامی افزایش می‌یابند. به همین دلیل، آغاز حرکت موتور به کندی صورت خواهد گرفت. برای غلبه بر این مشکل، از ولتاژ تغذیه‌ای بسیار بیشتر از مقدار مشخص شده برای موتور استفاده می‌کنیم. پس، جریان سیم‌پیچ‌ها به سرعت به مقدار دلخواه ما می‌رسند. در این هنگام، تغذیه را قطع می‌کنیم و جریان کاهش خواهد یافت. با کاهش جریان مجدداً عمل فوق را تکرار می‌کنیم. این مجموعه عملیات، سلسله‌ای از پالس‌های ولتاژ را به پایانه‌های موتور اعمال می‌کند[۱]. نتیجه‌ی این روش کنترل، سرعت گردش بیشتر (یا همان نرخ پله‌ی بیشتر) به همراه گشتاور بیشتر است. معمولاً، از ولتاژ تغذیه‌ای معادل ۵ برابر تا ۲۰ برابر تغذیه‌ی اصلی برای کنترل به روش Chopper استفاده می‌شود.

[۱] علت نام‌گذاری Chopper یا بریده شده، همین روش راه اندازی است



شکل ۱۳.۴

۱۳.۱.۲ تشریح مدار

برای راه‌اندازی موتور پله‌ای دوقطبی (که ۴ ترمینال دارد)، نیاز به ۴ مدار راه‌انداز با قابلیت جریان‌دهی مناسب داریم. در این‌جا از دو ماسفت[۱] نوع N ، برای هر ترمینال استفاده کرده‌ایم (یعنی ۸ ماسفت نوع N). به علت استفاده از روش Chopper، ولتاژ راه‌اندازی موتورها بالا است. پس، از ماسفت هایی با توانایی راه‌اندازی ولتاژ بالا (یعنی $V_{DSS} \geq 100V$)، باید استفاده شود. جریان راه‌اندازی موتورها از 2A بیشتر نخواهد بود. بنابراین از ماسفت هایی با جریان‌دهی بیش از 10A و ولتاژ V_{DSS} بیش از 100V، به راحتی می‌توان استفاده کرد.

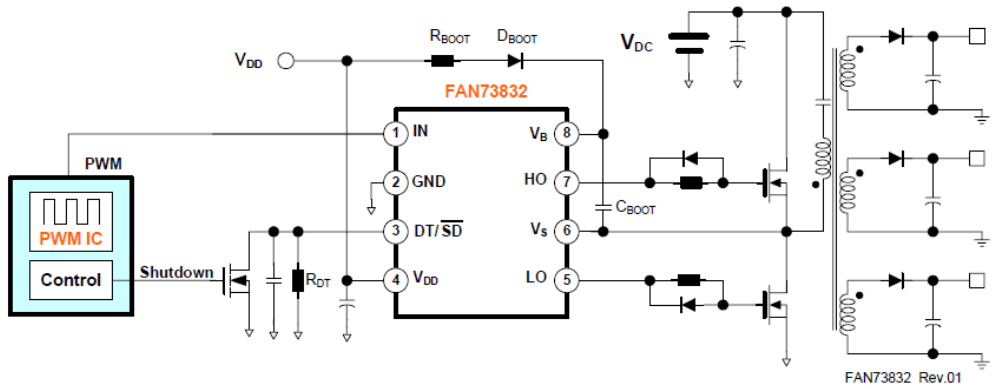
ماسفت ها را در این‌جا به صورت نیم‌پل[۲] می‌بندیم و برای راه‌اندازی‌شان از یک تراشه‌ی راه‌انداز استاندارد استفاده می‌کنیم. به عنوان نمونه، تراشه‌ی FAN73832 توانایی راه‌اندازی دو ماسفت به صورت نیم‌پل (HI-Side و LO-Side) را داراست و جریان راه‌اندازی آن‌ها را نیز فراهم می‌کند.

این تراشه، پایه‌ای برای کنترل میزان "زمان مرده"[۳] را نیز دارد. ترانزیستورهای Q1، Q2، Q7 و Q8 برای خاموش کردن FAN 73832 استفاده می‌شوند.

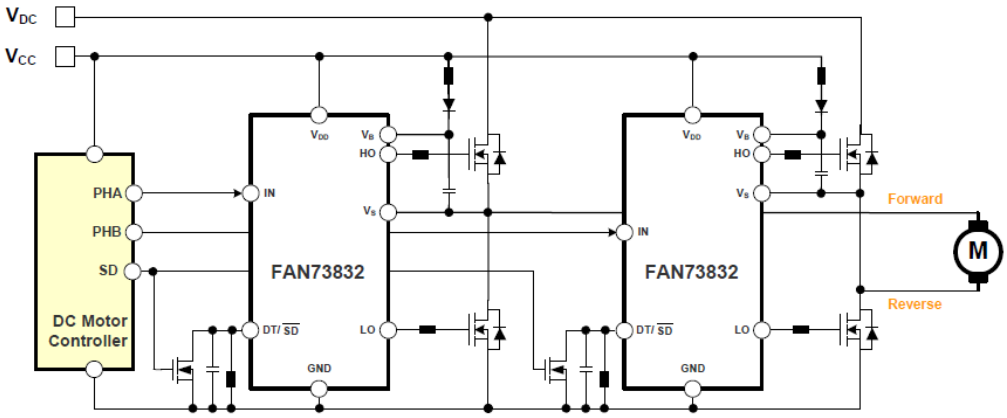
---

[۱] MOS Fet
[۲] Half Bridge
[۳] Dead Time

شکل ۱۳.۵

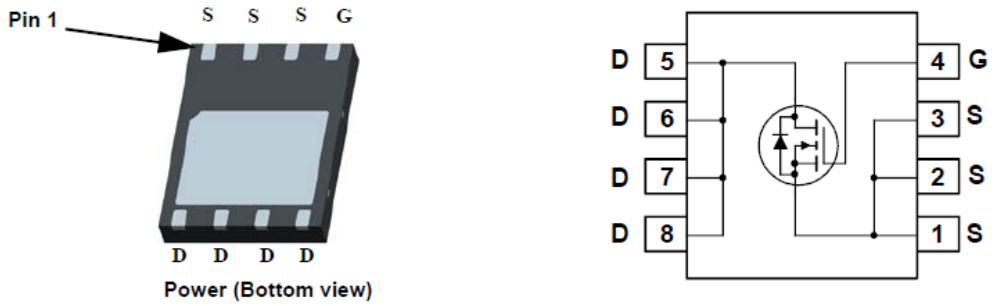


شکل ۱۳.۶



شکل ۱۳.۷

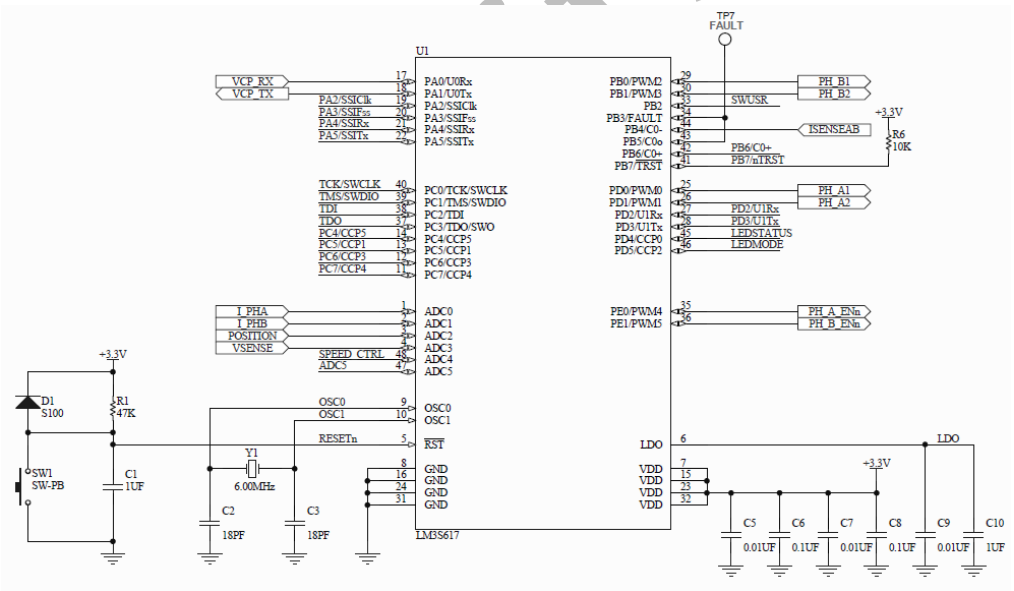
ترانزیستورهای راه‌انداز اصلی (Q6-Q3 و Q12-Q9) را FDMS3672 انتخاب کرده‌ایم. البته، می‌توان از هر نوع ترانزیستور با خواص مشابه نیز استفاده کرد.



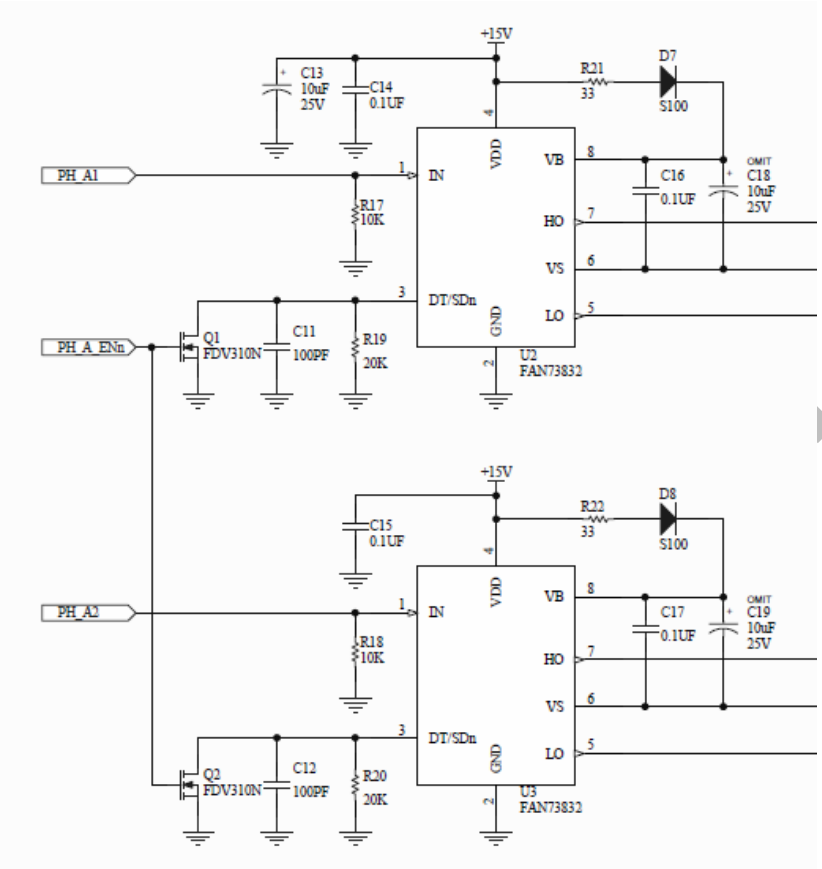
شکل ۱۳.۸

R27 و R42 ، به عنوان حسگر جریان مورد استفاده قرار گرفته‌اند. U5 و U8 ، وظیفه‌ی تقویت جریان (در واقع $I_{OUT} \times Rs$) را بر عهده دارند.

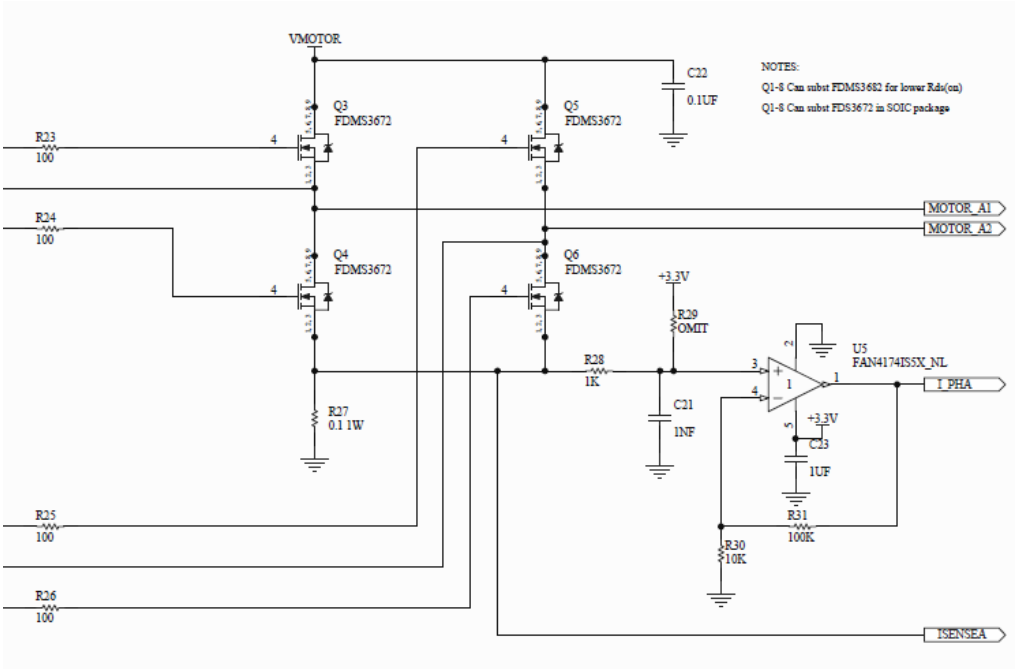
برای تولید سیگنال‌های PWM و دریافت ورودی‌های حس شده‌ی جریان، از میکروکنترولری با ۴ کانال خروجی PWM و چند ورودی ADC می‌توان استفاده کرد. در این‌جا، از میکروکنترولر STM32F103 استفاده کرده‌ایم.



شکل ۱۳.۹


+15V
C13
10uF
25V
C14
0.1UF
R21
33
D7
S100
PH_A1
IN
VDD
VB
HO
VS
DT/SDn
GND
LO
R17
10K
C16
0.1UF
OMIT
C18
10uF
25V
Q1
FDV310N
C11
100PF
R19
20K
U2
FAN73832
+15V
C15
0.1UF
R22
33
D8
S100
PH_A2
R18
10K
C17
0.1UF
OMIT
C19
10uF
25V
Q2
FDV310N
C12
100PF
R20
20K
U3
FAN73832

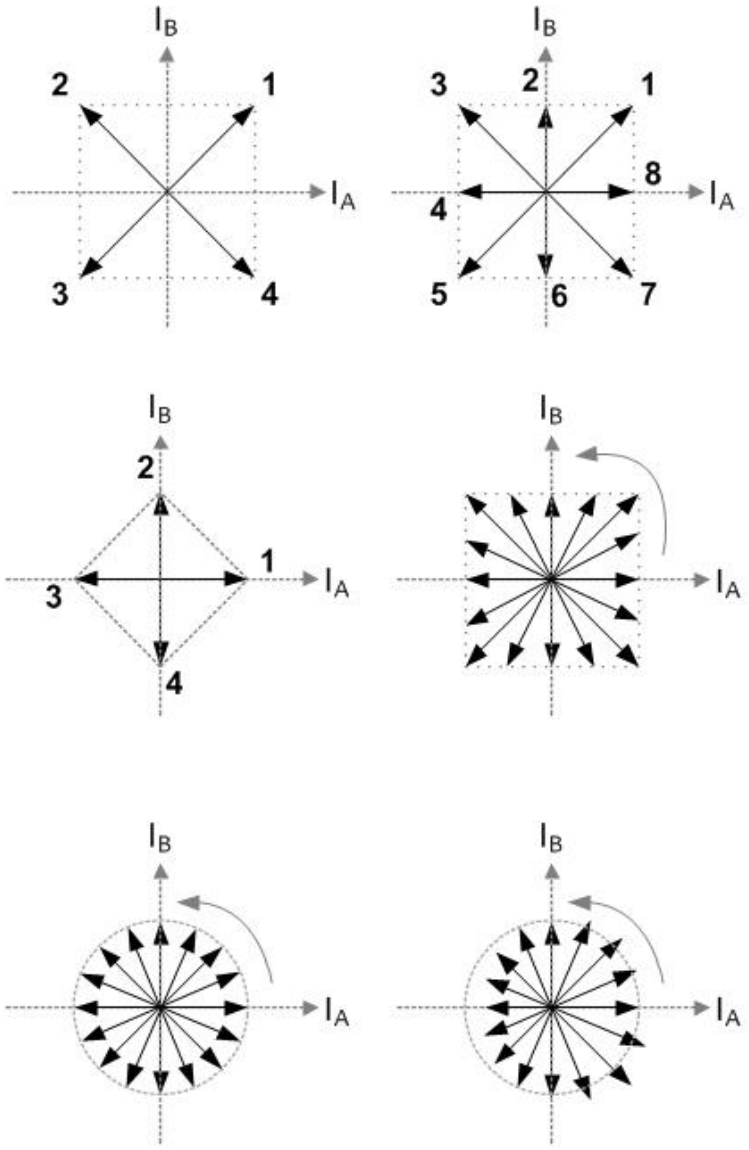
شکل ۱۳.۱۰



شکل ۱۳.۱۱

۱۳.۱.۳ مبانی حرکت پله‌ای نیم‌پله

در موتور پله‌ای، با چرخش جریان در سیم‌پیچ‌های مختلف، می‌توان به حرکت‌های مختلفی دست پیدا کرد. در شکل ۱۳.۱۲، نمونه‌ای از حرکت‌های مختلف را می‌بینیم. در این شکل، با تغییر جریان یک موتور دوقطبی (I_A و I_B) و با درنظر گرفتن علامت و جهت جریان، حرکت‌های مختلف حاصل شده است.



شکل ۱۳.۱۲ – در ردیف بالا، شکل سمت چپ حرکت تمام پله[۱] را داریم که در آن محور موتور در هر مرحله فقط یک پله حرکت می‌کند. شکل سمت راست، حرکت نیم‌پله[۲] را نشان می‌دهد. با تغییر جریان به نحوی که در شکل آمده است، در هر مرحله محور موتور فقط نیم‌پله حرکت می‌کند.

---

۱ Full Step

۲ Half Step

ردیف وسط، شکل سمت چپ، نشان دهنده‌ی حرکت نوع پله‌ی موجی[۱] است. این حرکت، نوعی حرکت تمام پله است. با این تفاوت که در این‌جا در هر لحظه فقط یکی از سیم‌پیچ‌ها حاوی جریان است. جریان مصرفی در این‌جا کاهش یافته ولی گشتاور نیز به همان اندازه کاهش دارد. شکل سمت راست، حرکت ریز- پله است[۲]. در این نوع حرکت، جریان سیم‌پیچ‌ها مقادیری به غیر از $0, \pm I_{MAX}$ نیز اتخاذ می‌کنند. بدین معنا که بسته به تعداد حرکت‌هایی که بین پله‌های اصلی نیاز است، مقدار مطلق جریان هر سیم‌پیچ می‌تواند بین 0 و I_{MAX} نیز باشد.

مسیر حرکت (مسیر حرکتِ فازور جریان یا گشتاور) این گردش ریزپله‌ای از نوع مربعی است.

در شکل‌های پایین، همان حرکت ریزپله را با مسیرهای دایره‌ای و یک مسیر دلخواه می‌بینیم. هرچند هر مسیر دلخواهی، همواره کاربردی نیست، اما در این‌جا قصد داریم مفهوم ریزپله را توضیح دهیم.

در صفحه‌ی جریان داده شده، هر بردار (یا فازور) نشان دهنده‌ی توان و یا جریان مصرفی هر سیم‌پیچ در هر یک از مراحل حرکتی است. یعنی، در هر مرحله، طول هر بردار نشان دهنده‌ی جریان مصرفی است. از آن‌جایی که گشتاور موتور با جریان مصرفی آن متناسب است (البته تا وقتی که به اشباع مغناطیسی نرسیده باشیم):

$$\boldsymbol{Torque} \propto \boldsymbol{I}$$

$$\boldsymbol{Power} = R \times (I_A{}^2 + I_B{}^2)$$

در این‌جا، R مقاومت اهمی هر سیم‌پیچ و P توان مصرفی است. در مورد طول هر بردار، داریم:

$$Phasor\ Length = \sqrt{I_A{}^2 + I_B{}^2} = \sqrt{\frac{\boldsymbol{Power}}{R}} \propto \boldsymbol{Torque}$$

از آن‌جایی که طول هر بردار متناسب است با گشتاور و گشتاور متناسب است با مجذور توان، برای افزایش دو برابری گشتاور موتور، توان مصرفی را باید چهار برابر کنیم.

از مباحث بالا نتیجه می‌گیریم که برای داشتن حرکت نرم و برخورداری از حداقل ریپل در گشتاور موتور، بهتر است مسیر حرکتی انتخاب شود که در آن تمامی بردارها دارای اندازه‌ی یکسانی باشند.

به عنوان مثال، حرکت در مسیر دایره‌ای بهترین نوع حرکت ریزپله‌ای است. این نوع حرکت دایره‌ای را با نام حرکت "ریزپله‌ی سینوسی- کسینوسی[۳] " نیز می‌شناسند. زیرا، هر یک از جریان‌های I_A و I_B ، دارای مقادیر سینوسی شکل هستند.

---

[۱] Wave Step

[۲] Micro Stepping

[۳] Sine-Cosine Microstepping

$$I_A{}^2 + I_B{}^2 = \boldsymbol{r}$$

$$\boldsymbol{I_A} = r \times sin\,\theta$$

$$\boldsymbol{I_B} = r \times cos\,\theta$$

برای تولید جریان دلخواه در نواحی بین 0 و IMAX باید از یک مبدل دیجیتال بـه آنـالوگ اسـتفاده کـرد. زیرا با استفاده از کلیدهای قطع و وصل جریان (در اینجا همان MOSfet ها)، نمی‌توان به ایـن مقـادیر دست یافت. استفاده از یک چنین مبدلی در عمل ممکن نیسـت. بـرای ایـن‌کـار، در راه‌انـدازی هـر یـک از کلیدهای جریان از شکل موج PWM استفاده می‌کنیم. با استفاده از ایـن روش و راه‌حلـی کـه در بخـش مربوط به Chopper گفته شد، می‌توان به جریان مذکور دست یافت.

۱۳.۲ موتورهای القایی

یک موتور القایی[۱] یا موتور آسنکرون[۲]، نوعی از موتورهای AC است که در آن تغذیه‌ی روتـور توسـط القای الکترومغناطیسی صورت می‌گیرد. علت چرخش محور موتور همین میدان‌های مغناطیسـی موجـود است. انواع مختلف موتورها با استفاده از نحوه‌ی تغذیه‌ی روتور شناخته می‌شوند.

به عنوان مثال، در موتورهای DC، تغذیه‌ی روتور توسط اتصالات الکتریکـی بـه نـام کموتـاتور[۳] انجـام می‌شود. در مقایسه با مدارهای DC، در موتورهای AC توان روتور توسـط القـای الکترومغناطیسـی از سمت استاتور به روتور، تأمین می‌شود.

موتور القایی را گاهی با نام "ترانسفورماتورِ چرخنده" می‌شناسند. زیرا در اینجا استاتور نقش اولیـه‌ی ترانسفورماتور را بازی می‌کند و روتور چرخنده نیز به عنوان ثانویه‌ی آن، نقش القاشونده‌ی مغناطیسی را بازی می‌کند.

برخلاف ترانسفورماتورها که از شار تغییر یابنده برای تغییر جریان استفاده می‌کنند، موتورهای القـایی از شار مغناطیسی چرخنده برای تبدیل ولتاژ استفاده می‌کنند. جریان اولیه در ایـن‌جـا تولیـد یـک میـدان الکترومغناطیسی کرده کـه در تعامـل بـا میـدان الکترومغناطیسـی ثانویـه، تولیـد گشـتاور مـی‌کننـد. در این‌جاست که انرژی الکتریکی به انرژی مکانیکی تبدیل می‌شود.

موتورهای القایی در صنعت، کاربرد گسترده‌ای دارند. ایـن اسـتقبال گسـترده بـه علـت سیسـتم سـاخت محکم و مقاوم، نبود جاروبک و وجود کنترل کننده‌های سرعت مدرن است.

---

۱ Induction Motor

۲ Asynchronous

۳ Commutator

در این‌جا، در مورد روش‌های گوناگون ساخت این موتورها بحث نخـواهیم کـرد و فقـط بـه نمونـه‌ای از راه‌اندازی موتورهای القایی سه فاز، خواهیم پرداخت.

هشدار

خطر شوک الکتریکی

میکروکنترولر به کار رفته در این قسمت، در بخش مربوط به پتانسیل AC قرار گرفته است. هیچ گونه ارتباط مستقیمی با JTAG و سایر مدارات مرتبط با میکروکنترولر برقرار نکنید و **از لمس آن‌ها جداً پرهیز کنید**.

سرعت موتور القایی توسط فرکانس جریان AC ورودی و تعداد قطب‌های آن (مربوط به طراحی داخلی) مشخص می‌شود.

رابطه به این صورت بیان می‌شود:

$$\textbf{\textit{Synchronous Speed}} = 120 \times \frac{Frequency}{Number\ of\ Poles}$$

وقتی روتور دارای بار باشد (همواره مقداری بار وجود دارد)، موتور بایـد گشـتاور لازم را تولیـد کنـد. گشتاور نیاز دارد که روتور، از میدان مغناطیسی استاتور آهسته‌تر بچرخد. این تفاوت در سـرعت را بـا نام لغزش[1] می‌شناسیم. به عنوان مثال، موتوری که دارای سرعت سنکرون 3600 rpm اسـت، در بـار کامل ممکن دارای سرعت روتور 3350 rpm باشد.

مثالی از یک موتور دوقطبی را درنظر بگیرید که با فرکانس 0 تا 340 Hz کار می‌کند. محور این موتور می‌تواند سرعتی معادل 0 تا 20400 rpm داشته باشد.

روش کنترلی که در این‌جا استفاده می‌شود، تغییر فرکانس جریان سیم‌پیچ استاتور اسـت. چنـدین روش مدولاسیون ولتاژ دیگر نیز وجود دارند، اما همگی یک جریان استاتور متناوب سینوسی را تولید می‌کنند.

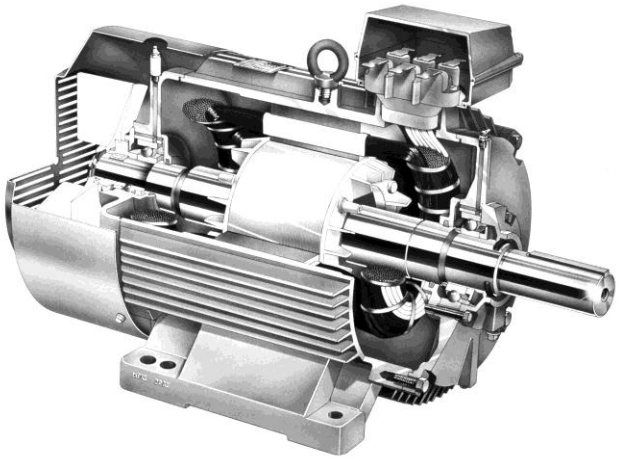
انواع مختلفی از تقسیم‌بندی موتورهای القایی وجود دارند. یکی از مهم‌ترین این تقسیمات براساس تعـداد فاز است. موتورهای تک‌فاز و سه فاز، از معروف‌ترین این موتورها هستند. موتورهای سه فاز بـه علـت بازدهی بیشتر، گشتاور بالاتر و محدوده‌ی سرعت وسیع‌تـر نسـبت بـه موتورهـای تـک‌فـاز، محبوبیـت بیشتری دارند. موتورهای سه فاز، تناسب خوبی با روش کنترل فرکانس دارند.

مدل‌های مختلفی از موتورهای تک‌فاز وجود دارند که با این روش به خـوبی کنتـرل مـی‌شـوند. دو نـوع موتور Permanent Split Capacitor (PSC) و موتورهای Shaded Pole در این‌جا مـی‌تواننـد استفاده شوند. موتور یک دهنده‌ی هوا که از نوع PSC است را در زیر می‌بینیم.

---

[1] Slip

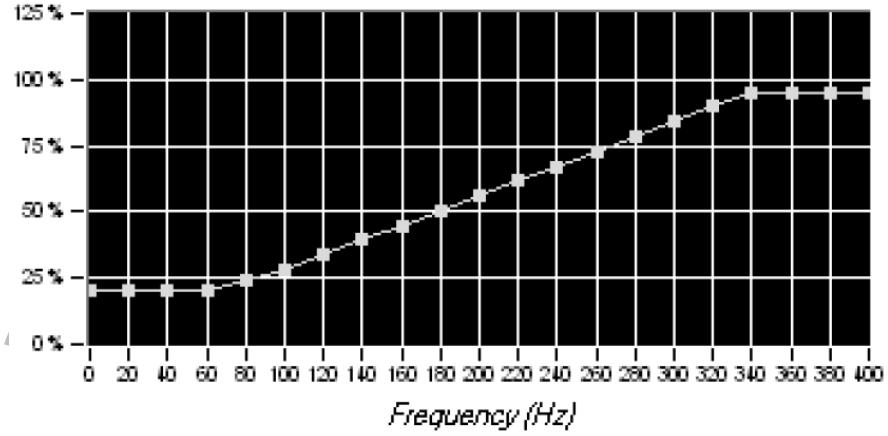

ZD
INDUCTION
MOTOR
25W
0.25A
220V
3Φ
1300rpm
50Hz
Cont
2010/12
www.zd-motor.com



شکل ۱۳.۱۳ – ACIM-1

۱۳.۲.۱ تغییر سرعت

کاهش ولتاژ به‌منظور افزایشِ میزانِ لغزش، روش مناسبی برای کنترل سرعت نیست. دلیل این امر کاهش گشتاور موتور است. تغییر فرکانس، روش مؤثرتری است. البته، باید به چند نکته توجه داشت. با کاهش فرکانس، امپدانس مؤثر موتور نیز کاهش می‌یابد. برای حفظ جریان و گشتاور ثابت، باید از تابع تبدیل VIF نمودار زیر استفاده کنیم.



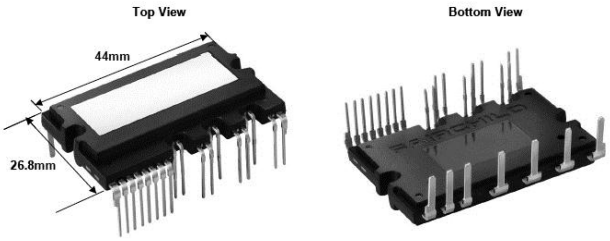
شکل ۱۳.۱۴

این نمودار، یک موتور 340 Hz را نشان می‌دهد. توجه کنید که ولتاژ، تا زمانی که فرکانس استاتور به فرکانس نامی برسد، افزایش می‌یابد.

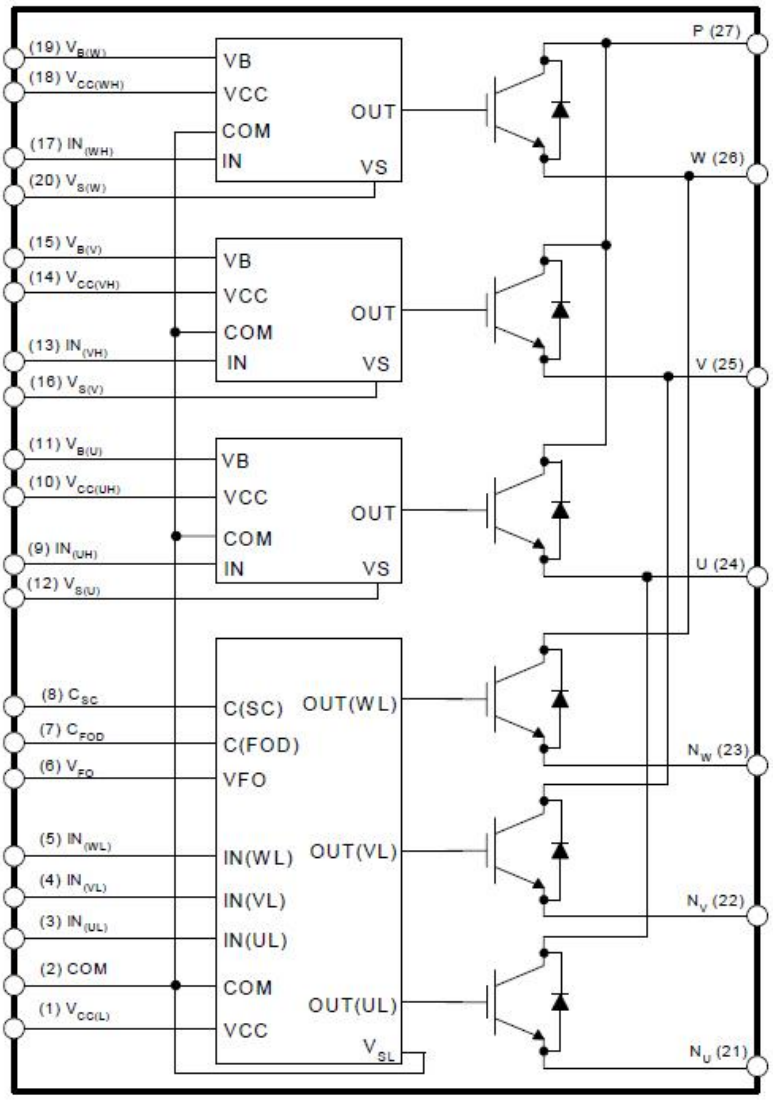
در مدار استفاده شده، می‌توان از ولتاژ VAC 230 استفاده کرد. جریان مصرفی حداکثر راه‌انداز موتور به A 10 می‌رسد. ایزولاسیون ولتاژ بین خروجی و ورودی، Vrms 2500 می‌باشد.
فرکانس مدار بین Hz 0-400 و با پله‌های Hz 0.1 قابل تنظیم است. فرکانس PWM را نیز می‌توان بین KHz 8-20 تغییر داد.

۱۳.۲.۲ توضیح سخت‌افزار

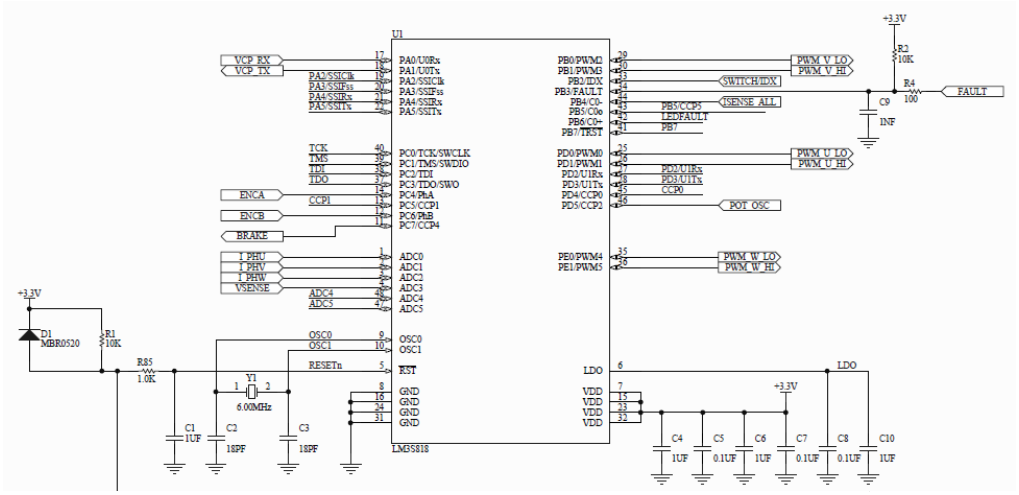
در این مدار، از یک میکروکنترولر LPC1768 با هسته‌ی ARM CortexM3 برای کنترل موتور استفاده شده است. قسمت راه‌انداز پرقدرت خروجی با استفاده از ماژول قدرت FSBS10CH60 ، ساخته شده است.
محافظت، شکل‌دهی سیگنال، حسگرهای مختلف و منبع تغذیه از دیگر مدارهای جانبی ارایه شده هستند.
میکروکنترولر استفاده شده دارای یک بخش کنترل موتور با توانایی تولید سیگنال‌های PWM سه فاز نیز هست که توانایی کنترل بیشتری را در اختیار می‌گذارد. در این مدار، از یک منبع تغذیه‌ی سه خروجی نیز استفاده شده است که منابع تغذیه‌ی لازم برای کارکرد صحیح مدار را فراهم می‌آورد. از این خروجی‌ها، تنها منبع 5V+ ایزوله است. منبع مزبور از تراشه‌ی FSD200 استفاده می‌کند که مداری کامل برای تبدیل ولتاژ برق شهر به ولتاژهای DC است (تکنولوژی سوییچینگ Offline) .
برای ایزوله کردن قسمت‌های JTAG و رایانه‌ی شخصی از بقیه‌ی مدار، از اپتوکوپلرهای FOD2200 و H11L1M استفاده شده است.
در مدار قدرت، R31-R33 نقش حسگرهای جریان را بازی می‌کنند و U2، U4 و U5 تقویت کننده‌های جریان هستند. ورودی‌های PWM-V، PWM-U و PWM-W به صورت جفت‌های راه‌انداز بالا (HI) و پایین (LO) مورد استفاده قرار گرفته‌اند (یعنی جمعاً ۶ ورودی PWM).
تراشه‌ی راه‌انداز قدرت FSBS10CH60 یک ماژول هیبرید چند لایه بود که برای راه‌اندازی موتور سه فاز مورد استفاده قرار گرفته است. این تراشه، دارای سه جفت IGBT به همراه مدار راه‌اندازشان است که در یک بسته‌بندی کوچک قرار گرفته است.



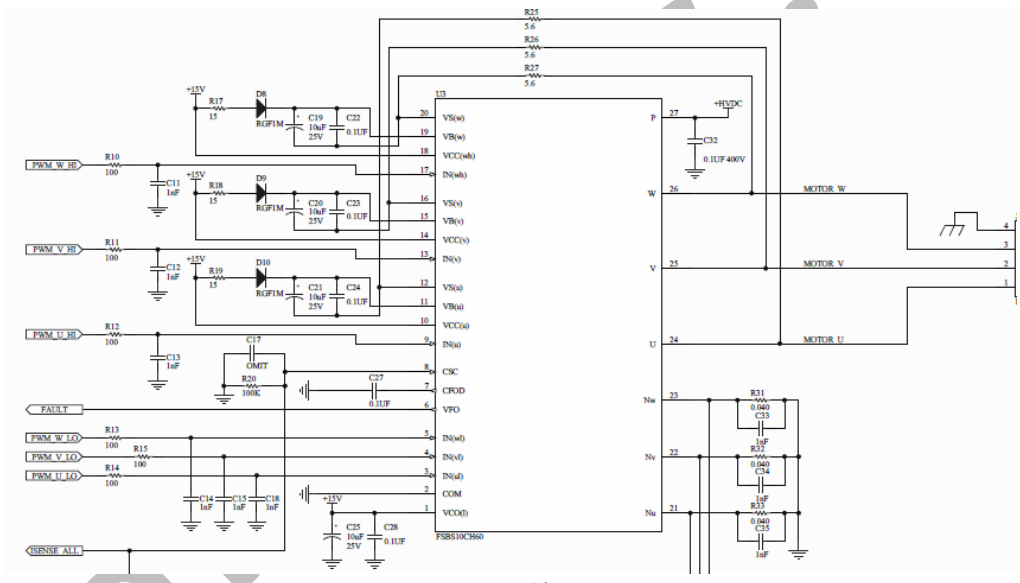
شکل ۱۳.۱۵


(19) $V_{B(W)}$
(18) $V_{CC(WH)}$
(17) $IN_{(WH)}$
(20) $V_{S(W)}$
(15) $V_{B(V)}$
(14) $V_{CC(VH)}$
(13) $IN_{(VH)}$
(16) $V_{S(V)}$
(11) $V_{B(U)}$
(10) $V_{CC(UH)}$
(9) $IN_{(UH)}$
(12) $V_{S(U)}$
(8) C_{SC}
(7) C_{FOD}
(6) V_{FO}
(5) $IN_{(WL)}$
(4) $IN_{(VL)}$
(3) $IN_{(UL)}$
(2) COM
(1) $V_{CC(L)}$
VB
VCC
COM
IN
OUT
VS
C(SC)
C(FOD)
VFO
IN(WL)
IN(VL)
IN(UL)
COM
VCC
OUT(WL)
OUT(VL)
OUT(UL)
V_{SL}
P (27)
W (26)
V (25)
U (24)
N_W (23)
N_V (22)
N_U (21)

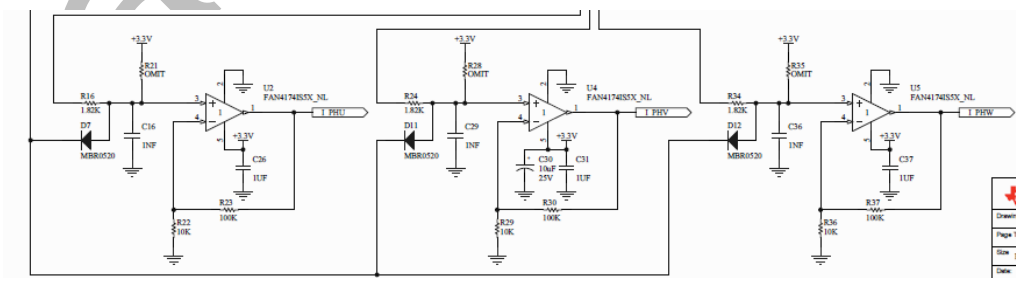
شکل ۱۳.۱۶



شکل ۱۳.۱۷



شکل ۱۳.۱۸



شکل ۱۳.۱۹

۱۳.۳ راه‌اندازی موتورهای بدون جاروبک DC[۱]

کموتاسیون موتورهای بدون جاروبک، الکترونیکی بوده و دارای آهن‌ربای دایم هستند.
این موتورها دارای عمر طولانی بوده، مشخصات گشتاوری خوب داشته و عملکردی با بازدهی بالا دارند. موتورهای بدون جاروبک، در موارد زیادی از جمله صنایع خودرویی، اتوماسیون و هچنین در خنک کننده‌های CPU رایانه‌ی شخصی نیز، به کار گرفته می‌شوند.

۱۳.۳.۱ ساختار داخلی

همان‌طور که در شکل ۱۳.۲۰ آمده است، استاتور این موتور دارای یک قاب و سیم‌پیچی از سیم‌های مسی است. در یک موتور بدون جاروبک، روتور شامل یک یا چند آهن‌ربای دایم است. روتور می‌تواند داخل و یا خارج از استاتور قرار داشته باشد. روتورهای داخلی، از آهن‌رباهایی با یک یا چند جفت قطب استفاده می‌کنند. روتورهای خارجی، اغلب به شکل آهن‌رباهای دایره‌ای هستند که می‌توانند تعداد بیشتری قطب داشته باشند. این به معنای گشتاور بیشتر است.

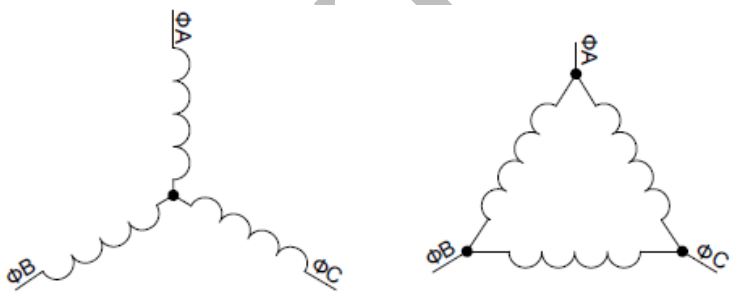
---

[۱] Brush Less DC-BLDC

شکل ۱۳.۲۰

شکل ۱۳.۲۱

بیشتر موتورهای بدون جاروبک، سه فاز هستند. یک موتور بدون جاروبک DC سه فاز، سه سیم‌پیچ دارد که هر یک از آن‌ها بر روی دو یا چند شیار استاتور توزیع شده‌اند. سیم‌پیچ‌ها می‌توانند به هر یک از اشکال Yو یا Δ به یکدیگر متصل شده باشند.



شکل ۱۳.۲۲

طریقه‌ی اتصال Y رایج‌تر است. هرچند، از لحاظ مشخصات الکتریکی، هر دو نحوه‌ی سیم‌بندی یکسان هستند.

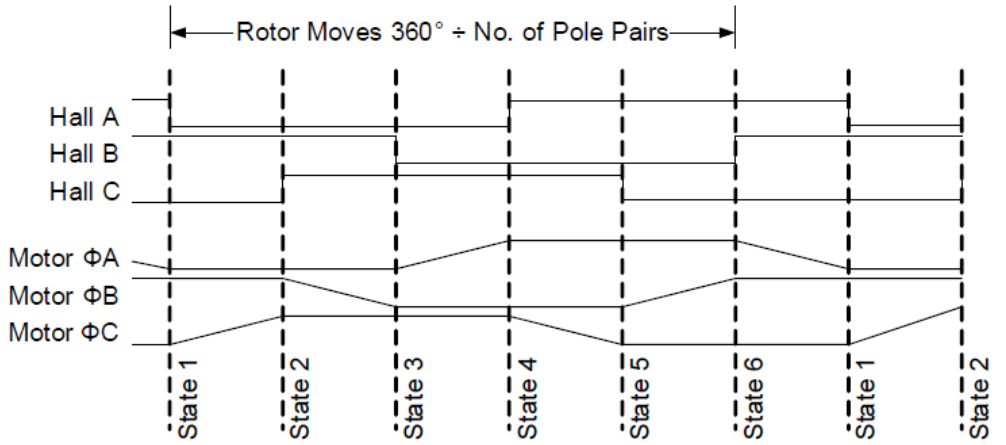
۱۳.۳.۲ کموتاسیون

موتور، با اتصال ولتاژ به دو سر پایانه‌های آن به راه می‌افتد. اتصال ولتاژ باعث عبور جریان از سیم-پیچ آن می‌شود. با عبور دادن جریان از سیم‌پیچ های مختلف موتور به صورت پشت سرهم، باعث چرخش محور موتور می‌شویم. موتور BLDC یک موتور سنکرون است. این بدان معنا است که اگر

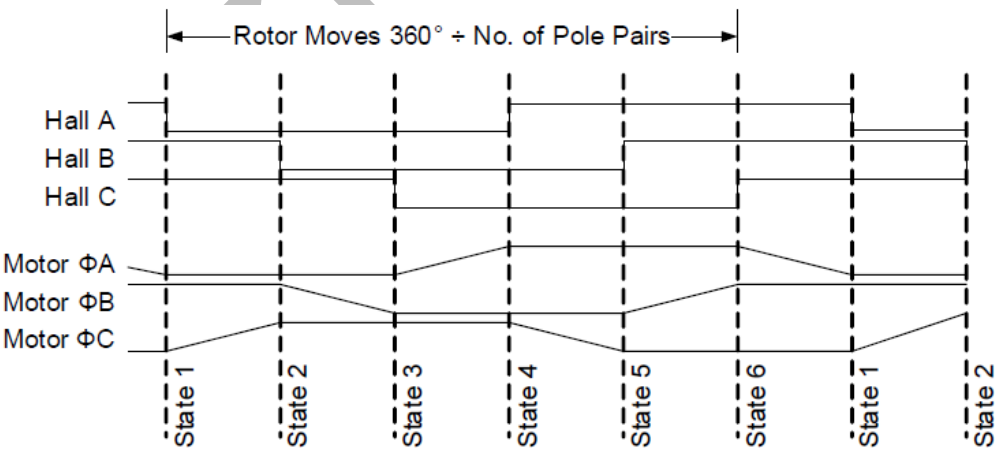
نحوه‌ی راه‌اندازی صحیح باشد، سرعت چرخش محور موتور با سرعت تحریک سیم‌پیچ‌های آن متناسب است. برای حفظ این هماهنگی[1]، در یک محدوده‌ی گشتاور/ سرعت، موقعیت روتور باید بازرسی شود. این کار از طریق یک سیستم فیدبک صورت می‌گیرد. دو سیستم فیدبک معروف سنسورهای اثرهال و انکودرهای نوری هستند.

در ساده‌ترین حالت، کموتاسیون موتور BLDC، ۶ مرحله دارد. این حالت‌های مختلف در جدول BLDC-1 نشان داده شده‌اند.

استفاده از سنسورهای اثر هال برای کاربردهایی که در آن گشتاور ثابت مورد نیاز است، ضروری است. سازمان داخلی سنسورهای اثرهال می‌توانند برای زوایای $60°$ و یا $120°$ تنظیم شده باشند. در شکل‌های زیر، رابطه‌ی بین کموتاسیون و سنسورها را در هر دو حالت مشاهده می‌کنید.



شکل ۱۳.۲۳



[1] Synchronity

شکل ۱۳.۲۴

سنسورهای اثرهال، عموماً دارای خروجی دیجیتال و درین باز[۱] هستند. روشی دیگر برای حس کردن موقعیت خروجی به نام Back-EMF وجود دارد که در آن نیازی به حسگر نیست. روش کار در این‌جا استفاده از میزان ولتاژ القا شده در **فاز غیرفعال** است. در شکل‌های ۱۳.۲۳ و ۱۳.۲۴، فاز غیرفعال با شیب‌های بالا و یا پایین‌رونده مشخص شده‌اند. خطوط شیب‌دار، تقریبی از ولتاژ القا شده در سیم‌پیچ‌ها هستند. این اثر را با نام Back-EMF می‌شناسیم. در عمل می‌توان از یک مقایسه کننده آنالوگ و یا از ADC میکروکنترولر، برای پی بردن به لحظه‌ی عبور از صفر استفاده کرد.

۱۳.۳.۳ سخت‌افزار مورد استفاده

در این‌جا نیز از روش‌های قبل برای راه‌اندازی هر یک از فازهای موتور استفاده کرده‌ایم. یک تراشه‌ی راه‌انداز گیت (FAN7382) و دو N-MOSfet (FDD13AN06A0) برای سوییچ شکل موج ولتاژ و R51-R53 ، در این‌جا حسگرهای جریان هستند.

در این‌جا، از مدار مشتمل بر Q1 و Q2 و R14 برای ایجاد ترمز استفاده کرده‌ایم. با فعال شدن سیگنال Breakn ، مقاومت R14 باعث کاهش V_{MOTOR} شده و به نوعی ترمز ایجاد می‌کند.

---

[۱] Open Drain

فصل چهاردهم

ارتباط با شبکه اترنت

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ شبکه‌ی اترنت و جزییات آن چیست؟
- ✓ شبکه‌های LAN و WAN چیست؟
- ✓ IP و TCP چیست؟

با گسترش روز افزون شبکه های کامپیوتری به ویژه اینترنت، این روزها کمتر رایانه ای یافت می شود که به چنین شبکه هایی دسترسی نداشته باشد. ارزش ارتباط با شبکه تا آن‌جاست که امروزه تمامی وسایل خانگی، تلفن های همراه و سایر دستگاه های کوچک و بزرگ تمایل به استفاده از آن را دارند. همه ما در خانه هایی زندگی می کنیم که کم و بیش مجهز به دستگاه های نیمه هوشمند تا کاملا هوشمند هستند؛ ولی آیا تا به حال فکر کرده اید اگر یخچال به اینترنت متصل شود و موجودی یا کمبود های خود را به سوپر مارکت محل اعلام کند، چه خواهد شد ؟ اگر کرکره های خانه به طور خودکار با روشنایی روز، میزان روشنایی خانه را کنترل کنند، چه ؟ کنترل دستگاه‌ها در خانه های هوشمند، به طور هوشمند و تحت کنترل یک سیستم مرکزی صورت می گیرد. مثال اول، مورد استفاده ای از شبکه اینترنت، و مثال دوم استفاده‌ی شبکه محلی (LAN یا WLAN) را در وسایل پر کاربرد روزمره تشریح می کردند.

این‌ها، تنها مثال های کوچکی از کاربردهای شبکه های کامپیوتری در دستگاه‌های تعبیه شده بودند. اما به عنوان مثال، از سایر کاربردها می توان به : کنترل و مانیتورینگ وسایل دور و نزدیک، بررسی میزان توان مصرفی، نظارت بر صحت عملکرد و پارامترهای محیطی همانند دما، رطوبت و... ، ارسال گزارش از وضعیت های گوناگون، اتصال وسایل گوناگون به اینترنت برای گرفتن فایل‌های ضروری و یا قرار دادن فایل‌های مورد نظر بر روی یک سایت و ... اشاره کرد.

استفاده از شبکه در صنعت نیز بسیار رواج دارد. در کنترل کننده های صنعتی PLC و پروتوکل‌های معروفی چون MODBUS ، استفاده از شبکه های اترنت، به چند دلیل رایج است:

- شبکه اترنت، به عنوان پل ارتباطی مقاوم در برابر نویز مطرح است. دلیل آن، استفاده از سیگنالینگ تفاضلی و سایر مشخصات الکتریکی است که برایش در نظر گرفته شده است؛
- طول کابل‌های ارتباطی شبکه، را می تواند بالغ بر چند صد متر باشد. با استفاده از تجهیزات کمکی نیز می توان این طول را افزایش داد؛

- امنیت این سیستم، هم در نرم افزار و هم در سخت افزار به خوبی تامین شده است.
- برنامه های گوناگون و پروتوکل‌های متفاوتی برای استفاده از این بستر فراهم آمده است.

در ادامه این فصل، در مورد شبکه ها و جزییات آن‌ها بیشتر خواهیم آموخت. نحوه‌ی اتصال میکروکنترولرها به رابط‌های شبکه را بررسی کرده و بر طرق مختلف ارتباط سیستم های تعبیه شده با شبکه های LAN و WLAN و Internet ، نگاهی خواهیم انداخت.

۱۴.۱ شبکه اترنت[۱] چیست؟

دستیابی به اطلاعات با روش های مطمئن و با سرعت بالا، یکی از رموز موفقیت هر سازمان و موسسه است .طی سالیان اخیر، هزاران پرونده و کاغذ که حاوی اطلاعات با ارزش برای یک سازمان بوده ، در کامپیوتر ذخیره شده اند. با ورود انبوهی از اطلاعات به کامپیوتر ، امکان مدیریت الکترونیکی اطلاعات فراهم شده است و کاربران مختلفی در سراسر جهان قادر به اشتراک اطلاعات هستند.

شبکه های کامپیوتری جهت نیل به اهداف فوق، نقش بسیار مهمی را ایفاء می نمایند. امروزه اینترنت، به عنوان عالی ترین نمود یک شبکه کامپیوتری در سطح جهان، کاربرد گسترده ای دارد. بسیاری از شرکت‌ها و کارخانه ها محصولات تولیدی و یا خدماتشان را از این طریق در اختیار استفاده کنندگان قرار می دهند. وب[۲] که عالی ترین سرویس خدماتی اینترنت می باشد، کاربران را قادر می سازد که در دورترین نقاط دنیا اقدام به خرید، آموزش ، مطالعه و ... نمایند.

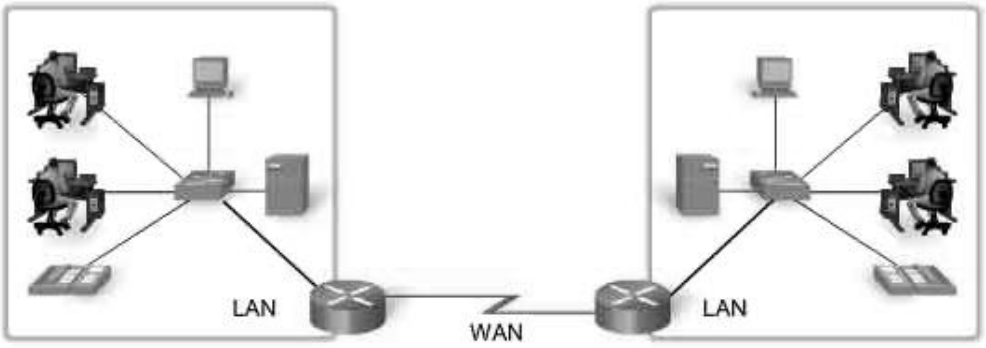
با استفاده از شبکه، یک کامپیوتر(امروزه حتی دستگاه‌های قابل حمل) قادر به ارسال و دریافت اطلاعات از کامپیوتر(یا دستگاه) دیگر است . اینترنت، نمونه ای عینی از یک شبکه کامپیوتری است که در آن میلیون ها کامپیوتر در اقصی نقاط جهان به یکدیگر متصل شده اند. اینترنت، شبکه ای است مشتمل بر زنجیره ای از شبکه های کوچک‌تراست. در ادامه این بخش، به بررسی شبکه های کامپیوتری و جایگاه مهم آنان در زمینه تکنولوژی اطلاعات و تبادل الکترونیکی اطلاعات، خواهیم داشت .

شبکه های محلی(LAN) و شبکه های وسیع (WAN)

یکی از این دسته بندی‌های معروف شبکه های کامپیوتری بر اساس واقع شدنشان در یک "محدوده‌ی مکانی" است. بر اساس این تقسیم بندی، شبکه ها به دو گروه عمده LAN (Local Area Network) و WAN (Wide Area Network) تقسیم می گردند. در شبکه های LAN ، مجموعه ای از دستگاه های موجود در یک محدوده محلی، نظیر یک ساختمان یا اداره به یکدیگر متصل می گردند. در شبکه های WAN تعدادی دستگاه که از یکدیگر کیلومترها فاصله دارند، به همدیگر متصل خواهند شد. برای مثال اگر دو شرکت که هر یک در یک ناحیه از شهر بزرگی مستقر هستند، قصد اشتراک اطلاعات را داشته باشند، می بایست از شبکه‌ی WAN برای اتصال آن‌ها به یکدیگر استفاده نمود. شکل ۱۴.۱ تصویری از شبکه های LAN و WAN و جایگاهشان در کنار هم ارایه می دهد.

---

[۱] Ethernet
[۲] Web



شکل ۱۴.۱

شبکه‌هایLAN نسبت به شبکه‌های WAN ، دارای سرعت بیشتری هستند. با رشد و توسعه دستگاه-های گوناگون مخابراتی، سرعت شبکه های WAN بهبود زیادی پیدا کرده است. امروزه با استفاده از فیبر نوری در شبکه‌های LAN ، امکان ارتباط دستگاه‌های متعدد در مسافت های طولانی نسبت به یکدیگر و با سرعت بالا، فراهم شده است .

۱۴.۲ اترنت

اولین شبکه اترنت، در سال ۱۹۷۳ در مرکز تحقیقات شرکت زیراکس به وجود آمد. هدف این شبکه، ارتباط فیزیکی یک رایانه شخصی به یک چاپگر بود. این شبکه، اترنت نام گرفت و در مدت زمان کوتاهی به عنوان یکی از تکنولوژی های رایج جهت برپاسازی شبکه در سطح دنیا مطرح گردید.

اترنت، یکی از تکنولوژیهای موجود بر روی LAN است. اکثر شبکه های اولیه در حد و اندازه یک ساختمان اداری بوده و دستگاهها به یکدیگر نزدیک بودند. امروزه با توجه به توسعه امکانات مخابراتی و محیط انتقال، دستگاههای موجود در یک شبکه اترنت می توانند در فواصل چند کیلومتری نسبت به هم نصب شوند .

۱۴.۲.۱ پروتکل

پروتکل در شبکه های کامپیوتری به مجموعه قوانینی اطلاق می گردد که نحوه ارتباطات را قانونمند می نماید. به منظور ارتباط موفقیت آمیز دو دستگاه در شبکه می بایست هر دو دستگاه از یک پروتکل مشابه استفاده نمایند.

۱۴.۲.۲ اصطلاحات اترنت

شبکه های اترنت از مجموعه قوانین محدودی به منظور قانونمند کردن عملیات اساسی خود استفاده می نمایند. بمنظور شناخت مناسب قوانین موجود لازم است کـه بـا برخـی از اصـطلاحات مربوطـه در ایـن زمینه بیشتر آشنا شویم :

Medium (محیط انتقال) – دستگاههای اترنت از طریق یک محیط انتقال به یکدیگر متصل می گردند.

Segment (سگمنت) – به یک محیط انتقال به اشتراک گذاشته شده منفرد، " سگمنت " می گویند.

Node (گره) – دستگاههای متصل شده به یک Segment را گره و یا " ایستگاه " می گویند.

Frame (فریم) – به یک بلاک اطلاعات که گره ها از طریق ارسال آنها بـا یکـدیگر مـرتبط مـی گردنـد، اطلاق می گردد

آدرسBroadcast – زمانیکه آدرس مقصد یـک فـریم از نـوع Broadcast باشـد، تمـام گـره هـای موجود در شبکه آن را دریافت و پردازش خواهند کرد.

۱۴.۲.۳ محدودیت های اترنت

یک شبکه اترنت دارای محدودیت های متفاوت از ابعاد گوناگون (بکارگیری تجهیزات) است .طول کـابلی که تمام ایستگاهها بصورت اشتراکی از آن بعنوان محیط انتقال استفاده می نمایند یکی از شاخص تـرین موارد در این زمنیه است . سیگنال های الکتریکی در طول کابل بسرعت منتشر می گردند. همزمان با طی مسافتی، سیگنال ها ضعیف می گردند. وچود میدان های الکتریکی که توسط دسـتگاههای مجـاور کابـل نظیرلامپ های فلورسنت ایجاد می گردد ، باعث تلف شدن سیگنال می گردد. طول کابل شبکه می بایست کوتاه بوده تا امکان دریافت سیگنال توسط دسـتگاه هـای موجـود در دو نقطـه ابتـدائی و انتهـائی کابـل بصورت شفاف و با حداقل تاخیر زمانی فراهم گردد. همین امـر باعـث بـروز محـدودیت در طـول کابـل استفاده شده، می گردد

پروتکل CSMA/CD امکان ارسال اطلاعات برای فقط یک دسـتگاه را در هـر لحظـه فـراهم مـی نمایـد، بنابراین محدودیت هائی از لحاظ تعداد دستگاههائی که می توانند بر روی یک شبکه مجزا وجـود داشـته باشند، نیز بوجود خواهد آمد. با اتصال دستگاه های متعـدد (فـراوان) بـر روی یـک سـگمنت مشـترک، شانس استفاده از محیط انتقال برای هر یک از دستگاه های موجود بر روی سگمنت کاهش پیـدا خواهـد کرد. در این حالت هر دستگاه بمنظور ارسال اطلاعات می بایست مدت زمان زیادی را در انتظار سـپری نماید .

تولید کنندگان تجهیزات شبکه دستگاه های متفاوتی را بمنظور غلبه بر مشکلات و محدودیت گفته شـده ، طراحی و عرضه نموده اند. اغلـب دسـتگاههای فـوق مخـتص شـبکه هـای اترنـت نبـوده ولـی در سـایر تکنولوژی های مرتبط با شبکه نقش مهمی را ایفاء می نمایند.

۱۴.۲.۴ تکرارکننده[۱]

اولین محیط انتقال استفاده شده در شبکه های اترنت کابل های مسی کواکسیال بودند. حداکثر طول یک کابل محدود است و ضمنا یک کابل طویل ممکن است جوابگوی یک تمامی دستگاه های یک ساختمان بزرگ نباشد. تکرارکننده ها ، سگمنت های متفاوت یک شبکه اترنت را به یکدیگر متصل می کنند. در این حالت تکرارکننده سیگنال ورودی خود را از یک سگمنت اخذ و با تقویت سیگنال آن را برای سگمنت بعدی ارسال خواهد کرد. بدین ترتیب با استفاده از چندین تکرار کننده و اتصال کابل های مربوطه توسط آنان ، می توان وسعت یک شبکه را افزایش داد. (مسافت بین دورترین دستگاههای موجود در شبکه)

۱۴.۲.۵ استاندارد IEEE 802.3

در سال ۱۹۸۰ موسسه IEEE کمیته ای را مسئول استاندار سازی تکنولوژی های مرتبط با شبکه کرد. موسسه IEEE نام گروه فوق را 802 نهاد. کمیته فوق از چندین کمیته جانبی دیگر تشکیل شده بود. هر یک از کمیته های فرعی نیز مسئول بررسی جنبه های خاصی از شبکه گردیدند. موسسه IEEE برای تمایز هر یک از کمیته های جانبی از روش نامگذاری 802.x.y استفاده کرد. (x یک عدد منصر بفرد بوده که برای هر یک از کمیته ها در نظر گرفته شده و y یک یا چند حرف نشان دهنده قابلیت های مختلف است)

۱۴.۲.۶ آینده اترنت

اترنت با یک روند ثابت همچنان به رشد خود ادامه می دهد. پس از گذشت حدود سی سال ازعمر شبکه های فوق استانداردهای مربوطه ایجاد و برای عموم متخصصین شناخته شده هستند و همین امر نگهداری و پشتیبانی شبکه های اترنت را آسان نموده است . اترنت با صلابت بسمت افزایش سرعت و بهبود کارآئی و عملکرد گام بر می دارد.

---

[۱] Repeater

۱۴.۳ کابل‌های ارتباطی و سرعت شبکه

۱۴.۳.۱ انواع مختلف کابل‌های ارتباطی

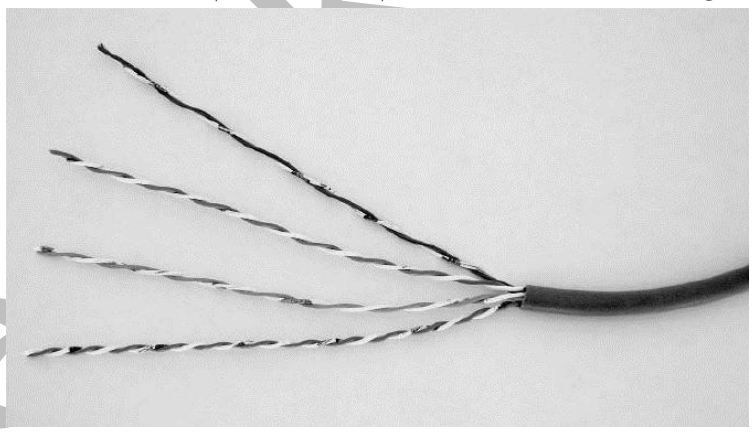
استاندارد اترنت از سه نوع کابل را مجاز کرده است: جفت شده‌ی پیچیده[۱]، فیبرنوری[۲] و کابل کواکسیال[۳]. مشخصات هر یک از کابل‌ها در جدول ۲.۱ آمده است.

جدول ۲.۱

کابل جفت شده‌ی پیچیده بسیار متداول است زیرا علاوه بر قیمت کم دارای کارآیی خوبی است.
فیبرنوری امنیت بیشتری نسبت به اختلالات الکترومغناطیسی داشته و در فواصل طولانی‌تری کاربرد دارد. ولی دارای قیمت بیشتری است. شبکه‌هایی که از کابل کواکسیال استفاده می‌کنند قدیمی‌تر بوده و امروزه به ندرت یافت می‌شوند.

۱۴.۳.۲ کابل جفت شده‌ی پیچیده

این کابل علی‌رغم قیمت کم در فواصل بلند می‌تواند داده‌ها را منتقل کند. پیچیدن سیم‌ها به دور هم، به دو طریق نویز را کاهش می‌دهد: با کاهش میدان مغناطیسی که از سیم‌ها نشت می‌کند و با حذف هرگونه نویز که سیم‌ها را تحت تأثیر قرار می‌دهد.
نمونه‌ای از این نوع کابل‌ها را در شکل ۱۴.۲ می‌بینیم. چهار جفت سیم در این کابل وجود دارند.



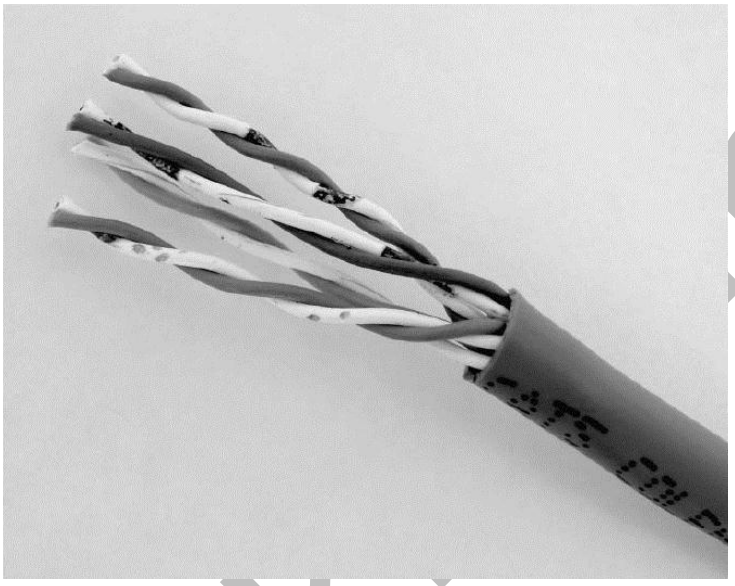
شکل ۱۴.۲

---

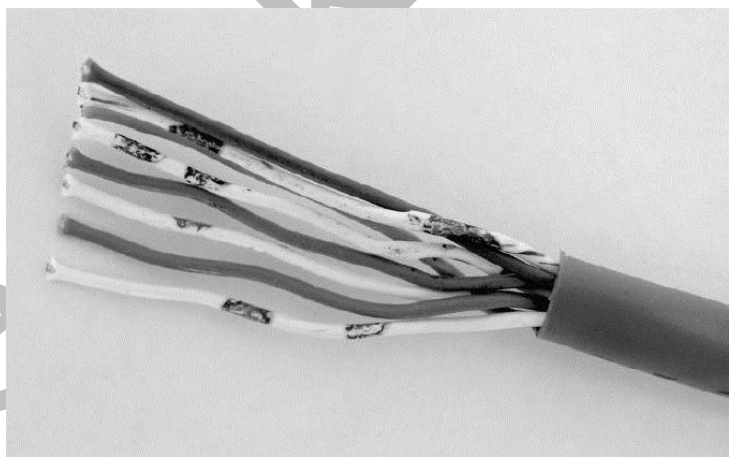
۱ Twisted Pair
۲ Fiber Optic
۳ Coaxial Cable

نحوه‌ی اتصال این کابل به کانکتور مربوطه را در جدول ۲.۲ می‌بینیم. در این جدول رنگ‌های سیم‌ها نیز مشخص‌اند.

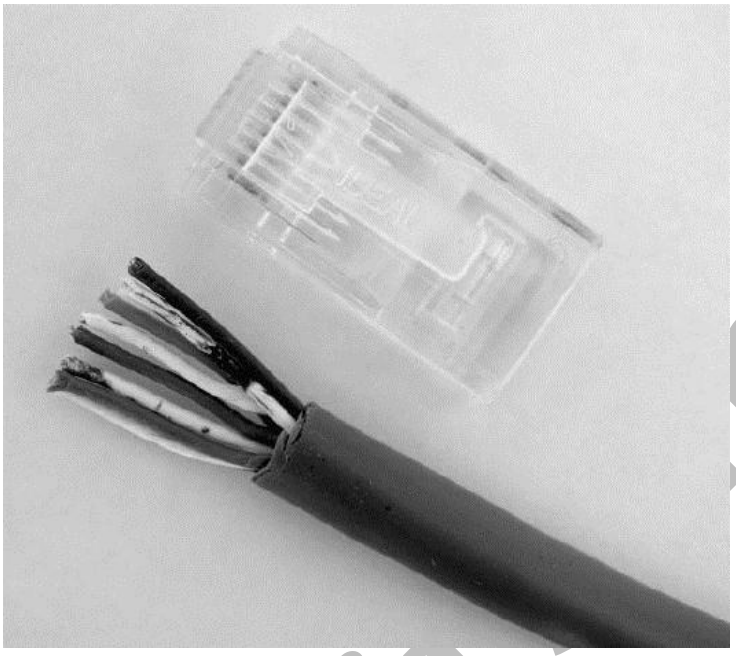
برای اتصال کانکتور RJ-45 به کابل اترنت فرآیندی باید طی شود که مراحل کار را در شکل زیر می‌بینیم.



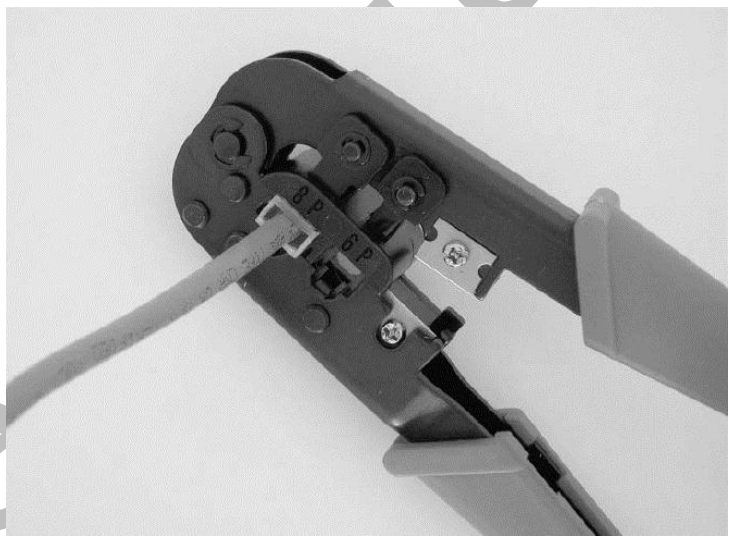
شکل ۱۴.۳



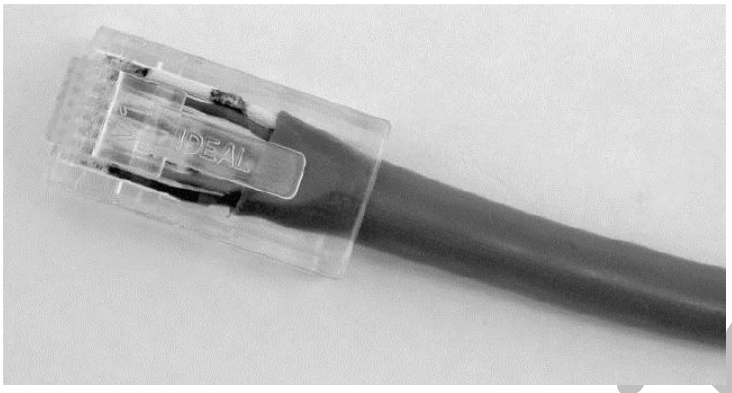
شکل ۱۴.۴



شکل ۱۴.۵



شکل ۱۴.۶



شکل ۱۴.۷

۱۴.۳.۳ تجهیز کابل‌های اترنت به تغذیه

بسیاری از سیستم‌های تعبیه شده، در نزدیکی منبع تغذیه‌ی مطمئنی قرار دارند که با استفاده از یک دیودپل، ورودی AC را به DC تبدیل کرده و جهت تغذیه‌ی سیستم استفاده می‌کنند.

اما وسایلی که در نزدیکی منبع تغذیه قرار ندارند، می‌توانند از باتری استفاده کنند که البته بعد از مدتی نیاز به تعویض آن خواهد داشت. راه‌حل دیگر استفاده از باتری‌های قابل شارژ و یا استفاده از انرژی خورشیدی و باتری خورشیدی است. راه‌حل مناسب دیگر استفاده از همان کابل داده برای انتقال انرژی مورد نیاز جهت تغذیه‌ی سیستم است.

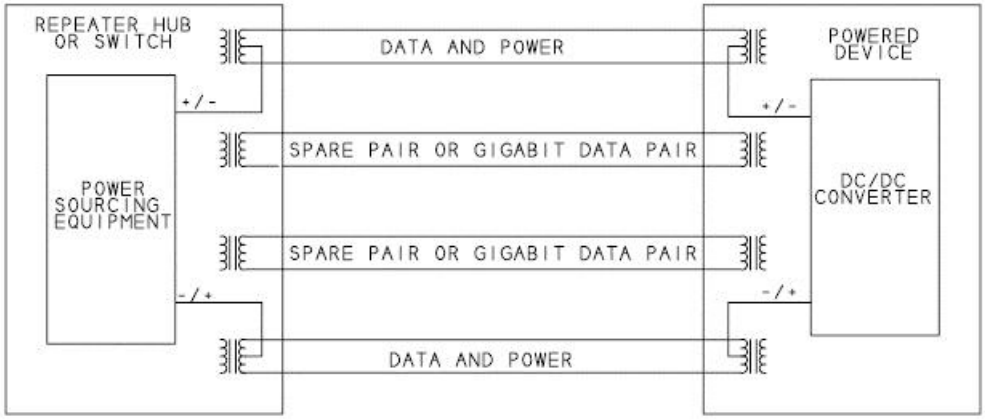
سیستم‌هایی که از راه‌حل آخر استفاده می‌کنند باید از استاندارد 802.3af (که به IEEE 802.3 استاندارد ملحق شده است) برای انتقال مقادیر متولی از انرژی، تبعیت کنند. عبارتی که اغلب برای انتقال انرژی از طریق کابل داده استفاده می‌شود، عبارت "توان بر روی اترنت[۱] " است.

کابل‌های با سرعت ۱۰، ۱۰۰ و ۱۰۰۰ مگابیت در ثانیه می‌توانند از روش فوق استفاده کنند.

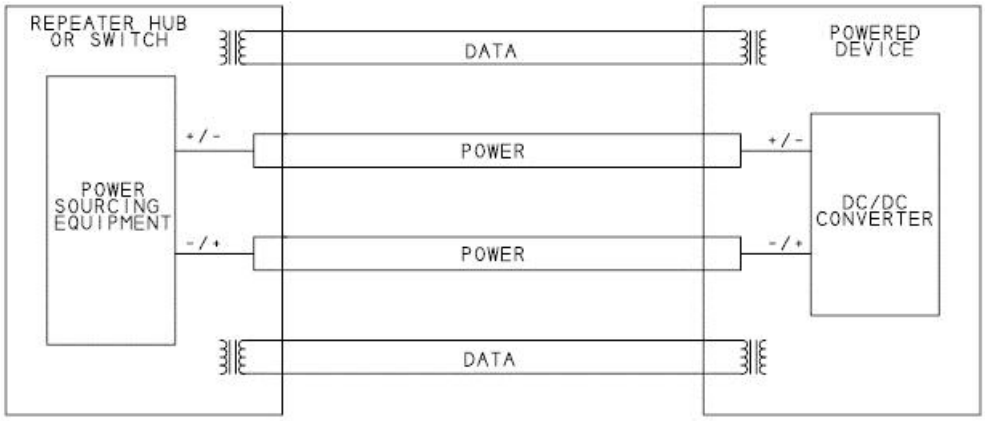
در شکل ۱۴.۸ دستگاه فراهم کننده‌ی منبع تغذیه (PSE) می‌تواند در هر یک از دو سمت واقع باشد. این شکل روش‌های مختلف فراهم آوردن تغذیه را نشان می‌دهد.

---

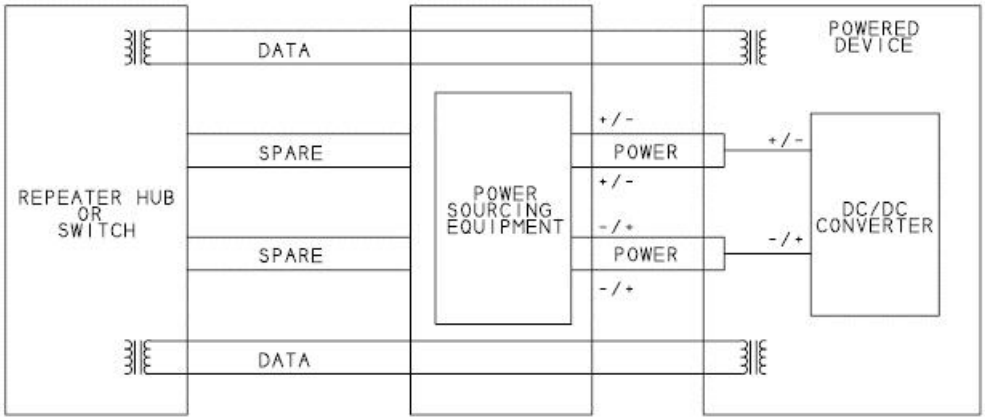
[۱] Power Over Ethernet-POE



ENDPOINT POWER SOURCING EQUIPMENT
USING DATA WIRES



ENDPOINT POWER SOURCING EQUIPMENT
USING SPARE WIRES (10 AND 100 MB/SEC ONLY)



MIDSPAN POWER SOURCING EQUIPMENT
USING SPARE WIRES (10 AND 100 MB/SEC ONLY)

شکل ۱۴.۸

PSE می‌تواند تغذیه را از طریق سیم‌های استفاده نشده‌ی کابل و یا بر روی همان سیم‌های داده انتقال دهند. دقت کنید که کابل‌های با سرعت ۱۰ و ۱۰۰ مگابیت، سیم‌های اضافی برای این منظور ندارند. در این حالت، جهت استفاده از خطوط داده برای انتقال توان، PSE یک ولتاژ DC را به سر وسط ترانسفورماتور ایزوله کننده متصل می‌کند.

PSE تشخیص می‌دهد که یک دستگاه به توان نیاز دارد. توان را ارایه می‌دهد. توان مصرفی را نمایش داده و در صورتی که دیگر نیازی به آن نباشد، توان تغذیه را قطع می‌کند. یک PSE استاندارد می‌تواند توان 13 W را به دستگاه مصرف کننده تحویل دهد. (معادل 350 mA در ولتاژ 37 V)

۱۴.۳.۴ سیستم‌های واسط و استانداردها

راه‌های متفاوتی برای اتصال شبکه‌ها به یکدیگر وجود دارند. هر سیستم واسط از کابل‌های خاص و سرعت انتقال داده‌های مخصوصی استفاده می‌کند. انواع این سیستم‌ها را در جدول ۲.۴ می‌بینیم.

جدول ۲.۴

۱۴.۴ کنترل کننده‌ی Ethernet

یک سیستم تعبیه شده که از اترنت پشتیبانی می‌کند، دارای یک سخت‌افزار کنترل کننده‌ی اترنت برای راه‌اندازی این واسط است. سیستم‌های تعبیه شده‌ی کوچک، عموماً به تمام قابلیت‌هایی که در رایانه شخصی مورد نیاز است نیازی ندارند.

ارتباطات اترنت را می‌توان به دو قسمت تراشه‌ی کنترل کننده‌ی اترنت و کد راه‌اندازی آن تقسیم کرد.

شکل ۱۴.۹ مکان سخت‌افزار اترنت و راه‌اندازش را در پشته‌ی شبکه[۱] نشان می‌دهد. بسیاری از سیستم‌های تعبیه شده از IP به همراه TCP یا UDP استفاده می‌کنند. اما در بعضی از آن‌ها ممکن است راه‌انداز اترنت مستقیماً با لایه‌ی برنامه‌ی کاربردی[۲] در ارتباط باشد.

کنترل کننده اترنت بسیاری از ریزه کاری‌های مرتبط با ارسال و دریافت فریم‌های اترنت را از دید نرم‌افزار اصلی پنهان می‌کند.

در ارسال یک فریم داده، کنترل کننده وظایف زیر را انجام می‌دهد:

- پیغامی که باید ارسال شود و آدرس مقصد را از نرم‌افزار سطح بالاتر، دریافت می‌کند.

---

[۱] Network Stack

[۲] Application Layer

- رشته‌ی آزمایش فریم[1] را محاسبه می‌کند.
- داده، آدرس‌ها و سایر اطلاعات را در فریم اترنت جای می‌دهد.
- هنگامی که شبکه بیکار است، اقدام به ارسال داده‌ها می‌کند.
- تصادم[2] را تشخیص داده و هر داده ارسال شده با تصادم را حذف می‌کند. سپس طبق پروتکل IEEE802.3 اقدام به ارسال مجدد اطلاعات می‌نماید.
- نشانگری برای نمایش موفقیت یا شکست ارسال داده‌ها دارد.

در دریافت فریم‌ها نیز عملیات زیر را انجام می‌دهد:

- یک فریم جدید را سنکرون کرده و دریافت می‌کند.
- هر فریم را که از حداقل اندازه‌ی مجاز کمتر باشد را حذف می‌کند.
- فریمی را که دارای آدرس واسط مشخص نباشد، و یا دارای آدرس ناصحیح پخش همگانی[3] و یا مالتی‌کست[4] باشد، را نادیده می‌گیرد.
- رشته‌ی آزمایش فریم را محاسبه کرده، مقدار حاصل را با مقدار دریافتی مقایسه می‌کند و اگر دو مقدار با یکدیگر مطابقت نداشته باشند اعلام خطا می‌کند.
- داده دریافت شده از فریم را در دسترس برنامه‌ی کاربردی قرار می‌دهد.

۱۴.۴.۱ ارتباط با کنترل کننده‌های اترنت

کابل‌ها و واسط‌های مختلف اترنت با سرعت‌های مختلف، نیازمند مدارهای سخت‌افزاری مختلف و متفاوتی نیز برای ایجاد ارتباط هستند.

استاندارد اترنت چندین واسط اترنت مختلف برای اتصال لایه‌ی " کنترل دستیابی واسط" (MAC ـ که کارش مدیریت ارسال و دریافت داده‌های اترنت است) به لایه "فیزیکی" (PHY ـ که شامل مدارات سخت‌افزاری متناسب با کابل و نرخ انتقال داده خاص است) را پیشنهاد می‌دهد.

بسته به نوع کابل و سرعت انتقال داده‌ها، قسمت PHY ممکن است شامل فرستنده- گیرنده‌ها، فیلترهای آنالوگ، قسمت مغناطیسی کابل و کابل‌های ارتباطی باشد.

شکل ۱۴.۹ تعدادی از راه‌های مرسوم این ارتباط را نشان می‌دهد. آن دسته از میکروکنترولرهای ARM که دارای کنترل کننده‌ی داخلی اترنت هستند، حاوی لایه MAC بوده و برای اتصال به کابل اترنت نیاز

---

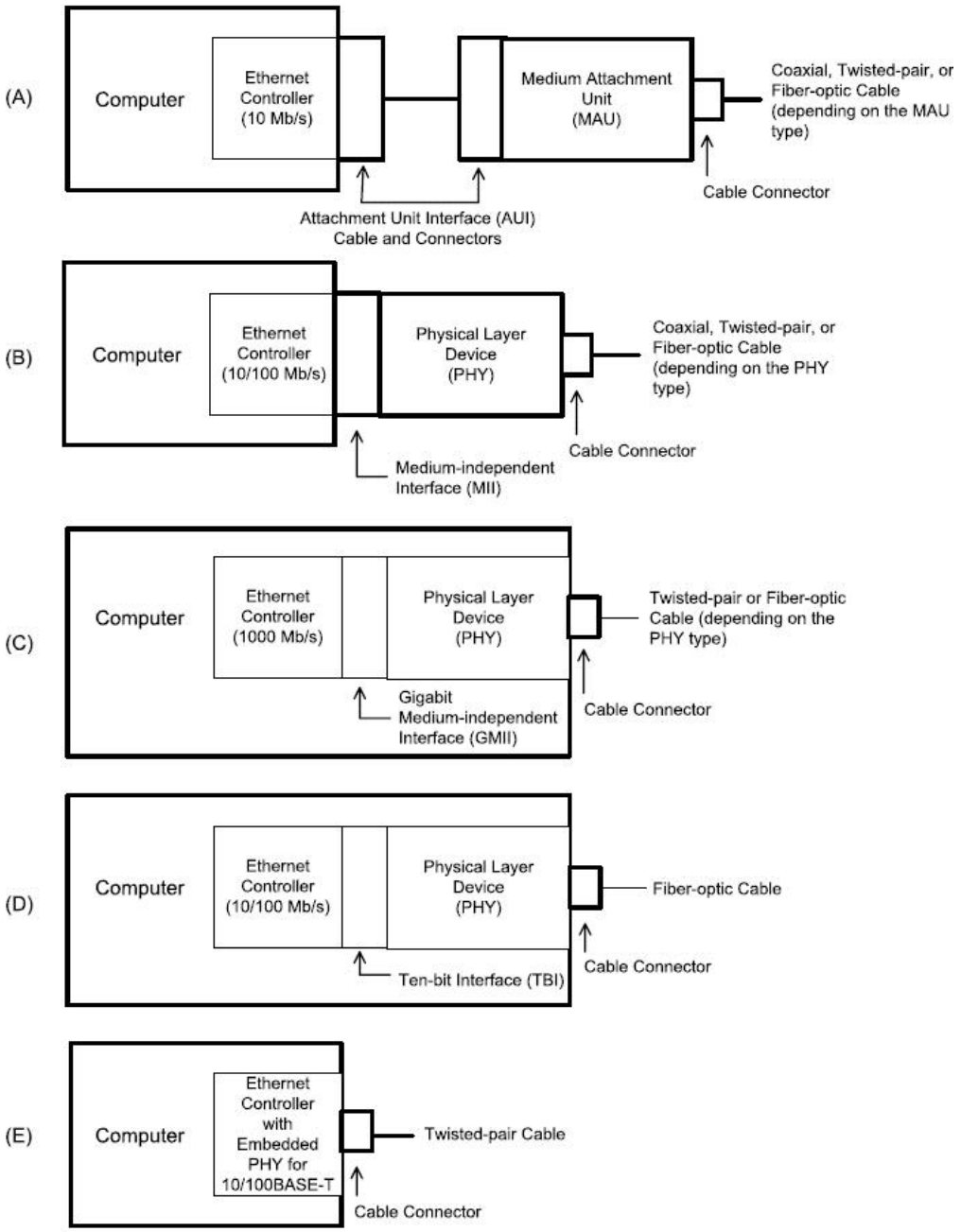
[1] Frame Check

[2] Collision

[3] Broad Cast

[4] Multi Cast

به یک تراشه‌ی PHY دارند. تعدادی از تراشه‌های استاندارد PHY که می‌توانند برای این منظور استفاده شوند در جدول ۱۸۲ فهرست شده‌اند.



شکل ۱۴.۹

در شکل ۱۴.۱۰ یک کنترل کننده‌ی با سرعت ۱۰ یا ۱۰۰ مگابیت در ثانیه با استفاده از "واسط نابسته به وسیله" (یا MII - Medium Independent Interface) به تراشه‌ی PHY و سپس به کابل اترنت متصل می‌شود.
MII داده‌های سریال شبکه را به داده‌های ۴ بیتی متصل شونده به کنترل کننده‌ی اترنت، متصل می‌کند.

توجه

فقط تعدادی معدود از میکروکنترولرهای ARM دارای PHY بر روی تراشه خود هستند. برای مثال، میکروکنترولرهای ساخت Luminary Micro (هم‌اکنون TI) این گونه اند.

انواع دیگری از ارتباط بین کنترل کننده‌ی اترنت و PHY وجود دارند که در شکل فوق آمده‌اند.
RMII نوع ساده شده‌ی ارتباط نوع MII است. GMII مخصوص ارتباط اترنت با سرعت 1 Gbps بوده و TBI مخصوص ارتباط با فیبرنوری است.

۱۴.۵ درون پروتکل اینترنت[۱]

IP به بسته‌های داده کمک می‌کنند که مسیر خود را به سمت مقصد پیدا کنند. این فرآیند شامل حالتی است که حتی داده‌ها باید به شبکه‌های دیگر و یا اینترنت وارد شوند. اگرچه این پروتکل را با نام پروتکل اینترنت می‌شناسند، ولی این پروتکل در شبکه‌های محلی نیز استفاده می‌شود.
در این بخش در مورد IP، نحوه‌ی به دست آوردن آدرس IP، فرمت دیتاگرام IP و نحوه‌ی مسیریابی داده‌ها به سمت مقصد بحث خواهیم کرد.
در شکل ۱۴.۱۱ جایگاه IP در ارتباطات شبکه نشان داده شده است. به هنگام ارسال داده‌ها، IP اطلاعات لازم را از لایه‌های بالاتر با پروتکل‌هایی چون TCP یا UDP دریافت می‌کند. لایه‌ی IP این بسته‌ها را گرفته و با قرار دادن اطلاعات اضافی، آن‌ها را به لایه‌های پایین‌تر برای ارسال می‌فرستد.
پروتکل اینترنت دو وظیفه‌ی اصلی بر عهده دارد:

- راهی برای مشخص کردن آدرس مقصد و مبدأ فراهم می‌آورد که از طریق آن داده‌ها در مسیر درست خود حرکت می‌کنند.

[۱] Internet Protocol-IP

- با استفاده از روش به کار رفته در دیتاگرام IP، داده‌ها با گذر از مسیریاب[1]ها و قسمت‌های مختلف شبکه با مشکل مواجه نمی‌شوند. در بعضی مواقع بسته‌ها با گذر از مسیریاب‌ها به بسته‌های کوچک‌تری تقسیم می‌شوند که بعد از گذر از شبکه، مجدداً به یکدیگر متصل می‌شوند.

دو پروتکل در نحوه‌ی انطباق آدرس‌های IP دستگاه‌های متصل به اترنت نقش دارند. پروتکل اول سیستم نام حوزه" (DNS) بوده که به یک وسیله‌ی اجازه‌ی انطباق یک نام با آدرس IP را می‌دهد. و دیگری "پروتکل تفکیک آدرس" (ARP) است که به فرستنده دیتاگرام IP اجازه می‌دهد، آدرس IP را با یک آدرس سخت‌افزاری شبکه‌ی اترنت، بر روی شبکه‌ی محلی انطباق دهد.

۱۴.۵.۱ آدرس‌های IP

هر کامپیوتری که از پروتکل اینترنت استفاده می‌کند باید یک آدرس IP داشته باشد. آدرس‌های IP می‌توانند به صورت دستی، یا به طور خودکار و با استفاده از پروتکل DHCP به هر کامپیوتر اختصاص داده شوند.

طبق IPV4 آدرس‌های IP، ۳۲ بیتی هستند. روش قراردادی نوشتن آدرس‌های IP به صورت x.y.z.w است که در آن هر یک از حروف x، y، z و w می‌توانند اعداد بین 0 تا 255 باشند. برای مثال 192.168.1.7.

۱۴.۵.۱.۱ اختصاص آدرس‌ها

هر دیتاگرام IP بر روی شبکه دارای یک آدرس مبدأ و یک آدرس مقصد است. آدرس IP یک کامپیوتر در درون شبکه و یا شبکه‌هایی که در آن مشغول به کار است، باید منحصربه‌فرد باشد.

آدرس‌های IP در عمل نمی‌توانند هر عددی باشند. آدرس‌های خاصی که طبق تأیید جامعه‌ی استفاده کنندگان از اینترنت، پذیرفته شده‌اند در جدول ۴.۲ آمده‌اند.

برای پی بردن به آدرس IP بر روی رایانه‌ی شخصی و یا تغییر آن‌ها می‌توانید از دستورهای ipconfig (بر روی ویندوز) و ifconfig (بر روی لینوکس) استفاده کنید.

---

[1] Router

۱۴.۶ تبادل پیغام‌ها با استفاده از UDP و TCP

در این بخش در مورد نحوه‌ی رساندن داده به مقصد توسط پروتکل‌های TCP و UDP بحث خواهیم کرد. داشتن دانش فنی در مورد پروتکل‌ها، به استفاده‌ی صحیح و مؤثر آن‌ها کمک شایانی می‌کند.
اترنت به انتقال فایل‌ها بر روی شبکه‌ی LAN کمک می‌کنند. اما اترنت به تنهایی، اطلاعات چندانی از آدرس، شماره‌ی پورت، برنامه‌ی استفاده کننده، نحوه‌ی دست‌دهی[۱] و... نمی‌دهد.
شماره‌ی فریم ارسالی، اطمینان از صحت اطلاعاتی ارسالی به مقصد و... از جمله عملیاتی است که TCP[۲] در اختیار می‌گذارد. UDP نیز پروتکلی ساده‌تر است که فقط شماره‌ی پورت‌ها و آزمایش خطا را دربردارند. جدول ۵.۱ مشخصات TCP و UDP را با یکدیگر مقایسه می‌کند.
شکل ۱۴.۱۲ جایگاه UDP و TCP را در پشته‌ی پروتکل‌های شبکه نشان می‌دهد. TCP و UDP با لایه IP و لایه‌ی برنامه کاربردی[۳] تبادل اطلاعات می‌کنند. بعضی از برنامه‌ها به UDP یا TCP نیاز نداشته و ممکن است مستقیماً با لایه‌ی IP و یا راه‌انداز اترنت ارتباط برقرار کنند.

۱۴.۶.۱ سوکت‌ها و پورت‌ها

هر ارتباط TCP یا UDP بین دو نقطه‌ی انتهایی یا سوکت[۴] انجام می‌شود. هر سوکت دارای یک شماره‌ی پورت و یک آدرس IP است.
در یک فریم اترنت، آدرس‌های مبدأ و مقصد، مشخص کننده‌ی واسط‌های اترنت فرستنده و یا گیرنده هستند. با استفاده از TCP و UDP آدرس گیرنده به نحوی دقیق‌تر مشخص می‌شود. این کار با استفاده از قرار دادن یک **شماره‌ی پورت**[۵] برای مقصد انجام می‌شود. در TCP، مبدأ نیز دارای یک شماره‌ی پورت است که نشان می‌دهد داده‌ها از کجا ارسال شده‌اند. ارتباطات UDP نیز دارای شماره‌ی پورت مبدأ نیز هستند ولی به این شماره در دیتاگرام UDP نیازی نمی‌باشد.
برای داشتن تصور صحیح از پورت‌ها، می‌توان هر یک از کامپیوترها در شبکه را به صورت یک آبکش درنظر گرفت که سوراخ‌های آن همانند پورت‌ها هستند. از هر یک از سوراخ‌ها به هر دوسمت داده‌ای منتقل می‌شود.

---

۱ Hand Shake
۲ Transfer Control Protocol
۳ Application Layer
۴ Socket
۵ Port

سه گروه مختلف از شماره‌های پورت موجودند: شماره‌های بین ۱ تا ۱۰۲۳ را "پورت‌های شناخته شده" می‌نامند که توسط برنامه‌های خاص استفاده می‌شوند. جدول ۵.۲ تعدادی از این شماره پورت‌ها را فهرست کرده است.
پورت‌های بین ۱۰۲۴ و ۴۹۱۵۱ را با نام "پورت‌های ثبت شده" می‌شناسیم. و پورت‌های بین ۴۹۱۵۲ تا ۶۵۵۳۵ به نام "پورت‌های خصوصی" شناخته می‌شوند. هر یک از این پورت‌ها و یا محدوده‌ی پورت‌ها متعلق به برنامه‌های خاصی هستند و یا ممکن است آزاد بوده و یا استفاده نشده باشند.

۱۴.۷ uIP

این پشته نرم‌افزاری برای ایجاد ارتباط آسان با پروتکل TCP/IP بر روی میکروکنترولرها (حتی انواع ساده ۸ بیتی) به وجو آمده است. اندازه کدی که این برنامه اشغال می‌کند به چندین کیلوبایت می‌رسد و حافظه RAM مورد استفاده‌اش نیز از چندین صد بایت بیشتر نیست.

بخش ۶

مباحث پیشرفته‌ی میکروکنترولرهای ARM

فصل پانزدهم

پردازش سیگنال دیجیتال

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ پردازش سیگنال دیجیتال چیست؟
- ✓ پیاده سازی فیلترهای FIR و IIR؛
- ✓ پیاده‌سازی FFT.

امروزه، میکروپروسسورها به خوبی توان حداکثر خود را برای سیگنال‌های دیجیتالی استفاده می‌کنند. احتمالاً، با پخش کننده‌های موسیقی (MP3 Player)، دوربین‌های دیجیتالی و تلفن‌های همراه یا سلولی، کاملاً آشنا هستید. پردازش سیگنال‌های دیجیتال به "سرعت تبادل اطلاعات بالا" و "حضور عملیات ضرب و جمع توأم[1] " (یا MAC) نیاز دارد. در این بخش، به راه‌های افزایش کارآیی ARM برای کاربردهای پردازش سیگنال دیجیتال، نگاهی می‌اندازیم.

از قدیم این‌گونه متداول بوده است که یک دستگاه تعبیه شده یا قابل حمل، دو نوع پردازنده دارد:

یک میکروکنترولر، به واسطه‌های کاربری رسیدگی کرده و یک پردازنده‌ی DSP مجزا، سیگنال‌های دیجیتال همانند صدا را پردازش می‌کند.

امروزه شما می‌توانید از یک پردازنده برای هر دو منظور فوق استفاده کنید. دلیل این امر توان پردازشی بالا و کلاک مرکزی بالای پردازنده‌های حرفه‌ای است که امروزه به راحتی در دسترس قرار دارند. طرحی براساس یک پردازنده "تک هسته‌ای[2] " نسبت به یک طرح "دوهسته‌ای"، هزینه‌ی کمتری را بر سیستم تحمیل کرده و توان کمتری را مصرف می‌کند.

معماری ARMv5TE که در ARM9E و پردازنده‌های بعد از آن موجود می‌باشند، از افزونه‌های پردازش سیگنال به‌منظور دستیابی به توان پردازشی بالا استفاده می‌کنند. تمامی پردازنده‌های ذکر شده دارای دستور MAC هستند که قلب پردازش سیگنال دیجیتال است. با به کار بردن دقت بیشتر هنگام کدنویسی، ARM9E کاملاً رفتار یک پردازنده‌ی DSP را تقلید می‌کند. این در حالی‌است که در قسمت-های کنترلی دستگاه مورد نظر، میکروکنترولر ARM کاملاً بر DSP برتری دارد.

---

[1] Multiply Accumulate

[2] Single Core

برنامه‌های DSP ، عموماً استفاده‌ی گسترده‌ای از دستورهای ضرب و بارگیری_ ذخیره سازی[1] می‌کنند. یک عملگر ابتدایی، MAC[2] است که شامل ضرب دو عدد علامت‌دار ۱۶ بیتی به همراه جمع نتیجه‌ی حاصل با انباره‌ی علامت‌دار ۳۲ بیتی می‌باشد.

جدول ۸.۱، افزایش کارآیی بر روی نسل‌های مختلف ARM را با یکدیگر مقایسه می‌کند. ستون دوم، تعداد سیکل‌های ماشین برای عملیات ضرب ۱۶ بیت در ۱۶ بیت به همراه جمع با یک کمیت ۳۲ بیتی است. ستون سوم، تعداد سیکل‌های مورد نیاز برای ضرب ۳۲ بیت در ۳۲ بیت به همراه جمع با یک کمیت ۶۴ بیتی است. مورد آخر برای الگوریتم‌های پیشرفته‌ای چون mp3 بسیار مفید است.

در جدول ۱۵.۱، فرض بر این‌است که شما از کاراترین مجموعه‌ی دستورها برای عملیات فوق استفاده کرده‌اید. به نحوی که از هرگونه interlock (که بعد از عملیات ضرب امکان وقوع دارند)، دوری جسته-اید. جزئیات این موضوع را در بخش ۱۵.۲ خواهیم دید.

از آن‌جایی که در نوشتن الگوریتم‌های DSP نیاز به سرعت تبادل اطلاعات و کارآیی بالا می‌باشد، اغلب در نوشتن این الگوریتم‌ها از کدهای نوشته شده به زبان اسمبلی استفاده می‌کنیم. شما باید کنترل دقیقی بر روی ثبات‌های مورد استفاده و زمان‌بندی دستورها داشته باشید. به علت گستردگی روش‌ها و الگوریتم‌های مختلف DSP، بررسی تمامی آن‌ها در این بخش امکان‌پذیر نیست. پس، فقط بر روی روش‌های پایه و کاربردی DSP که در بسیاری از الگوریتم‌ها استفاده می‌شوند، تمرکز خواهیم کرد.

بخش ۱۵.۱، نشان می‌دهد که یک سیگنال چگونه برای پردازش صحیح در پردازنده‌ی ARM فراهم می-شود. در بخش ۱۵.۲ ، قوانین کلی حاکم بر نوشتن الگوریتم‌های DSP را خواهیم آموخت.

فیلتر، احتمالاً یکی از پرکاربردترین عملیات در DSP است. از کاربردهای گوناگون "فیلتر کردن" می-توان به حذف نویز، تحلیل سیگنال‌ها و فشرده سازی سیگنال‌ها اشاره نمود. در بخش‌های ۱۵.۳ و ۱۵.۴ نگاهی دقیق به جزئیات فیلترهای صوتی خواهیم داشت. دیگر الگوریتم پرکاربرد، تبدیل فوریه‌ی گسسته[3] است که یک سیگنال را از نمایش در حوزه‌ی زمان به نمایش در حوزه فرکانس و یا برعکس تبدیل می-کند. DFT را نیز در بخش ۸.۵ بررسی خواهیم کرد.

۱۵.۱ نمایش یک سیگنال دیجیتال

قبل از پردازش سیگنال‌های دیجیتال، بهتر است که نحوه‌ی نمایشی برای آن‌ها انتخاب کنیم. چگونه یک سیگنال را فقط با اعداد صحیحِ موجود بر روی ARM نمایش دهیم؟ این یک مسأله مهم در طراحی نرم-افزار DSP است. در این بخش، از علامت‌گذاری x_t و $x[t]$ برای نمایش سیگنال x در زمان t استفاده

---

[1] Load_store

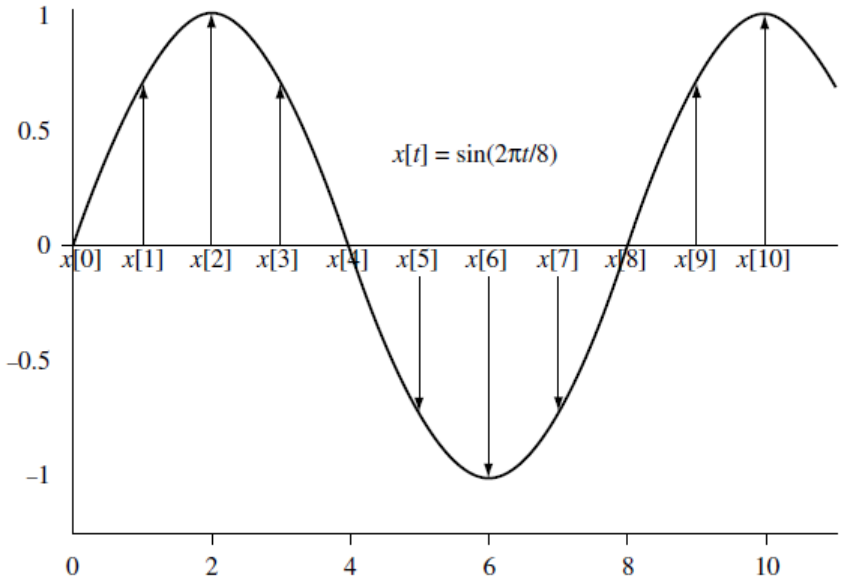
[2] Multiply-Accumulate-MAC

[3] Discrete Faurier Transform-DFT

می‌کنیم. علامت‌گذاری اول را به علت سادگی، در فرمول‌ها استفاده کرده و نمونه دوم را به دلیل شباهت به کد نویسی از زبان C، در برنامه نویسی به کار می‌بریم.

۱۵.۱.۱ انتخاب نحوه‌ی نمایش

در یک سیگنال آنالوگ x[t]، هر دو کمیت x و t، مقادیر حقیقی پیوسته هستند. برای تبدیل مقادیر آنالوگ به نمونه‌های دیجیتال، از تعداد محدودی از مقادیر نمونه‌برداری t_i استفاده می‌کنیم. مقادیر x[t] در این نمونه‌های زمانی، $x[t_i]$ هستند.



شکل ۱۵.۱

شکل ۱۵.۱ یک سیگنال سینوسی را که در نقاط نمونه‌برداری ۰، ۱، ۲، ۳ و... به نمونه‌های دیجیتال تبدیل شده‌اند را نشان می‌دهد. سیگنال‌های شبیه این شکل را در کاربردهای صوتی به وفور می‌بینیم. برای مثال، در پخش کننده‌ی CD[۱]، نرخ نمونه‌برداری 44,100 Hz (یعنی 44,100 نمونه‌ی دیجیتال در هر ثانیه) است. بنابراین t ، زمان برحسب واحد تناوب نمونه‌برداری، یعنی 1/44,100 Hz=22.7μS است. در مورد این پخش کننده‌ی موسیقی، x[t] ولتاژ علامت‌دار خروجی است که به بلندگوها اعمال می‌شود. در مورد انتخاب نحوه‌ی نمایش، دو چیز درخور اهمیت‌اند:

[۱] CD Player

۱. محدوده‌ی دینامیک[۱] سیگنال. بیشینه تغییرات نوسانی سیگنال که توسط معادله‌ی ۸.۱ تعریف می‌شود.

$$M = max|x[t]| \text{ over all } t = 0,1,2,3 \ldots$$

۲. دقت مورد نیاز در نحوه‌ی نمایش سیگنال. اغلب به عنوان درصدی از محدوده‌ی بیشینه‌ی سیگنال درنظر گرفته می‌شود. برای مثال، دقت 100 ppm بدین معنا است که x[t] باید در محدوده‌ی خطای

$$E = M\times 0.0001 = 0.0001 \text{ volts}$$

تعریف شود.

حال، بیایید بهترین راه ذخیره‌ی x[t] را با توجه به محدوده‌ی دینامیک و وقت مورد نیاز بیابیم.

ما می‌توانیم از نمایش با نقطه‌ی اعشاری سیگنال استفاده کنیم. این نحوه‌ی نمایش حتماً احتیاج‌های ما را در مورد محدوده‌ی دینامیک و دقت برآورده خواهند کرد. همچنین، دستکاری این اعداد، با متغیرهای نوع float در زبان C نیز بسیار ساده خواهد بود. اگرچه، مشکل از آن‌جا پیدا می‌شود که بیشتر پردازنده‌های ARM از محاسبات نقطه‌ی اعشار سخت‌افزاری پشتیبانی نمی‌کنند و پیاده سازی این نحوه‌ی نمایش زمان‌بر و کند است.

برای دستیابی به سرعت بیشتر، در این محاسبات از نحوه‌ی نمایش نقطه‌ی ثابت[۲] استفاده می‌کنیم. در این نوع نحوه‌ی نمایش از اعداد صحیح برای نمایش مقادیر اعشاری (البته با یک ضریب مقیاس معین) استفاده می‌شود.

برای مثال جهت دستیابی به خطای ± 0.0001 ولت (در این‌جا مقدار بیشینه‌ی سیگنال را 1 ولت درنظر می‌گیریم)، مقدار فاصله‌ی بین دو مقدار متوالی از اعداد قابل نمایش برابر 0.0002 ولت است. پس بهتر است برای نمایش این اعداد، از x[t] که توسط رابطه‌ی زیر داده می‌شود استفاده کنیم:

$$X[t] = round\_to\_nearest\_integer(5000 \times x[t])$$

ما در عمل از توان‌های 2 برای مقیاس‌بندی ورودی استفاده می‌کنیم. پیاده سازی دستورها ضرب و تقسیم در این حالت، در واقع از طریق دستورها شیفت انجام می‌شوند. در این حالت بزرگ‌ترین توان دویی که کمی بیشتر از 5000 را درنظر می‌گیریم. یعنی 2^{13}=8192 که اولین توان دوی بزرگ‌تر از 5000 است. در این‌جا می‌گوییم که X[t] نمایش نقطه‌ـ ثابت Qk کمیت x[t] است اگر:

$$X[t] = round\_to\_nearest\_integer(2^k x[t])$$

در مثال بالا، ما از Q13 برای نمایش بادقت مورد نیاز استفاده کردیم.

---

[۱] Dynamic Range
[۲] Fixed-Point

از آن‌جایی که x[t] بین 1- ولت و 1+ ولت تغییر می‌کند، X[t] بین 8192- و 8192+ تغییر خواهد کرد. این مقدار عددی، در یک متغیر ۱۶ بیتی از جنس short به راحتی جای می‌گیرد.
سیگنال‌ها معمولاً با Q15 نیز ذخیره می‌شوند، زیرا در Q15 از فضای در دسترس اعداد در متغیرهای short (یعنی 32767+ تا 32768-) حداکثر استفاده را می‌بریم. دقت کنید که 1+ در این نمایش اعداد، نمایش دقیقی نداشته و ما آن را با 32767+ که معادل $1-2^{-15}$ است، تقریب می‌زنیم. اگرچه همان‌طور که بعداً خواهیم دید، مقیاس‌بندی اعداد تا حدّ بیشینه‌ی قابل نمایش، ایده‌ی خوبی نیست. زیرا در آن با خطر سرریز اعداد روبه‌رو خواهیم بود.

۱۵.۱.۲ نمایش لگاریتمی

فرض کنید سیگنال ورودی دارای محدوده دینامیک بزرگی است. همچنین فرض کنید که عملگر ضرب از جمع بسیار بیشتر به کار برده می‌شود. در این حالت می‌توانید از نمایش مبنای دو لگاریتمی استفاده کنید:

$$y[t] = \log_2(x[t])$$

$y[t]$ را با قالب‌بندی نقطه ثابت نشان می‌دهد. عملیات شبیه

$$x[a] = x[b] \times x[c]$$

را با

$$y[a] = y[b] + y[c]$$

و عملیات شبیه

$$x[a] = x[b] + x[c]$$

را با

$$y[a] = y[b] + \log_2(1 + 2^{\,y[c]-y[b]})$$

جایگزین کنید. در مورد دوم، فرض براین‌است که داریم : $y[c] \le y[b]$. تابع $f(x) = \log_2(1+2^x)$ را با استفاده از جدول جستجو[1] و یا برون‌یابی محاسبه کنید.

۱۵.۱.۳ جمع و تفریق سیگنال‌های نقطه‌ی ثابت

حالت عمومی این عملیات، تبدیل معادله

$$y[t] = x[t] + c[t]$$

به قالب نقطه ثابت است. یعنی به طور تقریب داریم:

$$Y[t] = 2^d y[t] = 2^d\,(x[t] + c[t]) = 2^{d-n} X[t] + 2^{d-m} C[t]$$

یا در زبان C داریم:

```
Y[t] = (X[t]<<(d-n))+(C[t]<<(d-m));
```

[1] Look-Up Table - LUT

با استفاده از Barrel Shifter موجود در پردازنده ARM می‌توانیم عملیات فوق را در یک سیکل انجام دهیم:

```
Y[t] = X[t] + C[t]<<(d-m))
Y[t] = C[t] + (X[t]<<(d-n))
```

۱۵.۱.۴ ضرب سیگنال‌های نقطه‌ی ثابت

حالت عمومی این عملیات، تبدیل معادله

$$y[t] = x[t]\ c[t]$$

به قالب نقطه‌ی ثابت است. یعنی به طور تقریب داریم:

$$Y[t] = 2^d\, y[t] = 2^d\, x[t]c[t] = 2^{d-n-m}\, X[t]C[t]$$

یا در زبان C داریم:

```
Y[t] = (X[t]*C[t])>>(n+m-d);
```

اگر برای سادگی فرض کنیم d = m+n.

```
Y[t] = X[t]*C[t];
```

۱۵.۱.۵ تقسیم سیگنال‌های نقطه‌ی ثابت

حالت عمومی این عملیات، تبدیل معادله

$$y[t] = x[t]\ /\ c[t]$$

به قالب نقطه‌ی ثابت است. یعنی به طور تقریب داریم:

$$Y[t] = 2^d\, y[t] = 2^d\, x[t]/c[t] = 2^{d-n+m}\, X[t]/C[t]$$

یا در زبان C داریم:

```
Y[t] = (X[t]<<(d-n+m)) / C[t];
```

اگر برای سادگی فرض کنیم m = n.

```
Y[t] = (X[t]<<(d)) / C[t];
```

۱۵.۱.۶ ریشه دوم سیگنال‌های نقطه‌ی ثابت

حالت عمومی این عملیات، تبدیل معادله

$$y[t] = \sqrt{x[t]}$$

به قالب نقطه‌ی ثابت است. یعنی به طور تقریب داریم:

$$Y[t] = 2^d\, y[t] = 2^d\, \sqrt{x[t]} = \sqrt{\left(2^{2d-n}\, X[t]\right)}$$

یا در زبان C داریم:

```
Y[t] = isqrt(X[t]<<(2*d-n));
```

اگر برای سادگی فرض کنیم m = n.

```
Y[t] = (X[t]<<(d)) / C[t];
```

۱۵.۲ راهنمای نوشتن کد DSP برای ARM

در نوشتن کد برای DSP بر روی ARM به چندین نکته توجه کنید:

۱. الگوریتم‌های DSP را چنان بنویسید که از اشباع استفاده نکنند. استفاده از اشباع به معنی هدر رفتن سیکل‌های اضافی است. سیگنال‌ها را چنان مقیاس بندی کنید که نیازی به اشباع نداشته باشند؛
۲. الگوریتم‌ها را چنان بنویسید که از کمترین تعداد Load و Store استفاده کنند؛
۳. از دستورهای اسمبلی به منظور کاهش interlock در نوشتن کدها استفاده کنید. نتایج دستورهای Load و Store تا چندین سیکل در دسترس نخواهند بود؛
۴. تنها ۱۴ ثبات *r0* تا *r12* و *r14* برای استفاده‌های عمومی در دسترس هستند. طوری برنامه بنویسید که از ۱۴ ثبات یا کمتر استفاده کنند.

۱۵.۳ راهنمای نوشتن کد DSP برای ARM7TDMI

۱. دستور Load کُند بوده و برای بارگیری یک مقدار عددی ۳ سیکل کاری طول می‌کشد. برای دسترسی بهینه به حافظه، از دستورهای Load و Store چندگانه، یعنی LDM و STM استفاده کنید. این دستورها برای هر عمل Load و یا Store اضافی تنها یک سیکل تلف می‌کند؛
۲. دستور ضرب تنها یک سیکل از دستور ضرب–جمع سریع‌تر است. بعضی مواقع استفاده از دستورها جمع و ضرب جداگانه سودمند است. در این‌حالت می‌توانید از Barrel Shifter به همراه دستور ADD برای عملیات جمع مقیاس‌بندی شده استفاده کنید؛
۳. اغلب اوقات استفاده از ضرب با مقادیر ثابت، عملیات سریع‌تری را به دست می‌دهد. برای مثال $240x = (x<<8) - (x<<4)$. برای تمام ضرایب به شکل $\pm 2^A \pm 2^B \pm 2^C$ می‌توان از دستورالعملی مشابه فوق استفاده کرد.

مثال ۱۵.۱

این مثال، پیاده‌سازی ضرب نقطه‌ای ۱۶ بیتی بهینه شده برای ARM7TDMI را نشان می‌دهد. هر MLA در بدترین حالت، ۴ سیکل طول می‌کشد.

```
x   RN 0 ; input array x[]
c   RN 1 ; input array c[]
N   RN 2 ; number of samples (a multiple of 5(
acc RN 3 ; accumulator
x_0 RN 4 ; elements from array x[]
x_1 RN 5
x_2 RN 6
x_3 RN 7
x_4 RN 8
c_0 RN 9 ; elements from array c[]
c_1 RN 10
```

```
c_2 RN 11
c_3 RN 12
c_4 RN 14
     ; int dot_16by16_arm7m(int *x, int *c, unsigned N)
dot_16by16_arm7m
    STMFD sp!, {r4-r11, lr}
    MOV acc, #0
loop_7m ; accumulate 5 products
    LDMIA x!, {x_0, x_1, x_2, x_3, x_4{
    LDMIA c!, {c_0, c_1, c_2, c_3, c_4{
    MLA acc, x_0, c_0, acc
    MLA acc, x_1, c_1, acc
    MLA acc, x_2, c_2, acc
    MLA acc, x_3, c_3, acc
    MLA acc, x_4, c_4, acc
    SUBS N, N, #5
    BGT loop_7m
    MOV r0, acc
    LDMFD sp!, {r4-r11, pc}
```

در این مثال، فرض براین‌است که تعداد نمونه‌ها، ضریبی از ۵ است. بنابراین، می‌توانیم از دستور LDM برای بارگیری ۵-کلمه‌ای به منظور افزایش سرعت استفاده کنیم. زمان هر Load در این‌جا 7/4Cycle = 1.4 Cycle در مقایسه با دستور LDR که ۳ سیکل برای هر دستور است. حلقه‌ی داخلی در بدترین حالت 38=3+1+4*5+7+7 سیکل طول می‌کشد. این به معنای نرخ DSP معادل 38/5=7.6 سیکل به ازای هر انشعاب در ARM7TDMI است.

۱۵.۴ راهنمای نوشتن کد DSP برای ARM9E

هسته‌ی ARM9E ، آرایه‌ای از ضرب‌کننده‌های خط لوله‌ای سریع ۳۲ بیتی در ۱۶ بیتی دارد که در یک سیکل این عمل را انجام می‌دهد. نتیجه‌ی کار در سیکل بعد در دسترس نیست. مگر این‌که این نتیجه برای یک سری عملیات MAC ، مورد نیاز باشد.

۱. معماری ARMv5TE توانایی نصف کردن مقادیر ۳۲ بیتی به مقادیر ۱۶ بیتی و ضرب نمودن آن‌ها را دارد. برای بهترین استفاده از پهنای باند، بهتر از دستورهای Load ۳۲ بیتی استفاده کنیم؛

۲. سرعت ضرب با سرعت عمل MAC یکی است. به جای استفاده از عملیات ضرب و جمع جداگانه، از دستور SMLAxy استفاده کنید.

مثال ۱۵.۱

این مثال، پیاده‌سازی ضرب نقطه‌ای ۱۶ بیتی بهینه شده برای ARM9E را نشان می‌دهد. در این‌جا فرض شده است که سیستم حافظه در حالت Little-endian تنظیم شده است. در حالت Big-endian

می‌توانیم جای پسوندهای B و T را در دستورها عوض کنیم. البته، می‌توانید از macro ها برای انجـام این‌کار استفاده کنید.

```
x    RN 0 ; input array x[]
c    RN 1 ; input array c[]
N    RN 2 ; number of samples (a multiple of 8)
acc  RN 3 ; accumulator
x_10 RN 4 ; packed elements from array x[]
x_32 RN 5
c_10 RN 9 ; packed elements from array c[]
c_32 RN 10
      ; int dot_16by16_arm9e(short *x, short *c, unsigned N)
dot_16by16_arm9e
      STMFD sp!, {r4-r5, r9-r10, lr}
      MOV acc, #0
      LDR x_10, [x], #4
      LDR c_10, [c], #4
loop_9e ; accumulate 8 products
      SUBS N, N, #8
      LDR x_32, [x], #4
      SMLABB acc, x_10, c_10, acc
      LDR c_32, [c], #4
      SMLATT acc, x_10, c_10, acc
      LDR x_10, [x], #4
      SMLABB acc, x_32, c_32, acc
      LDR c_10, [c], #4
      SMLATT acc, x_32, c_32, acc
      LDR x_32, [x], #4
      SMLABB acc, x_10, c_10, acc
      LDR c_32, [c], #4
      SMLATT acc, x_10, c_10, acc
      LDRGT x_10, [x], #4
      SMLABB acc, x_32, c_32, acc
      LDRGT c_10, [c], #4
      SMLATT acc, x_32, c_32, acc
      BGT loop_9e
      MOV r0, acc
      LDMFD sp!, {r4-r5, r9-r10, pc}
```

حلقه داخلی ۲۰ سیکل، برای جمع کردن ۸ حاصل‌ضرب زمان نیاز دارد.

۱۵.۵ فیلترهای FIR

فیلترهای FIR ، از اساسی‌ترین عناصر ابتدایی بسیاری از کاربردهای DSP است. از این‌رو، نیاز به دقت و توجه بیشتری دارند. با استفاده از این فیلترها، می‌توان فرکانس‌های خاصـی را حـذف کـرده، بعضـی فرکانس‌ها را تقویت کرده و یا افکت‌های خاصی را به سیگنال اعمال کنید. در این‌جا به پیاده‌سازی بهینه-

ی این فیلترها بر روی ARM می‌پردازیم. فیلترهای FIR ، ساده‌ترین نوع فیلترهای دیجیتال هستند. خروجی فیلتر شده y_t به صورت خطی به تعداد محدودی از نمونه‌های ورودی x_t بستگی دارند:

$$y_t = \sum_{i=0}^{M-1} c_i x_{t-i}$$

بعضی کتاب‌ها c_i را به عنوان پاسخ ضربه[1] می‌شناسند. اگر به فیلتر فوق یک ورودی ضربه $x =$ (1,0,0,...) را به آن اعمال کنید، خروجی آن ضرایب فیلتر $y = (c0, c1, \ldots)$ است. پیاده‌سازی فیلتر به این صورت است:

A[t] = C[0]*X[t] + C[1]*X[t-1] + ... + C[M-1]*X[t-M+1]

X[t] و C[i] معمولا اعداد صحیح k بیتی و A[t] عددی 2k بیتی است. (k عموما ۸، ۱۶ و یا ۳۲ بیتی است) در جدول زیر، دقت مورد استفاده در بعضی کاربردهای معمول را می‌بینیم.

Application	دقت X[t] بر حسب بیت	دقت C[i] بر حسب بیت	دقت A[t] بر حسب بیت
تصویر	8	8	16
صدا در مخابرات	16	16	32
صدای با کیفیت	32	32	64

در این‌جا به فیلترهای بلند می‌پردازیم. فیلترهای بلند به فیلترهایی اشاره دارد که M در آن مقدار بزرگی است به نحوی که ضرایب آن در ثبات‌ها جای نمی‌گیرند.

مثال ۸.۱۰

این مثال، نمونه‌ای از پیاده‌سازی یک فیلتر بلوکی ۶ در ۶ برای پردازنده‌های ARMv5TE است. این روال نسبتا طولانی است. علت آن تلاش برای بهینه‌سازی سرعت بوده است.

```
a RN 0 ; array for output samples a[]
x RN 1 ; array of input samples x[] (32-bit aligned)
c RN 2 ; array of coefficients c[] (32-bit aligned)
N RN 3 ; number of outputs (a multiple of 6)
M RN 4 ; number of coefficients (a multiple of 6)
c_10 RN 0 ; coefficient pairs
c_32 RN 3
x_10 RN 5 ; sample pairs
x_32 RN 6
x_54 RN 7
a_0 RN 8 ; output accumulators
a_1 RN 9
a_2 RN 10
a_3 RN 11
```

[1] Impulse Response

```
a_4 RN 12
a_5 RN 14
        ;void fir_16by16_arm9e
        ;(int *a,
        ;short *x,
        ;struct { short *c; unsigned int M; } *c,
        ;unsigned int N(
fir_16by16_arm9e
       STMFD sp!, {r4-r11, lr}
       LDMIA c, {c, M}
next_sample_arm9e
       STMFD sp!, {a, N, M}
       LDMIA x!, {x_10, x_32, x_54} ; preload six samples
       MOV a_0, #0 ; zero accumulators
       MOV a_1, #0
       MOV a_2, #0
       MOV a_3, #0
       MOV a_4, #0
       MOV a_5, #0
next_tap_arm9e
        ;perform next block of 6x6=36 taps
       LDMIA c!, {c_10, c_32} ; load four coefficients
       SUBS M, M, #6
       SMLABB a_0, x_10, c_10, a_0
       SMLATB a_1, x_10, c_10, a_1
       SMLABB a_2, x_32, c_10, a_2
       SMLATB a_3, x_32, c_10, a_3
       SMLABB a_4, x_54, c_10, a_4
       SMLATB a_5, x_54, c_10, a_5
       SMLATT a_0, x_10, c_10, a_0
       LDR x_10, [x], #4 ; load two coefficients
       SMLABT a_1, x_32, c_10, a_1
       SMLATT a_2, x_32, c_10, a_2
       SMLABT a_3, x_54, c_10, a_3
       SMLATT a_4, x_54, c_10, a_4
       SMLABT a_5, x_10, c_10, a_5
       LDR c_10, [c], #4
       SMLABB a_0, x_32, c_32, a_0
       SMLATB a_1, x_32, c_32, a_1
       SMLABB a_2, x_54, c_32, a_2
       SMLATB a_3, x_54, c_32, a_3
       SMLABB a_4, x_10, c_32, a_4
       SMLATB a_5, x_10, c_32, a_5
       SMLATT a_0, x_32, c_32, a_0
       LDR x_32, [x], #4
       SMLABT a_1, x_54, c_32, a_1
       SMLATT a_2, x_54, c_32, a_2
       SMLABT a_3, x_10, c_32, a_3
       SMLATT a_4, x_10, c_32, a_4
       SMLABT a_5, x_32, c_32, a_5
       SMLABB a_0, x_54, c_10, a_0
       SMLATB a_1, x_54, c_10, a_1
       SMLABB a_2, x_10, c_10, a_2
       SMLATB a_3, x_10, c_10, a_3
       SMLABB a_4, x_32, c_10, a_4
```

```
SMLATB a_5, x_32, c_10, a_5
SMLATT a_0, x_54, c_10, a_0
LDR x_54, [x], #4
SMLABT a_1, x_10, c_10, a_1
SMLATT a_2, x_10, c_10, a_2
SMLABT a_3, x_32, c_10, a_3
SMLATT a_4, x_32, c_10, a_4
SMLABT a_5, x_54, c_10, a_5
BGT next_tap_arm9e
LDMFD sp!, {a, N, M}
STMIA a!, {a_0, a_1, a_2, a_3, a_4, a_5}
SUB c, c, M, LSL#1 ; restore coefficient pointer
SUB x, x, M, LSL#1 ; advance data pointer
SUBS N, N, #6
BGT next_sample_arm9e
LDMFD sp!, {r4-r11, pc}
```

۱۵.۵.۱ نوشتن فیلترهای FIR بر روی ARM

۱. اگر تعداد ضرایب فیلترهای FIR ، به اندازه کافی کوچک باشند، ضرایب و تاریخچه مقادیر را در ثبات‌ها نگهدارید. اغلب پیش می‌آید که ضرایب تکراری باشند. در این حالت، تعداد ثبات‌های مورد نیاز کاهش خواهند داشت؛

۲. اگر طول فیلتر FIR بلند باشد، از الگوریتم فیلتر بلوکی R×(R-1) یا R×R استفاده کنید. بزرگ‌ترین مقدار R ممکن را با استفاده از ۱۴ ثبات عمومی در دسترس، انتخاب کنید؛

۳. طول آرایه‌های ورودی را ضریبی از اندازه بلوک انتخاب کنید.

۱۵.۶ فیلترهای IIR

یک فیلتر با پاسخ ضربه نامحدود[۱] یک فیلتر دیجیتال است که به طور خطی به تعداد محدودی از نمونه-های ورودی فعلی و تعداد محدودی از نمونه‌های خروجی قبلی، بستگی دارد. به عبارت دیگر، این فیلتر ترکیبی از فیلتر FIR و فیدبکی از خروجی‌های قبلی فیلتر است. رابطه آن به این صورت بیان می‌شود:

$$y_t = \sum_{i=0}^{M} b_i x_{t-i} - \sum_{j=1}^{L} a_j y_{t-j}$$

در عمل، استفاده از M=L=2 روشی بسیار سریع‌تر و بهینه تر به‌دست می‌دهد. به این گروه‌های دوتایی، biquad گفته می‌شود.

$$y_t = b_0 x_t + b_1 x_{t-1} + b_2 x_{t-2} - a_1 y_{t-1} - a_2 y_{t-2}$$

[۱] Infinite Impulse Response - IIR

برای پیاده‌سازی فیلتر IIR ، می‌توانیم از یک سری از biquad ها استفاده کنیم. بـا اسـتفاده از تبـدیل z پیاده‌سازی زیر را داریم:

$$\left(1 + a_1 z^{-1} + \cdots + a_L z^{-L}\right) y(z) = \left(b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}\right) x(z)$$

یا به‌طور معادل:

$$y(z) = H(z)x(z), \text{ where } H(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_M z^{-M}}{1 + a_1 z^{-1} + \cdots + a_L z^{-L}}$$

سپس، H(z) را به عنوان نسبت دو چند جمله‌ای از z^{-1} در نظر می‌گیریم. چندجمله‌ای ها را مـی‌تـوان بـه عبارتهای درجه دوم فاکتور گرفت.

مثال ۸.۱۴

این مثال، نمونه‌ای از یک فیلتر بلوکی 2×1 را بر روی ARM7TDMI پیاده‌سـازی مـی‌کنـد. هـر حلقـه‌ی وروی یک عبارت درجه دوم را به دونمونه‌ی ورودی بعدی اعمال می‌کند.

```
y RN 0 ; address for output samples y[]
x RN 1 ; address of input samples x[]
b RN 2 ; address of biquads
N RN 3 ; number of samples to filter (a multiple of 2)
M RN 4 ; number of biquads to apply
x_0 RN 2 ; input samples
x_1 RN 4
a_1 RN 6 ; biquad coefficient -a[1] at Q14
a_2 RN 7 ; biquad coefficient -a[2] at Q14
b_1 RN 8 ; biquad coefficient +b[1] at Q14
b_2 RN 9 ; biquad coefficient +b[2] at Q14
s_1 RN 10 ; s[t-1] then s[t-2] (alternates)
s_2 RN 11 ; s[t-2] then s[t-1] (alternates)
acc0 RN 12 ; accumulators
acc1 RN 14
      ;typedef struct {
      ;int a1,a2; /* coefficients -a[1],-a[2] at Q14 */
      ;int b1,b2; /* coefficients +b[1],+b[2] at Q14 */
      ;int s1,s2; /* s[t-1], s[t-2]*/
      ; } biquad;
     ;
      ;void iir_q14_arm7m
      ;(int *y,
      ;int *x,
      ;struct { biquad *b; unsigned int M; } *b,
      ;unsigned int N);
iir_q14_arm7m
      STMFD sp!, {r4-r11, lr}
      LDMIA b, {b, M}
```

```
next_biquad_arm7m
       LDMIA b!, {a_1, a_2, b_1, b_2, s_1, s_2}
       STMFD sp!, {b, N, M}
next_sample_arm7m
        ;use a 2x1 block IIR
       LDMIA x!, {x_0, x_1}
        ;apply biquad to sample 0 (x_0)
       MUL acc0, s_1, a_1
       MLA acc0, s_2, a_2, acc0
       MUL acc1, s_1, b_1
       MLA acc1, s_2, b_2, acc1
       ADD s_2, x_0, acc0, ASR #14
       ADD x_0, s_2, acc1, ASR #14
        ;apply biquad to sample 1 (x_1)
       MUL acc0, s_2, a_1
       MLA acc0, s_1, a_2, acc0
       MUL acc1, s_2, b_1
       MLA acc1, s_1, b_2, acc1
       ADD s_1, x_1, acc0, ASR #14
       ADD x_1, s_1, acc1, ASR #14
       STMIA y!, {x_0, x_1}
       SUBS N, N, #2
       BGT next_sample_arm7m
       LDMFD sp!, {b, N, M}
       STMDB b, {s_1, s_2}
       SUB y, y, N, LSL#2
       MOV x, y
       SUBS M, M, #1
       BGT next_biquad_arm7m
       LDMFD sp!, {r4-r11, pc}
```

۱۵.۶.۱ پیاده‌سازی فیلترهای IIR ۱۶ بیتی

۱. فیلتر IIR را مجموعه‌هایی از جمله‌های درجه دوم فاکتورگیری کنید. دقت مناسبی از داده ها را برای این‌که سرریز نداشته باشید، انتخاب کنید؛

۲. سیگنال ورودی را به فریم‌های بزرگ تقسیم کرده و یک الگوریتم IIR بلوکی اعمال کنید.

۱۵.۷ تبدیل فوریه‌ی گسسته[۱]

تبدیل فوریه‌ی گسسته، یک سیگنال ورودی حوزه زمان را به سیگنال خروجی حوزه فرکانس تبدیل می‌کند. یک تبدیل معکوس مرتبط با تبدیل فوق به نام IDFT نیز وجود دارد که عکس عمل فوق را انجام می‌دهد.

این الگوریتم، به دفعات در پردازش سیگنال استفاده می‌شود. یک الگوریتم ساده‌تر به نام تبدیل فوریه سریع[۲] وجود دارد که پیاده‌سازی DFT را بسیار سریع‌تر و راحت‌تر نموده است. در این بخش به پیاده‌سازی چندین نمونه از FFT بر روی معماری ARM خواهیم پرداخت.

DFT بر روی N نمونه مختلط زمانی عمل کرده و آن‌ها را به N نمونه مختلط از ضرایب فرکانسی تبدیل می‌کند. فرمول‌های پایه برای پیاده‌سازی الگوریتم فوق را در زیر می‌بینیم:

$$y = DFT_N(x) \text{ means } y_k = \sum_{t=0}^{N-1} x_t w_N^{kt}, \quad \text{where } w_N = e^{-2\pi i/N}$$

$$x = IDFT_N(y) \text{ means } x_t = \frac{1}{N} \sum_{k=0}^{N-1} y_k w_N^{kt}, \quad \text{where } w_N = e^{2\pi i/N}$$

۱۵.۷.۱ تبدیل فوریه‌ی سریع

ایده تبدیل فوریه‌ی سریع، شکستن تبدیل فوریه‌ی گسسته، به فاکتورهای N تایی است. فرض کنید N=R×S. خروجی را به S بلوک با اندازه‌ی R و ورودی را به R بلوک با اندازه‌ی S تقسیم کنید.

$$k = nR + m \quad \text{for } n = 0, 1, \ldots, S-1 \quad \text{and} \quad m = 0, 1, \ldots, R-1$$
$$t = rS + s \quad \text{for } r = 0, 1, \ldots, R-1 \quad \text{and} \quad s = 0, 1, \ldots, S-1$$

سپس

$$y[nR + m] = \sum_{r=0}^{R-1} \sum_{s=0}^{S-1} x[rS + s] w_N^{(nR+m)(rS+s)}$$

$$y[nR + m] = \sum_{s=0}^{S-1} w_S^{ns} w_N^{ms} \left(\sum_{r=0}^{R-1} x[rS + s] w_R^{mr} \right)$$

معادله بالا، N DFT- نقطه‌ای را به S مجموعه R DFT- نقطه‌ای تبدیل می‌کند.

---

[۱] Discrete Fourier Transform - DFT

[۲] Fast Fourier Transform - FFT

نکته

بهترین حالت وقتی حاصل می‌شود که N توانی از ۲ باشد.

نمونه‌ای از پیاده‌سازی FFT را در زیر مـی‌بینـیم. در ایـن کـد، از Barrel Shifter بـرای مقیـاس‌بنـدی ورودی استفاده بهینه شده است.

```
x0_r RN 4 ; data register (real part)
x0_i RN 5 ; data register (imaginary part)
x1_r RN 6
x1_i RN 7
x2_r RN 8
x2_i RN 9
x3_r RN 10
x3_i RN 11
y3_r RN x3_i
y3_i RN x3_r
        ;Four-point complex Fast Fourier Transform
       ;
       ;  (x0,x1,x2,y3)=DFT4(x0,x2 >> s,x1 >> s,x3 >> s)/4
       ;
        ;x0 = (x0 + (x2 >> s) + (x1 >> s) + (x3 >> s))/4
        ;x1 = (x0 - i*(x2 >> s) - (x1 >> s) + i*(x3 >> s))/4
        ;x2 = (x0 - (x2 >> s) + (x1 >> s) - (x3 >> s))/4
        ;y3 = (x0 + i*(x2 >> s) - (x1 >> s) - i*(x3 >> s))/4
       ;
       MACRO
       C_FFT4 $s
        ;(x2,x3) = (x2+x3, x2-x3)
       ADD x2_r, x2_r, x3_r
       ADD x2_i, x2_i, x3_i
       SUB x3_r, x2_r, x3_r, LSL#1
       SUB x3_i, x2_i, x3_i, LSL#1
        ;(x0,x1) = (x0+(x1 >> s), x0-(x1 >> s))/4
       MOV x0_r, x0_r, ASR#2
       MOV x0_i, x0_i, ASR#2
       ADD x0_r, x0_r, x1_r, ASR#(2+$s)
       ADD x0_i, x0_i, x1_i, ASR#(2+$s)
       SUB x1_r, x0_r, x1_r, ASR#(1+$s)
       SUB x1_i, x0_i, x1_i, ASR#(1+$s)
        ;(x0,x2) = (x0+(x2 >> s)/4, x0-(x2 >> s)/4)
       ADD x0_r, x0_r, x2_r, ASR#(2+$s)
       ADD x0_i, x0_i, x2_i, ASR#(2+$s)
       SUB x2_r, x0_r, x2_r, ASR#(1+$s)
       SUB x2_i, x0_i, x2_i, ASR#(1+$s)
```

```
 ;(x1,y3) = (x1-i*(x3 >> s)/4, x1+i*(x3 >> s)/4)
ADD x1_r, x1_r, x3_i, ASR#(2+$s)
SUB x1_i, x1_i, x3_r, ASR#(2+$s)
SUB y3_r, x1_r, x3_i, ASR#(1+$s)
ADD y3_i, x1_i, x3_r, ASR#(1+$s)
MEND
```

همچنین، از ماکروهای زیر برای ذخیره و بارگیری مقادیر مختلط استفاده می‌کنیم:

```
; complex load, x=[a], a+=offset
MACRO
C_LDR $x, $a, $offset
LDRSH $x._i, [$a, #2]
LDRSH $x._r, [$a], $offset
MEND
; complex store, [a]=x, a+=offset
MACRO
C_STR $x, $a, $offset
STRH $x._i, [$a, #2]
STRH $x._r, [$a], $offset
MEND
```

مثال ۸.۱۷

در این مثال، نمونه‌ای از پیاده‌سازی FFT ۱۶ بیتی پایه ۴ را برای انواع مختلف معماری پردازنده‌ی ARMv4 می‌بینیم. فرض می‌کنیم که تعداد نقاط، $n=4^b$ است.

```
; Complex conjugate multiply a=(xr+i*xi)*(cr-i*ci)
; x = xr + i*xi
; w = (cr-ci) + i*ci
MACRO
C_MUL9m $a, $x, $w
SUB t1, $x._i, $x._r ; (xi-xr)
MUL t0, t1, $w._i ; (xi-xr)*ci
ADD t1, $w._r, $w._i, LSL#1 ; (cr+ci)
MLA $a._i, $x._i, $w._r, t0 ; xi*cr-xr*ci
MLA $a._r, $x._r, t1, t0 ; xr*cr+xi*ci
MEND
y RN 0 ; output complex array y[]
c RN 0 ; coefficient array
x RN 1 ; input complex array x[]
N RN 2 ; number of samples (a power of 2)
S RN 2 ; the number of blocks
R RN 3 ; the number of samples in each block
x0_r RN 4 ; data register (real part)
x0_i RN 5 ; data register (complex part)
x1_r RN 6
x1_i RN 7
x2_r RN 8
x2_i RN 9
x3_r RN 10
```

```
x3_i RN 11
y3_r RN x3_i
y3_i RN x3_r
t0 RN 12 ; scratch register
t1 RN 14
       ; void fft_16_arm9m(short *y, short *x, unsigned int N)
fft_16_arm9m
       STMFD sp!, {r4-r11, lr}
       MOV t0, #0 ; bit reversed counter
first_stage_arm9m
       ; first stage load and bit reverse
       ADD t1, x, t0, LSL#2
       C_LDR x0, t1, N
       C_LDR x2, t1, N
       C_LDR x1, t1, N
       C_LDR x3, t1, N
       C_FFT4 0
       C_STR x0, y, #4
       C_STR x1, y, #4
       C_STR x2, y, #4
       C_STR y3, y, #4
       EOR t0, t0, N, LSR#3 ; increment third bit
       TST t0, N, LSR#3 ; from the top
       BNE first_stage_arm9m
       EOR t0, t0, N, LSR#4 ; increment fourth bit
       TST t0, N, LSR#4 ; from the top
       BNE first_stage_arm9m
       MOV t1, N, LSR#5 ; increment fifth
bit_reversed_count_arm9m ; bits downward
       EOR t0, t0, t1
       TST t0, t1
       BNE first_stage_arm9m
       MOVS t1, t1, LSR#1
       BNE bit_reversed_count_arm9m
       ; finished the first stage
       SUB x, y, N, LSL#2 ; x = working buffer
       MOV R, #16
       MOVS S, N, LSR#4
       LDMEQFD sp!, {r4-r11, pc}
       ADR c, fft_table_arm9m
next_stage_arm9m
       ; S = the number of blocks
       ; R = the number of samples in each block
       STMFD sp!, {x, S}
       ADD t0, R, R, LSL#1
       ADD x, x, t0
       SUB S, S, #1 << 16
next_block_arm9m
       ADD S, S, R, LSL#(16-2)
next_butterfly_arm9m
       ; S=((number butterflies left-1) << 16)
       ; + (number of blocks left)
       C_LDR x0, x, -R
       C_LDR x3, c, #4
       C_MUL9m x3, x0, x3
```

```
        C_LDR x0, x, -R
        C_LDR x2, c, #4
        C_MUL9m x2, x0, x2
        C_LDR x0, x, -R
        C_LDR x1, c, #4
        C_MUL9m x1, x0, x1
        C_LDR x0, x, #0
        C_FFT4 14 ; coefficients are Q14
        C_STR x0, x, R
        C_STR x1, x, R
        C_STR x2, x, R
        C_STR y3, x, #4
        SUBS S, S, #1 << 16
        BGE next_butterfly_arm9m
        ADD t0, R, R, LSL#1
        ADD x, x, t0
        SUB S, S, #1
        MOVS t1, S, LSL#16
        SUBNE c, c, t0
        BNE next_block_arm9m
        LDMFD sp!, {x, S}
        MOV R, R, LSL#2 ; quadruple block size
        MOVS S, S, LSR#2 ; quarter number of blocks
        BNE next_stage_arm9m
        LDMFD sp!, {r4-r11, pc}
fft_table_arm9m
        ; FFT twiddle table of triplets E(3t), E(t), E(2t)
        ; Where E(t)=(cos(t)-sin(t))+i*sin(t) at Q14
        ; N=16 t=2*PI*k/N for k=0,1,2,..,N/4-1
        DCW 0x4000,0x0000, 0x4000,0x0000, 0x4000,0x0000
        DCW 0xdd5d,0x3b21, 0x22a3,0x187e, 0x0000,0x2d41
        DCW 0xa57e,0x2d41, 0x0000,0x2d41, 0xc000,0x4000
        DCW 0xdd5d,0xe782, 0xdd5d,0x3b21, 0xa57e,0x2d41
        ; N=64 t=2*PI*k/N for k=0,1,2,..,N/4-1
        DCW 0x4000,0x0000, 0x4000,0x0000, 0x4000,0x0000
        DCW 0x2aaa,0x1294, 0x396b,0x0646, 0x3249,0x0c7c
        DCW 0x11a8,0x238e, 0x3249,0x0c7c, 0x22a3,0x187e
        DCW 0xf721,0x3179, 0x2aaa,0x1294, 0x11a8,0x238e
        DCW 0xdd5d,0x3b21, 0x22a3,0x187e, 0x0000,0x2d41
        DCW 0xc695,0x3fb1, 0x1a46,0x1e2b, 0xee58,0x3537
        DCW 0xb4be,0x3ec5, 0x11a8,0x238e, 0xdd5d,0x3b21
        DCW 0xa963,0x3871, 0x08df,0x289a, 0xcdb7,0x3ec5
        DCW 0xa57e,0x2d41, 0x0000,0x2d41, 0xc000,0x4000
        DCW 0xa963,0x1e2b, 0xf721,0x3179, 0xb4be,0x3ec5
        DCW 0xb4be,0x0c7c, 0xee58,0x3537, 0xac61,0x3b21
        DCW 0xc695,0xf9ba, 0xe5ba,0x3871, 0xa73b,0x3537
        DCW 0xdd5d,0xe782, 0xdd5d,0x3b21, 0xa57e,0x2d41
        DCW 0xf721,0xd766, 0xd556,0x3d3f, 0xa73b,0x238e
        DCW 0x11a8,0xcac9, 0xcdb7,0x3ec5, 0xac61,0x187e
        DCW 0x2aaa,0xc2c1, 0xc695,0x3fb1, 0xb4be,0x0c7c
        ; N=256 t=2*PI*k/N for k=0,1,2,..,N/4-1
        ;... continue as necessary ...
```

مثال ۸.۱۸

در این مثال، نمونه‌ای از پیاده‌سازی FFT پایه ۴ را برای معماری پردازنده‌ی ARMv5TE همانند پردازنده‌ی ARM9E می‌بینیم. فرض می‌کنیم که تعداد نقاط، $n=4^b$ است.

```
        ; Complex conjugate multiply a=(xr+i*xi)*(cr-i*ci)
        ; x = xr + i*xi (two 16-bits packed in 32-bit)
        ; w = cr + i*ci (two 16-bits packed in 32-bit)
        MACRO
        C_MUL9e $a, $x, $w
        SMULBT t0, $x, $w ; xr*ci
        SMULBB $a._r, $x, $w ; xr*cr
        SMULTB $a._i, $x, $w ; xi*cr
        SMLATT $a._r, $x, $w, $a._r ; xr*cr+xi*ci
        SUB $a._i, $a._i, t0 ; xi*cr-xr*ci
        MEND
        ; void fft_16_arm9e(short *y, short *x, unsigned int N)
fft_16_arm9e
        STMFD sp!, {r4-r11, lr}
        MOV t0, #0 ; bit-reversed counter
        MVN R, #0x80000000 ; R=0x7FFFFFFF
first_stage_arm9e
        ; first stage load and bit reverse
        ADDS t1, x, t0, LSL#2 ; t1=&x[t0] and clear carry
        C_LDR x0, t1, N
        C_LDR x2, t1, N
        C_LDR x1, t1, N
        C_LDR x3, t1, N
        C_FFT4 0
        C_STR x0, y, #4
        C_STR x1, y, #4
        C_STR x2, y, #4
        C_STR y3, y, #4
        ; bit reversed increment modulo (N/4)
        RSC t0, t0, N, LSR #2 ; t0 = (N/4)-t0-1
        CLZ t1, t0 ; find leading 1
        EORS t0, t0, R, ASR t1 ; toggle bits below leading 1
        BNE first_stage_arm9e ; loop if count nonzero
        ; finished the first stage
        SUB x, y, N, LSL #2 ; x = working buffer
        MOV R, #16
        MOVS S, N, LSR#4
        LDMEQFD sp!, {r4-r11, pc}
        ADR c, fft_table_arm9e
next_stage_arm9e
        ; S = the number of blocks
        ; R = the number of samples in each block
        STMFD sp!, {x, S}
        ADD t0, R, R, LSL#1
        ADD x, x, t0
        SUB S, S, #1 << 16
next_block_arm9e
        ADD S, S, R, LSL#(16-2)
```

```
next_butterfly_arm9e
       ; S=((number butterflies left-1) << 16)
       ; + (number of blocks left)
       LDR x2_r, [x], -R ; packed data
       LDR x2_i, [c], #4 ; packed coefficients
       LDR x1_r, [x], -R
       LDR x1_i, [c], #4
       LDR x0_r, [x], -R
       LDR x0_i, [c], #4
       C_MUL9e x3, x2_r, x2_i
       C_MUL9e x2, x1_r, x1_i
       C_MUL9e x1, x0_r, x0_i
       C_LDR x0, x, #0
       C_FFT4 15 ; coefficients are Q15
       C_STR x0, x, R
       C_STR x1, x, R
       C_STR x2, x, R
       C_STR y3, x, #4
       SUBS S, S, #1 << 16
       BGE next_butterfly_arm9e
       ADD t0, R, R, LSL#1
       ADD x, x, t0
       SUB S, S, #1
       MOVS t1, S, LSL#16
       SUBNE c, c, t0
       BNE next_block_arm9e
       LDMFD sp!, {x, S}
       MOV R, R, LSL#2 ; quadruple block size
       MOVS S, S, LSR#2 ; quarter number of blocks
       BNE next_stage_arm9e
       LDMFD sp!, {r4-r11, pc}
fft_table_arm9e
       ; FFT twiddle table of triplets E(3t), E(t), E(2t)
       ; Where E(t)=cos(t)+i*sin(t) at Q15
       ; N=16 t=2*PI*k/N for k=0,1,2,..,N/4-1
       DCW 0x7fff,0x0000, 0x7fff,0x0000, 0x7fff,0x0000
       DCW 0x30fc,0x7642, 0x7642,0x30fc, 0x5a82,0x5a82
       DCW 0xa57e,0x5a82, 0x5a82,0x5a82, 0x0000,0x7fff
       DCW 0x89be,0xcf04, 0x30fc,0x7642, 0xa57e,0x5a82
       ; N=64 t=2*PI*k/N for k=0,1,2,..,N/4-1
       DCW 0x7fff,0x0000, 0x7fff,0x0000, 0x7fff,0x0000
       DCW 0x7a7d,0x2528, 0x7f62,0x0c8c, 0x7d8a,0x18f9
       DCW 0x6a6e,0x471d, 0x7d8a,0x18f9, 0x7642,0x30fc
       DCW 0x5134,0x62f2, 0x7a7d,0x2528, 0x6a6e,0x471d
       DCW 0x30fc,0x7642, 0x7642,0x30fc, 0x5a82,0x5a82
       DCW 0x0c8c,0x7f62, 0x70e3,0x3c57, 0x471d,0x6a6e
       DCW 0xe707,0x7d8a, 0x6a6e,0x471d, 0x30fc,0x7642
       DCW 0xc3a9,0x70e3, 0x62f2,0x5134, 0x18f9,0x7d8a
       DCW 0xa57e,0x5a82, 0x5a82,0x5a82, 0x0000,0x7fff
       DCW 0x8f1d,0x3c57, 0x5134,0x62f2, 0xe707,0x7d8a
       DCW 0x8276,0x18f9, 0x471d,0x6a6e, 0xcf04,0x7642
       DCW 0x809e,0xf374, 0x3c57,0x70e3, 0xb8e3,0x6a6e
       DCW 0x89be,0xcf04, 0x30fc,0x7642, 0xa57e,0x5a82
       DCW 0x9d0e,0xaecc, 0x2528,0x7a7d, 0x9592,0x471d
       DCW 0xb8e3,0x9592, 0x18f9,0x7d8a, 0x89be,0x30fc
```

```
DCW 0xdad8,0x8583, 0x0c8c,0x7f62, 0x8276,0x18f9
; N=256 t=2*PI*k/N for k=0,1,2,..,N/4-1
; .... continue as required ....
```

۱۵.۸ خلاصه

در این فصل، نمونه‌های مختلفی از بلوک‌های سازنده‌ی پایه‌ی DSP را برای معماری ARM پیاده‌سازی کردیم. در جداول زیر، کارایی به‌دست آمده از این کدها را برحسب تعداد سیکل مصرفی‌شان می‌بینیم.

Processor core	16-bit dot-product (cycles/tap)	16-bit block FIR filter (cycles/tap)	32-bit block FIR filter (cycles/tap)	16-bit block IIR filter (cycles/biquad)
ARM7TDMI	7.6	5.2	9.0	22.0
ARM9TDMI	7.0	4.8	8.3	22.0
StrongARM	4.8	3.8	5.2	16.5
ARM9E	2.5	1.3	4.3	8.0
XScale	1.8	1.3	3.7	7.7

16-bit complex FFT (radix-4)	ARM9TDMI (cycles/FFT)	ARM9E (cycles/FFT)
64 point	3,524	2,480
256 point	19,514	13,194
1,024 point	99,946	66,196
4,096 point	487,632	318,878

فصل شانزدهم

بهینه سازی کدهای اسمبلی و C

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ بهینه سازی کدها در زبان اسمبلی؛
- ✓ بهینه سازی کدها در زبان C؛
- ✓ بهینه سازی ساختارهای حلقه.

۱۶.۱ نوشتن کدهای زبان اسمبلی و بهینه سازی آن‌ها

سیستم‌های تعبیه شده، اغلب چندین زیربرنامه‌ی کلیدی دارند که کارآیی سیستم، تحت تأثیر آن‌هاست. با بهینه کردن این زیربرنامه‌ها، علاوه بر کاهش توان مصرفی کل سیستم، تعداد کلاک مورد نیاز برای عملکرد بی‌درنگ‌شان نیز کاهش دارد. بهینه سازی یک سیستم غیرممکن (از لحاظ پیاده سازی و سرعت قابل قبول) را به یک سیستم ممکن و یک دستگاه غیر قابل رقابت را به نوع قابل رقابتش تبدیل می‌کند.
اگر شما کدهایی به زبان C نوشته و سپس آن‌ها را بهینه کنید (زیربرنامه‌هایی که زیاد فراخوانی می‌شوند را بهینه کنید) اولین گام را جهت بهینه سازی برداشته‌اید. برای دست‌یابی به بیشترین کارآیی، بهتر است که از کدهای اسمبلی دست‌نویسی استفاده کنید. اگر خودتان کدهای اسمبلی بنویسید، به سه ابزاری دست خواهید یافت که در زبان C به راحتی در دسترس نیستند. این سه ابزار عبارت‌اند از:

۱. زمان‌بندی دقیق دستورالعمل‌ها: در زبان اسمبلی، زمان اجرای دقیقی دو دستورالعمل را می‌توانید از جداول مربوط استخراج کنید. در ARM از خط لوله استفاده می‌شود، پس، زمان اجرای هر دستورالعمل می‌تواند تحت تأثیر دستورهای جانبی‌اش نیز باشد؛
۲. اختصاص ثبات‌ها: تصمیم گیری در مورد این‌که ثبات‌ها چگونه به عنوان متغیرهای محلی اختصاص داده شوند و استفاده از مکان‌های پشته چگونه است، به‌منظور دست‌یابی به کارآیی حداکثر، در دسترس برنامه نویس است. هدفمان کاهش تعداد دست‌یابی‌ها به حافظه است؛
۳. اجرای شرطی: دسترسی کامل به طیف وسیعی از کدهای شرط و دستورالعمل‌های شرطی، فقط در اسمبلی امکان‌پذیر است. بهینه سازی دستورهای زبان اسمبلی، زحمت زیادی را بر برنامه نویس تحمیل می‌کند. از بهینه سازی کدهایی که پر مصرف نیستند (کمتر فراخوانی می‌شود)، خودداری کنید. هنگامی که مشغول به بهینه سازی کدها یک زیربرنامه هستید، مزیت‌هایی چون فهم بهتر الگوریتم مورد نظر، نقاط مبهم و نحوه‌ی گردش داده‌ها را به همراه خواهید داشت.

بخش ۶.۱، به معرفی برنامه نویسی اسمبلی ARM می‌پردازد. در این بخش، خواهیم دید که یک نمونه کد نوشته شده به زبان C را چگونه به زبان اسمبلی تبدیل کرده و سپس به بهینه سازی آن می‌پردازیم. سپس، روش‌های رایج بهینه سازی در رابطه با نوشتن اسمبلی برای معماری ARM را تشریح خواهیم کرد. در این‌جا، از کدهای نوشته شده در حالت Thumb استفاده نمی‌کنیم. زیرا این حالت اساساً برای کاهش حجم کدهای نوشته شده به زبان C کاربرد دارد و هنگامی که یک گذرگاه ۳۲ بیتی در دسترس است، برای دست‌یابی به بهترین کارآیی، منطقی‌ترین راه، استفاده از حالت ARM است. هرچند، روش-های مطرح شده در مباحث بهینه سازی، برای هر دو حالت کاری ARM و Thumb قابل پیاده سازی است.

بهترین سطح بهینه سازی در زیربرنامه‌ها، به نوع معماری مورد استفاده نیز بستگی دارد. این مسأله خصوصاً در برنامه‌های پردازش سیگنال از اهمیت ویژه‌ای برخوردار است. اگرچه، امکان نوشتن کدهایی که برای تمامی هسته‌های ARM بهینه هستند، دور از انتظار نیست. برای این‌که از بحث دور نشویم، در مثال‌های این بخش از ARM9TDMI ، برای محاسبه‌ی زمان و تعداد سیکل‌های کاری زیربرنامه‌ها استفاده می‌کنیم (هرچند تمامی مثال‌ها بر روی هسته‌های ARM7TDMI به بعد، بهینه هستند).

۱۶.۱.۱ نوشتن کدهای اسمبلی

در این بخش، با ذکر یک مثال چگونگی نوشتن کدهای اسمبلی ابتدایی را توضیح می‌دهیم. فرض بر این-است که با دستورالعمل‌های اسمبلی ARM که در ضمیمه‌ی A آمده است، آشناهستید (اگر این‌گونه نیست، بهتر است نگاهی هرچند گذرا به آن‌ها داشته باشید). همچنین، فرض بر این‌است که با روش استاندارد فراخوانی زیربرنامه‌ها در حالات ARM و Thumb ، آشنا هستید (این موضوع در بخش ۵.۴ توضیح داده شده است).

در ادامه‌ی مباحث و مثال‌ها از اسمبلر ARM با نام armasm استفاده کرده‌ایم. همچنین می‌توانید از اسمبلر GNU با نام gas نیز استفاده کنید.

مثال ۶.۱

در این مثال، نحوه‌ی تبدیل یک تابع C به نمونه‌ی اسمبلی‌اش را می‌بینید. (اولین مرحله از بهینه سازی کد اسمبلی). نمونه برنامه‌ی ساده به زبان C زیر را درنظر بگیرید که مربع اعداد بین ۰ تا ۹ را در خروجی چاپ می‌کند (این فایل را main.c نامیده‌ایم) :

```
#include <stdio.h>
int square(int i);
int main(void)
{
```

```
	int i;
	for (i=0; i<10; i++)
	{
		printf("Square of %d is %d\n", i, square(i));
	}
}
int square(int i)
{
	return i*i;
}
```

بیایید ببینیم چگونه می‌توان جایگزینی برای تابع Square در زبان اسمبلی پیدا کرد. تعریف C تابع Square را برداشته اما به اعلان آن (در خط دوم) دست نزنید. سپس، یک فایل با نام Square.S درست کرده و محتویات زیر را به درون آن کپی کنید (این فایل، یک فایل اسمبلی است).

```
	AREA |.text|, CODE, READONLY
	EXPORT square
	 ;int square(int i)
square
	MUL r1, r0, r0 ; r1 = r0 * r0
	MOV r0, r1 ; r0 = r1
	MOV pc, lr ; return r0
	END
```

راهنمای[1] AREA ، ناحیه‌ای از کد که برنامه‌ها در آن مستقر هستند را نام‌گذاری می‌کند. اگر از کاراکترهایی غیر از حروف و یا اعداد برای نام‌گذاری این نواحی استفاده می‌کنید، اسامی نواحی را در بین خطوط عمودی قرار دهید. در غیر این‌صورت، بسیاری از این کاراکترها برای اسمبلر معانی خاصی ایجاد می‌کنند. در نمونه کد بالا، یک ناحیه‌ی کدِ "فقط خواندنی[2]" به نام ".text" را تعریف کرده‌ایم.
راهنمای Export نماد Square را برای دسترسی خارج از این فایل (برای لینک با سایر برنامه‌ها) فراهم می‌کند. در خط ششم، Square را به عنوان برچسب کد تعریف کردیم.

توجه

در armasm متن‌های تو نرفته[3] به عنوان تعریف برچسب درنظر گرفته می‌شوند.

[1] Directive
[2] Read-only
[3] Non-Indent

در فراخوانی Square، نحوه انتقال پارامترها تابع قوانین ATPCS (فراخوانی استاندارد زیربرنامه‌ها در حالات ARM و Thumb) است. آرگومان ورودی از طریق r0 وارد شده و مقدار خروجی نیز از همان طریق برگشت داده می‌شود. دستورالعمل ضرب، دارای این محدودیت است که پارامتر مقصدش نباید با اولین پارامتر ورودی‌اش یکسان باشد. به همین علت، نتیجه‌ی خروجی را ابتدا در r1 قرار داده، سپس محتوای r1 را به r0 منتقل می‌کنیم.
راهنمای END نشان دهنده‌ی انتهای فایل اسمبلی است و توضیحات تک‌خطی نیز با یک سمی‌کولن ";" شروع می‌شوند. برای ساختن فایل‌های باینری مثال بالا از اسکریپت زیر استفاده می‌کنیم:

```
armcc -c main1.c
armasm square.s
armlink -o main1.axf main1.o square.o
```

مثال، فقط هنگامی که کد C را به عنوان کد ARM ، کامپایل می‌کنید، درست کار می‌کند. اگر از حالت Thumb استفاده شود، در انتهای فایل اسمبلی، از دستور BX باید استفاده کرد.

مثال ۶.۲

هنگام فراخوانی کدهای ARM از کدهای C کامپایل شده در حالت Thumb، تنها تغییر نسبت به مثال ۶.۱، تغییر دستور بازگشت به BX است. BX بسته به بیت 0 ثبات Lr به حالت ARM و یا Thumb بازمی‌گردد. بنابراین، این زیربرنامه چه از حالت ARM و چه از حالت Thumb می‌تواند فراخوانده شود. به جای *lr* و MOV PC ، از دستور BX *lr* استفاده کنید. فایل جدیدی به نام Square2.s با محتویات زیر درست کنید.

```
        AREA |.text|, CODE, READONLY
        EXPORT square
         ;int square(int i)
square
        MUL r1, r0, r0 ; r1 = r0 * r0
        MOV r0, r1 ; r0 = r1
        BX lr ; return r0
        END
```

در این مثال، فایل C را با استفاده از کامپایلر Thumb به نام tcc کامپایل می‌کنیم.

```
tcc -c main1.c
armasm -apcs /interwork square2.s
armlink -o main2.axf main1.o square2.o
```

مثال ۶.۳

در این مثال، خواهیم دید که چگونه از برنامه‌ی اسمبلی یک زیربرنامه را فراخوانی کنیم. مثال ۶.۱ را درنظر گرفته و تمام برنامه (حتی کدهای C) را به دستورهای اسمبلی تبدیل می‌کنیم. در این‌جا تابع کتابخانه‌ای C به نام Printf را به عنوان یک زیربرنامه فراخوانی می‌کنیم. فایل اسمبلی main3.s را با محتویات زیر ایجاد کنید:

```
        AREA |.text|, CODE, READONLY

        EXPORT main

        IMPORT |Lib$$Request$$armlib|, WEAK
        IMPORT __main ; C library entry
        IMPORT printf ; prints to stdout
i       RN4
         ;int main(void(
main
        STMFD sp!, {i, lr{
        MOV i, #0
loop
        ADR r0, print_string
        MOV r1, i
        MUL r2, i, i
        BL printf
        ADD i, i, #1
        CMP i, #10
        BLT loop
        LDMFD sp!, {i, pc{
print_string
        DCB "Square of %d is %d\n", 0

        END
```

راهنمای IMPORT ، برای تعریف نمادهای استفاده شده در فایل‌های دیگر به کار می‌رود. نماد وارد شده‌ی Lib$$Request$$armLib درخواستی مبنی بر این‌که لینکر با توابع کتابخانه‌ای C باید لینک شود را صادر می‌کند. نماد وارد شده‌ی دوم (main-) آغاز کد اولیه‌ی کتابخانه‌ی C است.

راهنمای RN ، اجازه‌ی انتخاب نام برای ثبات‌ها را می‌دهد. در این‌جا، از i به عنوان نام جایگزینی برای r4 استفاده کرده‌ایم. استفاده از این نام‌گذاری مجدد، کدها را واضح‌تر می‌کند.

به یاد بیاورید که طبق ATPCS، تابع باید مقادیر r4 تا r11 و sp را حفظ کند. چون در این‌جا مقادیر i(r4) و Lr (توسط فراخوانی Printf) را دستکاری می‌کنیم، در ابتدای برنامه این دو ثبات را توسط STMFD در پشته قرار می‌دهیم. دستور LDMFD ، این مقادیر را از پشته بازخوانده و آدرس بازگشت را در PC می‌نویسد.

راهنمای DCB ، داده‌هایی از نوع بایت را به صورت یک رشته از کاراکترها[1] و یا مجموعه‌ای از بایت‌های جدا که توسط کاما "," از هم جدا می‌شوند، تعریف می‌کند.
برای ساختن این مثال، از اسکریپت زیر استفاده می‌کنیم:

```
armasm main3.s
armlink -o main3.axf main3.o
```

در مثال ۶.۳، فرض بر این‌است که کد مورد نظر از یک کد ARM ، فراخوانی شده است.
در مثال بعد، فرض بر این‌است که پارامترهای ورودی به تابع، بیشتر از ۴ عدد هستند. طبق ATPCS، ۴ متغیر اول در درون r0 تا r3 قرار می‌گیرند و باقی مانده در درون پشته مستقر می‌شوند.

مثال ۶.۴

این مثال، تابعی به نام Sumof تعریف می‌کند که توانایی جمع کردن هر تعداد از اعداد صحیح را دارد و آرگومان‌های ورودی تعداد اعداد و خود اعداد به صورت پشت سر هم هستند. تابع Sumof در اسمبلی نوشته شده و هر تعداد آرگومان را به عنوان ورودی قبول می‌کند. قسمت نوشته شده در C را در فایل main4.c با محتویات زیر قرار دهید:

```
#include <stdio.h>

/* N is the number of values to sum in list  ... */
int sumof(int N, ...);

int main(void)
{
       printf("Empty sum=%d\n", sumof(0));
       printf("1=%d\n", sumof(1,1));
       printf("1+2=%d\n", sumof(2,1,2));
       printf("1+2+3=%d\n", sumof(3,1,2,3));
       printf("1+2+3+4=%d\n", sumof(4,1,2,3,4));
       printf("1+2+3+4+5=%d\n", sumof(5,1,2,3,4,5));
       printf("1+2+3+4+5+6=%d\n", sumof(6,1,2,3,4,5,6));
}
```

سپس، تابع Sumof را در فایل sumof.s ، این‌گونه تعریف کنید:

```
       AREA    |.text|, CODE, READONLY
       EXPORT sumof
N      RN 0   ; number of elements to sum
sum    RN 1   ; current sum
        ;int sumof(int N(... ,
sumof
       SUBS N, N, #1 ; do we have one element
```

---

[1] String

```
        MOVLT sum, #0 ; no elements to sum!
        SUBS N, N, #1 ; do we have two elements
        ADDGE sum, sum, r2
        SUBS N, N, #1 ; do we have three elements
        ADDGE sum, sum, r3
        MOV r2, sp ; top of stack
loop
        SUBS N, N, #1 ; do we have another element
        LDMGEFD r2!, {r3} ; load from the stack
        ADDGE sum, sum, r3
        BGE loop
        MOV r0, sum
        MOV pc, lr ; return r0
        END
```

مثال فوق را با اسکریپت زیر می‌سازیم.

```
armcc -c main4.c
armasm sumof.s
armlink -o main4.axf main4.o sumof.o
```

۱۶.۱.۲ بررسی مشخصات رفتاری و شمارش سیکل‌های کاری

اولین گام در فرآیند بهینه سازی، پیدا کردن زیربرنامه‌های بحرانی و اندازه‌گیری کارآیی فعلی آن‌هاست. Profiler ، ابزاری است که زمان صرف شده و یا تعداد سیکل‌های مصرفی در هر زیربرنامه را اندازه می‌گیرد. شما با استفاده از این ابزار، زیربرنامه‌های بحرانی را پیدا می‌کنید. Cycle Counter نیز ابزاری است که تعداد سیکل‌های سپری شده در هر زیربرنامه را می‌شمارد. با شمارش تعداد سیکل‌های سپری شده در هر قسمت از کد، قبل و بعد از بهینه سازی، می‌توانید میزان بهره‌وری در فرآیند بهینه سازی‌تان را کشف کنید. نمونه‌ای از این ابزارها در نرم‌افزار ADS 1.1 با نام ARMulator فراهم آمده-اند.

۱۶.۱.۳ زمان‌بندی دستورالعمل‌ها

زمان اجرای دستورالعمل‌ها، به نوع خط لوله مورد استفاده بستگی دارد. قوانین زیرمجموعه‌ای از اطلاعات در مورد دستورهای رایج ARM9TDMI در اختیار می‌گذارد.

در دستورهای شرطی که از بیت‌های شرط cpsr استفاده می‌کنند، در صورت برآورده نشدن شرط مورد نظر، یک سیکل کاری بیشتر زمان سپری می‌کنند. در صورت برقراری شرط، قوانین زیر حاکم است:

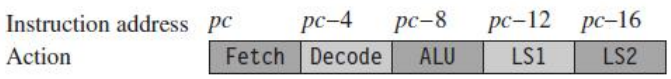
- عملیات ALU همانند جمع، تفریق و عملیات منطقی، یک سیکل را می‌گیرند. شیفت با یک مقدار ثابت نیز از همین قاعده پیروی می‌کند. در صورت استفاده از شیفت با مقدار یک ثبات، یک سیکل کاری اضافه می‌شود. اگر دستور مورد نظر به درون PC می‌نویسد، دو سیکل اضافه می‌شود؛
- دستورهای بارگیری که N کلمه‌ی ۳۲ بیتی را بارگیری می‌کنند، همانند LDR و LDM، N سیکل ماشین برای کار نیاز دارند.
- دستورهای بارگیری که مقادیر ۱۶ و ۳۲ بیتی را استفاده می‌کنند، همانند LDRB، LDRSB، LDRH و LDRSH، یک سیکل ماشین طول می‌کشند. نتایج بارگیری در دو سیکل بعد آماده‌اند؛
- دستورهای انشعاب، سه سیکل طول می‌کشند؛
- دستورهای ذخیره سازی N مقدار، N سیکل طول می‌کشند. STM تک مقداری مستثنا بوده و دو سیکل طول می‌کشد؛
- دستور ضرب، بسته به مقدار عملوند دوم ضرب، طول متغیری دارد.

به‌منظور فهم بهتر بهینه سازی، بایستی درک درستی از خط لوله‌ی ARM و وابستگی‌ها داشته باشیم. پردازنده‌ی ARM9TDMI پنج عمل موازی را انجام می‌دهد:

- واکشی: دستورالعمل را از آدرس PC در حافظه، واکشی می‌کند. دستور مورد نظر به درون هسته بارگیری شده و به درون خط لوله‌ی هسته هدایت می‌شود؛
- رمزگشایی: دستوری که در سیکل قبلی واکشی شده را رمزگشایی می‌کند؛
- ALU: دستور رمزگشایی شده در سیکل قبل را اجرا می‌کند. توجه کنید که دستورالعمل در واقع از آدرس PC - 8 (در حالت ARM) یا PC - 4 (در حالت Thumb) واکشی شده بود.

اجرای دستور، معمولاً شامل محاسبه‌ی جواب یک دستور پردازش داده و یا آدرس بارگیری و یا ذخیره سازی می‌شود. بعضی دستورها در این مرحله زمان بیشتری را صرف می‌کنند. به عنوان مثال، دستور ضرب و یا شیفت به تعداد موجود در یک ثبات، سیکل‌های بیشتری صرف می‌کنند؛

- LS1: اگر دستور یک دستور بارگیری و یا ذخیره سازی باشد، این عملیات در این سیکل اجرا می‌شوند؛
- LS2: در مورد دستورهای بارگیری و ذخیره سازی بایت، و یا دو بایت[۱] داده بارگیری شده را علامت‌دهی کرده یا توسعه می‌دهد.



شکل ۱۶.۱

شکل ۱۶.۱، شمایی ساده شده و کاربردی از خط لوله‌ی ARM9 را نشان می‌دهد.

[۱] Half Word

اگر در خط لوله، یک دستور به جواب حاصل از دستور قبلی نیاز داشته باشد (که در حال حاضر آماده نشده باشد)، حالتی اتفاق می‌افتد که در آن پردازنده از کار می‌ایستد. این حالت درمورد دستورهایی که طول اجرای بیشتری دارند، بسیار محتمل است. این حالت را با نام‌های hazard و یا Inter lock در خط لوله می‌شناسیم.

مثال ۶.۵

این مثال حالتی را نشان می‌دهد که در آن inter lock نداریم.

```
ADD r0, r0, r1
ADD r0, r0, r2
```

این دستورها در دو سیکل ماشین اجرا می‌شوند. ALU حاصل r0+r1 را در یک سیکل محاسبه می‌کند. بنابراین، نتیجه‌ی r0 در سیکل بعدی به عنوان پارامتر ورودی دستور دوم، حاضر است.

مثال ۶.۶

این مثال یک Inter lock تک سیکله ناشی از استفاده‌ی دستور بارگیری را می‌بینیم.

```
LDR r1, [r2, #4]
ADD r0, r0, r1
```

دستور دوم، در مرحله‌ی رمزگشایی دو سیکل ماشین می‌ایستد. این مسأله ناشی از این‌است که دستور بارگیری r1 را در دو سیکل بعد حاضر می‌کند. این Stall ایجاد شده در دستور ADD منجر به این می‌شود که اجرای کل دستورها، دو سیکل ماشین به درازا بیانجامند. شکل ۱۶.۲ نموداری از روند اجرای دستورها را نشان می‌دهند.

Pipeline	Fetch	Decode	ALU	LS1	LS2
Cycle 1	...	ADD	LDR	...	
Cycle 2		...	*ADD*	LDR	...
Cycle 3		...	ADD	—	LDR

شکل ۱۶.۲

۱۶.۱.۴ ساختارهای حلقه

بیشترین برنامه‌های با کارآیی بحرانی، احتمالاً شامل یک حلقه هستند. حلقه‌های ARM هنگامی که به سمت پایین حرکت کرده و به سمت صفر پیش می‌روند، سریع‌تر عمل می‌کنند. در این بخش می‌بینیم که

چگونه حلقه‌ها را در اسمبلی بهینه کنیم. همچنین به مثال‌هایی جهت افـزایش کـارآیی حلقـه‌هـا بـه حالـت پیشینه‌شان خواهیم پرداخت.

۱۶.۱.۴.۱ حلقه‌ی شمرده شونده و کاهش یابنده

برای یک حلقه‌ی کـاهش یابنـده بـا تکـرار N مرتبـه، شـمارنده‌ی حلقـه، i، از N تـا یـک را رو بـه پـایین می‌شمرد. هنگام رسیدن i به صفر، حلقه می‌ایستد. یک پیاده سازی بهینه این‌گونه است:

```
       MOV i, N
loop
        ;loop body goes here and i=N,N-1,...,1
       SUBS i, i, #1
       BGT loop
```

۱۶.۲ برنامه نویسی بهینه به زبان C

این بخش، به شما کمک می‌کند که برنامه‌هایی به زبان C بنویسید که به نحوی مؤثر و بهینه برای معماری ARM کامپایل شوند. به مثال‌های کوچکی نظر خواهیم داشت که نحوه تبدیل برنامه‌های C به اسمبلی توسط کامپایلر را نشان خواهند داد. هنگامی که نسبت به فرآیند "کامپایل" تسلط و احاطه پیدا کردید، شمایی از توانایی تشخیص برنامه C سریع از نوع کُند آن را خواهید داشت. این تکنیک‌ها عیناً برای C++ نیز صادق هستند ولی ما در این‌جا صرفاً بر روی C تأکید می‌ورزیم.
با مروری بر کامپایلرهای C و فرآیند بهینه سازی، نشان خواهیم داد که کامپایلرها با چه مسائلی هنگام بهینه سازی برنامه‌ها روبه‌رو هستند. با اشراف کامل بر این مسائل، می‌توانید برنامه‌هایی بنویسید که از لحاظ حجم برنامه‌ی کم و سرعت اجرای بالا، بهینه باشند.

۱۶.۲.۱ مروری بر کامپایلرهای C و بهینه سازی

در این بخش، فرض بر این‌است که با زبان C آشنا بوده و دانش نسبی از زبان برنامه نویسی اسمبلی نیز دارید. آشنایی با اسمبلی ضروری نیست فقط به شما در فهم خروجی‌هایی که کامپایلر تولید می‌کند، یاری می‌رساند.
بهینه سازی برنامه زمان‌بر بوده و خوانایی برنامه را کم می‌کند. اغلب تنها "زیربرنامه‌ها" و یا "توابعی" که به دفعات فراخوانی می‌شوند و در کارآیی تأثیرگذار هستند را بهینه می‌کنیم. به شما توصیه می‌کنیم از ابزارهای نرم‌افزاری که هسته ARM را شبیه سازی می‌کنند، استفاده کنید و از این طریق توابعی که به دفعات فراخوانی می‌شوند را بیابید. بهینه سازی‌های غیرمشهود را مستند سازی کنید و با افزودن توضیحات[۱]، روند بازخوانی برنامه در آینده را هموارتر سازید.
کامپایلرهای C، برنامه نوشته شده را به صورت لغت به لغت به زبان اسمبلی تبدیل می‌کنند. اجازه دهید با مثال زیر به یکی از مشکلاتی که کامپایلر با آن برخورد دارد، نگاهی بیاندازیم. تابع **memclr** تعداد N بایت داده را از آدرس شروع **data** پاک می‌کند.

```
void memclr(char *data, int N)
{
	for (; N>0; N--)
	{
		*data=0;
		data;++
	}
}
```

[۱] Comment

مهم نیست که کامپایلر چه‌قدر حرفه‌ای باشد، اما او نمی‌داند که N می‌تواند مقدار صفر و یا غیرصفر داشته باشد. بنابراین، کامپایلر بایستی قبل از وارد شدن به حلقه، رخداد این مسأله را صریحاً آزمایش کند.

کامپایلر، نمی‌داند که اشاره‌گر آرایه‌ای data از آدرسی ۴ـ بایتی شروع می‌شود یا نه. اگر این‌گونه باشد، کامپایلر برای پاک کردن آن از متغیرهای نوع **int** به جای نوع **char** استفاده می‌کند. همچنین، نمی‌داند که N ضریبی از ۴ است یا نه. اگر N ضریبی از ۴ نباشد، کامپایلر حلقه را ۴ مرتبه تکرار می‌کند و در صورتی که ضریبی از ۴ باشد، هر ۴ بایت به یک‌باره توسط متغیری از نوع **int** پاک می‌شوند.

کامپایلر، باید به تمام جوانب دقت لازم را داشته باشد. در این مثال، بایستی تمام مقادیر ممکن برای N و تمام ردیف‌های ممکن برای **data** را بسنجد.

برای نوشتن برنامه‌ای بهینه به زبان C ، بایستی حواستان به نکاتی که کامپایلر در آن‌ها محافظه کارانه عمل می‌کند، محدودیت‌های معماری سخت‌افزاری که کامپایلر C برای آن کامپایل می‌کند و همچنین محدودیت‌های هر کامپایلر بخصوص باشد. در این بخش، بیشتر بر روی دو مورد اول تمرکز داریم و مطالب گفته شده برای تمام کامپایلرهای C ، صادق هستند.

برای این‌که مثال‌های این بخش به‌هم پیوسته باشند، از دو کامپایلر نمونه‌ی زیر برای این منظور استفاده کرده‌ایم:

- کامپایلر armcc از برنامه ARM Developer Suite V1.1 (ADS 1.1) – می‌توانید مجوز این برنامه را از خود شرکت ARM خریداری کنید؛
- Arm-elf-gcc نسخه‌ی 2.95.2 – این کامپایلر رایگان بوده و جزئی از نرم‌افزارهای کامپایلر C (یعنی GNU (gcc است.

برای استفاده از armcc جهت تولید برنامه اسمبلی، از یک فایل نمونه test.c از اسکریپت‌های زیر استفاده می‌کنیم:

```
armcc -Otime -c -o test.o test.c
fromelf -text/c test.o > test.txt
```

arm تمام بهینه سازی‌هایش را به طور پیش‌فرض روشن کرده است (یعنی همان استفاده از سوییچ خط فرمان O2–). استفاده از سوییچ otime- اجرا برای سرعت بالا را بهینه می‌کند و عمدتاً بر روی بهینه سازی حلقه‌های for و While تمرکز دارد. اگر از دو کامپایلر gcc استفاده می‌کنید، چند خط زیر نیز خروجی اسمبلی مربوط به را تولید می‌کنند:

```
arm-elf-gcc -O2 -fomit-frame-pointer -c -o test.o test.c
arm-elf-objdump -d test.o > test.txt
```

در کامپایلرهای GNU ، بهینه سازی کامل به طور پیش‌فرض خاموش هستند. سوییچ -fomit-frame-pointer کامپایلر را مجبور به حذف ثبات اشاره‌گر فریم (Frame Pointer Register) می‌کند.

اشاره‌گرهای فریم در دیباگ کردن برنامه کمک می‌کنند و نگه‌داری آن‌ها در برنامه‌هایی که کارآیی بالایی دارند، غیرحرفه‌ای و غیرضروری است.

۱۶.۲.۲ انواع داده‌ها در زبان C

اجازه دهید نگاهی به چگونگی برخورد کامپایلرهای C با انواع داده‌ها بیاندازیم. خواهیم دید که بعضی از این داده‌ها نسبت به انواع دیگر در استفاده‌های محلی (Local Variables) سریع‌تر عمل می‌کنند. همچنین، تفاوت‌هایی در انواع آدرس‌دهی هنگام بارگیری و ذخیره داده‌ها وجود دارد.
پردازنده‌های ARM ثبات‌های ۳۲ بیتی دارند. عملیاتی که روی داده‌ها انجام می‌دهند نیز ۳۲ بیتی هستند. همان‌طور که می‌دانیم معماری ARM از انواع بارگیری/ ذخیره سازی RISC می‌باشد و هیچ دستوری مستقیماً بر روی حافظه عملیاتی صورت نمی‌دهد. نسخه‌های اولیه ARM (از ARMv1 تا ARMv3) سخت‌افزار لازم برای بارگیری و ذخیره سازی مقادیر بدون علامت ۸ بیتی (Unsigned 8 bit) و مقادیر بدون علامت یا باعلامت ۳۲ بیتی را داشتند. این ساختارها تا قبل از ARM7TDMI استفاده می‌شدند.

Architecture	Instruction Action	Instruction Action
Pre-ARMv4	LDRB	load an unsigned 8-bit value
	STRB	store a signed or unsigned 8-bit value
	LDR	load a signed or unsigned 32-bit value
	STR	store a signed or unsigned 32-bit value
ARMv4	LDRSB	load a signed 8-bit value
	LDRH	load an unsigned 16-bit value
	LDRSH	load a signed 16-bit value
	STRH	store a signed or unsigned 16-bit value
ARMv5	LDRD	load a signed or unsigned 64-bit value
	STRD	store a signed or unsigned 64-bit value

جدول ۵.۱ دستورهای بارگیری/ ذخیره سازی بر روی نسخه‌های مختلف ARM را نشان می‌دهد. در جدول ارایه شده، دستورهای بارگیری که بر روی مقادیر ۸ یا ۱۶ بیتی عمل می‌کنند، مقادیر را قبل از

نوشتن بر روی ثبات‌های داخلی به مقادیر ۳۲ بیتی تبدیل می‌کند. مشابهاً برای ذخیره سازی مقادیر ۸ یا ۱۶ بیتی، فقط ۸ یا ۱۶ بیت پایین ثبات در حافظه ذخیره می‌شوند.
معماری ARMv4 و بالاتر از آن از طریق دستورالعمل‌های جدید، بارگیری و ذخیره سازی ۸ و ۱۶ بیتی را پشتیبانی می‌کنند. به علت این‌که این دستورها در نسخه‌های جدیدتر معماری ARM ارایه شده‌اند، از تمامی حالت‌های آدرس‌دهی پشتیبانی نمی‌کنند.
معماری ARMv5 ، دستورهایی برای پشتیبانی از بارگیری و ذخیره سازی ۶۴ بیتی اضافه کرد. در هسته‌های ARM9E و بعد از ان حضور این دستورها را می‌بینیم.
قبل از ARMv4، پردازنده‌های ARM در پردازش اعداد ۸ بیتی علامت‌دار و هر مقدار ۱۶ بیتی مشکل داشتند. کامپایلرهای C ، مقادیر تعریف شده از جنس char را به عنوان مقادیر بدون علامت ۸ بیتی درنظر می‌گرفتند. در حالی‌که روش مرسوم، استفاده از مقادیر ۸ بیتی علامت‌دار بود.
کامپایلرهای armcc و gcc طبق جدول ۵.۲ انواع داده‌های مختلف را به یکدیگر نگاشت می‌کنند.
به عنوان مثالی برای یک حالت خاص char می‌توان از موردی یاد کرد که **i** شمارنده حلقه بوده و شرط ادامه حلقه **i≥0** است. از آن‌جایی که مقدار **i** برای کامپایلرهای ARM بدون علامت است، حلقه هرگز پایان نخواهد یافت. خوشبختانه، armcc در این مواقع یک پیغام اخطار با این مضمون می‌دهد:"مقایسه عدد بدون علامت با **0** "[1]. کامپایلرها از سوییچی برای علامت‌دار کردن **char** پشتیبانی می‌کنند.
به عنوان مثال، سوییچ خط فرمان fsigned- char- در کامپایلر gcc مقادیر **char** را علامت‌دار می‌کند. همین کار در کامپایلر armcc توسط zc- انجام می‌شود. در ادامه فرض بر این‌است که فقط از پردازنده‌های ARMv4 به بعد استفاده می‌کنیم. منظور ARM7TDMI و تمامی پردازنده‌های بعد از آن است.

C Data Type	Implementation
char unsigned 8	bit byte
short signed 16	bit halfword
int signed 32	bit word
long signed 32	bit word
long long signed 64	bit double word

[1] Unsigned Comparison with 0

۱۶.۲.۲.۱ انواع متغیرهای محلی

پردازنده‌های ARMv4 ، توانایی بارگیری و ذخیره سازی داده‌های ۸، ۱۶ و ۳۲ بیتی را دارند. اگرچه، اکثر دستورهای پردازش داده ARM فقط ۳۲ بیتی هستند. به همین علت بایستی در صورت امکان از انواع داده ۳۲ بیتی (یعنی **int** یا **long**) برای متغیرهای محلی استفاده کرد. از انواع داده **char** و یا **short** حتی در هنگام دستکاری مقادیر ۸ و یا ۱۶ بیتی ، استفاده نکنید. یک حالت استثنا در این مورد زمانی است که می‌خواهید مقادیر هنگامی که به حداکثر خود می‌رسند، سرریز شوند. مثلاً، متغیری از نوع **char** بعد از شمارش و رسیدن به ۲۵۵ به ۰ می‌رسد.

برای دیدن اثر متغیرهای محلی و شیوه انتخاب آن‌ها، نگاهی به یک مثال ساده خواهیم انداخت. نگاهی عمیق به تابع Checksum خواهیم داشت که مقادیر را در یک بسته داده جمع می‌کند. بیشتر پروتکل-های ارتباطی (از قبیل TCP/IP) روالی برای محاسبه‌ی Checksum ویا CRC در عیب‌یابی بسته‌های داده، دارند.

در ادامه، تابعی برای محاسبه Checksum در یک بسته داده با ۶۴ کلمه را می‌بینید. این مثال نشان می‌دهد که چرا از متغیرهای محلی **char** نباید استفاده کرد.

```
int checksum_v1(int *data)
{
	char i;
	int sum = 0;
	for (i = 0; i < 64; i++)
	{
		sum += data[i];
	}
	return sum;
}
```

در نگاه اول، به نظر می‌رسد که استفاده از متغیر **char** برای **i** راه‌حلی بهینه است. ممکن است تصور کنید که متغیر **char** نسبت به نوع **int** فضای حافظه‌ی کمتری را اشغال می‌کند. اما هردوی این تفکرات در مورد ARM اشتباه است. در ARM ثبات‌ها، و پشته و تمام ورودی‌های آن، ۳۲ بیتی هستند. برای پیاده سازی صحیح **++i** کامپایلر بایستی دقت خاص روی حالت **i=255** داشته باشد. افزایش 255 باید نتیجه‌ای برابر صفر داشته باشد. خروجی نمونه‌ی کامپایلر را در زیر درنظر بگیرید. برچسب‌ها و توضیحات فهم دستورها را واضح‌تر کرده‌اند.

```
checksum_v1
	MOV r2,r0 ; r2 = data
	MOV r0,#0 ; sum = 0
	MOV r1,#0 ; i = 0
checksum_v1_loop
	LDR r3,[r2,r1,LSL #2] ; r3 = data[i]
	ADD r1,r1,#1 ; r1 = i+1
	AND r1,r1,#0xff ; i = (char)r1
```

```
        CMP r1,#0x40 ; compare i, 64
        ADD r0,r3,r0 ; sum += r3
        BCC checksum_v1_loop ; if (i<64) loop
        MOV pc,r14 ; return sum
```

هم‌اکنون خروجی زیر را درنظر بگیرید که **i** را به عنوان **unsigned int** تعریف کرده‌ایم.

```
checksum_v2
        MOV r2,r0 ; r2 = data
        MOV r0,#0 ; sum = 0
        MOV r1,#0 ; i = 0
checksum_v2_loop
        LDR r3,[r2,r1,LSL #2] ; r3 = data[i]
        ADD r1,r1,#1 ; r1++
        CMP r1,#0x40 ; compare i, 64
        ADD r0,r3,r0 ; sum += r3
        BCC checksum_v2_loop ; if (i<64) goto loop
        MOV pc,r14 ; return sum
```

در مورد اول، کامپایلر یک دستور AND را برای محدود کردن i به مقادیر ۰ تا ۲۵۵، قبل از مقایسه با ۶۴، اضافه می کند. این دستور در حالت دوم وجود ندارد.

۱۶.۲.۲.۲ انواع آرگومان توابع

در بخش ۵.۲.۱ دیدیم که تبدیل متغیرهای محلی از نوع **short** و یا **char** به نوع **int**، کارآیی را افزایش داده و حجم کد را نیز کاهش می‌دهد. قانونی مشابه برای آرگومان‌های توابع صدق می‌کند.
تابع ساده زیر را درنظر بگیرید که دو مقدار ۱۶ بیتی را با هم جمع می‌کند:

```
short add_v1(short a, short b)
{
        return a + (b>>1);
}
```

این تابع، کمی مصنوعی است اما، برای سنجیدن مشکلاتی که کامپایلر با آن‌ها برخورد می‌کند، مفید است. مقادیر ورودی a و b و مقدار خروجی به صورت ثبات‌های ۳۲ بیتی استفاده می‌شوند.
آیا کامپایلر در مورد این‌که مقادیر a ، b و... ، ۱۶ بیتی هستند، اطلاعی دارد؟ برای پی بردن به این موضوع، به کدهای اسمبلی نمونه زیر، نگاهی بیاندازید.

```
add_v1
        ADD r0,r0,r1,ASR #1 ; r0 = (int)a + ((int)b >> 1(
        MOV r0,r0,LSL #16
        MOV r0,r0,ASR #16 ; r0 = (short)r0
        MOV pc,r14 ; return r0
```

```
add_v1_gcc
	MOV r0, r0, LSL #16
	MOV r1, r1, LSL #16
	MOV r1, r1, ASR #17 ; r1 = (int)b >> 1
	ADD r1, r1, r0, ASR #16 ; r1 += (int)a
	MOV r1, r1, LSL #16
	MOV r0, r1, ASR #16 ; r0 = (short)r1
	MOV pc, lr ; return r0
```

دو نمونه کد بالا از دو نمونه کامپایلر C مختلف استفاده می‌کنند. می‌بینیم که تبدیل‌های اضافی صورت گرفته‌اند.

نکته

بهتر است که برای آرگومان‌های توابع و مقادیر برگشتی از متغیرهای **int** ۳۲ بیتی استفاده کنیم. حتی اگر متغیرهای ۸ بیتی نیاز داشته باشیم. در این صورت، کارآیی بیشتری خواهیم داشت.

۱۶.۲.۲.۳ مقادیر علامت‌دار در برابر مقادیر بدون علامت

در بخش قبل، مزیت استفاده از مقادیر **int** را در مقابل استفاده از مقادیر **short** و **char** دیدیم. در این بخش کارآیی دو نوع **signed int** و **unsigned int** را مقایسه می‌کنیم.
اگر در کدتان از جمع، تفریق و یا ضرب استفاده می‌کنید، بین این دو نوع تفاوتی وجود ندارد. اما اگر از تقسیم استفاده می‌کنید، تفاوتی وجود دارد که در مثال زیر می‌بینیم:

```
int average_v1(int a, int b)
{
	return (a+b)/2;
}
```

بعد از کامپایل:

```
average_v1
	ADD r0,r0,r1 ; r0 = a + b
	ADD r0,r0,r0,LSR #31 ; if (r0<0) r0++
	MOV r0,r0,ASR #1 ; r0 = r0 >> 1
	MOV pc,r14 ; return r0
```

توجه کنید که کامپایلر، قبل از شیفت به راست (اگر حاصل‌جمع منفی باشد)، نتیجه را یک عدد اضافه می‌کند. به عبارت دیگر، $x/2$ را با عبارت زیر جایگزین می‌کند:

```
(x<0) ? ((x+1) >> 1): (x >> 1)
```

این کار به علت علامت‌دار بودن 0x یک ضرورت است. در زبان C ورودی یک پردازنده‌ی ARM، تقسیم بر دو هنگامی که منفی است، همان شیفت به راست نیست. برای مثال 2- =1<<3- اما 1- =3/2-.
تقسیم به سمت صفر گرد می‌شود، اما شیفت به راست به سمت ∞- گرد می‌شود. استفاده از انواع بدون علامت برای تقسیم، بسیار بهینه‌تر است.

نکته

در تقسیم، عموماً روال تقسیم موجود در کتابخانه‌های C برای مقادیر بدون علامت، سریع‌تر عمل می‌کنند.

۱۶.۲.۲.۴ خلاصه استفاده‌ی بهینه‌ی انواع داده در C

- برای متغیرهای محلی که در ثبات‌ها نگه‌داری می‌شوند، از انواع **int** استفاده کنید. (مگر مجبور به استفاده از ۸ و یا ۱۶ بیت داده، باشید.) استفاده از مقادیر بدون علامت هنگام استفاده از تقسیم، سریع‌تر است؛
- ورودی‌های آرایه و متغیرهای عمومی که در حافظه نگه‌داری می‌شوند را از نوع داده با کم‌ترین اندازه‌ی مورد نیاز انتخاب کنید. از آن‌جایی که معماری ARMv4 به بعد از بارگیری و ذخیره سازی انواع داده با طول‌های ۸، ۱۶ و ۳۲ بیتی، پشتیبانی می‌کند، این کار، حجم حافظه‌ی مورد نیاز را کاهش می‌دهد؛
- برای تبدیل انواع متغیرها به یکدیگر، می‌توان از تبدیل کننده‌های ضمنی در دستورهای بارگیری و ذخیره سازی استفاده کرد.

۱۶.۲.۳ ساختارهای حلقه‌ی C

در این بخش، در مورد نحوه‌ی پیاده سازی بهینه‌ی حلقه‌های **for** و **while** ، بحث خواهیم کرد. ابتدا، به حلقه‌های با تعداد تکرار ثابت، سپس به حلقه‌های با تعداد تکرار متغیر، نگاهی می‌اندازیم.

۱۶.۲.۳.۱ حلقه‌های با تعداد تکرار ثابت

مثال زیر، آخرین نمونه از روال checksum با ورودی ۶۴ کلمه‌ای است. این مثال، نشان می‌دهد که کامپایلر چگونه با حلقه‌ها برخورد می‌کند.

```
int checksum_v5(int *data)
{
       unsigned int i;
       int sum=0;
       for (i=0; i<64; i++)
       {
              sum += *(data++);
       }
       return sum;
}
```

کد کامپایل می‌شود به

```
checksum_v5
       MOV r2,r0                ; r2 = data
       MOV r0,#0                ; sum = 0
       MOV r1,#0                ; i = 0
checksum_v5_loop
       LDR r3,[r2],#4           ; r3 = *(data(++
       ADD r1,r1,#1             ; i++
       CMP r1,#0x40             ; compare i, 64
       ADD r0,r3,r0             ; sum += r3
       BCC checksum_v5_loop     ; if (i<64) goto loop
       MOV pc,r14               ; return sum
```

۳ دستور برای پیاده سازی حلقه‌ی for استفاده می‌شود:

- ADD ، برای اضافه نمودن i؛
- یک مقایسه که کوچک‌تر بودن i از ۶۴ را آزمایش می‌کند؛
- یک انشعاب شرطی که حلقه را مادامی که i<64 است، ادامه می‌دهد.

هرچند، این روش بهینه نیست. یک حلقه بر روی ARM باید فقط دو دستور داشته باشد:

- یک تفریق برای کاهش شمارنده‌ی حلقه (که البته پرچم‌های کد شرطی را بر روی نتیجه، تنظیم می‌کند)؛
- یک دستور انشعاب شرطی.

نکته مهم در این‌جا اینست که شمارنده‌ی حلقه باید به جای شمارش روبه بالا و به سمت یک عدد خاص، به سمت پایین و تا صفر بشمارد. در این‌جا مقایسه کردن نتیجه با مقدار صفر، زمان اضافی از ما نمی‌گیرد. زیرا نتیجه در درون پرچم‌های شرط ذخیره می‌شود.

مثال ۵.۲

این مثال میزان بهبود را هنگامی که از حلقه‌ی افزایش یابنده به حلقه‌ی کاهش یابنده تغییر کد می‌دهیم، نشان می‌دهد.

```
int checksum_v6(int *data)
{
	unsigned int i;
	int sum=0;
	for (i=64; i!=0; i--)
	{
		sum += *(data++);
	}
	return sum;
}
```

کد به کد زیر، کامپایل می‌شود:

```
checksum_v6
	MOV r2,r0 ; r2 = data
	MOV r0,#0 ; sum = 0
	MOV r1,#0x40 ; i = 64
checksum_v6_loop
	LDR r3,[r2],#4 ; r3 = *(data++)
	SUBS r1,r1,#1 ; i-- and set flags
	ADD r0,r3,r0 ; sum += r3
	BNE checksum_v6_loop ; if (i!=0) goto loop
	MOV pc,r14 ; return sum
```

دو دستور SUBS و BNE حلقه را پیاده سازی می‌کنند. روال checksum، در این‌جا، حداقل ۴ دستور دارد که نسبت به گذشته بسیار بهبود داشته است.

در مورد شمارنده‌ی حلقه‌ی بدون علامت i، می‌توانیم از هر دو نوع شرط ادامه‌ی حلقه‌ی i!=0 و یا i>0 استفاده کنیم. به علت این‌که i نمی‌تواند منفی باشد، هر دو شرط بالا یک معنی را می‌دهند.

اگر شمارنده حلقه علامت‌دار باشد، استفاده از شرط i>0 وسوسه انگیز است. کامپایلر باید کد زیر را برای آن تولید کند:

```
SUBS r1,r1,#1 ; compare i with 1, i=i-1
BGT loop ; if (i+1>1) goto loop
```

در واقع، کامپایلر کد زیر را استخراج می‌کند:

```
SUB r1,r1,#1 ; i--
CMP r1,#0 ; compare i with 0
BGT loop ; if (i>0) goto loop
```

کامپایلر بهینه است اما باید در مورد حالتی که i = -0x8000 0000 با دقت بیشتری عمل کند. زیـرا در این حالت، دو کد بالا نتایج مختلفی را ایجاد می‌کنند.
در حالت، اول دستور SUBS، i را با 0x1 مقایسه کرده، سپس یک واحد از آن کم می‌کند. از آن‌جایی که 1 > -0x8000 0000 ، حلقه پایان می‌پذیرد. در مـورد قسـمت دوم، i را کـم کـرده و سـپس بـا صـفر مقایسه می‌کنیم. در این‌جا i دارای مقدار 0x7fff ffff+ است که از صفر بزرگ‌تر است. پـس حلقـه بـرای دفعات بیشتری ادامه خواهد یافت.

البته، در عمل مقدار i به ندرت برابر 0x80000000- می‌شود. کامپایلر عموماً قادر به شناسایی این حالت نیست. خصوصاً اگر حلقه با تعداد تکرار نامشخصی شروع به کار کند.
بنابراین باید از شرط پایان دادن به حلقه‌ی0=!i برای مقادیر علامت‌دار و یا بدون علامت استفاده کنید. زیرا در مورد حالت i علامت‌دار، نسبت به شرط 0<i، یک دستور صرفه‌جویی داریم.

۱۶.۲.۳.۲ حلقه‌هایی که از تعداد تکرار حلقه‌ی متغیر استفاده می‌کنند

فرض کنید که روال checksum را به صورت ورودی با تعداد متغیر عوض کرده‌ایم. در این‌جا N نمایانگر تعداد ورودی‌ها در بسته‌ی داده است. از درس‌هایی که دو بخش قبل آموختیم، تا زمانی که **N = 0** به سمت پایین اقدام به شمارش می‌کنیم و نیازی به متغیر شمارنده اضافی i نداریم.
مثال زیر این مسأله را نشان می‌دهد:

```
int checksum_v7(int *data, unsigned int N)
{
	int sum=0;
	for (; N!=0; N--)
	{
		sum += *(data++);
	}
	return sum;
}
```

که این گونه کامپایل می‌شود:

```
checksum_v7
	MOV r2,#0 ; sum = 0
	CMP r1,#0 ; compare N, 0
	BEQ checksum_v7_end ; if (N==0) goto end
checksum_v7_loop
	LDR r3,[r0],#4 ; r3 = *(data++)
	SUBS r1,r1,#1 ; N-- and set flags
	ADD r2,r3,r2 ; sum += r3
	BNE checksum_v7_loop ; if (N!=0) goto loop
checksum_v7_end
	MOV r0,r2 ; r0 = sum
	MOV pc,r14 ; return r0
```

آزمایش مقدار N در ابتدای کد برای داشتن مقداری غیر صفر تقریباً غیرضروری است.
در این‌جا استفاده از حلقه‌های do-while کارآیی بهتری دارند.

مثال ۵.۳

در این‌جا برای پرهیز از آزمایش صفر بودن مقدار N، که در حلقه‌ی for رخ می‌دهـد، از do-while استفاده کرده‌ایم.

```
int checksum_v8(int *data, unsigned int N)
{
	int sum=0;
	do
	{
		sum += *(data++);
	} while (--N!=0);
	return sum;
}
```

اکنون خروجی کامپایلر این گونه است:

```
checksum_v8
	MOV r2,#0 ; sum = 0
checksum_v8_loop
	LDR r3,[r0],#4 ; r3 = *(data++)
	SUBS r1,r1,#1 ; N-- and set flags
	ADD r2,r3,r2 ; sum += r3
	BNE checksum_v8_loop ; if (N!=0) goto loop
	MOV r0,r2 ; r0 = sum
	MOV pc,r14 ; return r0
```

این خروجی را با خروجی checksum_V7 مقایسه کنید تا میزان دو سیکل صـرفه‌جـویی را بـه چشـم ببینید.

فصل هفدهم

سیستم های عامل بی درنگ

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ سیستم عامل تعبیه شده چیست؟
- ✓ روش انتخاب یک سیستم عامل تعبیه شده؛
- ✓ آشنایی با WinCE و eCos.

۱۷.۱ سیستم‌های عامل تعبیه شده

در این بخش، در مورد نحوه‌ی پیاده سازی سیستم‌عامل[۱] بحث خواهیم کرد. از آن‌جایی که از گذشته تا کنون سیستم‌های عامل بیشتر برای کاربردهای خاص استفاده می‌شدند، این سیستم‌های عامل، ساده و دارای محدودیت زمانی روی اجرای دستورها بودند. همچنین، روی حجم کمی از حافظه اجرا می‌شدند. هرچند، این مشخصات و محدودیت‌ها با گذشت زمان و ارایه‌ی سخت‌افزارهای جدید، تغییر کرده‌اند. مشخصاتی که در قدیم بر روی کامپیوترهای شخصی موجود بودند، به دنیای سیستم‌های عامل تعبیه شده پای گذاشتند.

حوزه‌ی مورد بحث این موضوع، بسیار گسترده است. به همین دلیل، ما خود را به بررسی اجزای پایه و شکل‌دهنده‌ی یک سیستم‌عامل محدود می‌کنیم.

۱۷.۱.۱ اجزای بنیادی

مجموعه‌ای از اجزای اساسی با اهداف و وظایف مشخص، در شکل‌گیری ساختار یک سیستم‌عامل مؤثراند. نحوه‌ی کارکرد هر یک از این اجزا و چگونگی تعامل آن‌ها با یکدیگر است که بیانگر مشخصات کلی یک سیستم‌عامل خاص است.

---

۱ Operating System_OS

مقداردهی اولیه[۱] نخستین زیربرنامه‌ای است که سیستم‌عامل آن را اجرا می‌کند. تنظیم ساختار داده‌ها، تعیین مقدار متغیرهای عمومی[۲] و تنظیم پارامترهای سخت‌افزار از جمله وظایفی است که در این زیربرنامه انجام می‌شود.

رسیدگی به حافظه[۳] شامل برپایی و تنظیم پشته‌ی مربوط به سیستم و وظایف کاری (Task) می‌شود. قرار دادن پشته در مکان خاص، بیانگر میزان حافظه‌ی در دسترس برای سیستم و یا وظایف کاری است.

تصمیم در مورد مکان قرار گیری پشته معمولاً در فاز **مقداردهی اولیه** انجام می‌شود ولی تنظیم پشته‌ی وظایف کاری، بستگی به ایستا[۴] و یا پویا[۵] بودن وظیفه‌ی مورد نظر دارد. یک وظیفه‌ی کاری ایستا در زمان ساختن فایل باینری سیستم‌عامل و فایل Image مربوط به آن ساخته شده و در آن به طور ثابت جاسازی می‌شود. پشته‌ی مربوط به این نوع وظایف، به راحتی در فاز مقداردهی اولیه اختصاص داده می‌شوند. در مقابل وظایف پویا، بعد از نصب و اجرای سیستم‌عامل، اجرا می‌شوند. بدین علت، جزئی از آن به حساب نمی‌آیند. به عنوان مثال، در سیستم‌عامل Linux (که نمونه‌ای از این نوع سیستم‌عامل است)، پشته، بعد از ایجاد شدن وظیفه‌ی کاری اختصاص داده می‌شود.

رسیدگی به حافظه در سیستم‌های عامل مختلف از محافظ پیچیدگی دارای انواع گوناگونی هستند. پیچیدگی مذکور، به عوامل مختلفی از جمله هسته‌ی ARM انتخاب شده، توانایی‌های میکروکنترولر و جانمایی حافظه‌ی فیزیکی[۶] در سخت‌افزار نهایی، بستگی دارند.

روش "رسیدگی به وقفه‌ها و استثناها"[۷]، جزئی از طراحی معماری یک سیستم‌عامل است. شما می‌بایستی در مورد نحوه‌ی رسیدگی به استثناهای مختلفی چون Data Abort، Fast Interrupt Request، Reset و... تصمیم گیری کنید.

البته، تمام استثناها نیاز به زیربرنامه‌ی رسیدگی ندارند. به عنوان مثال، اگر شما سخت‌افزاری در اختیار دارید که از وقفه‌ی FIQ استفاده نمی‌کند، بدیهی است که به زیربرنامه‌ی رسیدگی به FIQ نیز نیازی نخواهید داشت. البته، بهتر و ایمن‌تر است که از یک حلقه‌ی پایان‌ناپذیر به عنوان زیربرنامه‌ی پیش‌فرض برای روال‌های رسیدگی استفاده نشده، استفاده کنید. این کار روند عیب‌یابی را آسان‌تر می‌سازد. بدین صورت که هنگام توقف برنامه، مطمئن خواهید بود که در یکی از روال‌های رسیدگی، گیر افتاده‌اید.

---

[۱] Initialization

[۲] Global Variable

[۳] Memory Handling

[۴] Static

[۵] Dynamic

[۶] Physical Memory Layout

[۷] Interrupts and Exceptions

استفاده از حلقه‌ی بی‌پایان، سیستم‌عامل را از افتادن در سردرگمی و بروز رفتارهای ناخواسته نیز نجات می‌دهد.

در بعضی از سیستم‌های‌عامل از یک زمان سنج متناوب[۱] جهت ایجاد وقفه‌های تناوبی در زمان‌های خاص استفاده می‌شود. این زمان‌ها به سادگی توسط یک زمان سنج/ شمارنده محاسبه می‌شوند. تنظیم مشخصات این زمان سنج در فاز مقداردهی اولیه صورت می‌گیرد. بعد از آن شمارنده شروع به شمارش روبه پایین می‌کند و هنگام رسیدن به مقدار صفر، وقفه‌ای تولید می‌کند. روال رسیدگی به وقفه ابتدا مقدار جدیدی (مطابق آن‌چه قبلاً تعیین شده بود) را در درون ثبات زمان سنج بارگیری کرده سپس زمان‌بند[۲] و یا روتین خاص دیگری را اجرا می‌کند (در مورد زمان‌بندی بعداً توضیح داده می‌شود).

این نوع سیستم‌عامل‌ها با نام "انحصاری"[۳] شناخته می‌شوند. در مقابل سیستم‌عامل‌های "غیرانحصاری" قرار دارند که از وقفه‌های متناوب استفاده نمی‌کنند. درعوض، از تکنیک سرکشی[۴] استفاده می‌کنند. (سرکشی مداوم به‌منظور مشاهده‌ی تغییر حالت و وضعیت در دستگاه مورد نظر) به هنگام تغییر وضعیت سیستم، تصمیمی مرتبط با وضعیت مورد نظر (که از قبل تعیین شده است) اجرا می‌شود. "زمان‌بندی" الگوریتمی است که وظیفه‌ی کاری بعدی که بایستی اجرا شود را معین می‌کند. الگوریتم‌های مختلفی برای زمان‌بندی وجود دارند که یکی از ساده‌ترین آن‌ها روش "گردش نوبتی"[۵] است. در این شیوه، وظایف کاری یکی پس از دیگری در یک سیکل چرخشی اجرا می‌شوند.

وقتی "زمان‌بندی" کارش را به پایان رسانید، وظایف قبل و بعد می‌بایست از طریق Context Switch تعویض شوند. وظیفه Context Switch ذخیره‌ی تمام ثبات‌های پردازنده مربوط وضعیت وظیفه قبلی در یک ساختار داده و قرار دادن داده‌های وظیفه‌ی جدید، درون ثبات‌های پردازنده است.

جزء آخر این مجموعه "چارچوب راه‌انداز دستگاه[۶] " است (مکانیزمی که سیستم‌عامل ازطریق آن ارتباطی پایدار بین اجزای سخت‌افزار گوناگون خود برقرار می‌سازد).

این چارچوب، روشی استاندارد را درنظر می‌گیرد که از آن طریق اضافه نمودن سخت‌افزارهای جدید به سیستم‌عامل به راحتی صورت می‌گیرد. یک برنامه کاربردی برای دستیابی به یک جزء سخت‌افزاری، نیاز به راه‌انداز دستگاه[۷] دارد. چارچوب مذکور، باید ایمنی کامل در دسترسی به سخت‌افزار را فراهم نماید. به عنوان مثال، از دسترسی هم‌زمان دو برنامه‌ی مختلف به یک سخت‌افزار واحد، جلوگیری به عمل آورد.

---

۱ Periodic Timer

۲ Scheduler

۳ Preemptive

۴ Polling

۵ Round_Robin

۶ Device Driver Frame Work

۷ Device Driver

۱۷.۱.۲ واحد محافظت از حافظه

بعضی سیستم‌های تعبیه شده از سیستم‌های کنترلی و یا سیستم‌های عامل چند وظیفگی[۱] استفاده می‌کنند و بایستی اطمینان حاصل کنند که یک وظیفه کاری در کار وظیفه‌ی کاری دیگر، اختلال ایجاد نمی‌کند. "محافظت" به معنی محافظت یا حراست از منابع سیستمی و وظایف دیگر در مقابل دستیابی ناخواسته است.

۱۷.۱.۳ واحد مدیریت حافظه[۲]

هنگام استفاده از سیستم‌های تعبیه شده‌ی "چند وظیفگی"، منطقی است راه‌حلی آسان برای نوشتن، بارگیری و اجرای وظیفه‌های مختلف کاری داشته باشیم. بسیاری از سیستم‌های تعبیه شده‌ی امروزی به جای استفاده از سیستم‌های کنترلی اختصاصی به‌منظور تسهیل این سلسله عملیات، از سیستم‌های عامل استفاده می‌کنند. سیستم‌های عامل پیشرفته از MMU سخت‌افزاری استفاده می‌کنند.
یکی از کلیدی‌ترین مسئولیت‌های MMU ، توانایی مدیریت وظایف کاری به عنوان برنامه‌های مستقل در فضای آدرس اختصاصی خودشان است. یک وظیفه کاری که تحت یک سیستم‌عامل Z در کنار MMU کار می‌کند، نیازی ندارد که از میزان و محل قرار گیری حافظه مورد نیاز وظایف کاری دیگر اطلاع داشته باشد.
این مسأله‌ی طراحی هریک از وظایف کاری که تحت کنترل سیستم‌عامل قرار دارند، را بسیار ساده‌تر می‌سازد.
قبلاً در مورد محافظت از حافظه صحبت کردیم. این پردازنده‌ها فقط یک فضای حافظه‌ی فیزیکی قابل آدرس‌دهی دارند. هنگام اجرای یک وظیفه‌ی کاری، آدرس تولید شده توسط پردازنده به طور مستقیم برای دستیابی به حافظه‌ی اصلی استفاده می‌شود. دو برنامه کامپایل شده برای آدرس‌هایی هم‌پوشانی دارند، نمی‌توانند به طور هم‌زمان در حافظه مقیم باشند. این مسأله بدین معناست که عمل کردن در یک محیط چند وظیفگی، دشوار است. زیرا هر برنامه می‌بایست در بلوک آدرس مجزای خود در حافظه‌ی اصلی اجرا شود.

MMU ، با فراهم سازی بستر مناسب برای استفاده از حافظه‌ی مجازی[۳]، برنامه نویسی وظایف کاری را ساده می‌کند (حافظه‌ی مجازی، "فضای حافظه" ی جداگانه‌ای است که از حافظه‌ی واقعی متصل شده

---

[۱] Multi Tasking
[۲] Memory Management Unit_MMU
[۳] Virtual Memory

به سیستم مستقل است). MMU ، به عنوان یک مترجم، آدرس برنامه‌ها و داده‌هایی را که برای اجرای در فضای حافظه‌ی مجازی کامپایل شده‌اند را به آدرس فیزیکی[۱] مربوط به حافظه‌ی واقعی که برنامه‌ها و داده‌ها در آن‌جا ذخیره شده‌اند، تبدیل می‌کند. پروسه‌ی ترجمه، این قابلیت را می‌دهد: برنامه‌هایی که در مکان‌های مختلف حافظه قرار گرفته‌اند، در آدرس مجازی[۲] یکسانی اجرا شوند.
دید دوگانه از حافظه، منجر به وجود دو نوع آدرس جداگانه می‌شود: آدرس‌های مجازی و آدرس‌های فیزیکی. آدرس‌های مجازی توسط کامپایلر و لینکر[۳] هنگام قرار دادن یک برنامه در حافظه معین می‌شوند و آدرس‌های فیزیکی، برای اشاره به اجزای سخت‌افزاری واقعی، آن‌جایی که برنامه‌ها واقعاً در آن‌جا ذخیره شده‌اند، استفاده می‌شوند.
ARM ، چندین هسته‌ی پردازنده به همراه MMU داخلی ارایه کرده است. این پردازنده‌ها با استفاده از آدرس مجازی به طرزی مؤثر از محیط‌های چند وظیفگی پشتیبانی می‌کنند. در این بخش، اطلاعاتی در مورد مدیریت حافظه‌ی ARM و چند اصل پایه درباره‌ی استفاده از حافظه‌ی مجازی ارایه می‌دهیم.
با قابلیت‌های محافظتی ارایه شده توسط MPU شروع می‌کنیم و سپس به قابلیت‌هایی که توسط MMU افزوده می‌شوند، نگاهی می‌اندازیم.

۱۷.۱.۴ انتقال از یک MPU به MMU

در بخش مربوط به MPU، "نواحی محافظتی[۴] " را جهت ساماندهی و محافظت از حافظه معرفی کردیم. نواحی محافظتی یا **فعال** هستند و یا **تأخیری**[۵]. **نواحی فعال**، فضای شامل کد و یا داده‌ای هستند که در حال استفاده‌اند. در مقابل، **نواحی تأخیری**، کد یا داده‌ای را شامل هستند که در حال استفاده نیستند ولی احتمالاً در آینده نزدیک فعال می‌شوند. یک ناحیه‌ی تأخیری، محافظت شده است؛ بنابراین، توسط وظیفه-ی کاری در حال اجرا در دسترس نیست.
MPU سخت‌افزار اختصاصی جهت تخصیص مشخصات به نواحی مورد نظرشان را دارد. مشخصاتی که به یک ناحیه‌ی محافظت شده اختصاص داده می‌شوند را در جدول ۱۴.۱ می‌بینید. فرض بر این‌است که بر روش کار MPU و سخت‌افزار مربوط، اشراف کامل دارید. در این‌جا به پیکره‌بندی‌های سخت-افزاری MMU می‌پردازیم.

---

[۱] Physical Address
[۲] Virtual Address
[۳] Linker
[۴] region
[۵] Active Or Dormant

تفاوت اصلی بین MMU و MPU اضافه شدن سخت‌افزار اضافی به MMU جهت پشتیبانی از حافظه-ی مجازی است. MMU تعداد نواحی محافظتی در دسترس را نیز افزایش داده است. این کار از طریق انتقال جدول مشخصات (جدول ۱۴.۱) به حافظه‌ی اصلی ممکن شده است.

۱۷.۱.۵ حافظه‌ی مجازی چگونه کار می‌کند؟

در مبحث مربوط به MPU نشان داده شده که هر وظیفه کاری در محیط چند وظیفگی در آدرس‌های مجزا و ثابتی، کامپایل و اجرا می‌شدند. هر وظیفه، تنها در یک ناحیه‌ی محافظت شده اجرا می‌شود. هیچ یک از وظایف اجرا شده نمی‌تواند آدرس‌های هم‌پوشانی در حافظه‌ی اصلی داشته باشد.

در MMU وظایف، حتی اگر برای اجرا در نواحی هم‌پوشانی آدرس در حافظه اصلی، کامپایل و لینک شده باشند، می‌توانند اجرا شوند. MMU برای این منظور، از چندین نقشه‌ی حافظه‌ی مجازی[۱] و از یک نقشه‌ی حافظه فیزیکی منفرد استفاده می‌کند. وظایف برای این‌که نقشه‌های حافظه‌ی خودشان را داشته باشند، سخت‌افزار MMU از **تجدید محل آدرس**[۲] استفاده می‌کند. یعنی، آدرس‌های حافظه‌ی خروجی را توسط هسته‌ی پردازنده، قبل از رسیدن به حافظه‌ی اصلی، پردازش می‌کند. ساده‌ترین راه برای درک عملیات ترجمه‌ی آدرس، درنظر گرفتن یک **ثبات تجدید محل**[۳] درون MMU و بین هسته‌ی پردازنده و حافظه‌ی اصلی است.

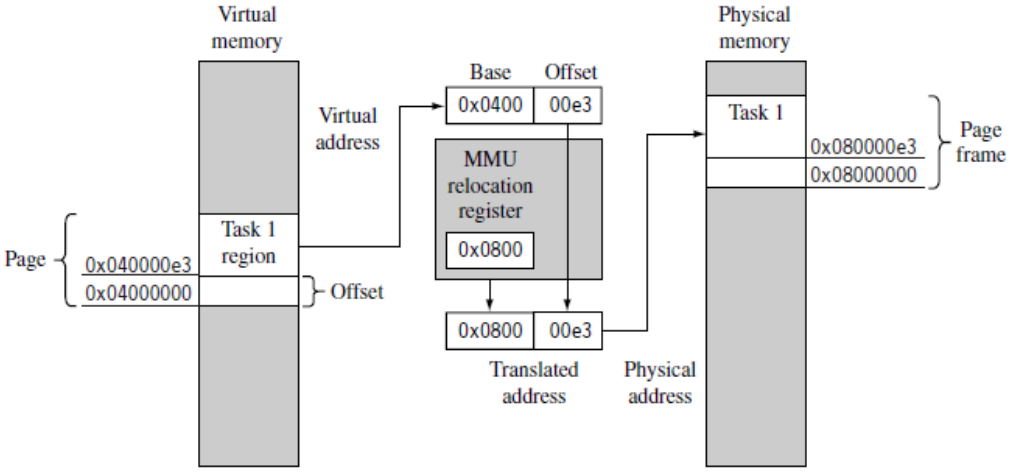
هنگامی که پردازنده یک آدرس مجازی تولید می‌کند، MMU بیت‌های بالای آدرس مجازی را گرفته و آن را با محتویات ثبات تجدید محل برای تولید آدرس فیزیکی، جایگزین می‌کند. این فرآیند در شکل ۱۷.۱ آمده است.

---

۱ Virtual Memory Map

۲ Address Relocation

۳ Relocation Register



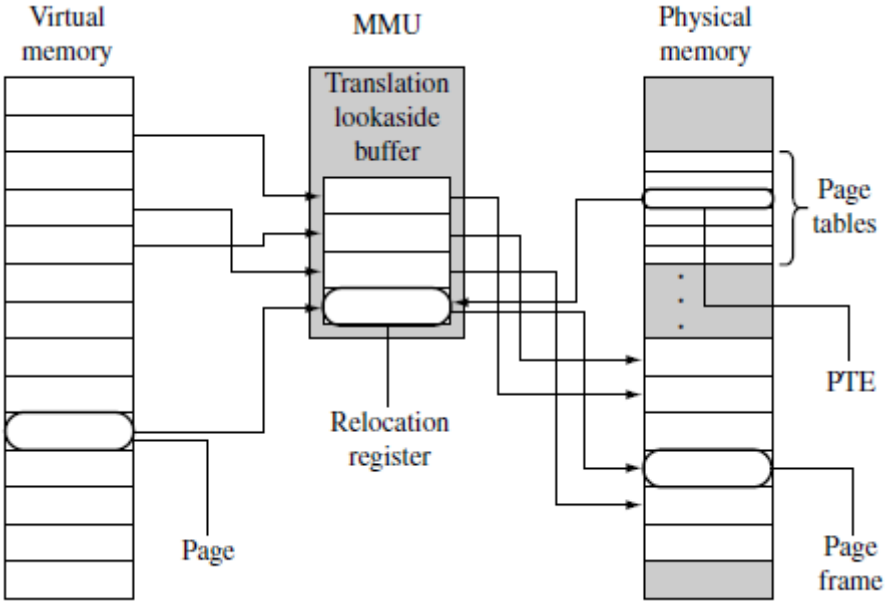
شکل ۱۷.۱

بخش پایینی آدرس مجازی، یک آفست است که به یک آدرس خاص در حافظه‌ی فیزیکی اشاره دارد. محدوده آدرس‌هایی که از این طریق می‌توانند ترجمه شوند، توسط حداکثر اندازه‌ی بخش آفستِ آدرس مجازی محدود می‌شود.

شکل ۱۷.۱ نشانگر یک وظیفه‌ی کاری نمونه است که برای اجرا در آدرس 0000 4000 0000 ×0 حافظه‌ی مجازی کامپایل شده است. ثبات تجدید محل آدرس حافظه‌ی مجازی 0000 4000 0000 ×0 را به آدرس حافظه‌ی فیزیکی 0000 8000 0000 ×0 ترجمه می‌کند.

اگر وظیفه‌ی کاری دومی برای اجرا در آدرس یکسانی از حافظه‌ی مجازی کامپایل شده باشد، در حافظه‌ی فیزیکی می‌تواند در هر ضریبی از 10000 ×0 (64 KB) جای بگیرد. با تغییر مقادیر موجود در ثبات تجدید محل به سادگی یک‌به‌یک بین آدرس مجازی یکسان و آدرس‌های فیزیکی گوناگون صورت می‌پذیرد.

یک ثبات تجدید محل منفرد، تنها قابلیت نگاشت **یک** ناحیه از حافظه را داراست. این ناحیه از حافظه‌ی مجازی با نام Page شناخته می‌شود. ناحیه‌ای از حافظه‌ی فیزیکی که توسط MMU و فرآیند ترجمه بدان اشاره می‌شود را با نام Page Frame می‌شناسیم.



شکل ۱۷.۲

رابطه‌ی بین Page، MMU و Page Frame را در شکل ۱۷.۲ می‌بینیم. سخت‌افزار MMU در ARM چندین ثبات تجدید محل برای عملیات ترجمه‌ی آدرس از حافظه‌ی مجازی به حافظه‌ی فیزیکی دارد. MMU برای موفقیت در امر تبدیل و ترجمه‌ی آدرس‌ها نیاز به تعداد زیادی از این ثبات‌های تجدید محل دارد.

مجموعه‌ی ثبات‌هایی که در MMU پردازنده‌های ARM موقتاً این تبدیلات را ذخیره می‌کنند، در حقیقت یک حافظه‌ی نهان مشتمل بر ۶۴ ثبات تجدید محل است. این حافظه‌ی نهان با نام TLB[۱] شناخته می‌شود. علاوه بر ثبات‌های تجدید محل، MMU از جداولی واقع در حافظه‌ی اصلی به‌منظور ذخیره کردن "نقشه-های حافظه‌ی مجازی" نیز استفاده می‌کند. این جدول‌های حاوی داده‌های عملیات ترجمه‌ی آدرس را با نام Page Table می‌شناسیم. هر یک از اطلاعات موجود در سلول‌های Page Table بیانگر اطلاعات لازم برای تبدیل یک Page از حافظه‌ی مجازی به یک Page Frame از حافظه‌ی فیزیکی است.

هر یک از این سلول‌ها (که با PTE[۲] می‌شناسیم) اطلاعات زیر را در مورد Page های حافظه‌ی مجازی دربردارند: آدرس Base فیزیکی که برای ترجمه استفاده می‌شود، مجوز دسترسی[۳] درنظر گرفته شده برای Page، پیکره‌بندی حافظه‌ی نهان و Write buffer برای Page مورد نظر. اگر به جدول ۱۴.۱

---

[۱] Translation Lookaside Buffer

[۲] Page Table Entry

[۳] Access Permission

مراجعه کنید می‌بینید که هم‌اکنون اطلاعات پیکره‌بندی نواحی محافظتی MPU در PTE ها نگه‌داری می‌شوند. **نواحی محافظتی** در MMU توسط نرم‌افزار و با گروه‌بندی بلوک‌های **Page های مجازی** در حافظه ایجاد می‌شوند.

۱۷.۲ سیستم‌عامل بی‌درنگ[۱]

RTOS ، سیستم‌عاملی است که برای خدمت‌رسانی به برنامه‌ها و کاربردهای بی‌درنگ درست شده است. یک سیستم‌عامل بی‌درنگ سخت[۲] دارای زمان مکث و درنگ کمتری در پاسخگویی به وظایف کاربردی[۳] نسبت به انواع نرم[۴] خود هستند. هدف اصلی در طراحی RTOS، لزوماً سریع بودن آن نیست. بلکه در هر یک از حوزه‌های سخت یا نرم باید تابع مشخصات آن حوزه باشد. به عنوان مثال، یک RTOS که دارای زمان پایان برای انجام وظایفش است، نوع نرم و RTOS ای که مشخصاً به زمان‌های اجرا پایبند است را نوع سخت می‌دانیم.
یک RTOS ، دارای یک الگوریتم پیشرفته برای زمان‌بندی[۵] است. کم‌ترین تأخیر در رسیدگی به وقفه (Interrupt Latency) و کم‌ترین زمان در تعویض ریسمان‌ها (Thread Suitching) از نقاط کلیدی در طراحی یک سیستم‌عامل بی‌درنگ هستند. اما بیشترین ارزش یک RTOS در سرعت و قابل پیش‌بینی بودن در پاسخ‌دهی است تا میزان کاری که در یک پریود زمانی می‌تواند انجام دهد.

۱۷.۲.۱ روش‌های طراحی

دو روش پرکاربرد در طراحی RTOS ها به این شرح است:

- رویدادگرا- وظایف را تنها هنگامی که یک رویداد با تقدم بالاتر نیاز به رسیدگی دارد، عوض می‌کند. این روش را با نام تقدم انحصاری[۶] و یا زمان‌بند تقدم[۷] می‌شناسیم؛
- طراحی با اشتراک زمانی[۸]- وظایف مختلف را براساس یک برنامه‌ی منظم و به ترتیب و پشت سرهم اجرا می‌کند. به این روش، چرخشی[۱] می‌گویند.

---

[۱] Real_time Operating System_RTOS
[۲] Hard RTOS
[۳] Task
[۴] Soft RTOS
[۵] Scheduling
[۶] Preemptive Priority
[۷] Priority Scheduling
[۸] Time Sharing

روش اشتراک زمانی وظایف را پشت سرهم اجرا می‌کند. این روش چند وظیفگی[۲] نرم‌تری ایجاد می‌کند و این تصور را در پروسه و یا کاربر ایجاد می‌کند که برنامه‌ی فعلی، تنها برنامه‌ی در حال اجرا بر روی سیستم است.

CPU های قدیمی تعداد سیکل بیشتری برای تعویض وظایف مختلف نیاز داشتند. در طی عملیات تعویض وظایف، پردازنده عملاً مورد استفاده‌ای نداشت. برای مثال در پردازنده‌ی قدیمی 68000 با فرکانس کاری 20 MHz یک نمونه زمان تعویض وظایف کاری، 20 mS طول می‌کشید. در مقایسه یک پردازنده‌ی ARM با فرکانس کاری 100 MHz، همین کار را در کمتر از 3 mS انجام می‌دهد. به همین دلیل در سیستم‌های عامل اولیه تلاش بر این بود که تا حد ممکن از تعویض‌های غیرضروری وظایف خودداری کنند.

۱۷.۲.۲ زمان‌بندی[۳]

در یک طرح نمونه، یک وظیفه سه حالت مختلف می‌تواند داشته باشد:

۱) در حال اجرا (running)؛
۲) آماده برای اجرا (ready)؛
۳) منتظر ورودی و یا خروجی (blocked).

بیشتر وظایف در حالت عادی در وضعیت‌های blocked و یا ready هستند. زیرا CPU در یک لحظه تنها یک وظیفه را می‌تواند اجرا کند.

۱۷.۲.۳ Window CE

Windows CE ، سیستم‌عاملی است که توسط شرکت مایکروسافت برای سیستم‌های تعبیه شده‌ی خاص، طراحی و ساخته شد. Windows CE یک سیستم‌عامل و هسته[۴] جدا از نسخه‌های رومیزی ویندوز است (نمونه‌ی ساده شده‌ی آن‌ها نیست). و نباید با سیستم‌های عامل دیگری چون Windows XP Embedded (که براساس NT ساخته شده است)، اشتباه گرفته شود.

---

۱ Round_robin
۲ Multi tasking
۳ Scheduling
۴ Kernel

شکل ۱۷.۳

مایکروسافت، مجوز استفاده از این سیستم‌عامل را به سازندگان دستگاه‌ها و OEM ها می‌دهد. سازندگان دستگاه‌ها می‌توانند به‌منظور دستیابی به محیط رابط کاربری دلخواه، آن را دستکاری کنند.
Windows CE در بسیاری از پردازنده‌های معروف چون Intel X86، MIPS، ARM و Super H به راحتی اجرا می‌شود.
این نسخه از ویندوز برای اجرا در دستگاه‌های تعبیه شده، بهینه گردیده است. (هسته‌ی Win CE در چندین مگابایت از حافظه به راحتی جای می‌گیرد.) Win CE با تعاریف یک سیستم‌عامل تعبیه شده با یک "تأخیر وقفه‌ی معین" تطابق دارد. این سیستم‌عامل از نسخه‌ی ۳ به بعد از ۲۵۶ سطح تقدم وقفه پشتیبانی می‌کند.
پایه‌ای‌ترین جزء اجرایی در این محیط، یک ریسمان[۱] است.
اولین نسخه‌های اجرایی این سیستم‌عامل، از محیط‌های گرافیکی مشابه ویندوز بهره می‌بردند. از آن پس Windows CE فقط محدود به دستگاه‌های دستی نشد، بلکه پایه و اساس بسیاری از سیستم‌های عامل همچون Auto PC، Pocket PC، Windows Mobile، Smart Phone و... قرار گرفت.
مشخصه‌ی متفاوت این سیستم‌عامل نسبت به دیگر سیستم‌های عامل مایکروسافت در این‌است که بسیاری از قسمت‌های آن به صورت کد منبع[۲] در دسترس قرار گرفتند. روش کار به این صورت است که کد منبع به فروشندگان تجهیزات سخت‌افزاری اولیه داده می‌شد. پس، آن‌ها می‌توانستند این کد را با سخت‌افزار خود تطبیق دهند. سپس، از ابزاری به نام Platform Builder برای سفارشی کردن محیط ویندوز و نهایتاً ساختن فایل‌های Image (فایل‌های باینری که بر روی سخت‌افزار اجرا می‌شوند) استفاده می‌کنند. بسیاری از فایل‌ها و کدهایی که نیاز به تغییرات ندارند و از CPU سیستم مستقل هستند، همچنان به صورت فایل‌های باینری ارایه می‌شوند.

۱۷.۲.۳.۱ ابزارهای توسعه‌ی نرم‌افزاری

MS Visual Studio

این محصول و دیگر محصولات قدیمی‌تر مایکروسافت می‌توانستند برای تولید فایل‌های اجرایی Win CE یا Windows Mobile به کار گرفته شوند.
قاب کاری فشرده‌ی Net. (Net Compact framework.) زیرمجموعه‌ای از قاب کاری Net. است. که از پروژه‌های C# و VB.Net پتیبانی می‌کند. مایکروسافت از نسخه‌ی 2010 این نرم‌افزار، مشخصه‌ی

---

[۱] Thread
[۲] Source Code

فوق را حذف کرد. در عوض، از یک محیط جدید و کاملی به نام Windows Phone7 برای این منظور استفاده می‌کند.
نرم‌افزار Delphi Prism شرکت Code gear نیز از Net Compact framework. پشتیبانی می‌کند. پس می‌توان از آن نیز برای توسعه برنامه‌های فوق استفاده کرد.
همچنین، نسخه‌هایی از برنامه‌های مختلف چون: Free Pascal، Basic 4PPC، GL Basic، Lab View و... برای توسعه نرم‌افزاری تحت Win CE در دسترس هستند.
مشخصات نسخه‌های مختلف Win CE را در جدول CE-1 می‌بینید.

۱۷.۲.۴ eCos[۱]

eCos ، یک سیستم‌عامل سورس‌باز، بدون حق امتیاز و بی‌درنگ است که برای سیستم‌های تعبیه شده طراحی گردیده است. این سیستم‌ها فقط می‌توانند دارای یک پروسه[۲] ولی چندین ریسمان[۳] باشند.
طراحی این سیستم‌عامل به گونه‌ای بوده است که کاملاً سفارشی شونده و مناسب برای سخت‌افزار هدف باشد. پیاده سازی این سیستم‌عامل به زبان C/C++ بوده است و دارای توابع و API هایی برای POSIX می‌باشد.

۱۷.۲.۴.۱ طراحی eCos

این سیستم‌عامل، برای وسایلی با ظرفیت حافظه‌ی بین ۱۰ تا چندین صد کیلوبایت و عملکرد بی‌درنگ طراحی شده است. این سیستم‌عامل، به راحتی می‌تواند در سیستم‌هایی که مقدار کمیRAM برای پشتیبانی از "لینوکس تعبیه شده" دارند (که حداقل به 2 MB حافظه‌ی RAM نیاز دارند) پیاده سازی شوند.
eCos در بسیاری از پلتفرم‌های سخت‌افزاری می‌توانند اجرا شوند. برای مثال: ARM، Calm RISC، IA-32، Motorola 68000، MIPS، NiosII، Power PC، SPARC، Super H و... .
به همراه eCos، Red Boot نیز توزیع می‌شود. Red Boot یک برنامه‌ی سورس‌باز است که در این‌جا برای عملیات بوت در لایه‌ی سخت‌افزاریeCos به کار گرفته می‌شود.

---

۱ embedded Configurable Operating System-eCOS
۲ Process
۳ Thread

۱۷.۲.۵ چگونه یک سیستم‌عامل بی‌درنگ را انتخاب کنیم؟

مهندسان اغلب از واژه‌ی "بی‌درنگ" برای توصیف مشکلاتی استفاده می‌کنند که در آن پاسخ‌های کُند و پاسخ‌های اشتباه، هردو نامطلوب هستند. این مشکلات اغلب درگیر مهلت، سررسید و یا آخرین فرصت[۱] برای انجام یک وظیفه هستند. برای مثال، اگر نرم‌افزاری که کنترل ترمز ABS را بر عهده دارد، در یکی از Deadline ها به مشکل برخورد کند، رخداد تصادف قطعی است. پس نرم‌افزار باید به موقع و تا قبل از پایان یافتن مهلت انجام، پاسخگو باشد.

[۱] Deadline

فصل هجدهم

سناریوهای طراحی

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

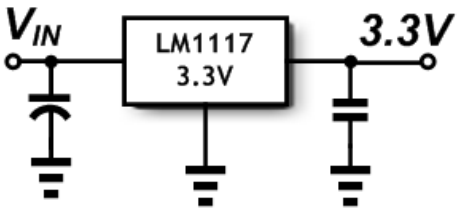
- ✓ سناریوهای طراحی بردهای آزمایشی ARM.

در این بخش، چندین سخت‌افزار مختلف براساس میکروکنترولرهای ARM را با هم بررسی خواهیم کرد. این طرح‌ها شامل میکروکنترولرهای نسبتاً ابتدایی، با حداقل امکانات جانبی تا میکروکنترولرهای پیشرفته‌تری که بر رویشان سیستم‌عامل اجرا می‌شود، می‌گردند. نخست، میکروکنترولری برای راه‌اندازی ادوات جانبی ساده را بررسی خواهیم کرد. نحوه‌ی پیکره‌بندی، اتصالات و سایر مسایل درگیر طراحی را بررسی می‌کنیم. سپس به ادوات جانبی پیشرفته‌تری چون USB و رابط‌های صوتی، اترنت و نمایشگرهای LCD خواهیم پرداخت. نقشه‌های این بردها به همراه کدهای راه‌اندازی اولیه‌شان در CD همراه کتاب آورده شده‌اند.

۱۸.۱ سناریو ۱ : طراحی برد پایه، براساس LPC2103

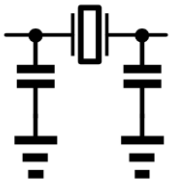
LPC2103 ، از ساده‌ترین میکروکنترولرهای خانواده LPC2000 است که 8 KB حافظه‌ی SRAM و 32 KB حافظه‌ی فلش دارد. این تراشه، از ساده‌ترین و ارزان‌ترین میکروکنترولرهای ARM به حساب می‌آید. با داشتن ADC ۱۰ بیتی، رابط‌های سریال استاندارد با قابلیت‌های خوب همانند SPI، SSP، UART و I^2C انواع مختلفی از تایمرهای کارآمد و یک واحد کلاک بی‌درنگ (RTC)، از ساده‌ترین و در عین حال مفیدترین میکروکنترولرهای ۳۲ بیتی به شمار می‌آید. پایه‌های این میکروکنترولر، قابلیت پذیرش ولتاژ 5 V را نیز دارند.

این میکروکنترولر، رگولاتور ولتاژ داخلی ندارد. برای راه‌اندازی هسته‌ی آن که 1.8 V نیاز دارد، باید ولتاژ 1.8 V را از خارج تراشه تأمین کرد. ولتاژ 3.3 V میکروکنترولر نیز برای ادوات جانبی سطح تراشه مورد نیاز است. پس، برای تأمین دو سطح ولتاژ، نیاز به دو رگولاتور با ولتاژهای 1.8 V و 3.3 V داریم. برای این کار می‌توان از دو رگولاتور LM1117 قابل تنظیم و یا یک رگولاتور دو خروجی همانند MIC5320 ، استفاده کرد. نحوه‌ی کار را در شکل ۱۸.۱ می‌بینیم.



شکل ۱۸.۱

راه‌اندازی کلاک سیستم با استفاده از یک کریستال 12 MHz و دو عدد خازن همانند شکل ۱۸.۲ صورت می‌گیرد:

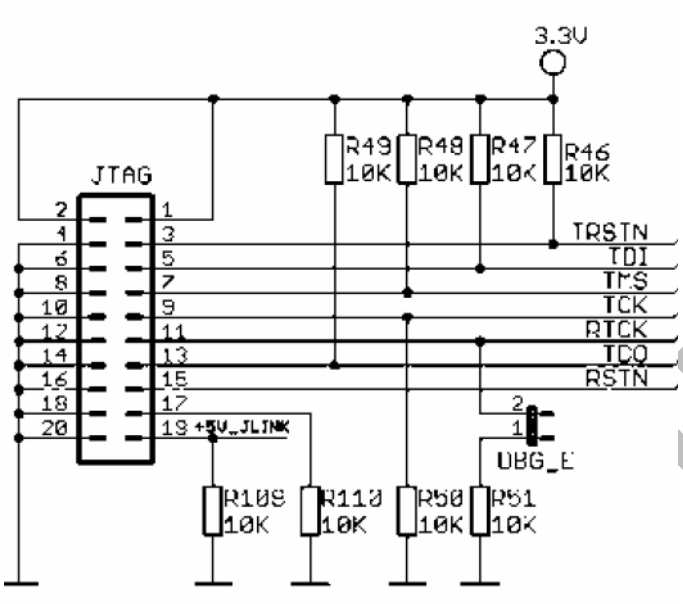


شکل ۱۸.۲

برای راه‌اندازی RTC، همان‌طور که مرسوم است، از کریستال 32.768 KHz به همراه دو خازن استفاده می‌کنیم. اتصال آن به پایه‌های RTxC1 و RTxC2 می‌باشد.

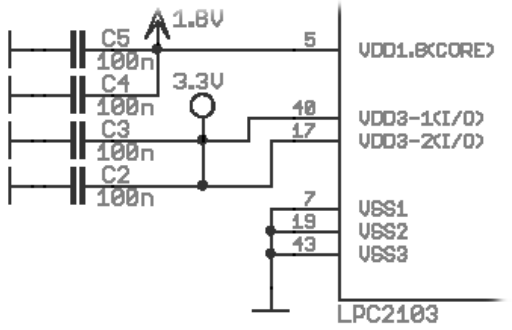
برای برنامه‌ریزی حافظه‌ی فلش روی تراشه از دو طریق می‌توان اقدام کرد. روش اول، استفاده از بوت‌لودر[۱] روی تراشه که اختصاصاً برای اتصال با رایانه و برنامه‌ریزی فلش، طراحی شده است (این کد در درون ROM روی تراشه مستقر است)، می‌باشد. در روش دوم، از یک واسط استاندارد JTAG برای برنامه‌ریزی و عیب‌یابی کدهای روی میکروکنترولر استفاده می‌کنیم. روش کار در شکل ۱۸.۳ نشان داده شده است.

[۱] Boot Loader



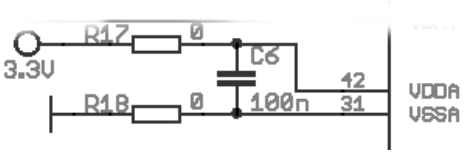
شکل ۱۸.۳

استفاده از خازن‌های دکوپلاژ در طراحی مدارات دیجیتالی که دارای پایه‌های تغذیه‌ی زیادی هستند، ضروری است. عدم به کار گیری این خازن‌ها منجر به کارکرد نادرست میکروکنترولر خواهد شد. در شکل زیر، خازن‌های مناسب در اماکنی که ضروری هستند، گذاشته شده است.



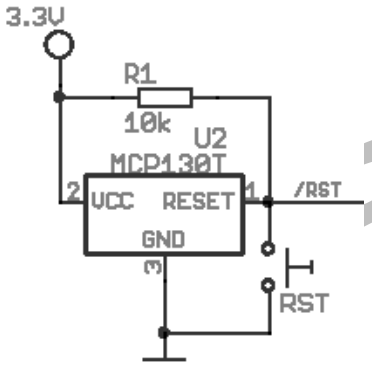
شکل ۱۸.۴

تغذیه‌ی پایه‌های مربوط به مدارات آنالوگ (مدار ADC در اینجا از طریق V_{DDA} و V_{SSA} تغذیه می‌شوند)، باید از طریق فیلترهای مناسبی صورت پذیرد. عموماً از فیلترهای ساده LC و یا RC برای این کار استفاده می‌شود.



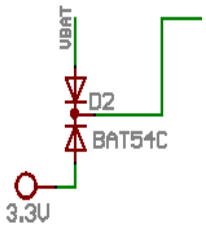
شکل ۱۸.۵

ریست، از پایه‌های مهم میکروکنترولرهاست. اگر در طراحی سخت‌افزار میکروکنترولر، توجه کافی به آن نشود، در فازهای بعدی طراحی و هنگام آزمایش سخت‌افزار به مشکلاتی (ممکن است جدی هم باشد) بر خواهیم خورد. در طراحی سخت‌افزار مربوط به ریست، از دو عنصر R و C و یک کلید برای اعمال ریست به صورت دستی بهره می‌بریم. مداری مشتمل بر MCP130T نیز برای انجام ریست هنگام اتصال تغذیه نیز می‌تواند انتخاب جایگزین خوبی باشد.



شکل ۱۸.۶

اتصال باتری برای عملیات مورد نیاز در بخش RTC، عموماً از طریق دو دیود که در شکل ۱۸.۷ نشان داده شده، انجام می‌شود. اگر باتری وصل باشد یا تغذیه‌ی 3.3 V فرقی نمی‌کند. این دو تغذیه همانند پشتیبان برای تأمین ولتاژ پایه‌ی V_{BAT} عمل می‌کنند. (البته چون معمولاً ولتاژ باتری از 3.3 V کمتر است، هنگام اتصال تغذیه، دیود سری باتری قطع می‌باشد. و از آن جریانی کشیده نمی‌شود):

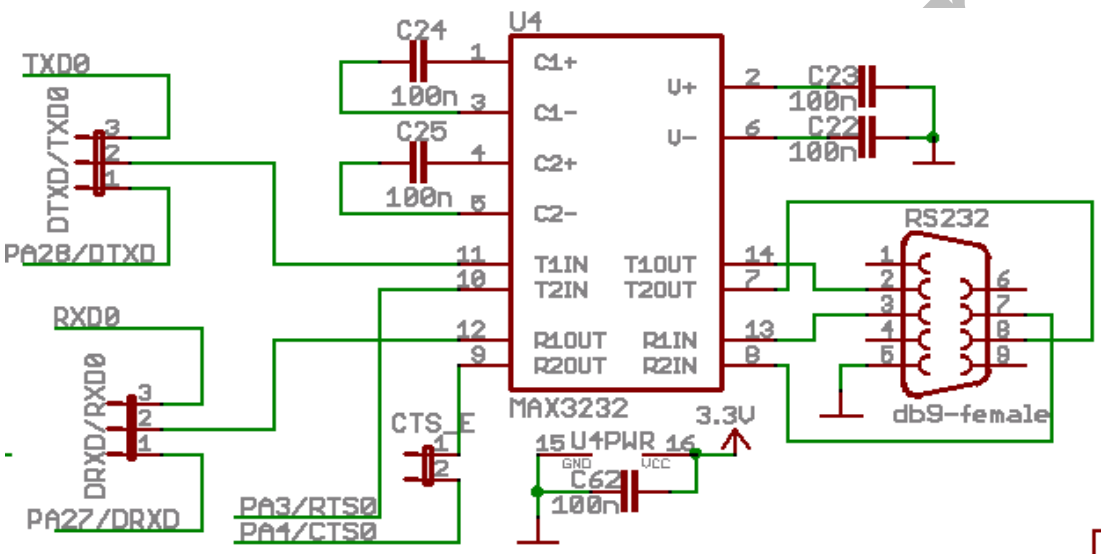


شکل ۱۸.۷

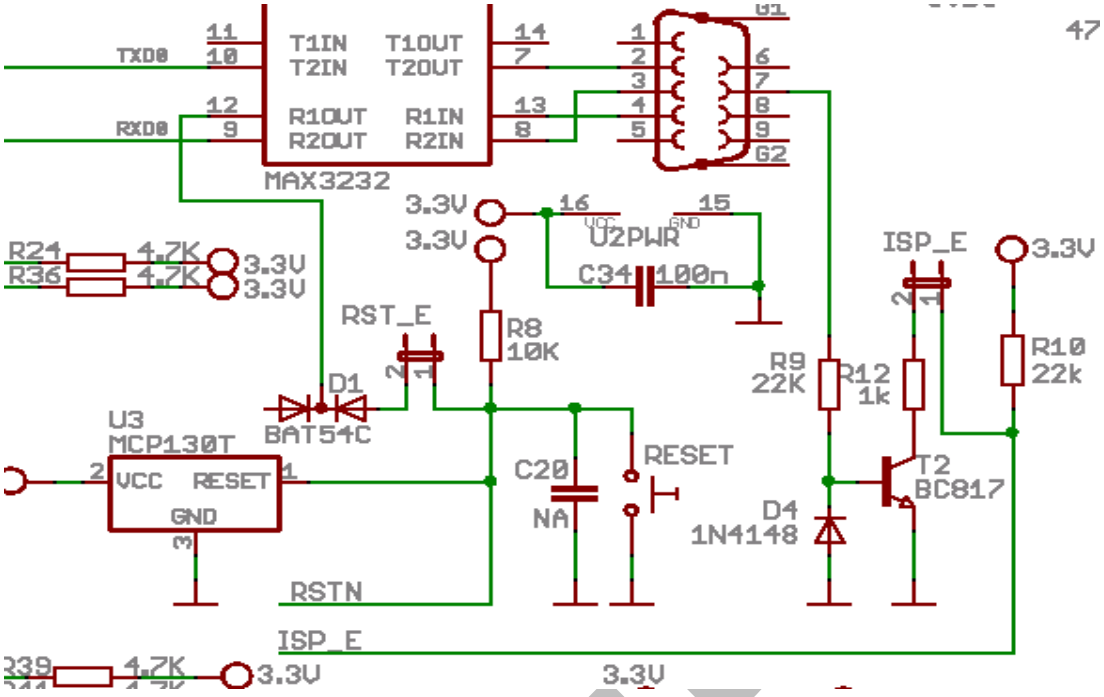
برای استفاده از رابط سریال UART و اتصال آن به رایانه و یا هر مدار مناسب دیگر، به یک تراشه‌ی مبدل سطح ولتاژ نیاز داریم. تراشه‌ی معروف MAX3232 این وظیفه را بر عهده دارد. روش کار در شکل ۱۸.۸ آمده است.

نکته

اگر از بوت‌لودر در سطح تراشه برای برنامه‌ریزی استفاده می‌کنید، این کار از طریق پایه‌های ارتباطی UART و دو پایه‌ی Reset و ISP_E (پایه‌ی P0.14 در این میکروکنترولر) انجام می‌شودو مدار مربوط برای این کار شبیه مداری است که در شکل ۱۸.۸ آمده است.



شکل ۱۸.۸



شکل ۱۸.۹

سایر مشخصات این برد و طراحی ارتباطات آن، بسیار ابتدایی بوده و از توضیحات آن اجتناب می‌ورزیم. نقشه‌ی کامل شده این برد را می‌توانید در پوشه‌ی ذی‌ربط در CD همراه کتاب پیدا کنید.

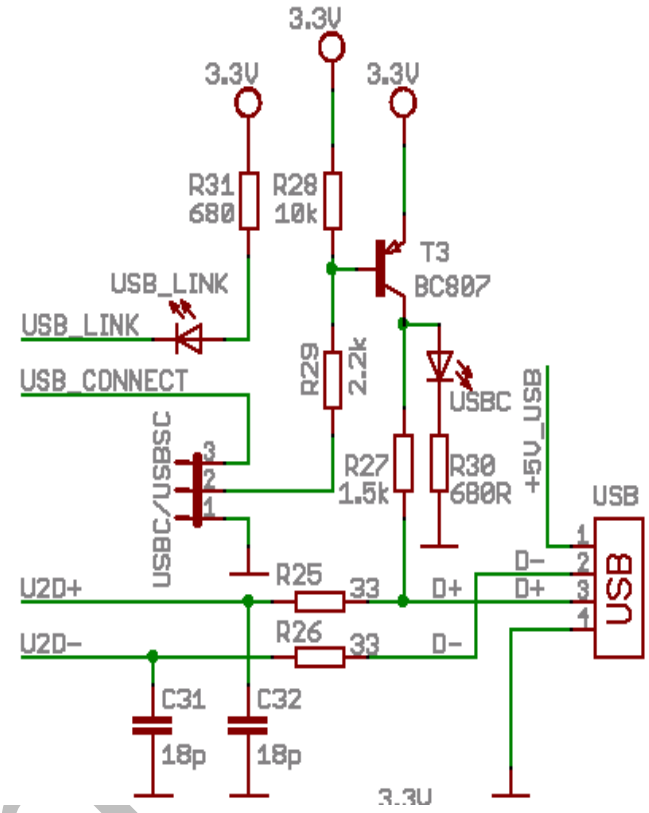
۱۸.۲ سناریو ۲ : طراحی برد پایه با استفاده از LPC2148

این پردازنده، همانند نمونه‌ای که در سناریوی ۱ آمده بود، به خانواده‌ی LPC2000 تعلق دارد. این میکروکنترولر دارای یک کنترل کننده‌ی USB Device در درون خود است. اتصال یک چنین میکروکنترولرهایی (که دارای کنترل کننده‌ی USB داخلی هستند) به کامپیوتر برای ارتباط از طریق USB بسیار راحت و جذاب شده است. در این طراحی دیگر نیازی به تراشه‌ی مبدل USB در خارج از میکروکنترولر نخواهید داشت.

طراحی این برد را همانند سناریوی ۱ آغاز می‌کنیم. منبع تغذیه مورد نیاز برای این میکروکنترولر همانند آن چیزی است که در مورد LPC2103 گفته شد. با این تفاوت که در این‌جا فقط نیاز به یک ولتاژ 3.3 V داریم.

تغذیه‌ی ADC ، نیز همانند قبل انجام می‌شود. پایه مرجع ADC (V_{REF}) نیاز به یک خازن تثبیت کننده محلی دارد. این خازن نویزهای احتمالی را از بین می‌برد.

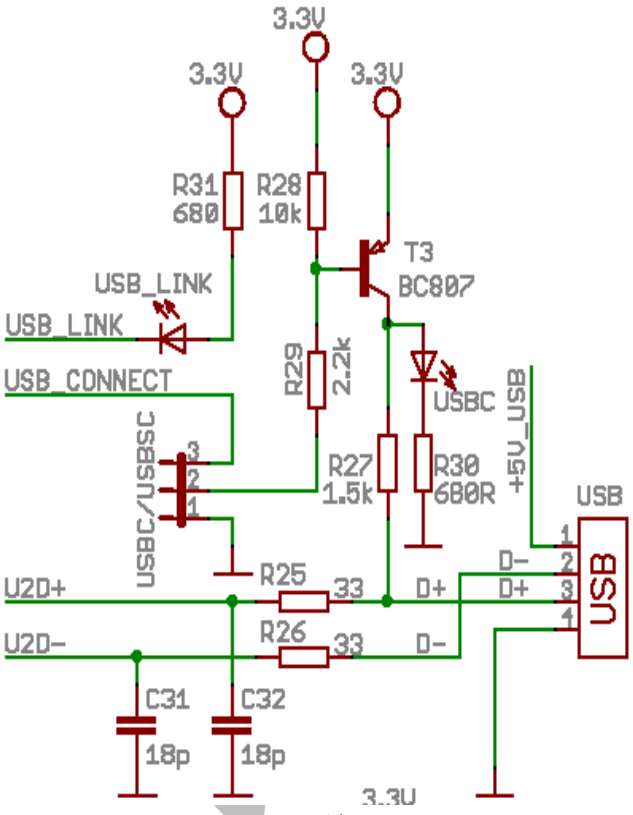
ارتباط با USB فقط نیاز به چندین قطعه‌ی غیرفعال[1] ساده دارد. کنترل کننده‌های USB دارای پایه‌های اضافی دیگری (علاوه بر پایه‌های +D و -D) برای نشان دهنده‌ی اتصال USB و... دارند. در این جا فقط از پایه‌ی اتصال USB (پایه‌های داخلی که هنگام برقراری صحیح اتصال، فعال می‌شود) یا همان USB-Link برای روشن کردن LED ، استفاده کرده‌ایم. در شکل ۱۸.۱۰، روش اتصالات فوق را می‌بینیم.



شکل ۱۸.۱۰

در طراحی این برد، از مدار راه‌اندازی کارت حافظه‌ی SD/MMC نیز استفاده کرده‌ایم. از آن‌جایی که این میکروکنترولر در درون خود، کنترل کننده‌ی حافظه‌ی SD/MMC ندارد، از ارتباط سریال SPI برای ارتباط با این حافظه استفاده کرده‌ایم. نحوه‌ی کار ساده است و با اتصال چند مقاومت، خازن و یک کانکتور کارت حافظه، این کار میسر شده است.

[1] Passive



شکل ۱۸.۱۱

فصل نوزدهم

الگوریتم‌های رمز کردن داده‌ها

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ روش رمز کردن داده‌ها با استفاده از RSA.

۱۹.۱ رمزدار کردن به روش RSA

دو روش پرکاربرد در رمزدار کردن داده‌ها، DES و RSA هستند. هر کدام از این روش‌ها، ضعف‌ها و قوت‌های خود را دارند. هرچند، به علت این که پیاده سازی نرم‌افزاری RSA آسان‌تر بوده و نیاز به کدهای ساده‌تری دارد. با این حال، در عین سادگی پیاده سازی خود الگوریتم، تولید کلیدهای خصوصی و عمومی برای کارکرد سیستم، مشکلات بیشتری را به همراه دارد. در این بخش، روش‌هایی ساده برای پیاده سازی الگوریتم RSA به همراه روش‌هایی برای تولید کلیدهای عمومی و خصوصی ارایه می‌کنیم.
توانایی موتور رمز کننده‌ی ارایه شده، محدود به اعداد صحیح ۶۴ بیتی (unsigned long long) می‌باشند. این موضوع به معنای امنیت کمتر نسبت به کدهایی است که با کلیدهای بزرگ‌تری رمزدار شده‌اند. هرچند، در عین سادگی و حجم کم کد حاصل، داده‌ها نسبت به انواع رمز نشده از امنیت بسیار بیشتری برخوردار هستند.
کد مذکور، بر روی یک نمونه تراشه‌ی با هسته‌ی Cortex-M3، 1.6 KB از حافظه‌ی فلش را اشغال می‌کند.

```
/************************************************************
************/
/*!
    \file     rsa.h

    Basic RSA-encryption using 64-bit math (32-bit keys).

*/
/************************************************************
************/

#ifndef _RSA_H_
#define _RSA_H_
```

```
/* In a secure implementation, huge_t should be at least 400
decimal digits, *
 * instead of the 20 provided by a 64-bit value.  This means
that key values *
 * can be no longer than 10 digits in length in the current
implementation.  */
typedef uint64_t huge_t;

/* Structure for RSA public keys. */
typedef struct rsaPubKey_s
{
  huge_t e;
  huge_t n;
}
rsaPubKey_t;

/* Define a structure for RSA private keys. */
typedef struct rsaPriKey_s
{
  huge_t d;
  huge_t n;
}
rsaPriKey_t;

void rsaTest();
void rsaEncrypt(huge_t plaintext, huge_t *ciphertext,
rsaPubKey_t pubkey);
void rsaDecrypt(huge_t ciphertext, huge_t *plaintext,
rsaPriKey_t prikey);

#endif

/************************************************************
************/
/*!
    \file     rsa.c
*/
/************************************************************
************/

#include <cross_studio_io.h>

#include "rsa.h"

static huge_t modexp(huge_t a, huge_t b, huge_t n)
{
  huge_t y;
  y = 1;
```

```
  /*  Compute pow(a, b) % n using the binary square and
multiply method. */
  while (b != 0)
  {
    /*  For each 1 in b, accumulate y. */
    if (b & 1)
    {
      y = (y * a) % n;
    }

    /* Square a for each bit in b. */
    a = (a * a) % n;

    /*  Prepare for the next bit in b. */
    b = b >> 1;
  }

  return y;
}

void rsaTest()
{
  huge_t      rsaOrig, rsaDecrypted, rsaEncrypted;
  rsaPubKey_t publicKey;
  rsaPriKey_t privateKey;
  int         i;

  debug_printf("Encrypting with RSA\n");

  // Values based on 64-bit math (huge_t = unsigned long long)
  // which will result in more secure encryption, but also
  // increases the size of the encrypted text
  publicKey.e = 21;
  publicKey.n = 16484947;
  privateKey.d = 15689981;
  privateKey.n = 16484947;

  // Alternative values with 32-bit math (huge_t = unsigned
long)
  // or when smaller encrypted text is desired
  // publicKey.e = 17;
  // publicKey.n = 209;
  // privateKey.d = 53;
  // privateKey.n = 209;

  debug_printf("d=%lld, e=%lld, n=%lld\n",
               privateKey.d, publicKey.e, publicKey.n);

  for (i = 0; i < 128; i++)
  {
```

```
      rsaOrig = i;
      rsaEncrypt(rsaOrig, &rsaEncrypted, publicKey);
      rsaDecrypt(rsaEncrypted, &rsaDecrypted, privateKey);

      if (rsaOrig == rsaDecrypted)
      {
         debug_printf("In=%5lld, Encrypted=%10lld, Out=%5lld
(OK)\n",
                      rsaOrig, rsaEncrypted, rsaDecrypted);
      }
      else
      {
         debug_printf("In=%5lld, Encrypted=%10lld, Out=%5lld
(ERROR)\n",
                      rsaOrig, rsaEncrypted, rsaDecrypted);
      }
   }
}

void rsaEncrypt(huge_t plaintext, huge_t *ciphertext,
rsaPubKey_t pubkey)
{
   *ciphertext = modexp(plaintext, pubkey.e, pubkey.n);

   return;
}

void rsaDecrypt(huge_t ciphertext, huge_t *plaintext,
rsaPriKey_t prikey)
{
   *plaintext = modexp(ciphertext, prikey.d, prikey.n);

   return;
}
```

۱۹.۲ تولید کلیدهای عمومی و خصوصی

رمزکنندگی در RSA **نامتقارن**[۱] است. بدین معنا که به هر دو کلید خصوصی و عمومی نیاز دارد. برای به دست آوردن مجموعه‌ای از کلیدها که با یکدیگر سازگار هستند، نیاز به یک سری اطلاعات ریاضی پایه و محاسبات مربوط داریم که در زیر به آن‌ها اشاره می‌کنیم.

گام نخست: پیمانه‌ی N[۲] را با استفاده از دو عدد اول p و q محاسبه کنید.

---

۱ Asymmetric

۲ Modulus

ابتدا دو عدد اول بزرگ را پیدا کنید. بزرگ‌ترین اندازه‌ی این داده‌ها بستگی به نوع پیاده سازی نرم‌افزاری برنامه دارند. در این‌جا از نوع داده unsigned long long (۶۴ بیتی) با قابلیت نگه‌داری مقادیر بین 0 تا 18,446,744,073,709,551,615 استفاده می‌کنیم. در این جا نیاز به عمل ضرب دو عدد اول داریم. بنابراین بزرگ‌ترین اعدادی که می‌توانیم انتخاب کنیم، باید از 2^{32} کمتر باشند. تا حاصل‌ضربشان از ۶۴ بیت بیشتر نشود.

مقدار پیمانه‌ی مورد نظر از فرمول N = p.q محاسبه می‌شود. به عنوان، مثال دو عدد اول 1931 و 8537 را انتخاب کرده‌ایم.

پس:

N = pq = 1931 × 8537 = 16,484,947

توجه

برای پیدا کردن اعداد اول، الگوریتم‌های متفاوتی وجود دارند. در این جا برای سادگی کار، از فهرست آماده‌ی اعداد اول استفاده می‌کنیم.

اینترنت

نمونه‌ای از فهرست اعداد اول آماده در آدرس http://www.walter-fendt.de/m14e/primes.htm یافت می‌شود.

گام دوم: یک عدد فرد صحیح که هیچ عامل مشترکی با (p-1)(q-1) نداشته باشد را، پیدا کنید.

در ادامه‌ی مثال بالا، داریم:

r = (p-1)(q-1) = 1930 × 8536 = 16,474,480

اینترنت

از ابزار موجود در آدرس
http://www.cs.drexel.edu/~introcs/Fa09/labs/LB-EndOfTerm/RSAWorksheetv4b.html می‌توان استفاده کرد.

گام سوم : ابزاری برای تعیین مقدار e (کلید عمومی) و d (کلید خصوصی) در آدرس فوق وجود دارند.

فصل بیستم

آینده معماری ARM و فناوری‌های برتر

اهداف فصل

با پایان این فصل، شما با موارد زیر آشنا می‌شوید:

- ✓ ریزپردازنده‌ها چگونه پا به عرصه دنیای دیجیتال گذاشتند؟
- ✓ پردازنده‌های ARM چگونه به وجود آمدند؟
- ✓ خانواده‌های گوناگون پردازنده‌های ARM کدامند؟
- ✓ میکروکنترلرها چگونه قطعاتی هستند و یک سیستم تعبیه شده چیست؟
- ✓ میکروکنترلرهای ۳۲ بیتی چه مزایایی نسبت به انواع قدیمی‌تر خود دارند؟
- ✓ چه مسایل و تکنیک‌هایی در طراحی‌های ۳۲ بیتی دخیل هستند؟
- ✓ کاربرد وسیع میکروکنترلرهای ۳۳ بیتی در صنعت
- ✓ بازار فروش میکروکنترلرهای جدید

در اکتبر سال ۱۹۹۹ آرم که در فکر طراحی یک معماری جدید برای آینده بود. محصول جدید عضوی از معماری نسخه ARMv6 به نام ARM1136J-S بود. در این زمان آرم درحال طراحی برنامه های کاربردی بسیاری بود که هر یک از اجزای طراحی شده در آینده به ارزیابی در مقوله ی قیمت نیازداشت. و اینکه آرم در آینده نیز کاربرد داشته باشد یکی از مزایای آرم محسوب می شود.

به عنوان مثال "آن چیپ سیستم" بسیار پیچیده طراحی شد، پردازنده ی آرم یک پردازنده ی مرکزی داشت بعلاوه ی یک پردازشگرچندگانه ی عملیاتی و چند زیر سیستم.به طوریکه قابل حمل ،دستی با قدرت محسبه ی بالا و بازار فروش مناسب باشد.همچنین قابلیت اتصال وسیله ی خروجی جدید را نیز داشت و می توانست خواسته ای آرم را برآورده سازد و انجام دهد.

محدوده ی موردنیاز آدرس دهی ،در مرحله ی ارقام تک رقمی (DSP(و انجام تصویری برای سیستمهای قابل حمل ،با یکدیگر میکس –ایندین سیستم را همچون TCP/IPوهمزمان سازی کاراوموثر در محیط پردازشگرهای چندگانه .

از دیگر دستاوردهای آرم پاسخگویی به نیاز بازار بوده است،و هنوز آرم در پشتیبانی سودمند مدعی ست و با رقابت مفید و محاسبات کارامد (قدرت محاسبه در هر مگاوات)در دنیای صنعت بهترین است.

در این فصل عناصر سازند ه ی داخلی آرم وی ۶ که به کمک آدرس دهی بازار درخواست این محصول وارد بازار شده ایت،شامل ارتقا پشتیبانی DSP و پشتیبانی محیطهای پردازشگرهای چندگانه است.

این فصل همچنین در آغاز سعی در شناختن عملکرد بالای آرم۶ وی در اجرا دارد. و بعـلاوه تکنولـوژی آرم وی ۶،یکی از نهایی ترین تکنولوژی های تراست اوزون است.

ظهورDSP وتقویت کردن ARMV6

با توجه به پروژه ARMV6.ARMمطرح میکند چگونه اصـلاح کـردن DSPو مقـدورات پـردازش کـردن رسانه ها ی متعلق به شاخه ساختار های ARMV5که بیان شده است در بخش۳.۷. .این کار ادامـه پیـدا کرد با دقـت بوسـیله ی گـروه فنـی_مهندسـی ARM1136J کـه بودنـد در اولـین مرحلـه از توسـعه ی MICROARCHITECTURE برای تولید. SIMD (واحد دستورالعملهای اطلاعات چندگانه) یک تکنیـک معروف است که برای استفاده در حجم اطلاعات مشابه در مقادیر زیاد و به ویژه مـوثر در بسـیاری از مسایل معمولی و مهم مورداستفاده است کـه اسـتفاده ی مرسـوم آنهـا در DSPو تصـاویر و پـردازش محاسبات عددی گرافیکی .SIMD دارای ظرفیت بالایی برای کددهی است و توان مصرف کم ،ازآنجاییکه در چیدمان شمارگان دستورالعملها (و پس ازآن دسترسی بـه حافظـه ی سیسـتم) کـم نگـاه میـدارد. و بطورموثری قیمت را کاهش میدهد و قابلیت انتصاب اجزای کـامپیوترو قـالبی بـرای الگوهـای اطلاعـاتی خاص خواهد شد. هر چند به نحوه ی مطلوبی در بسیاری از تصاویر و واحدهای پردازشگر محاسـباتی کار می کند .

استفاده از آرم استاندارد که براساس فلسفه ی محاسباتی موثر با توان مصرف کـم.آرم ارتقامییابـد بـه همراه یک مسیر هوشمند و سـاده از آرم ۳۲ بیتـی مسـیر اطلاعـات بـه همـراه چهـاربرش ۸ بیتـی و دو برش۱۶بیتی .برعکس بسیاری SIMDهی موجود طوری طراحی شده اند که مسیرهای اطلاعاتی خاصـی را جهت عملکردSIMD جمع آوری کنند.این روش به SIMD ها اجازه میدهدکه طراحـی پایـه ا ی آرم را بدون صرف هیچگونه هزینه ی سخت افزاری گزافی انجام دهد.

آرم V6 وزن سبکی دارد . SIMD برای رسیدن به قیمت واقعـی ،بـدون طـی مراحـل پیچیـده ی اضـافی (ورودی مقادیر) و بنابراین قدرت بالا.همزمان با دستورالعمل جدید می تواند پردازش خروجی برخی از محاسبات عددی را به وسیله ی ارتقا پردازش ب میزان دو بار برای اصلاعات ۱۶ بیتی یا به میزان چهار بار برای اطلاعات ۸ بیتی پردازش کند.به طـور معمـول ایـن عملکـرد در دسـتورالعمل آرم بـرای تنظـیم طراحی ها بکار می رود. تمام این دستورالعمل های جدید جهت اجرا مشروط بر آنچه است که در بخـش ۲.۲.۶شرح داده شده است.

شما می توانید بیابید شرح کاملی از تمام دستور العمل های ARM v6 که در دستورالعمل جداول تنظـیم شده در ضمیمه ی A موجود است.

جدول ۱۵-۱ خلاصه ای از عملکرد8 SIMD بیتی را نشان می دهد.

نتیجه هر بایت به وسیله ی عملکردهای طراحی روی هر بایت مشابه میـان منـابع اجرایـی دسـتورالعمل شکل گرفته است.در نتیجه عملکرد۸بیتی شاید به نماش ۹-بیتی منجر گردد،که باعث می شـود سـیری از دسترسی اتفاق بیافتد.وابسته به استفاده از دستورالعمل های خاص.

علاوه بر عملکرد SIMD8بیتی ، گستر ه ی وسیع ای از عملکرد۱۶-بیتی های دو واحدی که در جدول ۱۵-۲ نشان داده شده است نیز موجود است.هر لغت ۱۶-بیتی از طراحی عملکرد بر روی هر برش های ۱۶-بیتی مشابه میان منابع آن شکل گرفته است.در نتیجه شاید به ۱۷-بیتی ها جهت ذخیره سازی احتیاج شود .و در این مورد آنها می توانند مورداستفاده قرار دهند یا به همراه گستره ای ازنشان های ۱۶-بیتی هادر ورژن های مختلفی از دستورالعمل ها اشباع شوند .

جدول۱۵-۱

جدول۱۵-۲

عملگرهای دستورالعمل SIMD به طور صحیح به همراه ثباتهای منبع یافت نمی شوند؛جهت بهبود موثردر این موقعیت ،۱۶-بیت عملکرد SIMD وجود دارد که ۱۶-بیت کلمه ی را باعملگردمنبع تعویض می کند. این دستورالعمل هااجازه می دهند که با قابلیت تعویض پذیری وسیعی در توزیع با نصف کلمات که ممکن است در برخی روش هادر حافظه برابر نیز باشد ومخصوصا زمانی که در جفت های عددی پیچیده ی ۱۶-بیتی مفید است که در منابع ۳۲-بیتی بسته بندی شده اند.مدل های مختلف این دستورالعمل ها نشانه دار، نشانه دار اشباع شده،بی نشانه ی اشباع شده بی نشانه است که در جدول ۱۵-۶نمایش داده می شود.

Xدر ساختار کد حافظه دال بر این است که دو نصفه کلمه در Rm قابل تعویض است قبل از آنکه عملیات اجراشود بنابراین عملیات شبیه حالت قبل اتفاق می افتد.

افزایش عملیاتهای SIMDبدین معنی است که در این وضعیت به روش مشابه نمایش یک سرریز یا یک نقال از هر قسمت SIMD به درون بزرگراه اطلاعاتی وجود دارد.Sprبه طور کامل در سکشن۲.۲.۵شرح داده شده است که با جمع آوری چهاربیت پرچم اضافی برای آماده سازی هرقسمت ۸-بیتی از بزرگراه اطلاعاتی اصلاح شده است. اصلاح منابع cpsr به وسیله ی GEبیت که یکی از آن موارد " بیشتر یا مساوی " پرچم برای هر قسمت از میان راه اطلاعاتی است.

سیستم های عملیاتی آماده اند تا منابع cpsrرا روی یک راه گزین قرینه ذخیره سازند .افزایش این بیت ها برcpsrها تاثیر کمی روی پایه ی معماری OS دارد.

بعلاوه برای عملکردهای اساسی محاسباتی در قسمتهای اطلاعاتی SIMD ، عملیات منحصربفرد اطلاعاتی ای برای عملیات هایی که اجازه میدهد باقیمانده ی عناصر اطلاعات به همراه بزرگراه اطلاعاتی وشکل گیری دسته جمعی اجزا مورد استفاده قرار بگیرد. انتخاب ساختار SEL می تواند آزادنه انتخاب کند هرحوزه ی ۸-بیتی را از یکی از منایع ثبات های Rn یا منابع ثباتهای دیگرRm،مشروط بر اینکه با پرچمGE همراه شود.

این دستورالعمل ها ،باهم و باعملیاتهای SIMD،می تواند به طور موثری برای اجرای الگوریتم قانون گذار هسته مفید باشد. که به طور قابل ذکری برای علایم بازگشتی درسیستم های مخابراتی مورد استفاده است. از زمانی که الگوریتم قانون گذارضروری شده است ، در آمارگیری ها بالاترین احتمال

انتخاب را داراست،ازین رو همچنین در برخی سخنرانی ها و امضاهاودست خطهای ی رسمی مورد استفاده است.هسته ی یک Viterbi یک عملیات است که معمولا ACS)add-compare-select)شناخته می شود،و در حقیقت پردازگرهای DSPزیادی قابلیت داشتن ساختارACSرا دارند. و یک جمع کننده ی parallel(SIMD) ،کاهشدهنده (که می تواند در کامپیوترمورداستفاده قرار گیرد)،و انتخاب دستورالعمل است.ARMv6می تواندبه شدت قوی اضافه کردن–مقایسه – و انتخاب را اجرا کند

این هسته روی چهارراهی در پارالل اجرا می شود وچرخه ی۴ کلی ای را روی ARM11361میدهد.
ردیفهای کدشده برای تنظیم دستورالعمل ARMv5TE هریک از دستورالعملها را به طور پشت سرهم حداقل با ۱۶ حلقه اجرا می کند،
بنابراین عملکر add-compare-selectروی ARM1136jچهار بار سریعتر است .

۱۵.۱.۲چیدمان دستورالعملها
معماری ARMv6شامل یک ست جدید از چیدمان دستورالعملها می شود،که در جدول ۱۵.۵قابل مشاهده است،که برای ساختن بسته های اطلاعاتی ۳۲–بیتی جدید از جفتهای ۱۶–بیتی که هریک دارای ارزش خاصی است.
دومین عملوند می تواند بطور انتخابی حرکت کند.قراردهی دستورالعملها به طور ویژه ای برای مقادیر جفتهای ۱۶–بیتی مفید است بنابراین شما می توانید ازدستورالعملهای پردازشگر SIMDهای ۱۶بیتی ،که قبلا شرح داده شده است استفاده کنید.

۱۵.۱.۳پشتیبانی از محاسبات پیچیده ی عددی
محاسبات عددی پیچیده بطورمعمول درپردازش سیگنالهای ارتباطاتی استفاده می شود،مخصوصا جهت اجرای نقل و انتقالات محاسبات عددی از قبیل جابه جایی سریع Fourier که در بخش هشتم شرح داده شده است.جزئیات بیشتری از اجرا در این بخش که بطور موثر جهت استفاده از ARMv4یاARMv5Eهای متعدد و پیچیده در تنظیمات دستورالعمل بررسی شده است.
به ARMv6 دستورالعملهای متعددی اضافه شده است جهت سرعت بخشیدن اجراهای پیچیده که در جدول ۱۵.۶.
هردوی این عملکردها به طور انتخابی طبق دو نیمه ی ۱۶–بیتی از منابع عملوندRs قابل جابه جایی اند البته اگر آنها را با پسوندXتعیین کنید.

در این مثال RaوRb عدد های پیچیده را به طور مشترک با بسته های ۱۶–بیتی به وسیله ی قطعات اصلیشان در قسمت تحتانی ثبات و قسمت فرضی آن در قسمت بالایی ثبات قرار می گیرد.

۱۵.۱.۴سیری دستورالعمل ها

اشباع محاسباتی برای اولینبار به وسیله ی قسمت الحاقی E پیدا شد که به طراحی ARMv5TEاضافه شده بود،که این تولیدات به عنوان ARM966Eو ARM946E معرفی شد.

ARMv6دارای مزایایی از قبیل وسایل شخصی و دستورالعمل های اشباع قابل انعطافی است که می تواند روی کلمات ۳۲-بیتی و نیمه کلمات ۱۶-بیتی عمل کند.علاوه بر این دستورالعملها که در جدول ۱۵.۷نشان داده شده است محاسبات اشباع شده ی جدیدپردازشگر SIMD که در بخش ۱۵.۱.۱بیان شده است.

۱۵.۱.۵مجموعه کامل اختلافات ساختاری

این دوساختار جدید به احتمال زیاد دارای ساختمان کاربردی ویژه به همراه طراحی AR! USAD8 وUSADA8. این ها قابل استفاده اند برای محاسبات کامل تفاوت بین ارزش ۸بیتی ها و قابل استفاده اند مخصوصا برای محاسبه ی حرکت فشرده ی فیلم های ویدئویی.مانند MPEGیاH.263، که شامل محاسبه ی عددی که تفوت های محاسباتی را اندازه گیری می کند.که مجموعه ی کاملی از تفاوتها ی ساختاری است.(می توانید برای یافتن اطلاعات بیشتر به جدول ۱۵.۸ مراجعه کنید.

۱۵.۱.۶ دستورالعمل های متعدد ۱۶ بیتی های دوگانه

ARMv5 به طور قابل ملاحظه ای برای عملکرد DSP معرفی شده اند اما ARMv6 اطلاعات بیشتری در موردش موجود است. اجرای ARM v6 (از قبیل ARM1136j) دارای ظرفیتهای های ۱۶*۱۶ دوتایی اند،که قابل مقایسه با بسیاری ازقطعات DSPهای دارای تکنولوژی بالا و خاص هستند. که در جدول ۱۵-۹ قابل مشاهده است.

۱۵.۱.۷ بیشترین و کاربردی ترین مولتی پلی های دنیا

ARMv5TE به عملکردهای محاسباتی که به طور گسترده در یک خانواده وسیعی از محاسبات DSPها که شامل کنترل و ارتباطات می شود اضافه شده اند.آنها جهت استفاده از اطلاعات ورژن Q15 طراحی شده اند.

هرچند در پردازش وسایل صوتی ۱۶-بیتی ها رایج است که ناکافی است جهت شرح کیفیت سیگنالها.به طور معمول ۳۲-بیتی ها در این موارد استفاده می شوند و ARMv6اضافه تر دارد برخی از دستورالعملهای متعدد را که روی ورژن Q31قابل اجراست ، اضافه می کنند.ک(که محاسبه ی Q-formatدر بخش ۸ با جزئیات بیان شده است) این دستورالعملهای جدید به طور لیست در جدول ۱۵.۱۰ قابل مشاهد است.

۱۵.۲ سیستم و پشتیبانی از مراحل متعدد اضافه شده به ARMv6

سیستمها کمی بیشتر پیچیده شدند.که شامل مراحل پردازش و پردازش مهندسی متعددی.
این مهندسی ها به احتمال زیاد می تواندوجوه مختلف را درباره ی حافظه تقسیم کند و حتی در endiannesses
(طبق دستور)استفاده می شود.
جهت اصلاح وضعیت ارتباطات در این سیستم ، پشتیبانی می شود به کمک ARMv6 در سیستم میکس-ایندین. که پردازش سریعی ای که استثنایی است و همگن سازی های ابتدایی .

۱۵.۲.۱ پشتیبانی از میکی-ایندینسز

به طور معمول طراحی ARM ها که یک وجه از لیتل-ایندین های حافظه همراه با مدل بزرگ-ایندین داراست که می تواند تغییر کند در راه اندازی مجدد. این مدل بزرگ- ایندین در حافظه ی سیستم به عنوان یک بزرگ -ایندین که در دستورالعمل اطلاعات تعریف شده ؛طراحی شده است.
همانگونه که در دستورالعمل این بخش ذکر شد ،ARMها در بخشهای کاملی و بسیار ی از سیستمهای پیچیده یافت می شوند در قطعاتی از تراشه ی سیستم که با میکس-ایندین در ارتباط است.و همچنین بیشتر اوقات با هر دو مدل کوچک – ایندین و بزرگ – ایندین دراطلاعات شان در نرم افزارها مرتبط اند.
ARMv6 اضافه کرده دستورالعمل جدیدی را جهت تنظیم اطلاعات ایندین برای کدهای متوالی تنظیم کرده .(که در جدول ۱۵.۱۲ می بینید). همچنین برخی از دستورالعملهای متعدد ویژه می تواند افزایش دهد تاثیر ارتباط با محیط میکس- ایندین .
endian_specifier
به این نام خوانده می شود.BE برای بزرگ-ایندین و LE برای کوچک ایندین.یک برنامه می تواند به طورمعمول استفاده شود SETEND زمانی که وجود دارد مقادیر قابل توجهی کد وجود دارد.که حمل می کنند عملکردها را به وسیله ی یک ایندین ویژه حمل می کنن. که در شکل ۱۵.۳ آمده است.

۱۵.۲.۲ پردازش استثنا

متدوال است که در سیستمهای عامل، حالت بازگشت یک استثنا یا وقفه، درون پشته ذخیره می شود.ARMv6 دستورات جدول ۱۵.۱۳ را برای بهبود وضعیت این عملیات اضافه کرده است. این وضعیت در سیستم های پیشرفته به صورت وقفه و یا زمانبند های سیستم عامل به دفعات رخ می دهد.

۱۵.۲.۳ اجزای پایهٔ همزمان سازی چند پردازشی [Multiprocessing]

هر چه طراحی سیستمهای-روی-تراشه(SOC) پیچیده تر می شوند، در عوض ARM در بسیاری از وسایل، که نیاز به واحد پردازشی زیادی دارند یافت می شوند.

ساختارARM همواره دارای دستورالعمل SWP برای پیاده سازی سمافورها [Semaphor] جهت پایداری این سیستمها بوده است. هرچه SoC ها پیچیده تر می شوند ، هرچند مطمئناً وجوه دیگرSWP باعث خواهد شد مانند یک تنگراه که به عنوان مثال ذکر شد، عمل کند. به یاد بیاورید که SWP به عنوان یک عنصر "انسداد کننده"، در ابتدای کار گذرگاه خارجی پردازنده را قفل می کند و بیشتر پهنای داده اش را صرف پی بردن به آزادی منابعش [Reource] می کند. از این نظر دستور SWP بدبینانه عمل می کند زیرا که هیچگونه محاسباتی نمی تواند تا وقتی که این دستور با یک منبع آزاد شده باز گردد، ادامه یابد.

دستورالعملهای جدید LDREX و STREX (بارگیری و ذخیره سازی انحصاری) جهت حل این مشکل به معماری ARMv6 اضافه شدند. این دستورالعملها در جدول ۱۵.۱۴ آمده اند. پیاده سازی این دستورها بسیار سرراست است و اجرای آن با داشتن یک سیستم مونیتور در سیستم حافظه مقدور است. LDREX یک مقدار از حافظه را به درون یک ثبات بارگیری می کند. این دستور به طور خوش بینانه ای فرض می کند که هنگام اجرای آن، هیچ وسیله دیگری محتویات حافظه را دستکاری نمی کند. STREX یک مقدار را به درون حافظه باز می گرداند و برای نشان دادن اینکه "بین دستورالعمل LDREX اولیه و این دستور ذخیره سازی مقدار حافظه تغییری داشته است یا خیر" نشانه ای دارد.

به هرحال در ابتدا بدبین"optimistic" هستند که شما می توانید پردازش داده هایی را که بارگذاری کرده اید با LDREX حتی با وجود چندین وسیله ی خارجی که ممکن است مقادیر را اصلاح کند، ادامه دهید.تنهااگر یک اصلاح حقیقی صورت پذیرد در مکان های خارجی است که مقدار قرار می گیرد و دوباره بارگذاری می شود.

بزرگترین تفاوت سیستم اینست که پردازشگر چندان طولانی منتظر نمی ماند تا مسیر سیستم برای یک مخابره آزاد شود، و بنابراین بسیاری از مسیر پهنای باند سیستم را که جهت پردازش یا پردازشگرها؛ در دسترس است، ترک می نماید.

۱۵.۳ کاربردواجرای ARMv6

توسعه ی ARMاز ARM1136Jدر دسامبر۲۰۰۲کامل شد ،و در این متن محصولات مصرف کنندگان طراحی شد با این هسته. ARM1136Jلوله ای ،تا الان ، پیچیده ترین ARM از نظر کارآیی واجر است.ه در شکل ۱۵.۴ نشان داده شده است.که ۸ مرحله ی لوله ای ،با لوله های پارالل مجزا جهت بارگذاری/ذخیره سازی و افزایش / انباشتن دارد.

پاراال بارگذاری /ذخیره تا الان (LSU) با قابلیت hit-under-miss مجازند دستورالعملها را برای توزیع و اجرا ، بارگذاری کنند و ذخیره نمایند تا هنگامی که بارگذاری و ذخیره ساری ، در سیستم با حافظه ی پایین تر نیز کامل شود. به وسیله ی جداسازی جفتها(Decoupling) اجرای لوله جهت تکمیل

بارگذاری و ذخیره سازی ، هسته می تواند،به طور قابل توجهی سطوح کارآیی اضافی اش تقویت شود نسبت به زمانی که حافظه ی سیستم معمولا در بسیاری از مواقع آهسته تر از سرعت هسته است.
hit-under-miss ادامه می یابد با همین ی جداسازی جفتها(Decoupling) خارج ازحدواصل سیستم حافظه ی L1-L2 ،بنابراین یک L1 که منبعی برای ذخیره سازی پنهانی و موقت است می تواند اتفاق افتد و L2 بعنوان یک مبادله گر می تواند کامل شود تا زمانی که L1های دیگر هنوز ، به طور تصادفی به هم نزدیک می شوند.
از دیگر تغییرات بزرگ در microarchitecture جابه جایی آنهاست که با واقع روی نوک آنها برچسب زده شده ی پنهانی با برچسب فیزیکی پنهانی .
در متن ذکر شده است ، ARM واقعا قابل استفاده است به طور برچسبهای پنهانی که MMU بینشان پوشیده هست و بیرون از حافظه ی L2 سیستم است.
نوشته شد که ARMv6اینگونه تغییر کرد یه طوریکه MMU امروزه بین هسته و L1پنهانی قرار دارد ، بنابراین آدرس تمام حافظه های پنهانی به طور فیزیکی قابل دسترسی اند(وقابل حمل و نقل و جابجایی) .
یکی از بزرگترین محاسن این روش اینست که زمانی که ARM درحال اجرای دستورالعملهای سنگین سیستم است ، به طورقابل ملاحظه ای ، خالی کردن قسمتی از حافظه و محتویات ان را که به طور پنهانی انجام می شد را با همان سوئیچ قبلی ، کاهش میدهد .این کاهش دادن حافظه ، همچنین خواهد توانست توان مصرف را نیز کاهش دهد در نهایت سیستم در حالت فلاشینگ پنهانی ،سریعا حافظه ی خارجی بیشتری قابل دسترسی را می رساند.
در برخی موارد انتظار می رود با تغییر این ساختار بتواند تا ۲۰٪ کارآیی ، بهبود یابد.

۱۵.۴تکنیک های فراتر ARMv6 در آینده

در سال ۲۰۰۳،Arm اعلام کرد که تکنولوژی آینده شامل TrustZone و Thumb-2 دارد.در حالیکه این تکنولوژی ها بسیار جدید هستند ،در این متن ، آنها شامل هسته های ریزپردازنده های مدرنی هستند.در بخش بعد محاسن آنها ذکر شده است.

15.4.1 TrustZone

TrustZone یک صفحه نشانه ی وسیع از طراحی های پیچیده با انجام مخفیانه که ممکن است به وسیله ی تولیدات مصرف کننده اجراشود ، مانند موبایل و ممکن است در آینده جهت اجرا ی دانلود موسیقی و ویدئو استفاده شود. این تکنولوژی ابتدا در اکتبر ۲۰۰۳ ،زمانی که ARM1176JZ-S،ARM را اگهی داده بود وارد شد .

در ابتدا این ایده از عملیاتهای سیستم(حتی وقتی در یک وسیله جاسازی شده است) اینست که بسیار پیچیده است و اقدامات مختلف آن بسیار سخت است و بررسی صحت آن در نرم افزار .
Arm این مشکل را با اضافه کردن یک عملیات جدید به نام "states" حل کرده است . که معماری اش تنها یک نرم افزار ساده و قابل رسیدگی دارد که در هسته ی اصلی اجرا خواهد شد. و این خدماتی را جهت عملیاتهای بزرگ سیستم را تامین می کند.
سپس هسته ی این ریزپردازنده ها طبق یک قانون مشخص در کنترل دستگاههای جانبی که ممکن است درمنبع "state" قابل دسترس باشد هرچند جدیدبسیاری به طور واحد به طور مشترکی در مسیرها صادر می کنند.این قسمت از سیستم در شکل۱۵.۵ نشان داده شده است.

TrustZone

در وسایلی که قابل حمل اند و مقادیر ی از اطلاعات را دانلود می کنند بیشترین کاربرد را دارد .از قبیل : تلفن همراه و دیگر وسایل قابل جابجایی اند از جمله شبکه های قابل اتصال به زمین.
جزئیات این طراحی ها قابل نوشتن نیست در این متن.

THUMB-2 15.4.2

تامب -۲ یک طراحی وسیع است که جهت افزایش کارآیی درگنجایش High code ها در نظر گرفته شده است.این تکنولوژی اجازه می دهد تا ساختار ۳۲-بیتی های ARM-like را با۱۶ ساختار-بیتی های تامب ، در هم آمیزد.با ترکیب این توانایی ها ،شما بایست شماره ی گنجایش تامب را داشته باشید با ایجاد کارایی بیشتر جهت دسترسی به ساختار۳۲-بیتی ها.
تامب-۲ در اکتبر سال۲۰۰۳ وارد بازار شد و در پردازشگر ARM1156T2 اجرا خواهد شد.جزئیات این معماری قابل انتشار در این متن نیست.

۱۵.۵ خلاصه

معماری ARM نه تنها ثابت و دائمی نیست ،بلکه در حال توسعه است وبرای نیازهای کاربردی ساخته شده در وسایل مدرن ساخته شده توسط مصرف کنندگان ، رشد خواهد کرد و مدام تغییر می یابد.
هرچند معماری ARMv5TE ، در اضافه کردن پشتیبانی DSPهای بیشتراز ARM، بسیار موفق بود اما معماری ARMv6 توسعه یافت در پشتبانی DSPها و همچنین با اضافه کردن پشتیبانی برای ریزپردازنده های بزرگ سیستم . که در جدول ۱۵.۱۵ نشان داده شده است که چطور این تکنولوژی های جدید عمل می کنند.
ARMهنوز یک روش برای محاسن-کدهای گنجایش را متمرکز نکرده است. اخیرا با معرفی وسیع تامب-۲ ،تامب ها بسیار محبوبیت یافته اند.

جدیدترین توجهات روی امنیت با TrustZone ها اتفاق می افتد که ARM یک بارگـذاری در لبـه ی ایـن ناحیه می کند. انتظار می رود که در سالهای آیند ه ابداعات جدیدی به وجود آید!

بخش ۷

ضمایم

ضمیمه الف

مجموعه دستورات ARM و Thumb

انتظار می رود که در سالهای آیند ه ابداعات جدیدی به وجود آید!

ضمیمه ب

نحوه رمزگذاری ARM و Thumb

در این ضمیمه نحوه رمزگذاری دستورات ۳۲ بیتی ARM و ۱۶ بیتی Thumb را با هم می‌بینیم. همچنین در مورد بیتهای مختلف cpsr و spsr صحبت خواهیم کرد.

ب.۱ رمزگذاری دستورات ARM

جدول ب.۱ نحوه رمزگذاری دستورات ۳۲ بیتی ARM مربوط به معماری نسخه ARMv6 را به طور خلاصه فهرست کرده است. اگر دستورات را به طور دستی رمزگشایی می‌کنید، این جدول برایتان مفید خواهد بود. جدول مذکور برای راحتی در امر رمزگشایی، توسعه داده شده است. تمامی بیتهایی که در نقشه نشان داده نشده‌اند، یا قابل پیش بینی نیستند و یا برای معماری ARMv6 تعریف نشده‌اند.

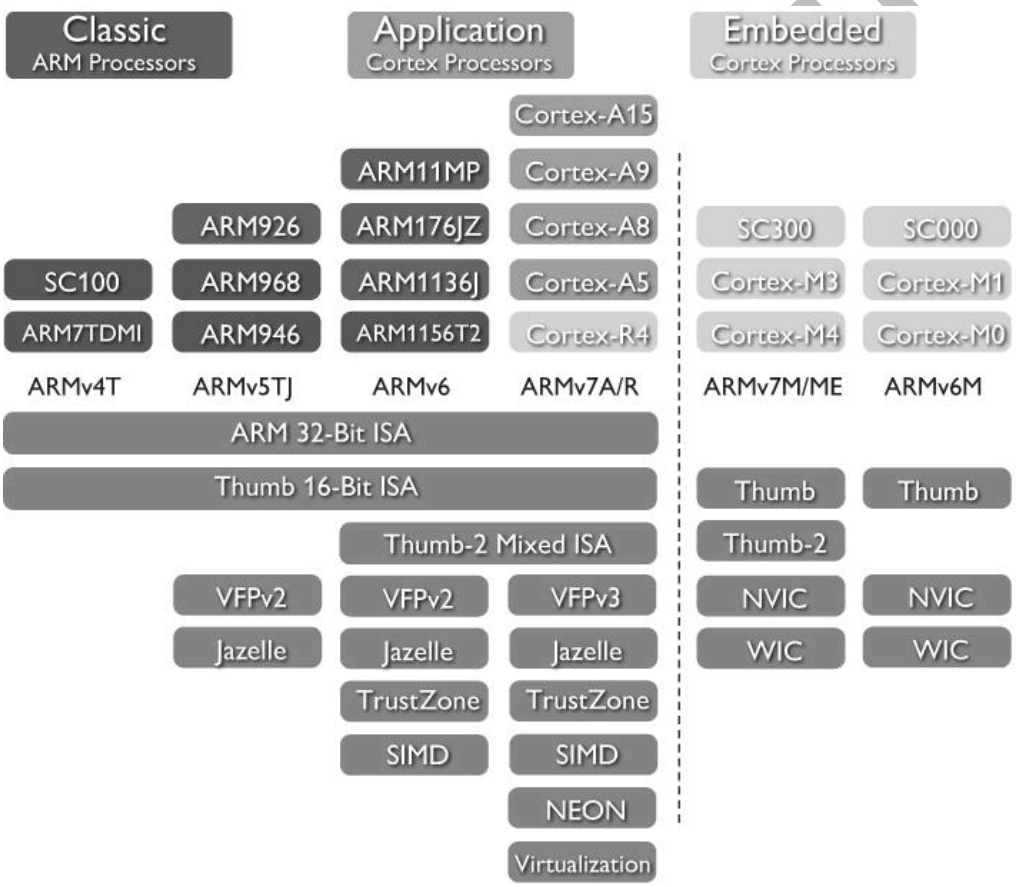
برای استفاده صحیح این جدول، فرآیند رمزگشایی زیر را دنبال کنید:

- به رقم Hex سمت چپ دستور (بیتهای ۲۸ تا ۳۱) نگاه کنید. اگر مقدارش برابر 0xF باشد، به انتهای جدول ب.۱ پرش کنید. در غیر اینصورت رقم Hex بالایی، نمایشگر یک شرط (یعنی cond) می‌باشد. cond را با استفاده از جدول ب.۲ رمزگشایی کنید.
- در مورد بیتهای ۲۴ تا ۲۷ (سایه‌دار) به جدول ب.۱ مراجعه کنید.

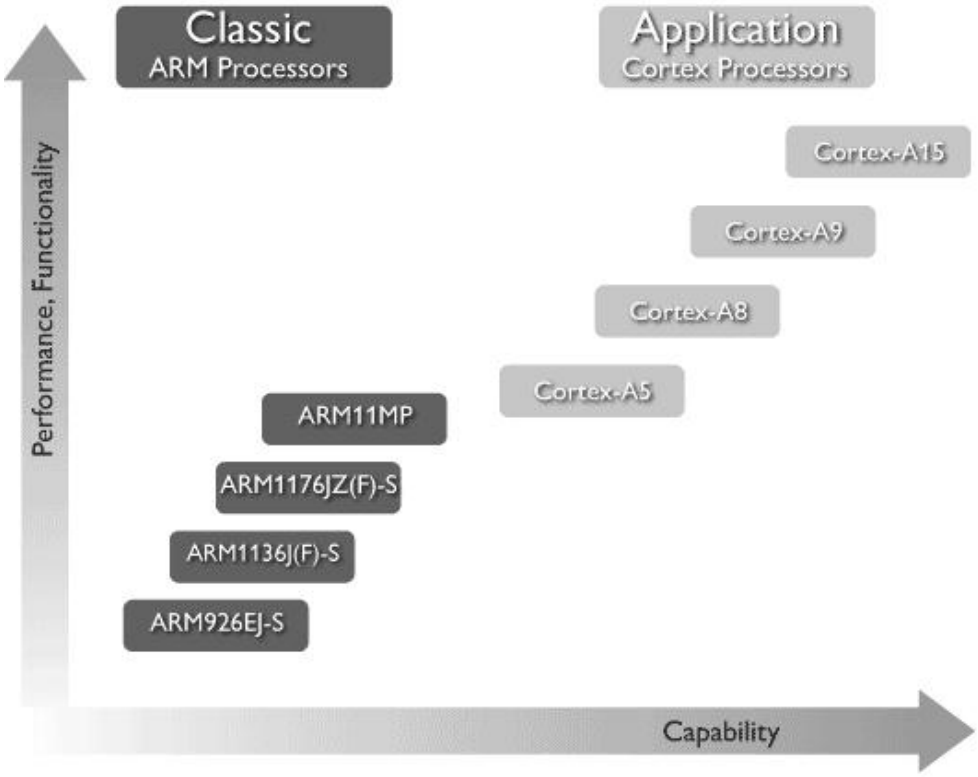
ضمیمه پ

خانواده‌های پردازنده ARM

خانواده‌های مختلفی از پردازنده ARM برای کاربردهای مختلف وجود دارند. شرکت ARM پردازنده هایش را در گروه‌های Classic، Application و Embedded Cortex تقسیم بندی کرده است. در شکل زیر نمایی کامل از انواع پردازنده‌ها را با نسخ معماری متناظرشان و قابلیت‌های ارایه شده‌شان میبینیم.



شکل پ.۱ –

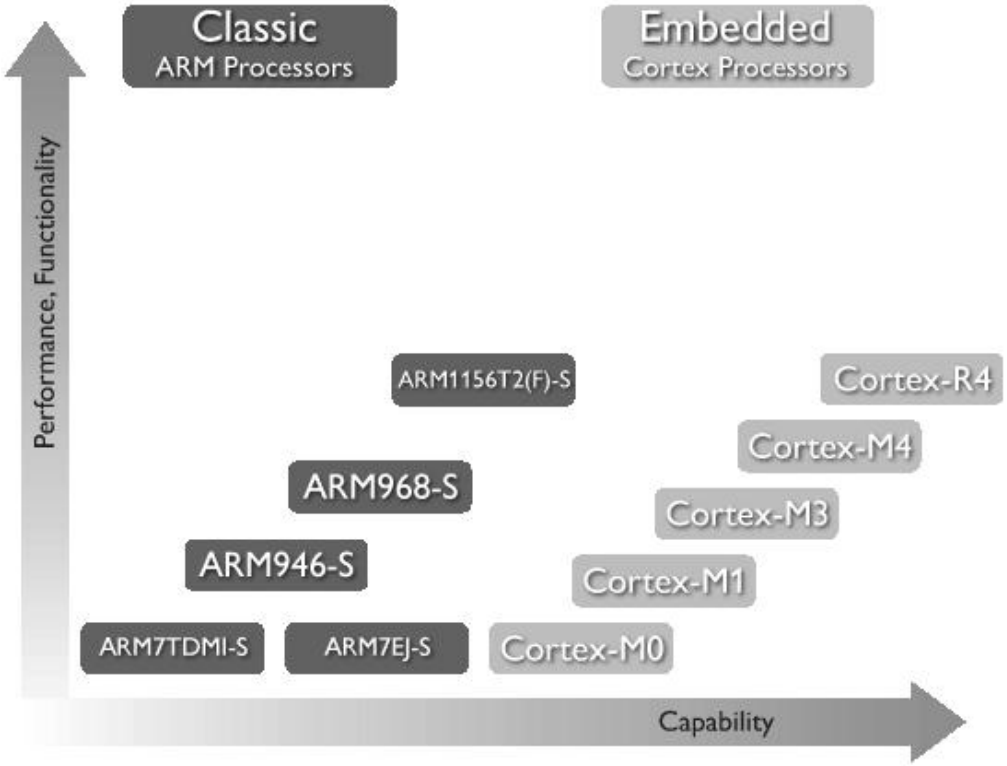


شکل پ.۱ – شکل پ.۲

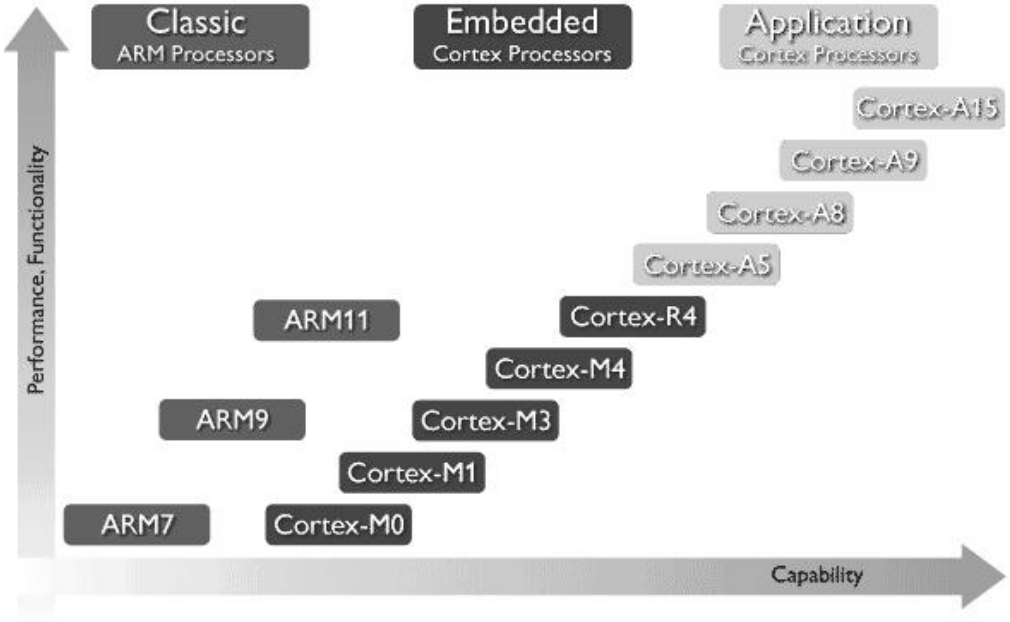
هر یک از این گروههای پردازنده برای کاربرد خاصی طراحی شده‌اند. در نمودار شکل پ.۲ دو گروه Application و Classic را از لحاظ کارایی و قابلیت ها با یکدیگر مقایسه می‌کنیم.

همانطور که مشهود است، پردازنده‌های Cortex Application از لحاظ کارایی از سری پردازنده‌های ARM7، ARM9 و حتی از انواع مختلف ARM11 بالاتر هستند.

در نمودار زیر نیز چند نمونه از هسته‌های ARM7TDMI، ARM946، ARM968 و ARM1156 از گروه Classic را با پردازنده‌های Cortex-M و Cortex-R خانواده Embedded از لحاظ کارایی و قابلیت ها با یکدیگر مقایسه می‌کنیم. پردازنده‌های Cortex-R برای کاربردهای خاص بلادرنگ طراحی شده‌اند. همین مساله دلیل موفقیت این گروه نسبت به پردازنده‌های ARM11 است.


Classic
ARM Processors
Embedded
Cortex Processors
Performance, Functionality
ARM1156T2(F)-S
Cortex-R4
Cortex-M4
ARM968-S
Cortex-M3
ARM946-S
Cortex-M1
ARM7TDMI-S
ARM7EJ-S
Cortex-M0
Capability

شکل پ.۱ –


Classic
ARM Processors
Embedded
Cortex Processors
Application
Cortex Processors
Cortex-A15
Cortex-A9
Cortex-A8
Cortex-A5
Performance, Functionality
ARM11
Cortex-R4
Cortex-M4
ARM9
Cortex-M3
Cortex-M1
ARM7
Cortex-M0
Capability

شکل پ.۱ –

در شکل آخر نیز هر سه خانواده فوق را با تمامی گروه‌ها در کنار هـم مـی‌بینـیم. در ایـن شـکل برتـری گروه Cortex-A را نسبت به پردازنده‌های دیگر به‌وضوح می‌بینیم.

جدول زیر خانواده‌های مختلف پردازنده‌های ARM به همراه معماری متناظر، مشخصه، حافظه نهان و فرکانس کاری را در کنار هم نشان می‌دهند.

خانواده ARM	معماری ARM	هسته ARM	مشخصه	Cache (I/D), MMU	Typical MIPS @ MHz
ARM1	ARMv1	ARM1	First implementation	None	
ARM2	ARMv2	ARM2	ARMv2 added the MUL (multiply) instruction	None	4 MIPS @ 8 MHz 0.33 DMIPS/ MHz
	ARMv2a	ARM250	Integrated MEMC (MMU), Graphics and IO processor. ARMv2a added the SWP and SWPB (swap) instructions.	None, MEMC1a	7 MIPS @ 12 MHz
ARM3	ARMv2a	ARM3	First integrated memory cache.	4K unified	12 MIPS @ 25 MHz 0.50 DMIPS/ MHz
ARM6	ARMv3	ARM60	ARMv3 first to support 32-bit memory address space (previously 26-bit)	None	10 MIPS @ 12 MHz
		ARM600	As ARM60, cache and coprocessor bus (for FPA10	4K unified	28 MIPS @ 33 MHz

خانواده ARM	معماری ARM	هسته ARM	مشخصه	Cache (I/D), MMU	Typical MIPS @ MHz
			floating-point unit).		
		ARM610	As ARM60, cache, no coprocessor bus.	4K unified	17 MIPS @ 20 MHz 0.65 DMIPS/ MHz
ARM7	ARMv3	ARM700		8 KB unified	40 MHz
		ARM710	As ARM700, no coprocessor bus.	8 KB unified	40 MHz
		ARM710a	As ARM710	8 KB unified	40 MHz 0.68 DMIPS/ MHz
ARM7TDMI	ARMv4T	ARM7TDMI(-S)	3-stage pipeline, Thumb	none	15 MIPS @ 16.8 MHz 63 DMIPS @ 70 MHz
		ARM710T	As ARM7TDMI, cache	8 KB unified, MMU	36 MIPS @ 40 MHz
		ARM720T	As ARM7TDMI, cache	8 KB unified, MMU with Fast Context Switch Extensio	60 MIPS @ 59.8 MHz

خانواده ARM	معماری ARM	هسته ARM	مشخصه	Cache (I/D), MMU	Typical MIPS @ MHz
				n	
		ARM740T	As ARM7TDMI, cache	MPU	
ARM7EJ	ARMv5TEJ	ARM7EJ-S	5-stage pipeline, Thumb, Jazelle DBX, Enhanced DSP instructions	none	
ARM8	ARMv4	ARM810	5-stage pipeline, static branch prediction, double-bandwidth memory	8 KB unified, MMU	84 MIPS @ 72 MHz 1.16 DMIPS/ MHz
StrongARM	ARMv4	SA-1	5-stage pipeline	16 KB/8–16 KB, MMU	203–206 MHz 1.0 DMIPS/ MHz
ARM9TDMI	ARMv4T	ARM9TDMI	5-stage pipeline, Thumb	none	
		ARM920T	As ARM9TDMI, cache	16 KB/16 KB, MMU with FCSE (Fast Context Switch Extensio	200 MIPS @ 180 MHz

خانواده ARM	معماری ARM	هسته ARM	مشخصه	Cache (I/D), MMU	Typical MIPS @ MHz
				n)	
		ARM922T	As ARM9TDMI, caches	8 KB/8 KB, MMU	
		ARM940T	As ARM9TDMI, caches	4 KB/4 KB, MPU	
ARM9E	ARMv5TE	ARM946E-S	Thumb, Enhanced DSP instructions, caches	variable, tightly coupled memories, MPU	
		ARM966E-S	Thumb, Enhanced DSP instructions	no cache, TCMs	
		ARM968E-S	As ARM966E-S	no cache, TCMs	
	ARMv5TEJ	ARM926EJ-S	Thumb, Jazelle DBX, Enhanced DSP instructions	variable, TCMs, MMU	220 MIPS @ 200 MHz,
	ARMv5TE	ARM996HS	Clockless processor, as ARM966E-S	no caches, TCMs, MPU	
ARM10E	ARMv5TE	ARM1020E	6-stage pipeline, Thumb, Enhanced DSP instructions, (VFP)	32 KB/32 KB, MMU	

خانواده ARM	معماری ARM	هسته ARM	مشخصه	Cache (I/D), MMU	Typical MIPS @ MHz
		ARM1022E	As ARM1020E	16 KB/16 KB, MMU	
	ARMv5TEJ	ARM1026EJ-S	Thumb, Jazelle DBX, Enhanced DSP instructions, (VFP)	variable, MMU or MPU	
XScale	ARMv5TE	XScale	7-stage pipeline, Thumb, Enhanced DSP instructions	32 KB/32 KB, MMU	133-400 MHz
		Bulverde	Wireless MMX, Wireless SpeedStep added	32 KB/32 KB, MMU	312-624 MHz
		Monahans	Wireless MMX2 added	32 KB/32 KB (L1), optional L2 cache up to 512 KB, MMU	up to 1.25 GHz
ARM11	ARMv6	ARM1136J(F)-S	8-stage pipeline, SIMD, Thumb, Jazelle DBX, (VFP), Enhanced DSP instructions	variable, MMU	740 @ 532–665 MHz (i.MX31 SoC), 400–528 MHz

خانواده ARM	معماری ARM	هسته ARM	مشخصه	Cache (I/D), MMU	Typical MIPS @ MHz
	ARMv6T2	ARM1156T2(F)-S	9-stage pipeline, SIMD, Thumb-2, (VFP), Enhanced DSP instructions	variable, MPU	
	ARMv6ZK	ARM1176JZ(F)-S	As ARM1136EJ(F)-S	variable, MMU+TrustZone	965 DMIPS @ 772MHz, up to 2600 DMIPS with four processors
	ARMv6K	ARM11 MPCore	As ARM1136EJ(F)-S, 1–4 core SMP	variable, MMU	
Cortex-A	ARMv7-A	Cortex-A5	VFP, NEON, Jazelle RCT, Thumb/Thumb-2, 1-4 cores	variable (L1+L2), MMU+TrustZone	1.57 DMIPS / MHz per core
		Cortex-A8	VFP, NEON, Jazelle RCT, Thumb-2, 13-stage superscalar pipeline	variable (L1+L2), MMU+TrustZone	up to 2000 (2.0 DMIPS/MHz in speed from 600 MHz to greater than 1 GHz)
		Cortex-A9 MPCore	Application profile, VFPv3 FPU, NEON, Thumb-2, Jazelle RCT/DBX, out-of-order speculative	32 KB/32 KB L1, up to 4 MB L2, MMU+Tr	2.5 DMIPS/MHz per core, 10 000 DMIPS @ 2 GHz on

خانواده ARM	معماری ARM	هسته ARM	مشخصه	Cache (I/D), MMU	Typical MIPS @ MHz
			issue superscalar, 1–4 core SMP	ustZone	Performance Optimized TSMC 40G (dual core)
		Cortex-A15 MPCore	Application profile, VFPv4 FPU, NEON, Thumb-2, Jazelle RCT/DBX, out-of-order speculative issue superscalar, Large Physical Address Extensions (LPAE), Hardware virtualization, 1-4 SMP cores	32 KB/32 KB L1, up to 4 MB L2, MMU+TrustZone	
Cortex-R	ARMv7-R	Cortex-R4(F)	Real-time profile, Thumb-2, (FPU)	variable cache, MPU optional	600 DMIPS @ 475 MHz
Cortex-M	ARMv6-M	Cortex-M0	Microcontroller profile, Thumb-2 subset (16-bit Thumb instructions & BL, MRS, MSR, ISB, DSB, and DMB). Hardware multiply instruction optional	No cache.	0.9 DMIPS/MHz

خانواده ARM	معماری ARM	هسته ARM	مشخصه	Cache (I/D), MMU	Typical MIPS @ MHz
		Cortex-M1	FPGA targeted, Microcontroller profile, Thumb-2 subset (16-bit Thumb instructions & BL, MRS, MSR, ISB, DSB, and DMB).	None, tightly coupled memory optional.	Up to 136 DMIPS @ 170 MHz (0.8 DMIPS/MHz, MHz achievable FPGA-dependent)
	ARMv7-M	Cortex-M3	Microcontroller profile, Thumb-2 only. Hardware divide instruction.	no cache, MPU optional.	125 DMIPS @ 100 MHz
	ARMv7-ME	Cortex-M4	Microcontroller profile, both Thumb and Thumb-2, FPU. Hardware MAC, SIMD and divide instructions.	MPU optional.	1.25 DMIPS/MHz

کاربردهای نمونه هسته های مختلف ARM نیز در جدول زیر گردآوری شده‌اند. در این جدول، تراشـه-های ساخته شده توسط شرکت‌های مختلف و دستگاه‌های تجاری گوناگونی کـه ایـن تراشـه‌هـا در آنهـا استفاده شده‌اند را با هم می‌بینیم.

هسته ARM	تراشه ARM	محصولات
ARM1	ARM1	ARM Evaluation System second processor for BBC Micro
ARM2	ARM2	Acorn Archimedes, Chessmachine
ARM250	ARM250	Acorn Archimedes
ARM3	ARM3	Acorn Archimedes
ARM60	ARM60	3DO Interactive Multiplayer, Zarlink GPS Receiver
ARM610	ARM610	Acorn Risc PC 600, Apple Newton 100 series
ARM700	ARM700	Acorn Risc PC prototype CPU card
ARM710	ARM710	Acorn Risc PC 700
ARM710a	ARM7100, ARM 7500 and ARM7500FE	Acorn Risc PC 700, Apple eMate 300, Psion Series 5 (ARM7100), Acorn A7000 (ARM7500), Acorn A7000+ (ARM7500FE), Network Computer (ARM7500FE)
ARM7TDMI(-S)	Atmel AT91SAM7, NXP Semiconductors LPC2000 and LH754xx, Actel CoreMP7	Game Boy Advance, Nintendo DS, Apple iPod, Lego NXT, Juice Box
ARM710T		Psion Series 5mx, Psion Revo/Revo Plus/Diamond Mako
ARM720T	NXP Semiconductors LH7952x	Zipit Wireless Messenger
StrongARM	Digital SA-110, SA-1100, SA-1110	SA-110 Apple Newton 2x00 series, Acorn Risc PC, Rebel/Corel

هسته ARM	تراشه ARM	محصولات
		Netwinder, Chalice CATS SA-1100 Psion netBook SA-1110 LART (computer), Intel Assabet, Ipaq H36x0, Balloon2, Zaurus SL-5x00, HP Jornada 7xx, Jornada 560 series, Palm Zire 31
ARM810		Acorn Risc PC prototype CPU card
ARM920T	Atmel AT91RM9200, AT91SAM9, Cirrus Logic EP9302, EP9307, EP9312, EP9315, Samsung S3C2442 and S3C2410	Armadillo, GP32, GP2X (first core), Tapwave Zodiac (Motorola i.MX1), Hewlett-Packard HP-49/50 Calculators, Sun SPOT, HTC TyTN, FIC Neo FreeRunner), TomTom navigation devices
ARM922T	NXP Semiconductors LH7A40x	
ARM940T		GP2X (second core), Meizu M6 Mini Player
ARM926EJ-S	Texas Instruments OMAP1710, OMAP1610, OMAP1611, OMAP1612, OMAP-L137, OMAP-L138; Qualcomm MSM6100, MSM6125, MSM6225, MSM6245, MSM6250, MSM6255A, MSM6260, MSM6275, MSM6280, MSM6300, MSM6500, MSM6800; Freescale i.MX21, i.MX27, Atmel AT91SAM9, NXP Semiconductors, Samsung S3C2412 LPC30xx, NEC C10046F5-211-PN2-A SoC – undocumented core in the ATi Hollywood graphics chip used in the Wii, Telechips TCC7801, TCC7901, ZiiLABS ZMS-05, Rockchip RK2806	Mobile phones: Sony Ericsson (K, W series); Siemens and Benq (x65 series and newer); LG Arena; , GPH Wiz, Squeezebox Duet Controller (Samsung S3C2412). Squeezebox Radio; Buffalo TeraStation Live (NAS); Drobo FS (NAS); Western Digital MyBook I World Edition; Western Digital MyBook II World Edition; Seagate FreeAgent

هسته ARM	تراشه ARM	محصولات
	and RK2808, NeoMagic MiMagic Family MM6, MM6+, MM8, MTV.	DockStar STDSD10G-RK; Seagate FreeAgent GoFlex Home; Chumby Classic
ARM946E-S		Nintendo DS, Nokia N-Gage, Canon PowerShot A470, Canon EOS 5D Mark II, Conexant 802.11 chips, Samsung S5L2010
ARM966E-S	ST Micro STR91xF	
ARM968E-S	NXP Semiconductors LPC29xx	
ARM1026EJ-S	Conexant so4610 and so4615 ADSL SoC	
XScale	Intel 80200, 80219, PXA210, PXA250, PXA255, PXA263, PXA26x, PXA27x, PXA3xx, PXA900, IXC1100, IXP42x	80219 Thecus N2100 IOP321 Iyonix PXA210/PXA250 Zaurus SL-5600, iPAQ H3900, Sony CLIÉ NX60, NX70V, NZ90 PXA255 Gumstix basix & connex, Palm Tungsten E2, Zaurus SL-C860, Mentor Ranger & Stryder, iRex ILiad PXA263 Sony CLIÉ NX73V, NX80V PXA26x Palm Tungsten T3 PXA27x Gumstix verdex, "Trizeps-Modules", "eSOM270-Module" PXA270 COM, HTC

هسته ARM	تراشه ARM	محصولات
		Universal, HP hx4700, Zaurus SL-C1000, 3000, 3100, 3200, Dell Axim x30, x50, and x51 series, Motorola Q, Balloon3, Trolltech Greenphone, Palm TX, Motorola Ezx Platform A728, A780, A910, A1200, E680, E680i, E680g, E690, E895, Rokr E2, Rokr E6, Fujitsu Siemens LOOX N560, Toshiba Portégé G500, Trēo 650-755p, Zipit Z2, HP iPaq 614c Business Navigator, I-mate PDA2 PXA3XX Samsung Omnia PXA900 Blackberry 8700, Blackberry Pearl (8100) IXP42x NSLU2
ARM1136J(F)-S	Texas Instruments OMAP2420, Qualcomm MSM7200, MSM7201A, MSM7227, Freescale i.MX31 and MXC300-30	OMAP2420 Nokia E90, Nokia N93, Nokia N95, Nokia N82, Zune, BUGbase, Nokia N800, Nokia N810 MSM7200

هسته ARM	تراشه ARM	محصولات
		Eten Glofiish, HTC TyTN II, HTC Nike Freescale i.MX31 original Zune 30gb, Toshiba Gigabeat S and Kindle DX Freescale MXC300-30 Nokia E63, Nokia E71, Nokia 5800, Nokia E51, Nokia 6700 Classic, Nokia 6120 Classic, Nokia 6210 Navigator, Nokia 6220 Classic, Nokia 6290, Nokia 6710 Navigator, Nokia 6720 Classic, Nokia E75, Nokia N97, Nokia N81 Qualcomm MSM7201A HTC Dream, HTC Magic, Motorola i1, Motorola Z6, HTC Hero, Samsung SGH-i627 (Propel Pro), Sony Ericsson Xperia X10 Mini Pro Qualcomm MSM7227 ZTE Link
ARM1176JZ(F)-S	Conexant CX2427X, Nvidia GoForce 6100; Telechips TCC9101, TCC9201, TCC8900, Fujitsu MB86H60, Samsung S3C6410, S3C6430, Qualcomm MSM7627, Infineon X-GOLD 213	Apple iPhone (original and 3G), Apple iPod touch (1st and 2nd Generation), Motorola RIZR Z8, Motorola RIZR Z10, Nintendo 3DS S3C6410

هسته ARM	تراشه ARM	محصولات
		Samsung Omnia II, Samsung Moment, SmartQ 5, Tablet PC Qualcomm MSM7627 Palm Pixi and Motorola Calgary/Devour
ARM11 MPCore	Nvidia APX 2500	
Cortex-A8	Texas Instruments OMAP3xxx series, FreeScale i.MX51-SOC, Apple A4, ZiiLABS ZMS-08, Qualcomm Snapdragon QSD8x50(A)/MSM7x30/MSM8255	SBM7000, Oregon State University OSWALD, Gumstix Overo Earth, Pandora, Apple iPhone 3GS, Apple iPod touch (3rd and 4th Generation), Apple iPad (A4), Apple iPhone 4 (A4), Archos 5, BeagleBoard, Motorola Droid, Motorola Droid X, Motorola Droid 2, Motorola Droid R2D2 Edition, Palm Pre, Samsung Omnia HD, Samsung Wave S8500, Samsung i9000 Galaxy S, Sony Ericsson Satio, Touch Book, Nokia N900, Meizu M9, Google Nexus S, Sharp PC-Z1 "Netwalker".
Cortex-A9	Texas Instruments OMAP4430/4440, ST-Ericsson U8500 / U5500, Nvidia Tegra2, Qualcomm Snapdragon QSD8672/MSM8260/MSM8660, Samsung Orion, STMicroelectronics SPEAr1310, Xilinx Extensible Processing Platform, Trident PNX847x/8x/9x STB SoC	
Cortex-A15	Qualcomm Snapdragon	

هسته ARM	تراشه ARM	محصولات
	MSM8270/MSM8960, Texas Instruments OMAP5, Samsung, ST Ericsson	
Cortex-R4(F)	Broadcom, Texas Instruments TMS570	
Cortex-M0	NXP Semiconductors LPC11xx, Triad Semiconductor, Melfas, Chungbuk Technopark, Nuvoton, austriamicrosystems, Rohm	
Cortex-M1	Actel ProASIC3, ProASIC3L, IGLOO and Fusion PSC devices, Altera Cyclone III, other FPGA products are also supported e.g. Synplicity	
Cortex-M3	Texas Instruments Stellaris, ST Microelectronics STM32, NXP Semiconductors LPC17xx, Toshiba TMPM330, Ember EM3xx, Atmel AT91SAM3, Europe Technologies EasyBCU, Energy Micro EFM32, Actel SmartFusion, mbed microcontroller	
Cortex-M4	Freescale Kinetis, NXP Semiconductors LCP4300,STMicro	

ضمیمه ت

زمان‌بندی اجرای دستورات

در این ضمیمه زمان‌بندی اجرای دستورات در چندین خانواده رایج ARM را می‌بینیم. زمان اجرای دستورات بین نسخ مختلف یک خانواده متغیر بوده و همچنین وقفه‌ها سرعت حافظه و... نیز بر روی سرعت مذکور تاثیر میگذارند. اعداد و ارقام ارایه شده برای شما باید نقش راهنما را داشته باشند. برای دقت بیشتر در محاسبه زمان اجرا، باید از شبیه سازی سخت‌افزاری استفاده کنید و برای اطلاع از به‌روزترین زمان‌بندیها، به برگه‌های داده سازندگان رجوع کنید.

زمان‌بندی اجرای دستورات در ARM7TDMI

هسته پردازنده ARM7TDMI بر اساس یک خطِ لوله ۳- طبقه‌ای با تنها یک مرحله‌ی اجرایی ساخته شده است. تعداد سیکلهای اجرایی عموما به دستورات قبل یا بعد بستگی ندارند.

مدار ضرب‌کننده شامل یک آرایه‌ی ۳۲ بیت در ۸ بیت است. تعداد سیکلهای تکرار M بستگی به مقدار ثبات Rs دارد.

M	*Rs* range (use the first applicable range)	Rs bitmap s = sign bit x = wildcard-bit
1	$-2^8 \le x < 2^8$	ssssssss ssssssss ssssssss xxxxxxxx
2	$-2^{16} \le x < 2^{16}$	ssssssss ssssssss xxxxxxxx xxxxxxxx
3	$-2^{24} \le x < 2^{24}$	ssssssss xxxxxxxx xxxxxxxx xxxxxxxx
4	remaining *x*	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Instruction class	Cycles	Notes
ALU	1	+1 if you use a register-specified shift *Rs*. +2 if *Rd* is *pc*.
B, BL, BX	3	
CDP	$1+B$	
LDC	$1+B+N$	
LDR/B/H/SB/SH	3	+2 if *Rd* is *pc*.
LDM	$2+N$	+2 if *pc* is in the register list.
MCR	$2+B$	
MLA	$2+M$	
xMLAL	$3+M$	
MRC	$3+B$	
MRS, MSR	1	
MUL	$1+M$	
xMULL	$2+M$	
STC	$1+B+N$	
STR/B/H	2	
STM	$1+N$	
SWI	3	
SWP/B	4	

زمان‌بندی اجرای دستورات در ARM9TDMI

هسته پردازنده ARM9TDMI بر اساس یک خطِ لوله ۵- طبقه‌ای با تنها یک مرحله‌ی اجرایی و دو طبقـه واکشی از حافظه ساخته شده است. معمولا یک یا دو سیکل تاخیر بعد از دستور Load وجود دارند قبل از اینکه بتوان به داده دست یافت.

مدار ضرب‌کننده شامل یک آرایه‌ی ۳۲ بیت در ۸ بیت است. تعداد سیکلهای تکـرار M بسـتگی بـه مقـدار ثبات Rs دارد.

M	*Rs* range (use the first applicable range)	Rs bitmap s = sign bit x = wildcard-bit
1	$-2^8 \le x < 2^8$	sssssssss sssssssss sssssssss xxxxxxxx
2	$-2^{16} \le x < 2^{16}$	sssssssss sssssssss xxxxxxxx xxxxxxxx
3	$-2^{24} \le x < 2^{24}$	sssssssss xxxxxxxx xxxxxxxx xxxxxxxx
4	remaining *x*	xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxxx

Instruction class	Cycles	Notes
ALU	1	+1 if a register-specified shift *Rs* is used. +2 if *Rd* is *pc*.
B, BL, BX	3	
CDP	$1 + B$	
LDC	$B + N$	
LDRB/H/SB/SH	1	*Rd* is not available for two cycles.
LDR *Rd* not *pc*	1	*Rd* is not available for one cycle.
LDR *Rd* is *pc*	5	
LDM not loading *pc*	N	+1 if $N = 1$ or the last loaded register used in the next cycle.
LDM loading *pc*	$N + 4$	
MCR	$1 + B$	
MRC *Rd* not *pc*	$1 + B$	*Rd* is not available for one cycle.
MRC *Rd* is *pc*	$3 + B$	
MRS	1	
MSR	1	+2 if any of the *csx* fields are updated.
MUL, MLA	$2 + M$	
xMULL, xMLAL	$3 + M$	
STC	$B + N$	
STR/B/H	1	
STM	N	+1 if $N = 1$.
SWI	3	
SWP/B	2	*Rd* is not available for one cycle.

زمان‌بندی اجرای دستورات در ARM9E

هسته پردازنده ARM9E بر اساس یک خطِ لوله ۵- طبقه‌ای ساخته شده است. معمولا یک یـا دو سـیکل تاخیر بعد از دستور Load و یا ضرب وجود دارند قبل از اینکه بتوان به داده دست یافت.

مدار ضرب‌کننده شامل یک آرایه‌ی ۳۲ بیت در ۱۶ بیت است. جدول زیـر زمـان‌بنـدی اجـرای دسـتورات ARM9E را نشان می‌دهد.

Instruction Class	Cycles	Notes
ALU *Rd* not *pc*	1	+1 if a register-specified shift is used.
ALU *Rd* is *pc*	3	+1 if the operation is logical or any shift is used.
B, BL, BX, BLX	3	
CDP	$1+B$	
LDC	$B+N$	
LDRB/H/SB/SH	1	*Rd* is not available for two cycles. +1 if the load offset is shifted.
LDR *Rd* not *pc*	1	*Rd* is not available for one cycle. +1 if the load offset is shifted.
LDR *Rd* is *pc*	5	+1 if the load offset is shifted.
LDRD	2	$R(d+1)$ is not available for one cycle.
LDM not loading *pc*	N	+1 if $N = 1$ or the last loaded register used in the next cycle.
LDM loading *pc*	$N+4$	
MCR	$1+B$	
MCRR	$2+B$	
MRC *Rd* not *pc*	$1+B$	*Rd* is not available for one cycle.
MRC *Rd* is *pc*	$4+B$	
MRRC	$2+B$	*Rn* is not available for one cycle.
MRS	2	
MSR	1	+2 if any of the *csx* fields are updated.
MUL, MLA	2	*Rd* is not available for one cycle, except as an accumulator input for a multiply accumulate.
MULS, MLAS	4	
xMULL, xMLAL	3	*RdHi* is not available for one cycle, except as an accumulator input for a multiply accumulate.
xMULLS, xMLALS	5	
PLD	1	
QxADD, QxSUB	1	*Rd* is not available for one cycle.
SMULxy, SMLAxy, SMULWx, SMLAWx	1	*Rd* is not available for one cycle, except as an accumulator input for a multiply accumulate.
SMLALxy	2	*RdHi* is not available for one cycle, except as an accumulator input for a multiply accumulate.
STC	$B+N$	
STR/B/H	1	+1 if a shifted offset is used.
STRD	2	
STM	N	+1 if $N = 1$.
SWI	3	
SWP/B	2	*Rd* is not available for one cycle.

زمان‌بندی اجرای دستورات در ARM10E

هسته پردازنده ARM10E بر اساس یک خطِ لوله ۵- طبقه‌ای با پیش‌بینی کننده انشعاب ساخته شده است. معمولا یک سیکل تاخیر بعد از دستور Load و یا ضرب وجود دارند قبل از اینکه بتوان به داده دست یافت. ARM10E دارای پهنای داده ۶۴ بیتی است. پس دستورات بارگیری و ذخیره‌سازی می توانند در هر سیکل ۶۴ بیت را منتقل کنند. جدول زیر زمان‌بندی اجرای دستورات ARM10E را نشان می‌دهد.

Instruction class	Cycles	Notes
ALU	1	+1 if a register-specified shift, or RRX, is used. +4 if *Rd* is *pc*. An exception is MOV *pc*, *Rn*. This takes 4 cycles.
B, BX	0-2	+4 if the branch is mispredicted.
BL, BLX	1-2	+4 if the branch is mispredicted.
CDP	1	
LDC	1	Data availability depends on the coprocessor.
LDR/B/H/SB/SH *Rd* not *pc*	1	*Rd* is not available for one cycle. +1 if the addressing mode is register preindexed with the option of a (constant) shift.
LDR *Rd* is *pc*	6	+1 if the offset (pre- or postindex) is a shifted register. [2 cycles if not executed].
LDRD	1	*Rd* and $R(d+1)$ are not available for one cycle.
LDM not loading *pc*	1	The first data item is not available for one cycle. Once the address is 8-byte aligned, data items are loaded in pairs, at two per cycle. Therefore the *k*th data item will be available after $(k+a+1)/2$ cycles, where *a* is bit 2 of the base address. You cannot start another load or store until this one has finished.
LDM loading *pc*	$L+6$	$L=(N+a)/2$, and *a* is bit 2 of the base address.
MCR, MCCR	1	
MR{R}C *Rd* not *pc*	1	*Rd* is not available for one cycle.
MRC *Rd* is *pc*	2	
MRS	1	
MSR to *cpsr*	1	+3 if any of the *csx* fields are updated.
MSR to *spsr*	3	[2 if the instruction is not executed.]
MUL, MLA	2	*Rd* is not available for one cycle.
MULS, MLAS	4	
xMULL, xMLAL	3	*RdHi* is not available for one cycle.
xMULLS, xMLALS	5	

Instruction class	Cycles	Notes
PLD	1	+1 if a shifted register offset is used.
QxADD, QxSUB	1	*Rd* is not available for one cycle.
SMULxy, SMULWx	1	*Rd* is not available for one cycle.
SMLAxy, SMLAWx	2	
SMLALxy	2	*RdHi* is not available for one cycle.
STC	1	
STR/B/H	1	+1 if a preindexed shifted register offset is used.
STRD	1	
STM	1	Registers are stored two per cycle once the address is 8-byte aligned. You cannot write a register in the register list until its value has been stored. You cannot start another load or store until this one is complete.
SWP/B	2	

زمان‌بندی اجرای دستورات در ARM11

هسته پردازنده ARM11 بر اساس یک خطِ لوله ۸- طبقه‌ای با ۳- طبقه اجرایی ساخته شده است. معمولا دو سیکل تاخیر بعد از دستور Load وجود دارند قبل از اینکه بتوان به داده دست یافت. ثباتهای

داده ورودی بعضی دستورات همانند شیفت، ضرب و محاسبه آدرس باید از یک سیکل قبل در دسترس باشند.
برای مثال دستورات زیر، هسته را برای ۳ سیکل متوقف می‌کنند. زیرا نتیجه عملیات بارگیری ۲ سیکل بعد در دسترس قرار می‌گیرد و ورودی دستور شیفت نیز یک سیکل قبل باید در دسترس باشد:

```
LDR r0, [r1]        ; r0 not available for 2 cycles
MOV r2, r0, ASR#3   ; r0 required one cycle early
```

هسته ARM11 دارای واحد تولیدکننده آدرس جداگانه‌ایست که می‌تواند آدرسهای ساده را در یک سیکل تولید کند. آدرسهای پیچیده‌تر، دوسیکل کاری طول می‌کشند. جدول زیر تعداد سیکلهای حالتهای آدرس‌دهی متفاوت را نشان می‌دهد.

A	Addressing modes
1	[*Rn*, #<*signed-offset*>]{!} [*Rn*], #<*signed-offset*> [*Rn*, *Rm* {, *LSL #2*}]{!} [*Rn*], *Rm* {, *LSL #2*}
2	[*Rn*, -*Rm*] {!} [*Rn*], -*Rm* [*Rn*, {-}<*shifted_Rm*>]{!} where shift is not *LSL #0* or *LSL #2* [*Rn*], {-}<*shifted_Rm*> where shift is not *LSL #0* or *LSL #2*

جدول زیر زمان‌بندی اجرای دستورات ARM11 را نشان می‌دهد.

Instruction class	Cycles	Notes
ALU operations except a MOV to *pc* (for MOV to *pc*, see BX)	1	*Rm* is required one cycle early if shifted by a constant shift. +1 if a register-specified shift is used. In this case *Rs* is required one cycle early and *Rn* is not required until the second cycle. +6 if *Rd* is *pc*.
B <*immed*> BL <*immed*> BLX <*immed*>	1	Assumes successful dynamic prediction. Some dynamically predicted branches may be folded, to be zero cycles. +3 for successful static prediction. +4 for unsuccessful static or dynamic prediction. In this case the flags are required two cycles early.
BX *lr* MOV *pc*, *lr*	4	+1 if unconditional and return stack is empty. +3 if unconditional and return stack mispredicts. +1 if conditional. In this case the flags are required two cycles early.
BX *Rm* (not *lr*) BLX *Rm* MOV *pc*, *Rm* (not *lr*)	5	If no shift on MOV and conditional, the flags are required two cycles early. +1 if a constant shift is used for MOV. In this case *Rm* is required one cycle early. If conditional, then the flags are required one cycle early. +2 if a register-specified shift is used for MOV. In this case *Rs* is required one cycle early, and *Rn* is not used until the second cycle.
CPS	1	+1 if a mode change occurs.
LDR/B/H/SB/SH/D *Rd* not *pc*	*A*	*Rd* is not available for two cycles. $R(d+1)$is not available for two cycles for LDRD. If the load is potentially unaligned (base or offset unaligned), then you cannot start another memory access on the next cycle. If the load is unaligned, then *Rd* is not available for three cycles for LDR/H/SH. For LDRD *Rd* is not available for two cycles and $R(d+1)$ for three cycles.

Instruction class	Cycles	Notes
LDR *pc*, [*sp*, #*off*] {!} LDR *pc*, [*sp*], #*off*	4	+4 if unconditional and return stack is empty. +5 if unconditional and return stack mispredicts +4 if conditional.
LDR *pc* not using a constant stack offset	$A + 7$	
LDM not loading *pc*	1	You cannot start another memory access for the next $(N + a - 1)/2$ cycles, where a is bit 2 of the address. The kth register in the list not available for $(k + a + 3)/2$ cycles.
LDM *sp*{!} loading *pc*	4	+5 if conditional or return stack empty or return stack mispredicts. You cannot start another memory access for $(N + a)/2$ cycles. The kth register in the list not available for $(k + a + 5)/2$ cycles.
LDM loading *pc* not from the stack	8	You cannot start another memory access for $(N + a)/2$ cycles. The kth register in the list not available for $(k + a + 5)/2$ cycles.
MCR/MCRR	1	This counts as a memory access.
MRC/MRRC	1	This counts as a memory access. The result registers are not available for two cycles.
MRS	1	
MSR to *cpsr*	1	+3 if any of the *csx* fields are updated.
MSR to *spsr*	5	
MUL, MLA	2	*Rd* is not available for two cycles, except as an accumulator input for another multiply accumulate when it is not available for one cycle. *Rm* and *Rs* are required one cycle early. *Rn* is not required until the second cycle for MLA.
MULS, MLAS	5	*Rm* and *Rs* are required one cycle early. *Rn* is not required until the second cycle for MLAS.
xMULL, xMLAL	3	*RdLo* is not available for one cycle. *RdHi* is not available for two cycles. Reduce these latencies by one if these registers are used as accumulator inputs for another multiply accumulate. *Rm* and *Rs* are required one cycle early. *RdLo* is not required until the second cycle for MLAL.
xMULLS, xMLALS	6	*Rm* and *Rs* are required one cycle early. *RdLo* is not required until the second cycle for MLAL.
PKHBT, PKHTB	1	*Rm* is required one cycle early.

Instruction class	Cycles	Notes
PLD	*A*	
QxADD, QxSUB	1	*Rd* is not available for one cycle. *Rn* is required one cycle early for QDADD and QDSUB.
REV, REV16, REVSH	1	*Rm* is required one cycle early.
{S,SH,Q,U,UH,UQ} ADD16, ADDSUBX, SUBADDX, SUB16, ADD8, SUB8	1	*Rd* is not available for one cycle for saturating or halving operations (SH, Q, UH, UQ prefix). *Rm* is required one cycle early for ADDSUBX and SUBADDX operations.
SEL	1	
SETEND	1	
SMULxy, SMLAxy, SMULWy, SMLAWy SMUAD, SMLAD, SMUSD, SMLSD	1	*Rd* is not available for two cycles, except as an accumulator input for another multiply accumulate when it is not available for one cycle. *Rm* and *Rs* are required one cycle early.
SMLALxy, SMLALD{X}, SMLSLD{X}	2	*RdLo* is not available for one cycle. *RdHi* is not available for two cycles. Reduce these latencies by one if these registers are used as accumulator inputs for another multiply accumulate. *Rm* and *Rs* are required one cycle early. *RdHi* is not required until the second cycle.
SMMUL{R}, SMMLA{R}, SMMLS{R}	2	*Rd* is not available for two cycles, except as an accumulator input for another multiply accumulate when it is not available for one cycle. *Rm* and *Rs* are required one cycle early. *Rn* is not required until the second cycle.
SSAT, USAT, SSAT16, USAT16	1	*Rd* is not available for one cycle. *Rm* is required one cycle early for SSAT and USAT.
STR/B/H/D	A	If the store is potentially unaligned (base or offset unaligned), then you cannot start a memory access on the next cycle. For STRD you cannot start another instruction that writes to $R(d+1)$ for one cycle.
STM	1	You cannot start another memory access for the next $(N+a-1)/2$ cycles, where a is bit 2 of the address. You cannot start an instruction that writes to the kth register in the list for $k/2$ cycles.
SWI	8	
SWP/B	2	*Rd* is not available for one cycle.
SXT, UXT	1	*Rm* is required one cycle early.

Instruction class	Cycles	Notes
UMAAL	3	*RdLo* is not available for one cycle. *RdHi* is not available for two cycles. These latencies are reduced by one for another accumulate. *Rm* and *Rs* are required one cycle early. *RdLo* is not required until the second cycle.
USAD8, USADA8	1	*Rd* is not available for two cycles, with the exception that the result of USAD8 is available as the accumulator for USADA8 after one cycle. *Rm* and *Rs* are required one cycle early.

ضمیمه ث

سایت های اینترنتی

در این ضمیمه نحوه رمزگذاری دستورات ۳۲ بیتی ARM و ۱۶ بیتی Thumb را با هم می‌بینیم. همچنین در مورد بیتهای مختلف cpsr و spsr صحبت خواهیم کرد.

S3C2440

ضمیمه ج

بردهای آموزشی ARM

در این ضمیمه فهرستی گسترده از انواع میکروکنترولرهای ARM ساخته شده توسط شرکتهای مختلف سازنده نیمه‌هادی را می‌بینیم. این فهرست دایما به‌روز می‌شود و اصل آن در سایت شرکت KEIL و همچنین در DVD همراه کتاب موجود است.

ضمیمه چ

مطالعات بیشتر

در این ضمیمه فهرستی گسترده از انواع میکروکنترولرهای ARM ساخته شده توسط شرکتهای مختلف سازنده نیمه‌هادی را می‌بینیم. این فهرست دایما به‌روز می‌شود و اصل آن در سایت شرکت KEIL و همچنین در DVD همراه کتاب موجود است.

ضمیمه ح

راهنمای سریع برنامه نویسی C/C++

در این ضمیمه فهرستی گسترده از انواع میکروکنترولرهای ARM ساخته شده توسط شرکتهای مختلف سازنده نیمه‌هادی را می‌بینیم. این فهرست دایما به‌روز می‌شود و اصل آن در سایت شرکت KEIL و همچنین در DVD همراه کتاب موجود است.

مراجع