آموزش میکروکنترلرهای ARM7

Electrovolt.ir

مولف : مهندس شجاع داودی

ناشر : وبسایت الکترو ولت

ویرایش دوم

شهریور ۱۳۹۵

مقدمه مولف :

امروزه از میکروکنترلرهای ARM به علت یکپارچه کردن سیستم های کنترلی با سرعت پردازش بالا ، توان مصرفی کم ، قیمت ارزان تر و حجم کمتر بیشترین استفاده را می شود. به طوری که امروزه هسته‌های پردازنده ARM به عنوان رایج‌ترین پردازنده ۳۲ بیتی با سرعت پردازش چند مگاهرتز تا چند گیگاهرتز در طیف وسیعی از سیستم‌های نهفته و قابل حمل مورد استفاده قرار می‌گیرند. برای نمونه امروزه اغلب تلویزیون ها ، تلفن‌های هوشمند ، تبلت ها ، خودروها و ... از این هسته پردازشی بهره می‌برند.

در این جزوه آموزشی شما ابتدا با انواع میکروکنترلرهای ARM ، ضرورت و مزایای هر یک آشنا می شوید. سپس از میان انواع مختلف ، با میکروکنترلرهای ARM7 از شرکت NXP آشنا می شوید و در ادامه با تشریح رجیسترها ، معماری داخلی، واحدهای مختلف و نحوه عملکرد آنها ، نحوه برنامه نویسی به زبان C ، پایه های میکروکنترلر و نحوه راه اندازی آن ، شبیه سازی با نرم افزار Proteus ، برنامه ریزی و عیب یابی با نرم افزار KEIL این سری از میکروکنترلرها را فرا خواهید گرفت.

از مزایای استفاده از این دوره میتوان به یادگیری اصول اولیه میکروکنترلرهای ARM از پایه ، آموزش نرم افزار KEIL از پایه ، آموزش شبیه سازی میکروکنترلرهای ARM در Proteus ، آموزش واحدهای مختلف ارتباطی و تنظیمات آن اشاره کرد. همچنین مهمترین مزیت این دوره تهیه و ساخت هدر فایلهایی برای کامپایلر KEIL است که با اضافه کردن آنها به برنامه میتوان میکروکنترلرهای ARM7 را بسیار راحت و همانند میکروکنترلرهای AVR برنامه نویسی کرد.

همانطور که برای یادگیری مثلا توابع مثلثاتی ابتدا لازم است جدول ضرب و سپس هندسه و پیش نیازهای آن را بلد باشیم، برای یادگیری دوره ARM مقدماتی نیز باید پیش نیازهای آن شامل اصول الکترونیک دیجیتال ، اجزای مدارهای الکترونیکی دیجیتال و زبان برنامه نویسی C را آموخته باشیم. در نتیجه در این دوره فرض بر این است که شما اصول برنامه نویسی C و اصول اولیه الکترونیک دیجیتال را به خوبی یاد گرفته اید.

Electrovolt.ir

سرفصل های آموزش :

فصل ۱ – آشنایی با انواع تراشه های مبتنی بر ARM

1-1- معرفی و تاریخچه میکروکنترلرهای ARM

1-2- نسل های مختلف پردازنده های ARM

1-3- انواع معماری های به کار رفته در ARM

1-4- مقایسه سری های مختلف پردازنده های ARM

1-5- شرکت های سازنده میکروکنترلرهای ARM

فصل ۲ – آشنایی با انواع میکروکنترلرهای ARM شرکت NXP

2-1- معرفی خانواده های مختلف میکروکنترلرهای ARM شرکت NXP

2-2- معرفی سری های پرکاربرد میکروکنترلرهای ARM شرکت NXP

فصل ۳ – معرفی، معماری و تشریح میکروکنترلرهای ARM7

3-1- معرفی هسته های پردازشیARM7

3-2- واحدهای اصلی یک میکروکنترلر در معماریARM7

3-3- انواع واحدهای جانبی موجود در میکروکنترلرهای ARM7 شرکتNXP

3-4- تشریح معماری و بلوک دیاگرام سری LPC213X

3-5- مقایسه شباهت ها و تفاوت های موجود بین ARM7 و AVR

فصل ۴ – اصول راه اندازی میکروکنترلرهای سری LPC213X

4-1- مقایسه بین میکروکنترلرهای سری LPC213X

4-2- تشریح پایه های میکروکنترلر LPC2138

4-3- طراحی برد راه انداز LPC2138

4-4- انواع روش های پروگرام کردن میکروکنترلرهای ARM

4-5- معرفی انواع کامپایلرهای موجود برای میکروکنترلرهای ARM

فصل ۵ – شروع به کار با نرم افزار KEIL و Proteus

5-1- دانلود و نصب نرم افزار KEIL و Proteus

5-2- آموزش نحوه ایجاد پروژه در KEIL

5-3- آموزش تنظیمات پروژه در KEIL

5-4- آموزش رسم مدار در پروتئوس

5-5- آموزش نحوه شبیه سازی در Proteus

فصل ۶ – آموزش برنامه ریزی و راه اندازی میکروکنترلرLPC2138

6-1- معرفی رجیسترهای GPIO و راه اندازی پورت ها

6-2- آموزش نوشتن برنامه برای راه اندازیLED

6-3- آموزش نحوه پیاده سازی پروژه بر روی برد

6-4- آموزش برنامه ریزی سریال با نرم افزار Flash Magic

6-5- آموزش برنامه ریزی و عیب یابی با JLINK

فصل ۷ – آموزش واحد کنترل سیستم و راه اندازی PLL

7-1- معرفی و تشریح واحد کنترل سیستم

7-2- آموزش نحوه مدیریت توان مصرفی

7-3- آموزش نحوه مدیریت کلاک سیستم

7-4- آموزش راه اندازی واحد PLL

فصل ۸ – آموزش کار با پورت ها و راه اندازی وسایل جانبی

8-1- آموزش راه اندازی کلید

8-2- آموزش راه اندازی صفحه کلید

8-3- آموزش راه اندازی سون سگمنت

8-4- آموزش راه اندازی LCD کاراکتری

8-5- آموزش نحوه راه اندازی LCD های گرافیکی

فصل ۹ – راه اندازی واحد وقفه برداری (VIC)

9-1- معرفی واحد VIC و تشریح نحوه عملکرد آن

9-2- راه اندازی عوامل وقفه های خارجی و داخلی

فصل ۱۰ – راه اندازی واحد مبدل آنالوگ به دیجیتال(ADC)

10-1- معرفی واحد ADC و نحوه عملکرد آن

10-2- راه اندازی واحد ADC و نحوه کار با رجیسترهای تنظیمات

10-3- توابع کار با واحد ADC و هدر فایل ADC.h

فصل ۱۱ – راه اندازی واحد DAC

11-1- معرفی و تشریح عملکرد واحد DAC

11-2- راه اندازی واحد DAC و تنظیمات آن

فصل ۱۲ – راه اندازی واحد تایمر/کانتر Timer/Counter

12-1- معرفی واحد T/C و تشریح نحوه عملکرد آن

12-2- نحوه راه اندازی واحد T/C و تنظیمات آن

فصل ۱۳ – راه اندازی واحد PWM

13-1- مفهوم PWM و انواع آن

13-2- معرفی و تشریح عملکرد واحدPWM

13-3- راه اندازی واحد PWM به صورت تک لبه

فصل ۱۴ – راه اندازی واحد RTC

14-1- معرفی و تشریح عملکرد واحدRTC

14-2- راه اندازی واحد RTC و رجیسترهای آن

فصل ۱۵ – راه اندازی تایمر سگ نگهبان Whatchdog

15-1- معرفی و تشریح عملکرد واحد تایمر سگ نگهبان

15-2- راه اندازی واحد WD و تنظیمات آن

فصل ۱۶ – راه اندازی واحد UART

16-1- معرفی و تشریح عملکرد واحدUART

16-2- راه اندازی واحد UART و تنظیمات آن

16-3- معرفی انواع ماژول های سریال

فصل ۱۷ – راه اندازی واحد SPI

17-1- معرفی و تشریح عملکرد واحدSPI

17-2- راه اندازی واحد SPI و تنظیمات آن

فصل ۱۸ – راه اندازی واحد I2C

18-1- معرفی و تشریح عملکرد واحد I2C

18-2- راه اندازی واحد I2C سخت افزاری و تنظیمات آن

18-3- راه اندازی واحد I2C نرم افزاری

پیش نیاز دوره : برنامه نویسی C ویژه میکروکنترلرها (جزوه ۰ تا ۱۰۰ برنامه نویسی C)

ایمیل استاد : hosainshoja@yahoo.com

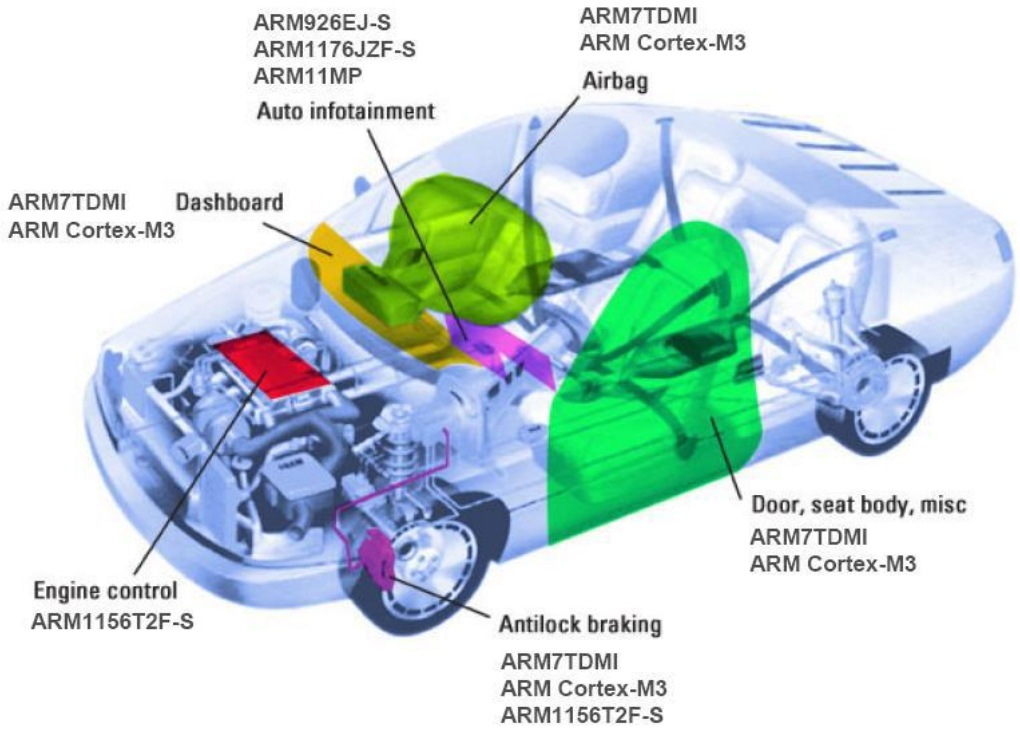
آی دی استاد : Telegram.me/electro_volt

فصل ۱ – آشنایی با انواع تراشه های مبتنی بر ARM

مقدمه

مدت هاست که از سیستم های مبتنی بر معماری ARM در محصولاتی با کارایی بالا استفاده می شود. به عنوان مثال در کاربردهای تجاری و خانگی نظیر یخچال ، تلویزیون ، لباسشویی ها ، گوشی های همراه ، تبلت ها و ... در کاربردهای صنعتی نظیر کنترل تجهیزات صنعتی کارخانه ها ، کنترل تجهیزات خودروها و ... و در بسیاری کاربردهای تفریحی و سرگرمی نظیر اسباب بازی ها ، کوادکوپترها و... از این تراشه ها استفاده می شود. در شکل زیر بخشی از کاربردهای قدرت گرفته از ARM را مشاهده می کنید.

پردازنده های ARM آنقدر ویژگی های خوب و بهینه داشته اند که پا از مرز کارایی فراتر گذاشته و در هر وسیله هوشمند چندین نوع از آن به چشم می خورد به طوری که هم اکنون بیش از ۱۰۰ بیلیون تراشه ARM در دنیا ساخته شده و استفاده می شود. برای مثال در یک خودروی هوشمند برای بسیاری از ویژگی ها از قبیل کنترل موتور ، کابین ، ترمزها ، GPS و... از انواع آنها استفاده می شود. شکل زیر یک نمونه خودرو هوشمند امروزی و پردازنده های ARM به کار رفته در آن را نشان می دهد.



ARMچیست ؟

ARM مخفف عبارت Advanced Risc Machine به معنای "ماشین ساختار یافته با دستورالعمل های کاهش یافته" می باشد. در حقیقت ARM معرف یک پردازنده (CPU) با معماری RISC و مجموعه دستورالعمل های ۳۲ یا ۶۴ بیتی است که مخترع آن شرکت سهامی تجاری ARM یا ARM Holding واقع در شهر کمبریج انگلستان است. این شرکت که در سال ۱۹۹۰ تاسیس شده است ، شهرت عمده خود را به خاطر طراحی پردازنده های ARM کسب نموده است. از دیگر محصولات این شرکت میتوان به طراحی انواع سیستم های روی تراشه (System On Chip) و محصولات نرم افزاری نظیر Real View و KEIL اشاره نمود.

ویژگی های منحصر به فرد ARM

- مصرف توان بهینه که ARM را به گزینه برتر برای استفاده در تجهیزات قابل حمل تبدیل نموده است.
- استاندارد بودن تراشه ARM ؛ یعنی میتوان برنامه ی نوشته شده برای یک تراشه را بدون نیاز به تغییر ، توسط تراشه های دیگر تولید کنندگان نیز استفاده نمود.
- معماری ساده ARM که با استفاده از تعداد ترانزیستورهای بسیار کمتر از پردازنده های CISC قابل پیاده سازی است.

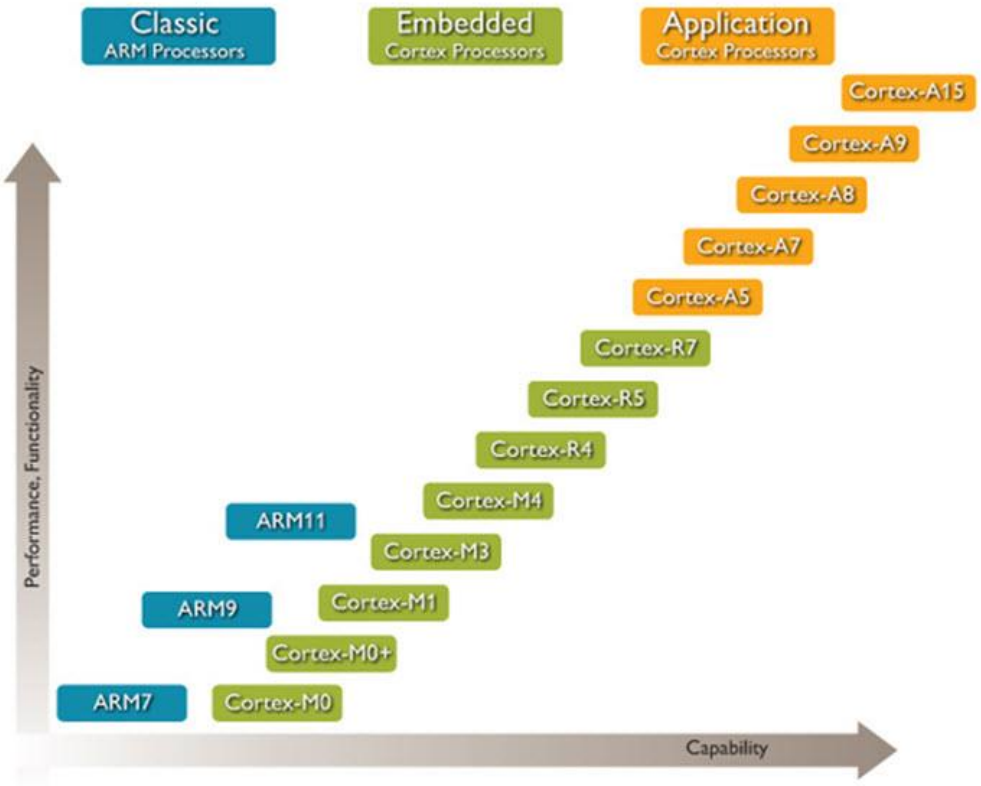
- کارایی بالا در عین ابعاد کوچک ؛ برای مثال کارایی پردازنده ARM که با فرکانس 400Mhz کار می کند ، با کارایی پردازنده PENTIUM2 با فرکانس 300Mhz تقریبا برابر است اما مصرف توان ARM مثلا یک پنجاهم پردازنده پنتیوم است.

- قابلیت استفاده از سیستم عامل هایی نظیر Linux ، Windows CE، Android و... که به صورت رایگان و متن باز در دسترس است.

نسل های مختلف پردازنده هایARM

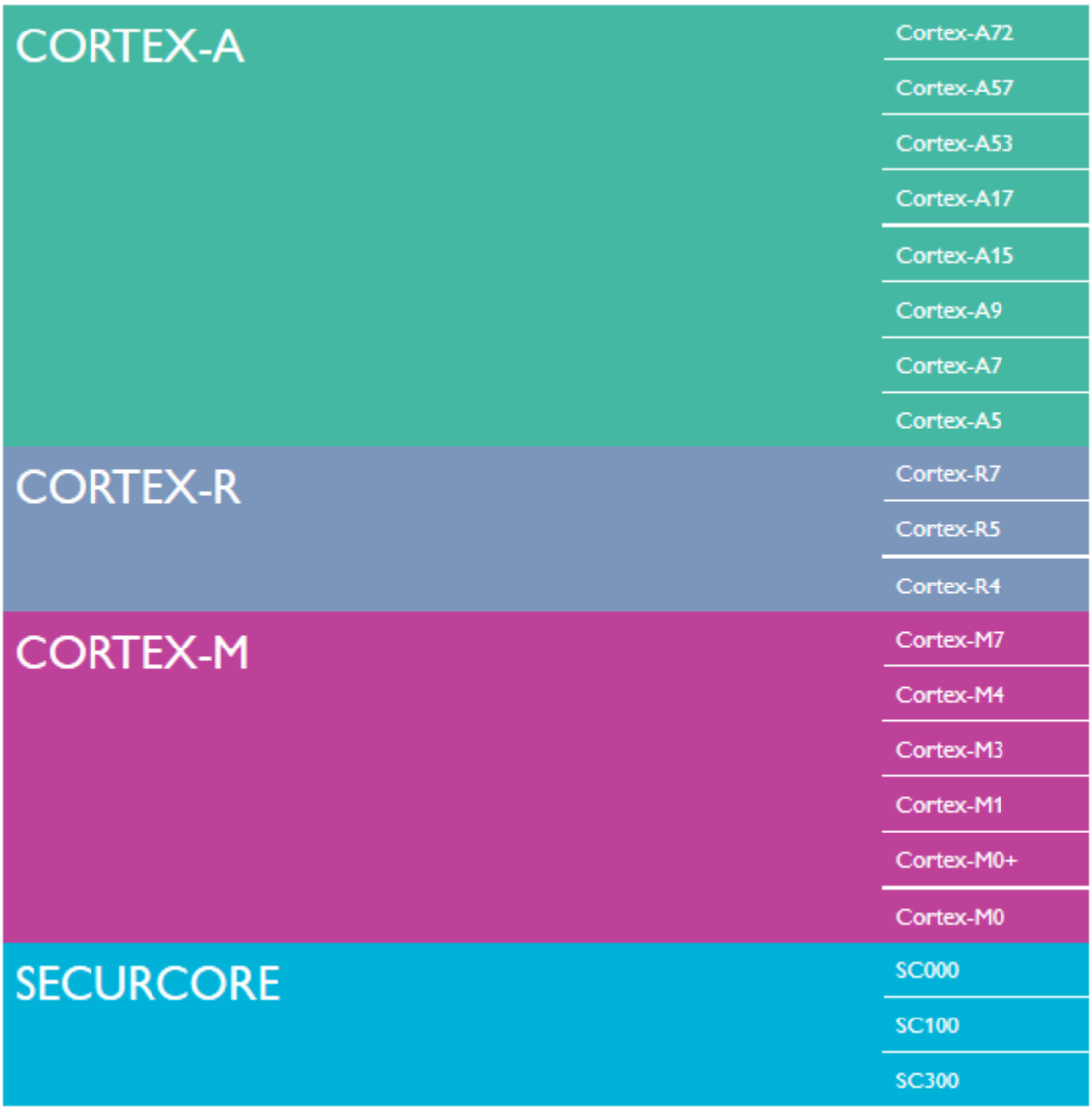
پردازنده های ARM بر اساس **معماری** ساخت هسته و همچنین **دستورالعمل** های پیاده سازی شده بر روی آن دسته بندی می شود.

انواع خانواده های پردازنده های ARM شامل ARM7 ، ARM9 ، ARM11 ، Cortex و SecurCore می باشد. هر یک از این خانواده ها خود تقسیم بندی های جزئی تری دارند که در نمودار شکل زیر مشاهده می کنید. خانواده Cortex خود به سه سری Cortex M، Cortex R و Cortex A تقسیم می شوند که هر یک ویژگی های خاص خود را دارند. در این نمودار هر خانواده براساس قابلیت و کارایی در مکان خود قرار گرفته است.



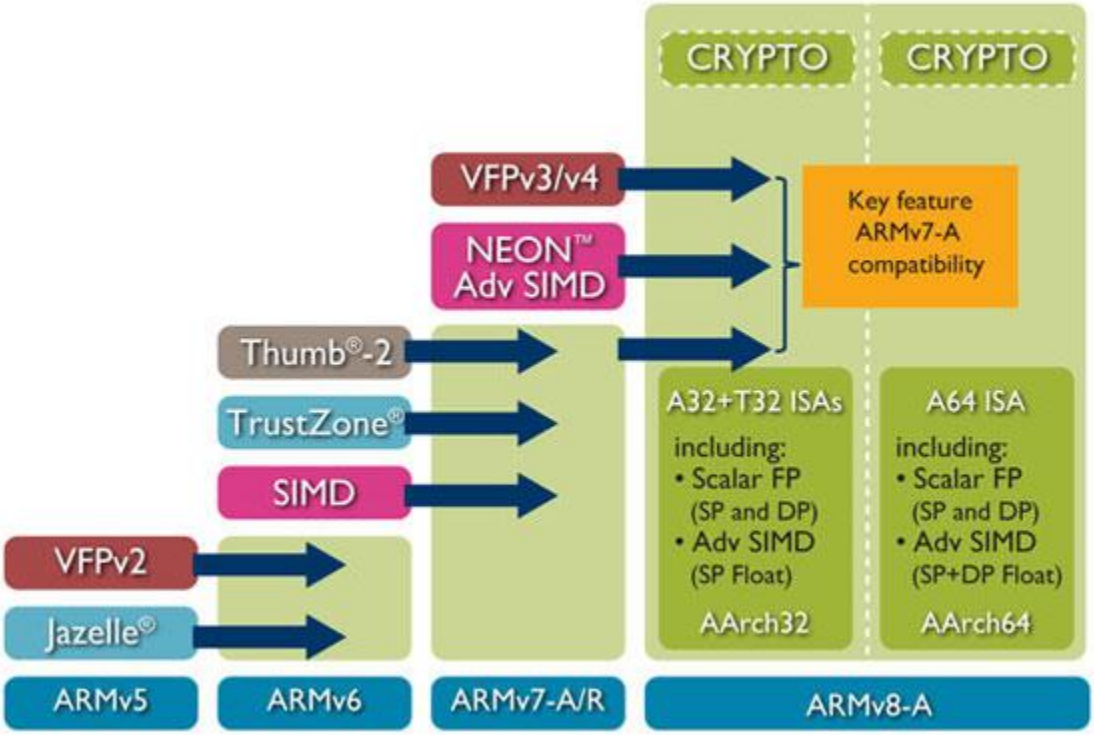
نکته : خانواده های قدیمی تر از ARM7 منسوخ شده است و دیگر تولید نمی شود. همچنین به علت کارایی و قابلیت های یکسان یا بالاتر پردازنده های Embedded نسبت به سری Classic ، در اکثر پروژه ها از آنها استفاده شده و بر سری کلاسیک ترجیح داده می شوند. بنابراین این سری نیز تقریبا در حال منسوخ شدن است.

در شکل زیر خانواده های جدیدتر پردازنده های ARM که هم اکنون بیشترین استفاده را دارند، نشان شده است.



هر یک از خانواده های یاد شده دارای معماری براساس مجموعه دستورالعمل هایی می باشد که با نسخه های یکتا معرفی می شود. این نسخه ها که ARM Architecture Versions نام دارند شامل : ARM V4 ، ARM V5 ، ARM V6 ، ARM V7 و ARM V8 می باشند. هر کدام از این نسخه ها خود بر اساس ویژگی های اضافه شده به دستورالعمل های آن دارای استانداردهای اضافه تری نیز می باشند که در برخی پردازنده ها یک یا چند ویژگی وجود دارد. این ویژگی ها که Instruction Set Architecture یا ISA نام دارند؛ نظیر Thumb ، Jazelle، DSP، NEON و ... باعث افزایش

کارایی پردازنده می شود. این ویژگی ها و اضافه شدن آن به ورژن های بالاتر معماری ARM را در شکل زیر مشاهده می کنید.



نکته : نسخه های قبل از ARMv4 منسوخ شده است و نسخه ARMv4 نیز که در معماری دستورالعمل های پردازنده های خانواده ARM7 و ARM9 از آن استفاده شده است ، در حال منسوخ شدن می باشد.

Thumb

مجموعه دستورالعمل های Thumb در نسخه چهارم معماری ARM معرفی شده است که کدهای بیشتری را برای کاربردهای تولیدکنندگان مختلف فراهم می کند Thumb .یک زیرمجموعه از عمومی ترین دستورالعمل های ARM سی و دو بیتی را که درون کدهای عملیاتی -Opcode) کدهای عملیاتی بخشی از دستورالعمل های زبان ماشین هستند که اجرا شدن عملیات را مشخص می کند) با اندازه شانزده بیت فشرده شده است، فراهم می کند. در زمان اجرا، این دستورالعمل های شانزده بیتی می توانند از وضعیت فشرده خارج شده و به دستورالعمل های ARM سی و دو بیتی مبدل شود یا به طور مستقیم توسط یک واحد رمزگشایی اختصاصی Thumb اجرا شوند. اگر چه کد Thumbنسبت به کد ARMسی و دو بیتی معادل، چهل درصد بیشتر دستورالعمل استفاده می کند، اما به سی درصد فضای کمتر نیاز دارد. همچنین کد Thumb نسبت به کد ARM چهل درصد آهسته تر است. بنابراین Thumb موجب کاهش کارایی می شود و به طور معمول در کارهایی که حساس به کارایی نیستند، کاربرد دارد Thumb .اگرچه موجب کاهش کارایی می شود، اما موجب کاهش مصرف توان سیستم می شود که یک معیار بسیار مناسب برای سیستم های همراه به شمار می آید.

Thumb 2

یک مجموعه از دستورالعمل های سی و دو بیتی است که در کنار دستورالعمل های شانزده بیتی سنتی که در Thumb بکار گرفته شده بود، اجرا می شود Thumb 2 .می تواند نیاز به Thumb را در یک سیستم کاهش داده یا به طور کامل حذف کند .این فناوری موجب بهبود کارایی در برخی مواقع نیز موجب کاهش مصرف توان می شود.

Jazell

فناوری Jazell سخت افزاری است که به پردازنده های ARM اجازه می دهد تا «بایت کدهای» جاوا را اجرا کند. آن دسته از معماری هایی که دارای چنین ویژگی هستند، در کاربردهای جاوا کارایی بالاتر به همراه مصرف توان پایین تر را فراهم می کنند.

بسط هایDSP

سیستم های مبتنی بر ARM وظایف پردازش سیگنال را با استفاده از کمک پردازنده DSP اختصاصی اجرا می کنند. در برخی از مواقع پشتیبانی از DSP به وسیله هسته اصلی ARM مناسب است. یک بسط ISA متناظر در پنجمین نسخه این معماری معرفی شده است. این ویژگی هنگامی که عملیات جمع و تفریق و ضرب شانزده بیتی اشباع شود از آن ها پشتیبانی می کند. این ویژگی در ششمین نسخه ARM به کار گرفته شد و به بسط مجموعه دستورالعمل های SIMD اضافه شد که اجازه می دهد دو دستورالعمل محاسباتی شانزده بیتی یا چهار دستورالعمل محاسباتی هشت بیتی به طور همزمان اجرا شود.

NEON

این فناوری که گاهی با نام Advanced SIMD شناخته می شود، در معماری نسل هفتم مورد استفاده قرار گرفته و برای کاربردهای حرفه ای، وسایل همراه کم مصرف و علاقمندان به رسانه ها طراحی شده است. فناوری NEON یک معماری هایبرید SIMD نوع ۶۴ یا ۱۲۸ بیت است که به وسیله ARM برای شتاب دادن به کارایی چند رسانه ای و برنامه های کاربردی پردازش سیگنال شامل رمزگذاری/ رمزگشایی ویدئویی، گرافیک سه بعدی، فشرده سازی رمزگشایی صوتی و پردازش تصاویر توسعه داده شده است.

معماری خط لوله ای پردازنده هایARM

پردازنده های ARM از خط لوله برای پردازش استفاده می کنند. پردازنده های مبتنی بر خط لوله دارای سه وضعیت کاری برای اجرای یک دستورالعمل هستند FETCH، DECODE وEXECUTE . به عبارت ساده تر، در سیکل اول، دستور اول واکشی می شود، در سیکل دوم، دستور اول رمزگشایی و دستور دوم واکشی می شود و در سیکل سوم، دستور اول اجرا و دستور دوم رمزگشایی و دستور سوم واکشی می شود. به این نوع سیستم خط لوله سه مرحله ای گفته می شود. خط لوله ها از اولین پردازنده ARM تا هسته ARM7TDMI سه مرحله ای هستند و در نسخه های بالاتر تعداد مراحل خط لوله ها افزایش پیدا کرده است. به طور مثال، پردازنده های ARM9 دارای خط لوله های پنج مرحله ای هستند که عملیات خواندن و نوشتن از حافظه ها نیز جزء این عملیات قرار گرفته است. خط لوله ها در ARM10 شش مرحله ای و در ARM11 هشت مرحله ای هستند. به طور کلی، هرچه تعداد مراحل خط لوله ها افزایش پیدا کند، قدرت پردازش پردازنده و در نتیجه کارایی آن افزایش می یابد.

هدف هر دو طراحی خط لوله و ARM ISA ، به حداقل رساندن مصرف انرژی است. مصرف توان کمتر یکی از فاکتورهای مهمی است که در وسایل همراه مورد توجه قرار می گیرد. سازندگان چنین وسایلی همواره تمایل دارند از قطعاتی با مصرف توان پایین تر در محصولات خود استفاده کنند. معماری ARM انعطاف پذیری بالایی دارد ، به طوری که تنها بخش اجباری یک پردازنده ARM ، مسیر پردازشی عدد صحیح آن است و اجزای دیگر شامل حافظه نهان، MMU ، ممیز شناور و دیگر اجزای پردازنده اختیاری هستند. این موضوع نیز انعطاف پذیری بالایی را در ساختمان پردازنده های مبتنی بر ARM فراهم کرده است. در نهایت اگر چه این پردازنده ها کوچک و کم مصرف هستند، اما کارایی بالایی در برنامه های کاربردی فراهم می کنند. به عنوان مثال، پردازنده PXA255 XScale با فرکانس چهارصد مگاهرتز کارایی تقریباً برابری با «پنتیوم ۲» سیصد مگاهرتزی ارائه می کند، در حالی که مصرف توان آن پنجاه برابر کمتر است.

هسته های کمک پردازشی(Coprocessors)

معماری ARMاز یک مکانیزم عمومی برای توسعه مجموعه دستورالعمل ها از طریق افزایش کمک پردازنده ها پشتیبانی می کند. برای نمونه، واحدهای ممیز شناور به عنوان یک کمک پردازنده به کار گرفته می شوند. یک مثال دیگر از کمک پردازنده، کمک پردازنده کنترل سیستم است که بافر نوشتن، TLB، حافظه نهان وMMU ها را مدیریت می کند.
معماری ARM یک پروتکل برای تعامل بین هسته ARM و کمک پردازنده ها و همچنین دستورالعمل هایی برای انتقال داده بین ARM و کمک پردازنده ها معین کرده است. کمک پردازنده ها نیازمند یک معماری بارگذاری ذخیره سازی (Load-Store) هستند. هر کمک پردازنده می تواند حداکثر شانزده ثبات از هر نوع اندازه ای داشته باشد. به طور کلی، سه نوع دستورالعمل مربوط به کمک پردازنده توسط هسته ARM اصلی به رسمیت شناخته شده است.

جمع بندی انواع تراشه های ARM

همانطور که پیش‌تر اشاره کردیم، شرکت ARM Holding نسبت به طراحی هسته براساس معماری ARM اقدام می‌کند و هسته‌های متفاوتی را براساس نسل‌های مختلف این معماری عرضه کرده است، جدیدترین معماری این شرکت ARM v8است که از دستورات ۶۴ بیتی پشتیبانی می‌کند. در جدول زیر کل هسته‌های طراحی شده توسط ARM را به همراه ویژگی ها و مشخصات آن مشاهده می کنید.

ARM Family	ARM Architecture	ARM Core	Feature	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.	4 KB 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 floating-point unit).	4 KB unified	28 MIPS @ 33 MHz
		ARM610	As ARM60, cache, no coprocessor bus.	4 KB unified	17 MIPS @ 20 MHz 0.65 DMIPS/MHz
ARM7	ARMv3	ARM700		4 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)	3stage 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 Extension	60 MIPS @ 59.8 MHz
		ARM740T	As ARM7TDMI, cache	MPU	
ARM7EJ	ARMv5TEJ	ARM7EJ-S	5stage pipeline, Thumb, Jazelle DBX, Enhanced DSP instructions	none	
ARM8	ARMv4	ARM810	5stage pipeline, static branch prediction, double-bandwidth memory	8 KB unified, MMU	84 MIPS @ 72 MHz 1.16 DMIPS/MHz

ARM9TDMI	ARMv4T	ARM9TDMI	5stage pipeline, Thumb	none	
		ARM920T	As ARM9TDMI, cache	16 KB/16 KB, MMU with FCSE (Fast Context Switch Extension)	200 MIPS @ 180 MHz
		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	6stage pipeline, Thumb, Enhanced DSP instructions, (VFP)	32 KB/32 KB, MMU	
		ARM1022E	As ARM1020E	16 KB/16 KB, MMU	
	ARMv5TEJ	ARM1026EJ-S	Thumb, Jazelle DBX, Enhanced DSP instructions, (VFP)	variable, MMU or MPU	
ARM11	ARMv6	ARM1136J(F)-S	8stage pipeline, SIMD, Thumb, Jazelle DBX, (VFP), Enhanced DSP instructions	variable, MMU	740 @ 665-532 MHz (i.MX31 SoC), 400–528 MHz
	ARMv6T2	ARM1156T2(F)-S	8stage pipeline, SIMD, Thumb-2, (VFP), Enhanced DSP instructions	variable, MPU	
	ARMv6Z	ARM1176JZ(F)-S	As ARM1136EJ(F)-S	variable, MMU + TrustZone	965 DMIPS @ 772 MHz, up to 2 600 DMIPS with four processors
	ARMv6K	ARM11 MPCore	As ARM1136EJ(F)-S, 14 core SMP	variable, MMU	
SecureCore	ARMv6-M	SC000			0.9 DMIPS/MHz
	ARMv4T	SC100			
	ARMv7-M	SC300			1.25 DMIPS/MHz
Cortex-M	ARMv6-M	Cortex-M0	Microcontroller profile, Thumb + Thumb-2 subset (BL, MRS, MSR, ISB, DSB, DMB), hardware multiply instruction (optional small), optional system timer, optional bit-banding memory	No cache, No TCM, No MPU	0.84 DMIPS/MHz
		Cortex-M0+	Microcontroller profile, Thumb + Thumb-2 subset (BL, MRS, MSR, ISB, DSB, DMB),hardware multiply instruction (optional small), optional system	No cache, No TCM, optional MPU with 8 regions	0.93 DMIPS/MHz

			timer, optional bit-banding memory		
		Cortex-M1	Microcontroller profile, Thumb + Thumb-2 subset (BL, MRS, MSR, ISB, DSB, DMB),hardware multiply instruction (optional small), OS option adds SVC / banked stack pointer, optional system timer, no bit-banding memory	No cache, 0-1024 KB I-TCM, 0-1024 KB D-TCM, No MPU	136 DMIPS @ 170 MHz ,(0.8 DMIPS/MHz FPGA-dependent)
	ARMv7-M	Cortex-M3	Microcontroller profile, Thumb / Thumb-2, hardware multiply and divide instructions, optional bit-banding memory	No cache, No TCM, optional MPU with 8 regions	1.25 DMIPS/MHz
	ARMv7E-M	Cortex-M4	Microcontroller profile, Thumb / Thumb-2 / DSP / optional FPv4 single-precision FPU, hardware multiply and divide instructions, optional bit-banding memory	No cache, No TCM, optional MPU with 8 regions	1.25 DMIPS/MHz
Cortex-R	ARMv7-R	Cortex-R4	Real-time profile, Thumb / Thumb-2 / DSP / optional VFPv3 FPU, hardware multiply and optional divide instructions, optional parity & ECC for internal buses / cache / TCM, 8-stage pipeline dual-core running lockstep with fault logic	0-64 KB / 0-64 KB, 0-2 of 0-8 MB TCM, opt MPU with 8/12 regions	
		Cortex-R5 (MPCore)	Real-time profile, Thumb / Thumb-2 / DSP / optional VFPv3 FPU and precision, hardware multiply and optional divide instructions, optional parity & ECC for internal buses / cache / TCM, 8-stage pipeline dual-core running lock-step with fault logic / optional as 2 independent cores, low-latency peripheral port (LLPP), accelerator coherency port (ACP)	0-64 KB / 0-64 KB, 0-2 of 0-8 MB TCM, opt MPU with 12/16 regions	
		Cortex-R7 (MPCore)	Real-time profile, Thumb / Thumb-2 / DSP / optional VFPv3 FPU and precision, hardware multiply and optional	0-64 KB / 0-64 KB, ? of 0-128 KB TCM, opt MPU with 16 regions	

			divide instructions, optional parity & ECC for internal buses / cache / TCM, 11-stage pipeline dual-core running lock-step with fault logic / out-of-order execution / dynamic register renaming / optional as 2 independent cores, low-latency peripheral port (LLPP), ACP [19]		
Cortex-A	ARMv7-A	Cortex-A5	Application profile, ARM / Thumb / Thumb-2 / DSP / SIMD / Optional VFPv4-D16 FPU / Optional NEON / Jazelle RCT and DBX, 1-4 cores / optional MPCore, snoop control unit (SCU), generic interrupt controller (GIC), accelerator coherence port (ACP)	4-64 KB / 4-64 KB L1, MMU + TrustZone	1.57 DMIPS / MHz per core
		Cortex-A7 MPCore	Application profile, ARM / Thumb / Thumb-2 / DSP / VFPv4-D16 FPU / NEON / Jazelle RCT and DBX / Hardware virtualization, in-order execution, SMP ۴superscalar, 1–cores, Large Physical Address Extensions (LPAE), snoop control unit (SCU), generic interrupt controller (GIC), ACP, architecture and feature set are identical to A15, 8-10 stage pipeline, low-power design	32 KB / 32 KB L1, 0-4 MB L2, L1 & L2 have Parity & ECC, MMU + TrustZone	1.9 DMIPS / MHz per core
		Cortex-A8	Application profile, ARM / Thumb / Thumb-2 / VFPv3 FPU / Optional NEON / Jazelle RCT and DAC, 13-stage superscalar pipeline	16-32 KB / 16-32 KB L1, 0-1 MB L2 opt ECC, MMU + TrustZone	up to 2000 (2.0 DMIPS/MHz in speed from 600 MHz to greater than 1 GHz)
		Cortex-A9 MPCore	Application profile, ARM / Thumb / Thumb-2 / DSP / Optional VFPv3 FPU / Optional NEON / Jazelle RCT and DBX, out-of-order speculative issue superscalar, 1-4 SMP cores, snoop control unit (SCU), generic interrupt controller (GIC), accelerator coherence	16-64 KB / 16-64 KB L1, 0-8 MB L2 opt Parity, MMU + TrustZone	2.5 DMIPS/MHz per core, 10,000 DMIPS @ 2 GHz on Performance Optimized TSMC 40G (dual core)

			port (ACP)		
		Cortex-A15 MPCore	Application profile, ARM / Thumb / Thumb-2 / DSP / VFPv4 FPU / NEON / Jazelle RCT / Hardware virtualization, out-of-order speculative issue SMP ۴superscalar, 1– cores, Large Physical Address Extensions (LPAE), snoop control unit (SCU), generic interrupt controller (GIC), ACP, 15-24 stage pipeline[23]	32 KB / 32 KB L1, 0-4 MB L2, L1 & L2 have Parity & ECC, MMU + TrustZone	At least 3.5 DMIPS/MHz per core (Up to 4.01 DMIPS/MHz depending on implementation).
	ARMv8-A	Cortex-A53	Application profile, AArch32 and AArch64, 1-4 SMP cores, Trustzone, NEON advanced SIMD, VFPv4, hardware virtualization, dual issue, in-order pipeline	8~64 KB/8~64 KB L1 per core, 128 KB~2 MB L2 shared, 40-bit physical addresses	2.3 DMIPS/MHz
		Cortex-A57	Application profile, AArch32 and AArch64, 1-4 SMP cores, Trustzone, NEON advanced SIMD, VFPv4, hardware virtualization, multi-issue, deeply out-of-order pipeline	48 KB/32 KB L1 per core, 512 KB~2 MB L2 shared, 44-bit physical addresses	At least 4.1 DMIPS/MHz per core (Up to 4.76 DMIPS/MHz depending on implementation).
ARM Family	**ARM Architecture**	**ARM Core**	**Feature**	**Cache (I/D), MMU**	**Typical MIPS @ MHz**

اما برخی از تولیدکنندگان مانند کوالکام، انویدیا یا اپل، با اجازه از شرکت ARM خود نسبت به طراحی هسته سفارشی بر مبنای معماری ARM اقدام می‌کنند. در جدول زیر هسته‌های طراحی شده توسط شرکت‌های دیگر که البته بر مبنای معماری یکی از خانواده‌های ARM هستند را مشاهده می‌کنید.

Family	ARM Architecture	Core	Feature	Cache (I/D), MMU	Typical MIPS @ MHz
StrongARM	ARMv4	SA-1	5 stage pipeline	16 KB/8–16 KB, MMU	203-206 MHz 1.0 DMIPS/MHz
XScale	ARMv5TE	XScale	7stage 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
Snapdragon	ARMv7-A	Scorpion	Used by some members of the Snapdragon S1, S2, and S3 families. 1 or 2 cores. ARM / Thumb	256 KB L2 per core	2.1 DMIPS / MHz per core

			/ Thumb-2 / DSP / SIMD / VFPv3 FPU / NEON (128-bit wide)		
		Krait	Used by some members of the Snapdragon S4 family. 1, 2, or 4 cores. ARM / Thumb / Thumb-2 / DSP / SIMD / VFPv4 FPU / NEON (128-bit wide)	4 KB / 4 KB L0, 16 KB / 16 KB L1, 512 KB L2 per core	3.3 DMIPS / MHz per core
Apple Ax	ARMv7-A	Apple Swift	Custom ARM core used in the Apple A6 and Apple A6X. 2 cores. ARM / Thumb / Thumb-2 / DSP / SIMD / VFPv4 FPU / NEON	L1: 32 kB instruction + 32 kB data, L2: 1 MB	3.5 DMIPS / MHz Per Core
Family	**ARM Architecture**	**Core**	**Feature**	**Cache (I/D), MMU**	**Typical MIPS @ MHz**

مقایسه ویژگی های مختلف پردازنده هایARM

پردازنده های ARM از نظر ویژگی ها و قابلیت های زیر قابل مقایسه با یکدیگر هستند:

- **حداکثر کلاک سیستم (Max Clk)**: سرعت پردازش CPU را مشخص می کند.
- **معماری دستور العمل ها (Architecture)** : به دو صورت ۳۲ بیتی یا ۶۴ بیتی هستند.
- **ویژگی های دستور العمل ها (ISA)** : ویژگی های اضافه شده نظیر Thumb و... که باعث افزایش کارایی سیستم می شود.
- **معماری حافظه (Memory sys)** : به یکی از دو صورت قدیمی (Von Neuman) و مدرن (Harvard) می باشد.
- **تعداد طبقات معماری خط لوله ای (Pipeline Stage)** : در پردازنده های مختلف بین ۳ تا ۱۳ طبقه متفاوت است.
- **میزان حافظه نهان (Cashe)** : هر چه میزان این حافظه بیشتر باشد قابلیت های پردازنده افزایش می یابد.

دو پردازنده ARM7 و Cortex M3 در جدول شکل زیر با از نظر ویژگی های مختلف با یکدیگر مقایسه شده اند.

Features	ARM7TDMI-S	Cortex-M3
Architecture	ARMv4T (von Neumann)	ARMv7-M (Harvard)
ISA Support	Thumb / ARM	Thumb / Thumb-2
Pipeline	3-Stage	3-Stage + branch speculation
Interrupts	FIQ / IRQ	NMI + 1 to 240 Physical Interrupts
Interrupt Latency	24-42 Cycles	12 Cycles
Sleep Modes	None	Integrated
Memory Protection	None	8 region Memory Protection Unit
Dhrystone	0.95 DMIPS/MHz (ARM mode)	1.25 DMIPS/MHz
Power Consumption	0.28mW/MHz	0.19mW/MHz
Area	0.62mm2 (Core Only)	0.86mm2 (Core & Peripherals)*

تفاوت پردازنده های سری Cortex با یکدیگر

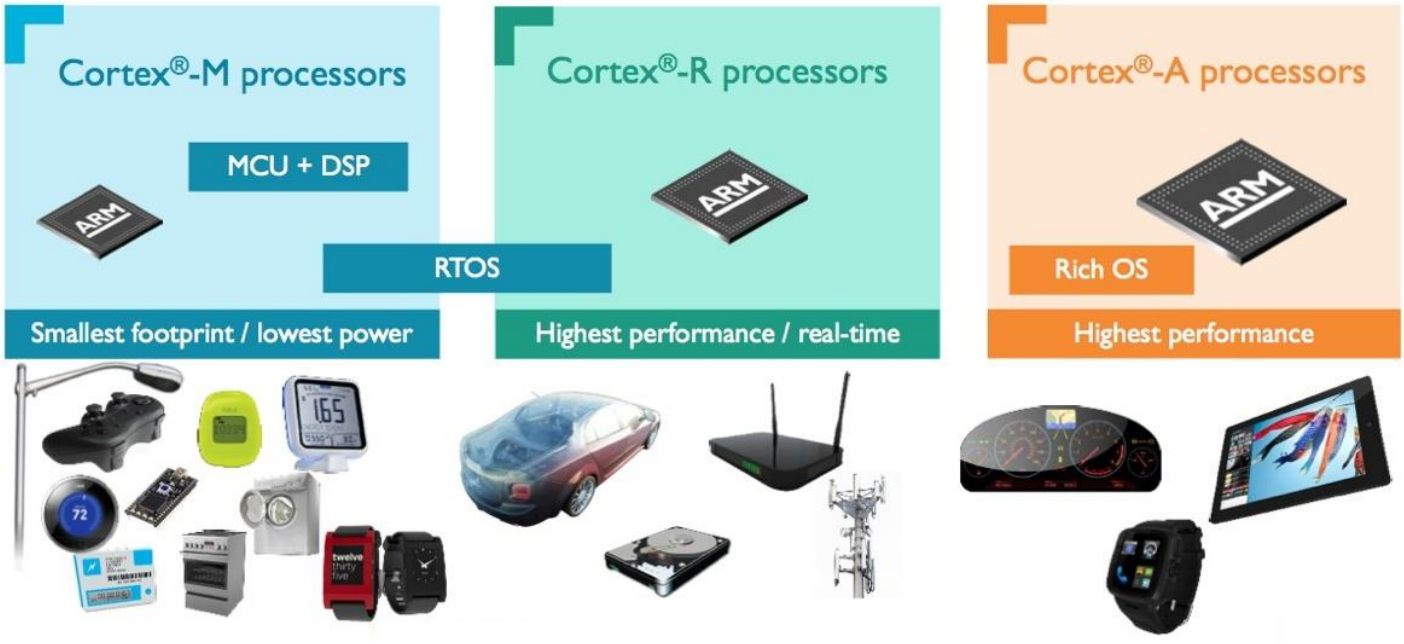
سری های مختلف پردازنده های Cortex به نام های Cortex M (مخفف Microcontroller) ، Cortex R (مخفف Real-Time) و Cortex A (مخفف Application) وجود دارند که با یکدیگر علاوه بر معماری هسته و دستورالعمل از سه دیدگاه دیگر زیر دارای تفاوت هستند :

۱. کاربرد ها

۲. مصرف توان

۳. قابلیت ها

از نظر کاربردی هر یک از سه سری پردازنده های Cortex ویژگی های خاص خود را دارند که در شکل زیر تفاوت میان آن ها را مشاهده می کنید. از این میان ، پردازنده های Cortex M در کارهای کوچکتر و با توان مصرفی بسیار پایین مورد استفاده قرار می گیرد. پردازنده های Cortex R در کارهای صنعتی و کنترلی دقیق که نیازمند پردازش سریع و به موقع (Real-Time) هستند ، مورد استفاده قرار می گیرد. همچنین پردازنده های Cortex A در کارهای گرافیکی (نمایشگر ها) و پردازش های سطح بالا و انجام اپلیکشن های مختلف مورد استفاده قرار می گیرد.



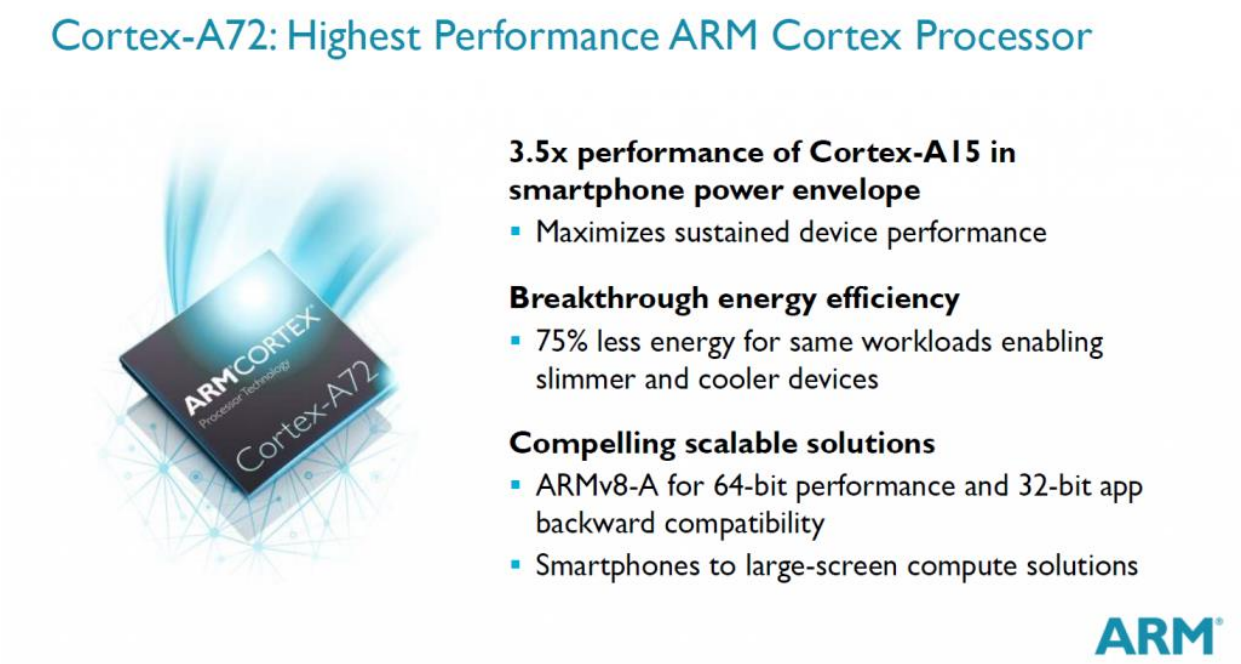
از بین سه سری تنها پردازنده های سری Cortex A است که قابلیت نصب سیستم عامل های مختلف لینوکسی و اندرویدی بر روی آن را دارد. در حالی که در سری M و R تنها میتوان سیستم عامل های RTOS نصب نمود که سرعت پایین تر و حجم کمتری دارند.

در سری Cortex R قابلیت ها و ویژگی های بیشتری نسبت به Cortex M در نظر گرفته شده است که همین مسئله باعث افزایش توان مصرفی این پردازنده های نیز می شود. این سری از پردازنده های کورتکس توانایی های زیادی دارند و عملکرد مطمئن تری در کاربرد های صنعتی و مخابراتی دارند. شکل زیر محدوده این توانایی ها را نشان می دهد.

با کنار گذاشتن موارد فنی، ARM Cortex A8 در مدل‌های یک هسته‌ای و Cortex A9 و A15 در مدل‌های چندهسته‌ای به کار گرفته می‌شود. از جمله معروفترین مواردی که پردازشگرهای ARM Cortex A9 در آن‌ها به کار گرفته شده است می‌توان به سیستم تک تراشه‌ای A5 شرکت اپل یا تگرا ۲ و ۳ شرکت انویدیا اشاره نمود.

هسته‌های ۶۴ بیتی Cortex-A57 و Cortex-A53 به نوعی حاصل تکامل هسته‌های ۳۲ بیتی Cortex-A15 و Cortex-A7 هستند و معمولاً با نوع مشابه مقایسه می‌شوند. به این صورت که A57 جانشین ۶۴ بیتی هسته‌ی A15 و A53 جانشین ۶۴ بیتی A7 است Cortex-A53 قدرت پردازشی تقریبا معادل با Cortex-A15 خواهد داشت ولی مصرف انرژی آن یک چهارم است. Cortex-A57 مصرف انرژی تقریبا معادل با Cortex-A15 دارد، در حالی که قدرت پردازشی آن به سه برابر می‌رسد.

پردازنده ۶۴ بیتی Cortex-A72 جانشین Cortex-A57 است که 3.5 برابر از نسل Cortex-A15 سریعتر می باشد و همچنین همان مقدار از انرژی را مصرف می کند Cortex-A72 از 75 درصد توان کمتر نسبت به دستگاههای مبتنی بر Cortex-A15 تحت میزان کار مشابه استفاده خواهد کرد.



شرکت های سازنده تراشه هایARM

همانطور که متوجه شدید شرکت ARM خود تولید کننده تراشه های ARM نیست بلکه تنها آن ها را طراحی می کند و سپس به دیگر شرکت های نیمه هادی می فروشد. شرکت هایی نظیر SUMSUNG ، Sony، Apple، Atmel، ST

Micro، NXP، Texas Instrumentو بسیاری کمپانی‌های دیگر به راحتی تراشه‌های خود را براساس معماری ARM تولید می‌کنند.

سیستم-روی-یک-چیپ (System on a Chip) که آن را به اختصار SoC می‌نامند در واقع یک تراشه است که در آن پردازنده اصلی(CPU) ، پردازنده گرافیک(GPU) ، حافظه رم، کنترلرهای ورودی و خروجی و بعضا کنترلر باند رادیویی قرار دارند. پس لازم است بدانید که کل SoC براساس معماری ARM تولید نمی‌شود و تنها بخش CPU آن بر مبنای معماری ARM طراحی و تولید می‌گردد. پس این باور که فلان SoC براساس معماری ARM ساخته شده، اشتباه است و بخش پردازنده اصلی اکثرSoC ها براساس یکی از طراح‌های معماری ARM ساخته می‌شوند.

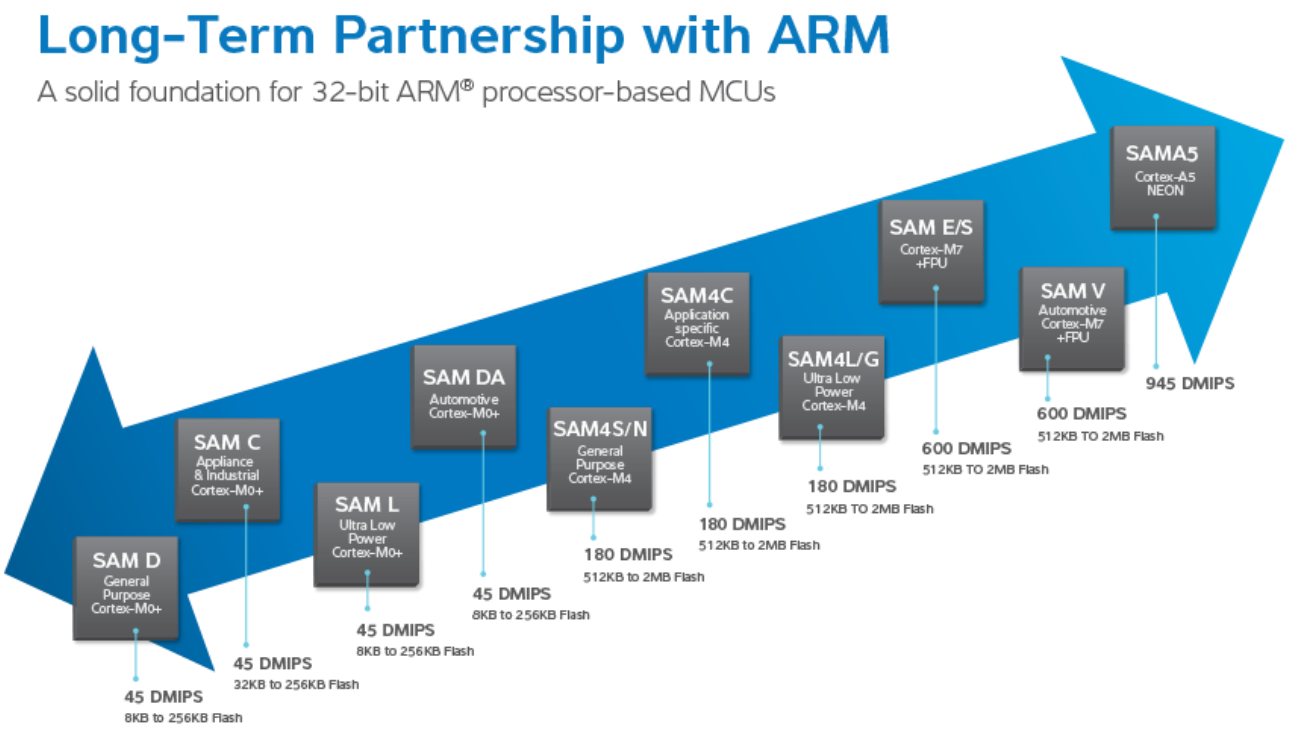
از جمله سیستم-روی-یک-چیپ‌هایی که هسته اصلی آن‌ها براساس معماری ARM طراحی شده‌اند می‌توان به ۳ نسل اول تگرا انویدیا، Quatro شرکتCSRT ، نوا شرکت اریکسون، OMAP شرکت تکزاس، Exynos شرکت سامسونگ و Ax شرکت اپل اشاره کرد. این شرکت ها از معماری ARM و همچنین معماری یکی از هسته‌های طراحی شده توسط این شرکت بهره برده‌اند.

اما شرکت‌ها می‌توانند گواهی استفاده از معماری ARM را تهیه کرده و سپس بر اساس آن هسته سفارشی مورد نظرشان را طراحی کنند یعنی به جای اینکه هسته CPU را براساس Cortex-A9 یا Cortex-A15 یا دیگر هسته‌های ARM بسازند خودشان براساس معماری یکی از خانواده‌هایARM ، هسته خاص خود را طراحی کنند. به عنوان مثال سیستم-روی-یک-چیپ A6 اپل، X-Gene، Krait کوالکام، StrongARM شرکت DEC ، XScale شرکت Marvell اینتل یا Project Denver شرکت انویدیا اینگونه هستند و اگر چه بخش CPU از سیستم-روی-یک-چیپ آنها براساس معماری ARM طراحی شده‌اند، اما طراحی هسته‌ها با آنچه ARM پیشنهاد کرده متفاوت هستند.

شرکت های سازنده میکروکنترلرهای مبتنی بر پردازندهARM

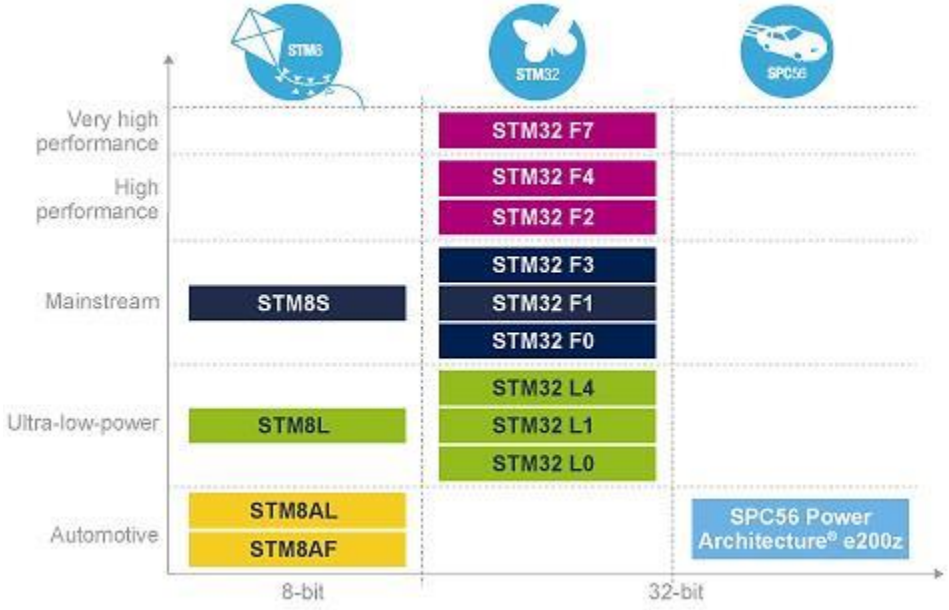
اما شرکت هایی که از هسته های پردازشی ARM در ساخت **میکروکنترلرهای** خود بهره می برند عبارتند از:

۱. **شرکت Atmel** : این شرکت که همان شرکت معروف سازنده میکروکنترلرهای AVR است ، میکروکنترلرهای ARM خود را در خانواده های متعدد ارائه می کند به طوری که میکروکنترلرهای با هسته ARM7TDMI در خانواده AT91SAM7 ، میکروکنترلرهای ARM9در خانوادهAT91SAM9 ، میکروکنترلرهای Cortex M3 در خانواده AT91SAM3 ، میکروکنترلرهای Cortex A5 در خانواده SAMA5D3 قرار می گیرند. شکل زیر میکروکنترلرهای سری جدید این شرکت را نشان می دهد.



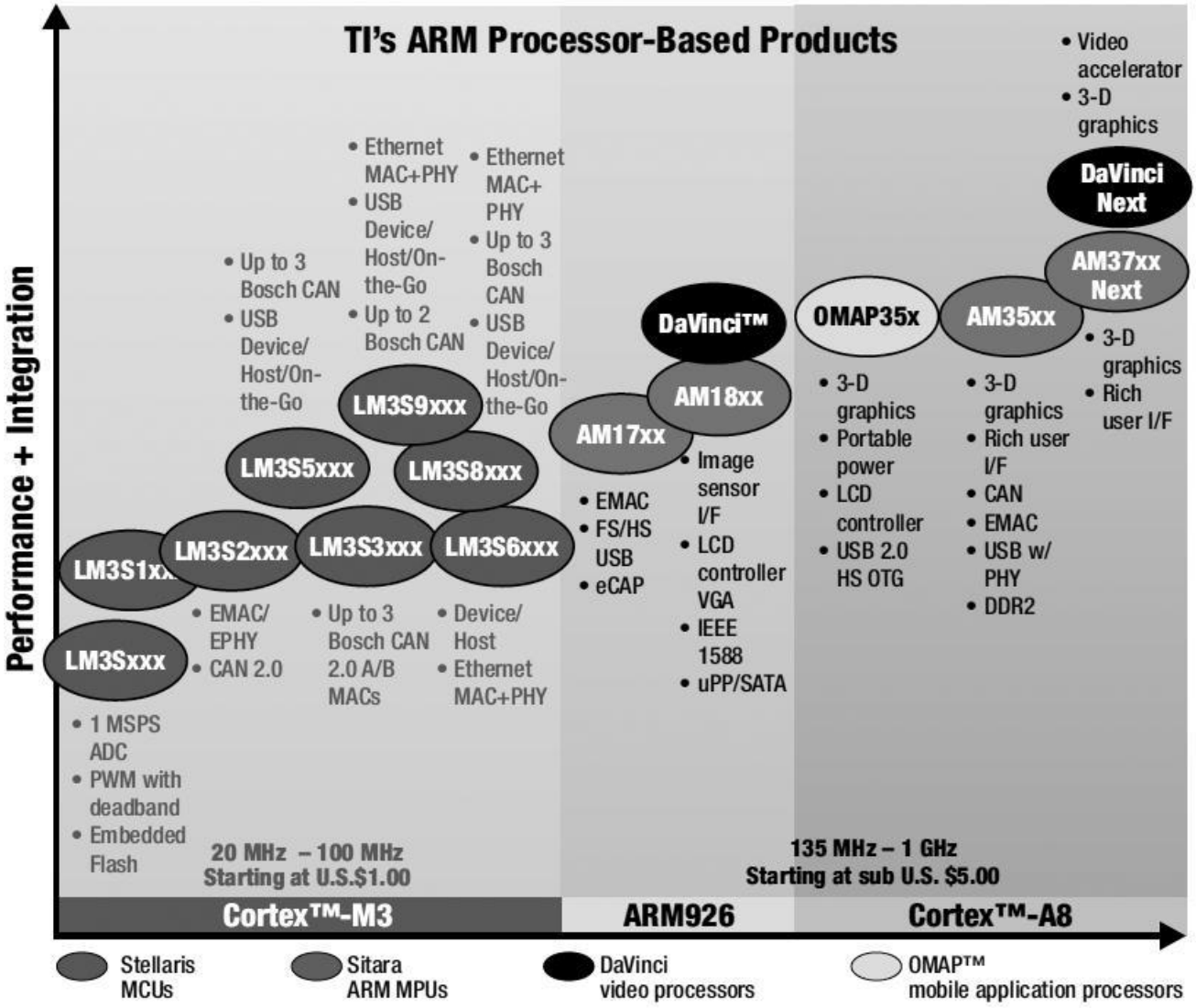
۲. **شرکت NXP** : یا همان شرکت فیلیپس می باشد که از پردازنده های ARM7TDMI ، Cortex M0/M1/M3/M4و ARM9 در ساخت میکروکنترلرهای خانواده LPC استفاده نموده است .خانواده LPC خود به سری هایLPC1000 ، LPC2000 ، LPC3000و LPC4000 تقسیم می شود که هر یک ویژگی های خاص خود را دارند .

۳. **شرکت ST Micro** : این شرکت نیز از هسته هایARM7 ، ARM9 و Cortex M0/M3/M4/M7 در ساخت میکروکنترلرهای خود استفاده می کند. شکل زیر انواع خانواده های موجود این شرکت را نشان می دهد.



میکروکنترلرهای STM32 این شرکت از قابلیت های زیادی پشتیبانی می کند که جزئیات آن در شکل زیر آورده شده است .

۴. **شرکت Texas Instrument** : این شرکت از پردازنده های Cortex M3 ، Cortex A8 و ARM9 در ساخت میکروکنترلرهای خانواده Stellaris ، Sitara، DaVinci و OMAP استفاده نموده است. شکل زیر انواع میکروکنترلرهای ARM این شرکت را نشان می دهد.



ARMv8 و پلتفرم ۶۴ بیتی

در سال ۲۰۱۱ نسل جدید ARMv8 رسما معرفی شد و پشتیبانی از معماری ۶۴ بیتی به آن اضافه گردید. در ARMv8 دستورات ۳۲ بیتی برروی سیستم‌عامل ۶۴ بیتی قابل اجرا هستند و در آن سیستم‌عامل‌های ۳۲ بیتی نیز از طریق مجازی سازی ۶۴ بیتی اجرا می‌شوند. شرکت‌های AMD, Micro, Brodom, Calxeda, Hisilicon, Samsung و ST Microelectronicsگواهی استفاده از معماری ARMv8 را دریافت کرده‌اند و اعلام نموده‌اندSoC های مبتنی بر این معماری را تولید خواهند کرد. خود ARM نیز دو طراحی Cortex-A53 و Cortex-A57 را در ۳۰ اکتبر ۲۰۱۲ معرفی کرد که هر دو مبتنی بر معماری ARMv8 هستند.

لینوکس که هسته اندروید نیز است به تازگی هسته اصلی سیستم‌عامل (Kernel) خود را بروز کرده تا از ARMv8 پشتیبانی کند. انتظار می‌رود در سال ۲۰۱۳ بسیاری از سیستم-روی-یک-چیپ‌های دنیا از معماری ARMv8 بهره ببرند.

چه سیستم‌عامل‌هایی از ARM پشتیبانی می‌کنند؟

سیستم‌های Acorn : اولین کامپیوتر مبتنی بر معماری ARM ، کامپیوتر شخصی Acorn بود که از سیستم‌عاملی به نام Arthur بهره می‌برد. سیستم‌عاملی مبتنی بر RISC OS که از معماری ARM پشتیانی می‌کرد و Acorn و برخی دیگر از تولیدکنندگان از آن استفاده می‌کردند.

سیستم‌عامل‌های توکار : معماری ARM از طیف وسیعی از سیستم‌عامل‌های توکار مانند Windows CE, Windows RT, Symbian, ChibiOS/RT, FreeRTOS, eCos, Integrity, Nucleus PLUS, MicroC/OS-II, QNX, RTEMS, CoOS, BRTOS, RTXC Quadros, ThreadX, Unison Operating System, uTasker, VxWorks, MQXو OSE پشتیبانی می‌کند.

یونیکس : یونیکس و برخی از سیستم‌عامل‌های مبتنی بر یونیکس مانند QNX ,Plan 9 ,Inferno و Solaris از ARM پشتیبانی می‌کنند.

لینوکس : بسیاری از توزیع‌های لینوکس از ARM پشتیبانی می‌کنند از آن جمله می‌توان به اندروید و کروم گوگل، Arch Linux، بادا سامسونگ، Debian، Fedora،OpenSuse، Ubuntu و WebOS اشاره کرد.

BSD : برخی از مشتقات BSD مانند OpenBSD و iOS و OS X اپل نیز از ARM پشتیبانی می‌کند.

ویندوز : معماری‌های ARMv 5, 6 و ۷ از ویندوز CE که در ابزارهای صنعتی وPDA ها استفاده می‌شود، پشتیبانی می‌کند. ویندوز RT و ویندوز فون نیز از معماری ARMv7 پشتیبانی می‌کنند.

گواهی و هزینه استفاده از معماری ARM

ARMخود تولیدکننده نیمه هادی نیست و در عوض از راه صدور مجوز استفاده از طراحی‌های خود، درآمد کسب می‌کند. گواهی استفاده از معماری ARM شرایط خاص و متنوعی را دارد و در شرایط مختلف هزینه مربوط به استفاده از آن نیز تفاوت می‌کند ARM .به همراه گواهی‌نامه خود اطلاعات جامعی در مورد نحوه یکپارچگی قسمت‌های مختلف با هسته‌ها را ارائه می‌کند تا تولیدکنندگان به راحتی بتوانند از این معماری در سیستم-روی-یک-چیپ‌های خود بهره ببرند.

ARMدر سال ۲۰۰۶ و در گزارش سالانه خود اعلام کرد که ۱۶۴۰۱ میلیون دلار از بابت حق امتیاز یا حق اختراع، درآمد داشته که این مبلغ از بابت فروش گواهی استفاده از معماری این شرکت در ۲۰۴۵ میلیارد دستگاه مبتنی بر ARM بدست آمده است. این یعنی ARM Holding بابت هر گواهی ۰۰۰۶۷ دلار درآمد کسب نموده، اما این رقم میانگین است و براساس نسل‌های مختلف و نوع هسته‌ها متفاوت خواهد بود. مثلا هسته‌های قدیمی ارزان‌تر و معماری جدید گران‌تر است.

اما در سال ۲۰۰۶ این شرکت از بابت گواهی استفاده از طراحی هسته پردازنده، نزدیک به ۱۱۹٫۵ میلیون دلار درآمد بدست آورده است. در آن سال ۶۵ پردازنده براساس معماری هسته های ARM ساخته شده بودند که به این ترتیب بابت هر گواهی پردازنده مبلغ ۱٫۸۴ میلیون دلار درآمد کسب کرده است. این عدد نیز بصورت میانگین می‌باشد و براساس نوع و نسل هسته‌ها متفاوت خواهد بود.

در واقع شرکت ARM Holding از معماری ARM دو نوع درآمد دارد یکی بابت استفاده از معماری این شرکت در ابزارهای مختلف که بابت هر تلفن یا تبلت یا هر ابزار دیگری مبلغی بدست می‌آورد و دیگری بابت هر پردازنده مبتنی بر معماری هسته‌های ARM نیز یک رقم نسبتا سنگین حدود ۲ میلیون دلار دریافت می‌کند. در سال ۲۰۰۶ نزدیک به ۶۰ درصد درآمد ARM از بابت حق امتیاز و ۴۰ درصد بابت گواهی ساخت پردازنده براساس معماری ARM بوده است.

فصل ۲ – آشنایی با انواع میکروکنترلرهای ARM شرکت NXP

مقدمه

یکی از شرکت های سازنده میکروکنترلرهای ARM ، شرکت NXP می باشد. این شرکت در حقیقت بخشی از شرکت Philips است که از سال ۲۰۰۶ تولیدات نیمه هادی خود را با برند تجاری NXP تولید و عرضه کرده است. تفاوت اصلی تولیدات ARM این شرکت با سایر شرکت ها که باعث استفاده چشمگیر آنها در محصولات صنعتی و تجاری گشته است، در طراحی سخت افزار ساده تر و در عین حال با کارایی بالاتر می باشد. این طراحی منحصر به فرد در میکروکنترلرهای NXP باعث شده است که در بسیاری از دانشگاه ها به آموزش آن پرداخته شود.

انواع میکروکنترلرهای ARM شرکتNXP

در ابتدای نامگذاری همه میکروکنترلرهای ARM شرکت NXP از واژه LPC استفاده شده است. در یک دسته بندی کلی میکروکنترلرهای LPC به ۴ خانواده تقسیم می شوند به طوری که عدد اول بعد از LPC مشخص کننده خانواده آن ها است. هر خانواده خود به چندین سری تقسیم بندی می شود که عدد بعد از خانواده مشخص کننده آن است.

۱. **خانواده LPC1xxx** : میکروکنترلرهای این خانواده دارای هسته Cortex M0 یا Cortex M3 هستند .حداکثر فرکانس کاری آن ها ۱۰۰ مگاهرتز است. در این خانواده سری های LPC13xx ، LPC15xx، LPC17xx و LPC18xx به صورت Cortex M3 و سری های LPC11xx و LPC12xx به صورت Cortex M0 وجود دارد.

۲. **خانواده LPC2xxx** : بیشتر میکروکنترلرهای این خانواده دارای هسته ARM7TDMI-S هستند که فرکانس کاری آنها حداکثر ۸۰ مگاهرتز می باشد. در این خانواده میتوان به سری های LPC21xx، LPC22xx، LPC23xx و LPC24xx اشاره کرد. البته تنها سری LPC29xx در این خانواده بر اساس ARM968E-S ساخته شده است.

۳. **خانواده LPC3xxx** : میکروکنترلرهای این خانواده که هسته ARM9EJ-S دارند ، به دو صورت ۱۶ و ۳۲ بیتی با فرکانس کاری ۲۰۰ مگاهرتز ساخته شده اند. در این خانواده سری های LPC31xx و LPC32xx وجود دارد.

۴. **خانواده LPC4xxx** : میکروکنترلرهای این خانواده که هسته Cortex M4 دارند ، مخصوص کارهای پردازش سیگنال (dsp) می باشند. در این خانواده که در حال رشد است ، تنها سری LPC43xx وجود دارد.

سری های پرکاربرد میکروکنترلرهای ARM شرکتNXP

در این قسمت به معرفی خصوصیات و ویژگی های اصلی برخی از سری های پرکاربرد ، که در بالا به آن اشاره شد ، می پردازیم.

سری LPC21xx : این میکروکنترلرها بر اساس هسته ۱۶ یا ۳۲ بیتی ARM7TDMI-S با حداکثر سرعت ۷۲ مگاهرتز ، به همراه یک حافظه فلش سرعت بالا بین ۸ تا ۵۱۲ کیلوبایت با قابلیت برنامه ریزی به صورت ISP ، به صورت معماری سنتی Von Neuman ، با دو باس محلی سرعت بالا AHB و سرعت پایین APB برای کلاک اجزای سیستم ، طراحی شده است. در این سری امکانات جانبی زیر وجود دارند.

- یک واحد مبدل ADC شش کاناله ۱۰ بیتی با سرعت نمونه برداری 4.5MSPS
- یک واحد مبدل DAC با دقت ۱۰ بیت

- دو واحد ارتباط سریالUART
- یک واحد ارتباط سریالSPI
- یک واحد ارتباط سریال همزمانSSP
- دو رابط سریالI2C
- واحد تایمر/کانتر ۳۲ بیتی
- واحد ۳۲ بیتی مولد PWM مجزا
- تایمر Whatchdog مجزا
- واحد RTC مجزا
- واحد PLL مجزا
- واحد کنترل وقفه برداری(VIC)

نکته : این سری خود به چهار سری کوچکتر LPC210x ، LPC211x، LPC213xو LPC214x تقسیم می شود.

سری LPC23xx : این میکروکنترلرها بر اساس هسته ۱۶ یا ۳۲ بیتی ARM7TDMI-S با حداکثر سرعت ۷۲ مگاهرتز ، به همراه یک حافظه فلش سرعت بالا بین ۱۲۸ تا ۵۱۲ کیلوبایت با قابلیت برنامه ریزی به صورت ISP ، به صورت معماری سنتی Von Neuman ، با دو باس محلی سرعت بالا AHB و سرعت پایین APB برای کلاک اجزای سیستم ، طراحی شده است. یکی از ویژگی های اضافه شده به این سری مدیریت بهتر روی مصرف توان آی سی بواسطه قطع تغذیه واحدهای بیکار درون آی سی می باشد. در این سری امکانات جانبی زیر وجود دارند.

- یک واحد مبدل ADC شش یا هشت کاناله ۱۰ بیتی با سرعت نمونه برداری 4.5MSPS
- یک واحد مبدل DAC با دقت ۱۰ بیت
- واحد ارتباطی سریال اترنتEthernet 10/100 Mbps
- واحد ارتباطی سریال USB به سه صورتHost/Device/OTG
- چهار واحد ارتباط سریالUART

- دو واحد ارتباط سریالCAN
- واحد ارتباط سریالSPI
- دو واحد ارتباط سریال همزمانSSP
- سه رابط سریالI2C
- رابط سریالI2S
- واحد سرعت دهنده اجرای دستورات حافظه فلش(MAM)
- دارای پورت ارتباط با کارت حافظه MMC وSD
- چهار واحد تایمر/کانتر ۳۲ بیتی
- واحد ۳۲ بیتی مولد PWM مجزا
- تایمر Whatchdog مجزا
- دارای اسیلاتور RC داخلی
- واحد RTC کاملا مجزا
- واحد PLL مجزا
- واحد کنترل وقفه برداری(VIC)

نکته : این سری خود به سه سری کوچکتر LPC236x ، LPC237xو LPC238x تقسیم می شود.

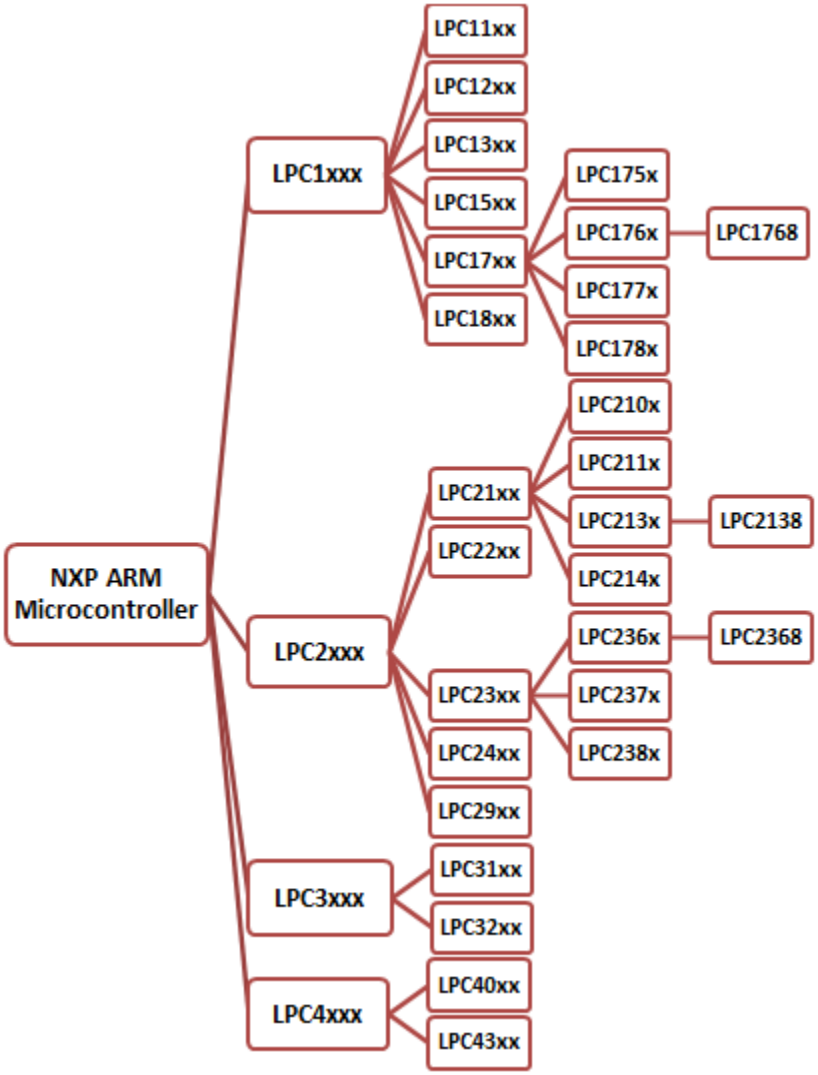
سری LPC17xx : این میکروکنترلرهای ۳۲ بیتی بر اساس هسته Cortex M3 ساخته شده اند که با حداکثر سرعت ۱۲۰ مگاهرتز ، معماری حافظه مدرن Harvard ، دارای اسیلاتور داخلی ، دو واحد PLL ، باس USB مجزا ، دو باس پرسرعت AHB و یک باس کم سرعت APB می باشند. ویژگی های این سری کاهش توان مصرفی بواسطه معماری بهبود یافته و نیز مدیریت بهتر روی مصرف توان آی سی بواسطه قطع تغذیه واحدهای بیکار درون آی سی می باشد.

- واحد ADC هشت کاناله با دقت ۱۲ بیت و نرخ نمونه برداری 4.5MSPS
- یک واحد مبدل DAC با دقت ۱۰ بیت

- واحد ارتباطی سریال اترنت Ethernet 10/100 Mbps
- واحد ارتباطی سریال USB به سه صورتHost/Device/OTG
- چهار واحد ارتباط سریالUART
- دو واحد ارتباط سریالCAN
- واحد ارتباط سریالSPI
- دو واحد ارتباط سریال همزمانSSP
- سه رابط سریالI2C
- رابط سریالI2S
- واحد شتاب دهنده حافظه فلش
- دارای واحد دسترسی همه منظوره به حافظه داده(GPDMA)
- چهار واحد تایمر/کانتر ۳۲ بیتی
- یک واحد ۳۲ بیتی مولد PWM مجزا
- دارای واحد مجزای تولید PWM مخصوص راه اندازی موتورهای سه فاز
- دارای واحد ارتباطی انکودر تربیعی(QEI)
- تایمر Whatchdog مجزا
- دارای اسیلاتور RC داخلی
- واحد RTC کاملا مجزا
- دارای دو واحد PLL مجزا یکی مخصوصUSB
- واحد کنترل وقفه برداری تو در تو(NVIC)

نکته : این سری خود به چهار سری کوچکتر LPC175x ، LPC176x، LPC177xو LPC178x تقسیم می شود.

در شکل زیر همه خانواده های شرکت NXP ، به همراه سری های پرکاربرد نشان داده شده است. توجه کنید که فقط برخی از سری ها نشان داده شده است.



سه عدد از میکروکنترلرهای معروف و پرکاربرد LPC2138 ، LPC1768 و LPC2368 در شکل فوق مشخص شده است که در بخش های بعدی با آنها کار خواهیم کرد. بهترین منابعی در رابطه با معرفی ، معماری ، ویژگی ها ، رجیسترها و راه اندازی این میکروکنترلرها ارائه شده است ، فایلی تحت عنوان User Manual است که شرکت NXP خود آن ها را تالیف و منتشر کرده است.

فصل ۳ – معرفی ، معماری و تشریح میکروکنترلرهای ARM7

مقدمه

همانطور که در فصل قبلی آموزش به آن اشاره کردیم ، میکروکنترلرهای ARM سری LPC213X ، دارای معماری ARM7TDMI-S هستند. در این بخش می خواهیم معماری این میکروکنترلرها و به طور کلی هر نوع میکروکنترلر ARM ی که دارای معماری ARM7 است را معرفی و تشریح نماییم. در این بخش مفاهیم پایه ای وجود دارد که پیش نیاز مباحث بعدی بوده و در بخش های بعدی دیگر توضیح داده نخواهند شد.

معرفی هسته های خانواده ARM7

به طور کلی خانواده ARM7 شامل هسته های پردازشی ARM7TDMI ، ARM7TDMI-S، ARM720T، ARM7EJ-S می باشد.

- ARM7TDMI: این هسته ها با حجم کم و مصرف توان پایین هستند که مناسب برای کاربردهای قابل حمل می باشند.

- ARM7TDMI-S: این هسته ورژن قابل سنتز (synthesizable) یا کد شده هسته قبلی است که در هر دو زبان Verilogو VHDL موجود بوده و برای شرکت هایی که از کد برای تولید تراشه های خود استفاده می کنند بسیار راحت تر است.

توضیح اینکه معمولا شرکت ها ترجیح می دهند برای کاهش زمان تولید به مصرف و افزایش انعطاف پذیری محصولات خود به جای پیاده سازی آنها در سطح ترانزیستوری ، از کدهای توصیف سخت افزاری و پیاده سازی آنها روی ماکروسل های آماده (همانند FPGA) استفاده نمایند.

نتیجه : دو هسته ARM7TDMI و ARM7TDMI-S از نظر عملکرد مداری تفاوتی با یکدیگر ندارند و فقط نوع توصیف و پیاده سازی آنها با یکدیگر تفاوت دارد.

- ARM720T: همان هسته قبلی به همراه 8Kb حافظه Cashe و واحد مدیریت حافظه MMU (مخفف Memory Management Unit) به منظور نصب سیستم عامل های ساده ای همچون WinCE، Symbian می باشد.

- ARM7EJ-S: این هسته تمام مزیت های هسته ARM7TDMI-S نظیر حجم کم ، مصرف توان پایین و قابل سنتز بودن را دارد ضمن اینکه قابلیت های تکنولوژی Jazlle مخصوص کاربردهای Jave و مجموعه بسط های DSP مخصوص کارهای پردازش سیگنال به آن اضافه شده است.

معرفی هسته پردازشیARM7TDMI

هسته ARM7TDMI بر پایه معماری سنتی Von Neuman با پهنای باس ۳۲ بیت برای دیتا و دستور العمل می باشند. دیتای ذخیره شده در حافظه میتواند ۸ ، ۱۶ یا ۳۲ بیتی باشد. علت نامگذاری این معماری این است که هسته پردازشی این میکروکنترلرها علاوه بر ARM7 بودن ، دارای ویژگی های دیگری نیز می باشد که در شکل زیر آورده شده است:

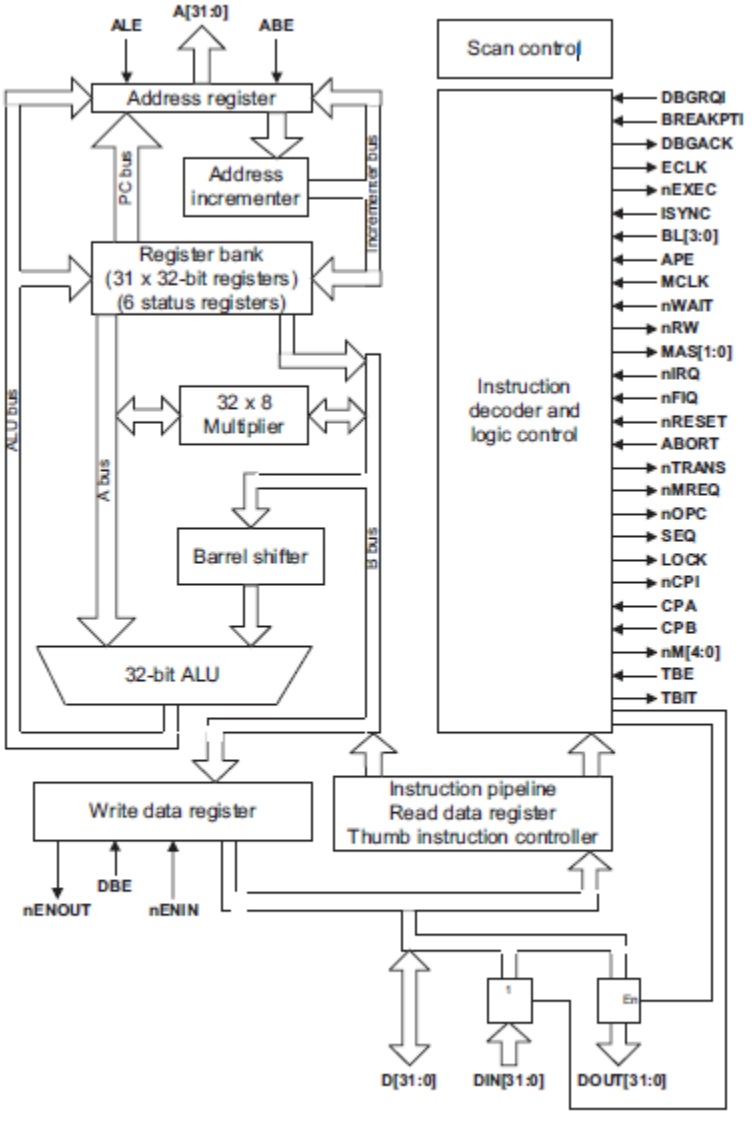
در این معماری سه مرحله Pipelining به صورت Fetch ، Decode و Execute وجود دارد. دستورات میکروکنترلر از یک حافظه Flash خط به خط اجرا می شود. این دستورات که به صورت ARM V4T می باشند ، میتوانند به یکی از دو صورت ARM و Thumb اجرا شوند. در مد ARM دستورات به صورت ۳۲ بیتی خواهند بود که حداکثر قابلیت ها با کمترین تعداد دستور العمل را محقق می سازد. در مد Thumb دستورات به صورت ۱۶ بیتی و ساده تر می باشند که در عوض این سادگی ، توان مصرفی آی سی و حجم کدهای حافظه کاهش می یابد.

نکته : در این آموزش ما فقط در مد ARM کار می کنیم زیرا در مد Thumb کلیه دستورات و داده ها ۱۶ بیتی می شوند و تمامی عملکرد میکروکنترلر ۱۶ بیتی می گردد.

واحد های اصلی یک میکروکنترلر ARM7

در یک میکروکنترلر که بر مبنای معماری ARM7 طراحی شده است ، یک سری واحد های اصلی وجود دارد که با معماری ARM7 در کنار هم قرار گرفته اند. این واحد های اصلی در همه میکروکنترلرهای ARM7 وجود دارد. اما واحدهای جانبی دیگری نیز در کنار این واحدهای اصلی وجود دارند که در میکروکنترلرهای مختلف متفاوت هستند. این واحد های اصلی عبارتند از:

- **CPU** : هسته اصلی پردازشی که خود شامل ۳۱ رجیستر همه منظوره ۳۲ بیتی (Register Bank) ، واحد ALU، واحد کنترل و ... می باشد. شکل زیر اجزای داخلی هسته ARM7 را به صورت بلوک دیاگرام نشان می دهد.



ARM7TDMI core diagram

- **واحد JTAG** : در کنار هسته اصلی یک رابط سریال JTAG به منظور عیب یابی سیستم تعبیه شده است.

- **واحد حافظه Memory** : که شامل حافظه SRAM برای ذخیره موقت داده ها و حافظه Flash که برای ذخیره دائمی برنامه و داده مورد استفاده قرار می گیرد.

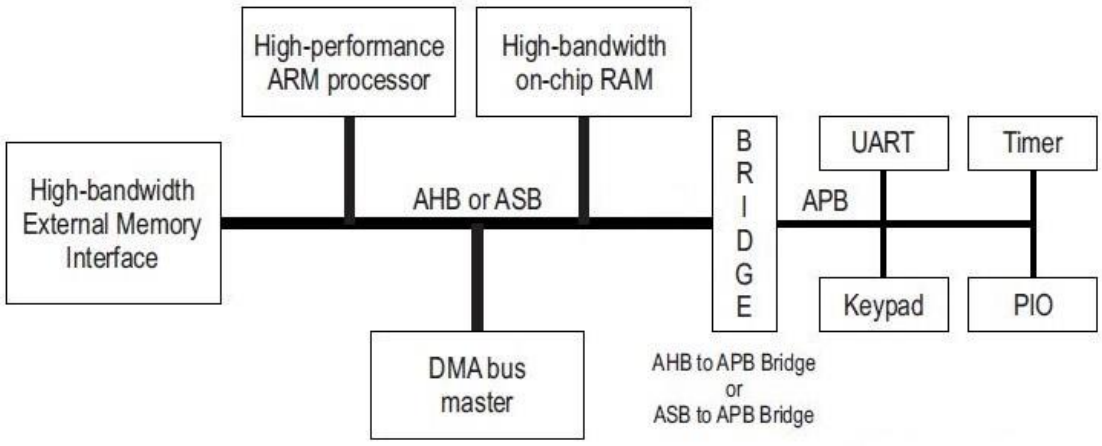
- **واحد AMBA یا باس بهبود یافته :** در معماری هسته های ARM7 از معماری باس بهبود یافته یا AMBA (مخفف Advanced Microcontroller Bus Architecture) استفاده می شود. در حقیقت AMBA یک باس استاندارد است که مدیریت اتصالات ماژول های مختلف درون سیستم را بر عهده دارد. در این معماری سه Bus با سرعت های مختلف به صورت زیر وجود دارد:

۱. باس ASB (مخفف Advanced System Bus)

۲. باس AHB (مخفف Advanced High-Performance Bus)

۳. باس APB (مخفف Advanced Peripheral Bus)

دو باس AHB و ASB به منظور اتصال ماژول های سرعت بالا نظیر CPU و Memory به کار می رود. ماژول های سرعت پایین تر به باس APB متصل می شوند. شکل زیر این موضوع را نشان می دهد.



همانطور که در شکل فوق مشاهده می کنید ، هسته اصلی (CPU) و کلیه ماژول های سرعت بالا نظیر Memory به باس پرسرعت AHB یا ASB متصل می شود. واحد های جانبی دیگر میکروکنترلر نظیر واحد ورودی/خروجی ، تایمر ، UARTو ... همگی به باس کم سرعت APB متصل می شوند. واحد APB Bridge یک تقسیم کننده می باشد که میتواند با ضرایب قابل برنامه ریزی ۱ ، ۲ و ۴ فرکانس AHB را تقسیم کند.

معرفی ویژگی های میکروکنترلرهای سریLPC213X

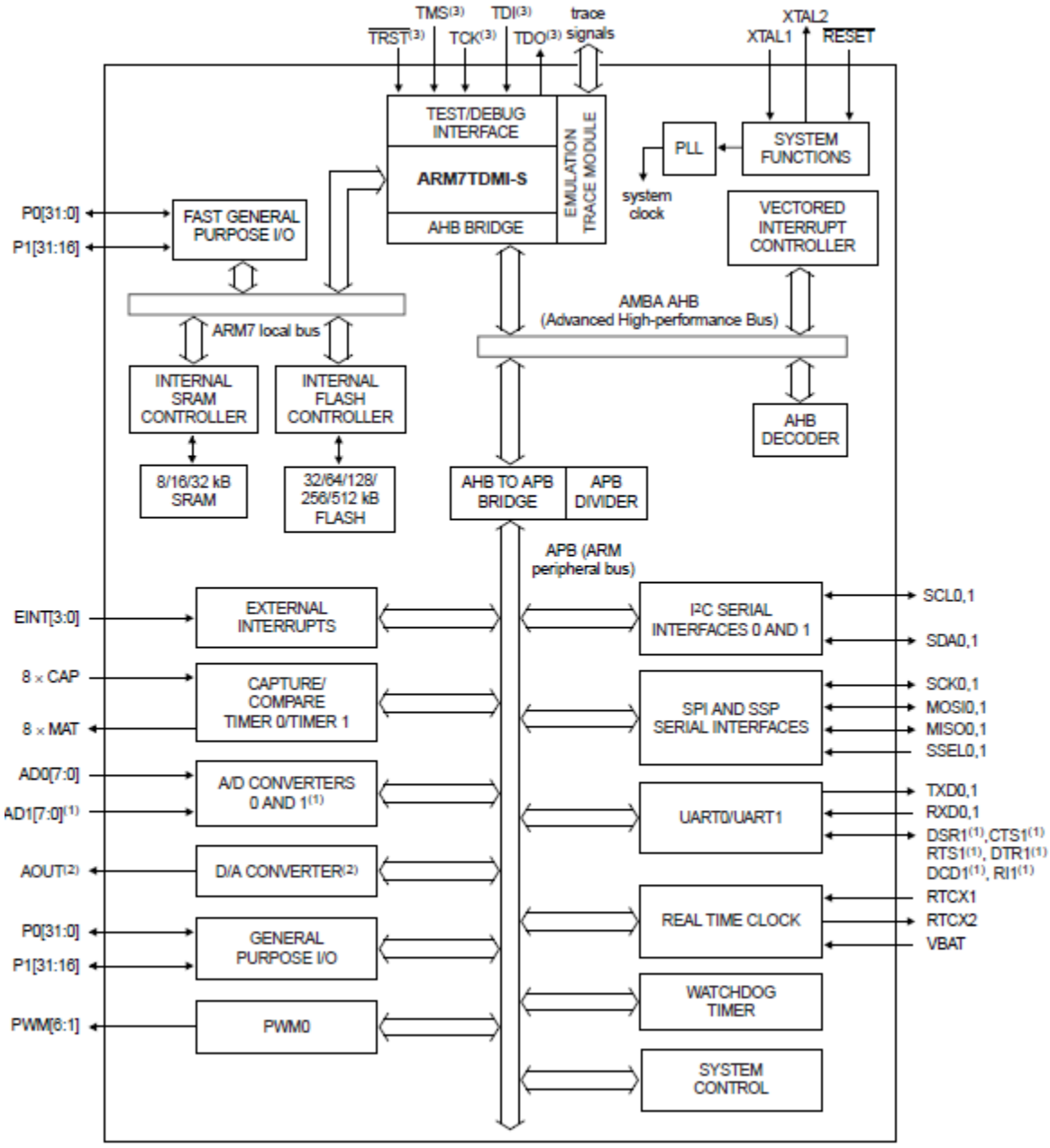
- یک میکروکنترلر ARM7TDMI به صورت ۱۶/۳۲ بیتی در بسته بندی کوچک ۶۴ پایه
- دارای حافظه SRAM داخلی ۳۲/۱۶/۸ کیلوبایت

- دارای حافظه FLASH داخلی ۵۱۲/۲۵۶/۱۲۸/۶۴/۳۲ کیلوبایت
- دارای امکان ذخیره سریع در حافظه به صورت ۱۲۸ بیتی در فرکانس ۶۰ مگاهرتز
- توانایی پروگرام شدن به دو صورت ISP (درون سیستم) و IAP (درون برنامه)
- توانایی پاک شدن سریع کل حافظه یا قسمتی از آن در ۴۰۰ میلی ثانیه
- توانایی پروگرام شدن ۲۵۶ بایت از حافظه ظرف مدت ۱ میلی ثانیه
- دارای رابط EMbedded ICE و Embedded Trace که به منظور عیب یابی برنامه به صورت آنلاین و توسط رابط JTAG انجام می گیرد.
- یک واحد مبدل A/D برای LPC2131/2 و دو واحد مبدل A/D برای LPC2134/6/8 که هریک ۸ کاناله و ۱۰ بیتی هستند و زمان تبدیل کمتر از ۲،۴۴ میکروثانیه بر کانال می باشد.
- یک واحد مبدل ۱۰ بیتی D/A که میتواند ولتاژهای مختلف آنالوگ را تولید کند (در LPC2131 این واحد وجود ندارد)
- دو واحد تایمر/کانتر ۳۲ بیتی با چهار پایه Capture (تسخیر) و چهار پایه Compare (مقایسه)
- واحد PWM با ۶ خروجی
- واحد تایمر Watchdog
- واحد RTC با مصرف توان پایین
- دارای واحدهای سریال UART ، I2C، SPI وSSP
- دارای واحد کنترل وقفه برداری و با اولویت بندی قابل تنظیم و آدرس برداری
- دارای ۴۷ پایه ورودی/خروجی که قابلیت تحمل ۵ ولت را دارند
- دارای ۹ پایه به منظور وقفه خارجی به صورت حساس به لبه یا پالس
- دارای حداکثر سرعت کلاک ۶۰ مگاهرتز زمانی که واحد PLL روی حداکثر تنظیم شده باشد
- دارای واحد اسیلاتور داخلی که با اتصال کریستال بین ۱ تا ۳۰ مگاهرتز کار می کند.
- دارای مدهای کاهش توان Idle و Power Down که قابلیت بیدار شدن از این حالت ها توسط وقفه خارجی یا واحد RTC محیا شده است .

- دارای مدارهای داخلی Power On Reset (ریست در زمان روشن شدن) و Brown Out Detection (ریست شدن میکرو در ولتاژ خاص)
- ولتاژ عملکرد بین ۳ تا ۳،۵ ولت (۳،۳ ولت)

معماری و ساختار میکروکنترلرLPC213X

شکل زیر بلوک دیاگرام این سری از میکروکنترلرهای شرکت NXP را نشان می دهد.



همانطور که در شکل فوق مشاهده می کنید ، هر سه باس AMBA در این میکروکنترلر وجود دارد. به طوری که واحد حافظه (SRAM و FLASH) و واحد ورودی/خروجی همه منظوره سریع (FAST GPIO) به باس ASB متصل شده اند. واحد کنترل وقفه برداری (VIC) نیز به باس AHB و دیگر واحد ها به باس APB متصل شده اند.

نکته : باس ASB و باس AHB دارای سرعت یکسان هستند اما باس APB بسته به ضرایب ۱ ، ۲ یا ۴ میتواند تقسیمی از آن یا خود آن باشد.

نکته : پایه هایی که روی شکل فوق با عدد (۱) کنار آن نشان داده شده است در LPC2131 وجود ندارند.

نکته : پایه هایی که روی شکل فوق با عدد (۲) کنار آن نشان داده شده است در LPC2131 و LPC2132 وجود ندارند.

نکته : پایه های JTAG که در شکل فوق با عدد (۳) کنار آن نشان داده شده است ، با پایه های واحد ورودی/خروجی همه منظوره (GPIO) به صورت مشترک قرار دارند.

واحد حافظه

این واحد شامل یک حافظه SRAM و یک حافظه Flash و کنترلر مخصوص هر یک می باشد و میتواند هر دوی آنها همزمان به عنوان حافظه داده و و حافظه برنامه استفاده شوند. حافظه فلش بسته به نوع میکرو یکی از مقادیر ۵۱۲/۲۵۶/۱۲۸/۶۴/۳۲ کیلوبایت و حافظه SRAM یکی از مقادیر ۳۲/۱۶/۸ کیلوبایت می باشد. پروگرام شدن حافظه فلش از طریق یکی از دوحالت زیر امکان پذیر است:

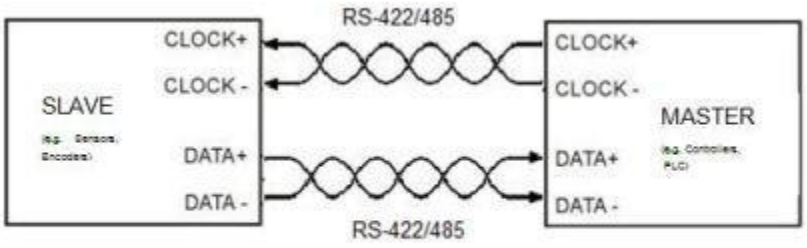
- توسط رابط JTAG و پروگرامرJlink
- توسط رابط UART0 و از طریق پورت سریال

واحد حلقه قفل فاز (PLL)

این واحد که در درون میکروکنترلر قرار دارد ، پالس تولید شده توسط واحد کلاک میکرو را دریافت کرده و فرکانس آن پالس را چند برابر می کند. سپس این پالس را که چند برابر فرکانس کلاک (بین ۲ تا ۵ برابر) است در اختیار CPU و واحدهای جانبی دیگر قرار می دهد. با افزایش فرکانس ، سرعت اجرای دستورات نیز افزایش می یابد. همچنین این واحد امکان مدیریت فرکانس کلاک در زمان اجرای برنامه را می دهد. به طوری که با کاهش ضریب چند برابر شدن کلاک ، توان مصرفی توسط آی سی به طرز چشمگیری کاهش می یابد.

واحدSSP

SSPکه مخفف Synchronous Serial Port می باشد ، یک پروتکل ارتباطی سریال همانند SPI می باشد که در کاربردهای صنعتی میان یک Master و یک Slave میتواند دیتا را به صورت سنکرون و با سرعت بالا جابجا نماید. این پروتکل معروف برای ارتباط میکرو با سنسورهای صنعتی ، بر اساس استاندارد RS422 عمل می کند که خاصیت طولانی بودن مسیر انتقال را به پروتکل SPI به ارمغان می آورد. شکل زیر نحوه ارتباط توسط این پروتکل را نشان می دهد.



SSI BLOCK DIAGRAM

منابع تولید کلاک درLPC213X

۱. **منبع کلاک خارجی :** اتصال کلاک آماده به پایه XTAL1 و آزاد گذاشتن پایهXTAL2

۲. **کریستال خارجی :** اتصال کریستال بین ۱ تا ۳۰ مگاهرتز به همراه دو عدد خازن.

مقایسه تفاوت های کلی بین میکروکنترلرهای AVR وARM7

- در میکروکنترلرهای AVR ، حافظه EEPROM نیز وجود دارد اما در میکروکنترلرهایARM7 ، فقط حافظه های SRAMو Flash وجود دارند.
- در میکروکنترلرهای AVR ، پایه ها میتوانند به صورت داخلی PullUp شوند اما در میکروکنترلرهای ARM7 این قابلیت وجود ندارد.
- حداکثر سرعت پردازش در میکروکنترلرهای AVR ، برابر ۱۶ مگاهرتز بود که در میکروکنترلرهای ARM7 به ۶۰ مگاهرتز افزایش یافته است.
- در میکروکنترلرهای AVR یک باس داده و یک باس برنامه وجود داشت (معماری هاروارد) اما در میکروکنترلرهای ARM7معماری سنتی است اما در عوض یک معماری باس بهبود یافته AMBA اضافه شده است.

- منابع کلاک در AVR میتوانست به صورت های مختلف از جمله کریستال داخلی باشد اما در میکروکنترلرهای ARM7 این قابلیت وجود ندارد و فقط میتوان از کریستال خارجی یا کلاک خارجی استفاده نمود.

- واحد کنترل وقفه در AVR بسیار ساده بود اما در ARM7 واحد کنترل وقفه برداری ، با قابلیت ها و پیچیدگی های بیشتر وجود دارد.

- در میکروکنترلرهای AVR میتوانستیم به صورت ISP و از طریق رابط SPI یا از طریق رابط JTAG میکروکنترلر را پروگرام کنیم اما در میکروکنترلرهای ARM7 علاوه بر استفاده از رابط JTAG برای پروگرام کردن از رابط سریال UART و به صورت های ISP و IAP استفاده می شود.

- واحد مقایسه کننده آنالوگ به طور کلی در میکروکنترلرهای ARM7 وجود ندارد.

- پروتکل جدید سریال SSP در میکروکنترلرهای ARM7 وجود دارد.

- واحد جدید PLL برای افزایش فرکانس کلاک در میکروکنترلرهای ARM7 وجود دارد.

- در هردو میکروکنترلر AVR و ARM7 از سه خط لوله Pipelining استفاده شده است.

- در میکروکنترلرهای AVR واحد USART وجود داشت که به صورت سنکرون یا آسنکرون عمل می کرد اما در ARM7 فقط آسنکرن وجود دارد.

- تعداد واحد های جانبی نظیر USART و I2C در میکروکنترلرهای AVR حداکثر یک بود اما در میکروکنترلرهای ARM7تعداد آنها حداکثر دو می باشد.

- در میکروکنترلرهای AVR راه اندازی PWM و RTC جزئی از واحد تایمر/کانتر بود اما در میکروکنترلرهای ARM7 هر یک واحدی جداگانه دارند.

- در میکروکنترلرهای AVR ، حداکثر قدرت واحد تایمر/کانتر یک واحد و آن هم ۱۶ بیتی بود اما در میکروکنترلرهای ARM7دو واحد ۳۲ بیتی وجود دارد.

- در میکروکنترلرهای AVR واحد تایمر/کانتر دارای مدهای مختلف است اما در میکروکنترلرهای ARM7 فقط یک مد تسخیر و یک مد مقایسه وجود دارد.

- و...

فصل ۴ – اصول راه اندازی میکروکنترلرهای سری LPC213X

مقدمه

در قسمت قبل با معماری میکروکنترلرهای ARM7 و سری LPC213x آشنا شدیم. اما معمولا میکروکنترلرهای Cortex M3 ، یعنی سری LPC176x در پروژه ها به علت قابلیت های بیشتر و سرعت بالاتر ترجیح داده می شود. با این حال ویژگی های مفیدی ، باعث می شود که ابتدا میکروکنترلرهای سری ARM7 را یادبگیریم. یکی از این ویژگی ها ، ساده تر بودن کار با آن هاست به طوری که در هنگام برنامه نویسی با تعداد رجیسترهای کمتری سر و کار داریم و نیز دسترسی به رجیسترها راحت تر است. ویژگی دیگر پشتیبانی کردن نرم افزار پروتئوس از میکروکنترلرهای ARM7 است. به طوری که میتوان در این نرم افزار میکروکنترلرهای سری LPC213x را شبیه سازی نمود. (نرم افزار پروتئوس سری LPC176x یعنی Cortex M3 را پشتیبانی نمی کند) . بنابراین این ویژگی ها میتواند کمک کند تا به طور ملموس تر و راحت تر میکروکنترلرهای ARM را فرا گرفته و ضمن یادگیری مفاهیم به طور عمیق تر ، با برنامه نویسی راحت تر و شبیه سازی بیشتر جلو برویم.

60-MHz, 32-bit microcontroller with ARM7TDMI-S™ core LPC213x

ARM7-based microcontrollers with two 10-bit ADCs and 10-bit DAC

مقایسه بین میکروکنترلرهای سریLPC213X

همانطور که در جدول زیر مشاهده می کنید ، در این سری ۵ میکروکنترلر LPC2131, LPC2132, LPC2134, LPC2136 و LPC2138 وجود دارد و ۵ میکروکنترلر دیگر به صورت LPC213X/01 نیز در کنار آن مشاهده می شود. میکروکنترلرهایی که به آن پسوند 01 اضافه شده است ، نسخه جدیدتر و با قابلیت های بهبود یافته ی نسخه بدون پسوند می باشد.

Type	Memory		Serial interfaces			ADC/DAC options		Enhanced UARTs, ADC, Fast I/Os, and BOD	Packages
	Flash (KB)	SRAM (KB)	I²C	UART	SPI and SSP	ADC channels (10-bit)	DAC channels (10-bit)		
LPC2131	32	8	2	2	1	8			LQFP64
LPC2131/01	32	8	2	2	1	8		•	LQFP64
LPC2132	64	16	2	2	1	8	1		LQFP64,HVQFN64
LPC2132/01	64	16	2	2	1	8	1	•	LQFP64,HVQFN64
LPC2134	128	16	2	2	1	16	1		LQFP64
LPC2134/01	128	16	2	2	1	16	1	•	LQFP64
LPC2136	256	32	2	2	1	16	1		LQFP64
LPC2136/01	256	32	2	2	1	16	1	•	LQFP64
LPC2138	512	32	2	2	1	16	1		LQFP64,HVQFN64
LPC2138/01	512	32	2	2	1	16	1	•	LQFP64,HVQFN64

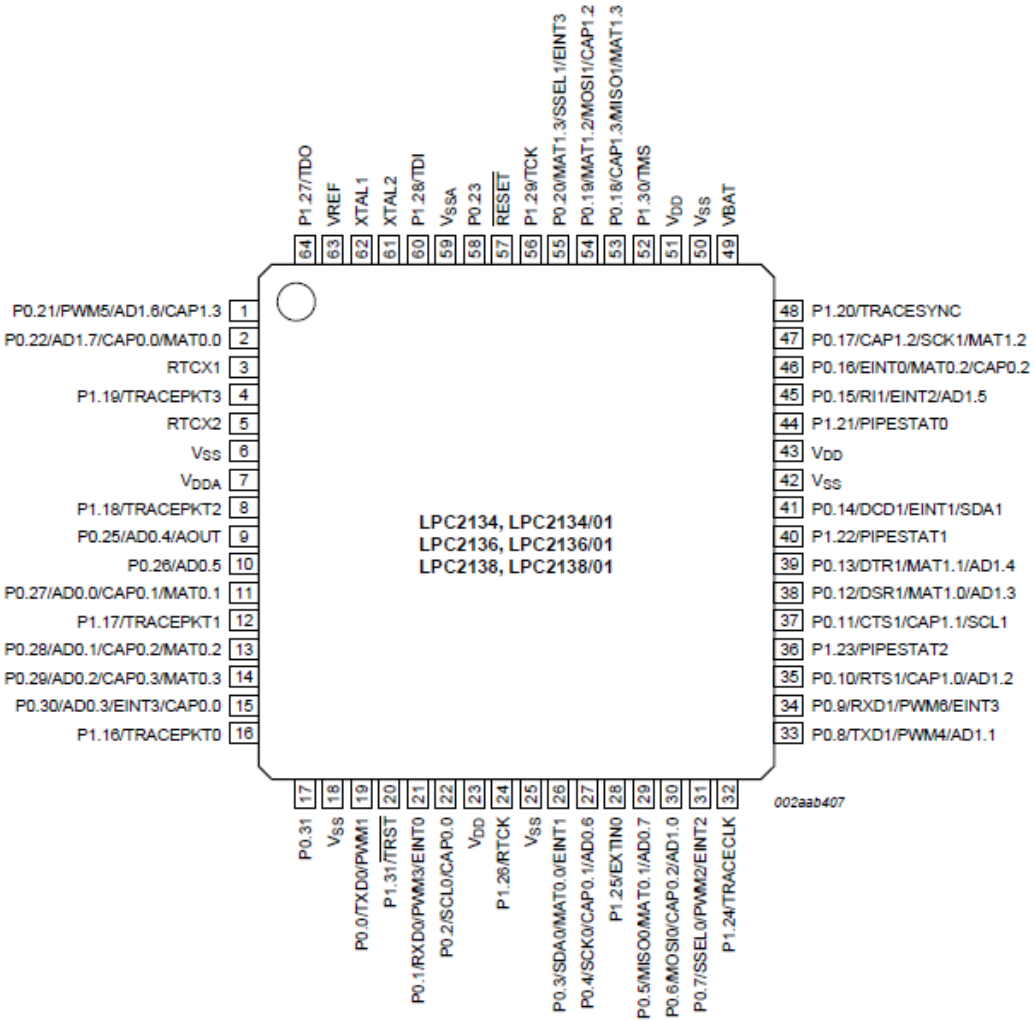
به طور کلی بهبود هایی که در LPC213X/01 وجود دارند عبارت است از:

- سریع تر شدن قابلیت تغییر در پورت های GPIO تا 3.5 برابر
- تخصیص رجیستری برای ذخیره نتایج مبدل هایAnalog/Digital
- اضافه شدن مولد BuadRate بهUART0/1 ، قابلیت تشخیص خودکار نرخ ارسال/دریافت (AutoBaud) و اضافه شدن قابلیت کنترل به صورت HandShake در آن
- بهبود یافتن BrownOutDetection و رجیستر کنترل توان مصرفی(PCON)

تذکر : هر کجا که میگوییم LPC213X منظور همان آخرین نسخه یعنی LPC213X/01 است.

پایه های میکروکنترلرLPC2138

از میان میکروکنترلرهای سری LPC213x ، میکروی دارای بیشترین قابلیت یعنی LPC2138 انتخاب می شود. این میکروکنترلر در بسته بندی TQFP و ۶۴ پایه عرضه می شود. پایه های این میکروکنترلر را در شکل زیر مشاهده می کنید.



همانطور که در شکل فوق مشاهده می کنید ، میکرو LPC2138 دارای دو پورت همه منظوره ورودی/خروجی (GPIO) می باشد که به صورت P0 و P1 نامگذاری شده است. عملکردهای دیگر هر پایه که مربوط به دیگر واحدهای میکروکنترلر می باشد نیز در شکل مشخص شده است.

چند نکته مهم:

- همانطور که مشاهده می کنید هر یک از پایه های حداقل **یک** و حداکثر **چهار** عملکرد مختلف دارند که به طور همزمان فقط از یک عملکرد آن استفاده می گردد.

- پهنای داده ، رجیستر ها و پورت های GPIO در میکروکنترلر ۳۲ بیتی هستند اما به علت زیاد شدن پایه های خروجی پورت ها ، فقط برخی از پایه های هر پورت از آی سی بیرون آمده و در دسترس است. بقیه آنها به صورت نرم افزاری (در برنامه نویسی) در دسترس می باشند.

- تمامی پایه های مربوت به پورت صفرام (P0) از P0.0 تا P0.31 به جز P0.24 در دسترس است. (پایه P0.24 وجود ندارد)

- در پورت یکم (P1) فقط ۱۶ پایه دوم یعنی پایه های P0.16 تا P0.31 بیرون میکرو در دسترس هستند و بقیه به صورت نرم افزاری وجود دارند. (پایه های P0.0 تا P0.15 وجود ندارد)

تشریح پایه های LPC2138

تغذیه دیجیتال آی سی : پایه هایی که با نام VDD وجود دارند تغذیه مثبت آی سی و پایه های VSS تغذیه منفی (زمین) آی سی می باشند.

پایه ریست : پایه RST به عنوان پایه ریست آی سی می باشد که به صورت Active Low عمل می کند و همانند میکروکنترلرهای AVR باید به مدار Power On Reset متصل شود.

کریستال خارجی آی سی : پایه های XTAL1 و XTAL2 به منظور اتصال کریستال خارجی (کلاک اصلی میکرو) می باشد.

کریستال واحد RTC : این آی سی دارای واحد RTC مجزا است که برای فعالسازی آن نیاز به اتصال کریستال ساعت به پایه های RTXC1 و RTXC2 می باشد. همچنین پایه VBAT ، تغذیه مثبت واحد RTC است که به منظور فعال بودن دائمی این واحد به باتری پشتیبان (Backup Battery) متصل می گردد.

واحد GPIO : تمام پایه هایی که به نام P0.X و P1.X هستند ، میتواند از آنها به عنوان پورت ورودی/خروجی استفاده شود. X میتواند عددی بین ۰ تا ۳۱ باشد.

واحد ADC : تمام پایه هایی که به نام AD0.X و AD1.X هستند ، میتوانند به عنوان ورودی واحد آنالوگ به دیجیتال استفاده شوند. X میتواند عددی بین ۰ تا ۷ باشد.

واحد DAC : پایه AOUT مربوط به خروجی آنالوگ واحد DAC می باشد.

پایه های تغذیه آنالوگ آی سی (تغذیه واحدهای ADC و DAC) : شامل پایه های VREF (ولتاژ مرجع) ، VDDA تغذیه مثبت آنالوگ و VSSA تغذیه منفی آنالوگ می باشند.

واحد UART0 : پایه های RXD0 وTXD0

واحد UART1 : پایه های RXD1 ، TXD1 ، CTS1 ، DCD1 ، DSR1 ، DTR1 ، RI1 ، RTS1 مربوط به واحد ارتباط سریال می باشد. پایه هایی که اضافه بر RXD1 و TXD1 برای این واحد وجود دارد مربوط به قابلیت ارتباط دو طرفه با اطمینان بالا (Handshaking) می باشد.

واحد SPI0 : پایه های SCK0 ، SSEL0 ، MISO0 و MOSI0

واحد SPI1/SSP : پایه های SCK1 ، SSEL1 ، MISO1 و MOSI1 که قابلیت برقراری هر دو پروتکل ارتباط سریال SPI و SSP را دارد.

واحد I2C0 : پایه های SDA0 و SCL0

واحد I2C1 : پایه های SDA1 و SCL1

واحد Timer : تمام پایه های به نام های CAP0.X و CAP1.X به عنوان ورودی واحد تایمر (کانتر) به کار می رود که در آن ها X عددی بین ۰ تا ۳ می باشد. همچنین تمام پایه های به نام های MAT0.X و MAT1.X به عنوان خروجی واحد تایمر به کار می رود که در آن X عددی بین ۰ تا ۳ می باشد.

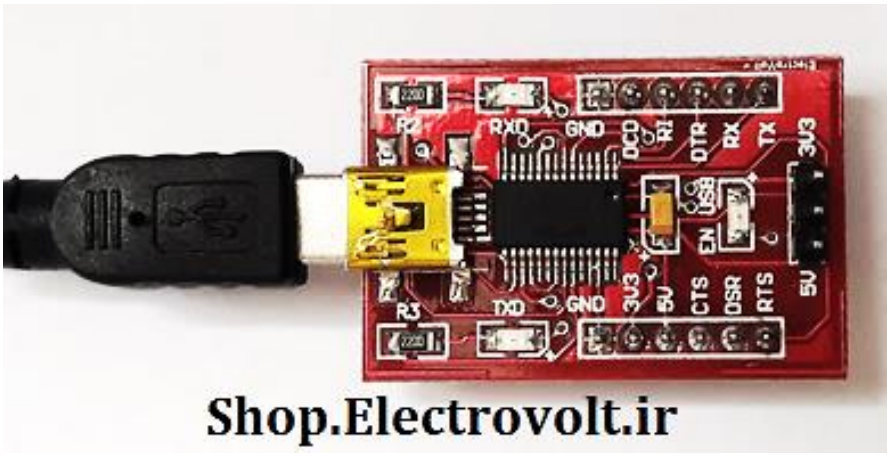
واحد PWM : تمام پایه های به نام های PWMX که در آن X بین ۱ تا ۶ می باشد ، مربوط به واحد مجزای PWM است.

واحد JTAG : پایه هایی که به نام های TRST ، TMS، RTCK، TDI، TDO، TCKوجود دارند ، مربوط به واحد برنامه ریزی و عیب یابی میکرو می باشند.

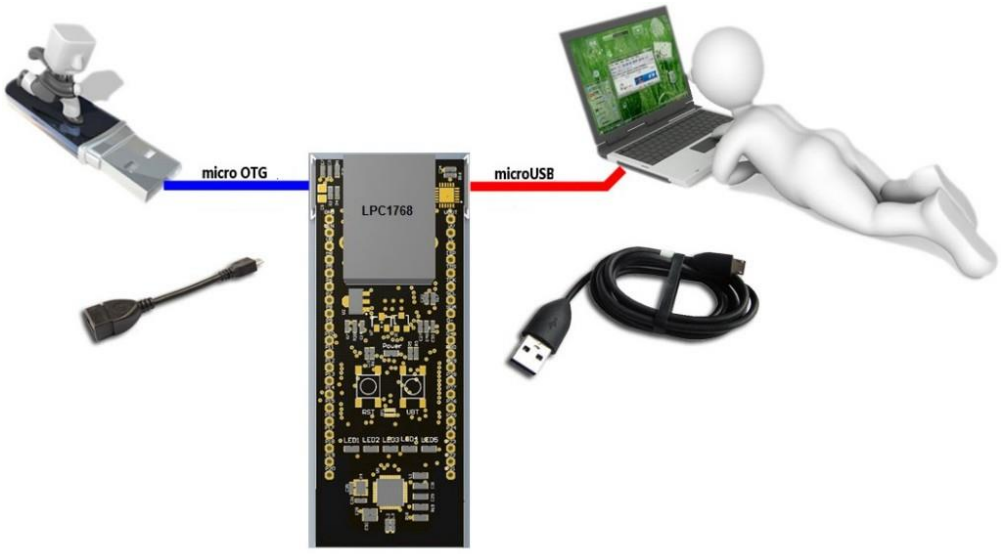
انواع روش های پروگرام کردن میکروکنترلرهایARM

۱- **از طریق واسط استاندارد JTAG** : با استفاده از پورت JTAG و پروگرامر-J-Link از شرکت Segger و انجام تنظیمات مربوطه در کامپایلر میتوان به صورت مستقیم و با سرعت بالا برنامه را به میکرو Download و سپس Debug نمود.

۲- **از طریق پورت سریال صفرام :** همه میکروکنترلرهای ARM میتوانند از طریق پورت UART0 برنامه بگیرند. به این صورت که توسط مبدل USB به سریال یک سمت به کامپیوتر وصل می شود و در سمت دیگر Rx مبدل به TXD0 میکرو ، و Tx مبدل به RXD0 میکرو متصل می شود .سپس توسط نرم افزار Flash Magic (فقط برای میکروکنترلرهای شرکت NXP) میتوان فایل Hex ساخته شده را به میکروکنترلر انتقال و میکرو را پروگرام نمود. در شکل زیر ماژول USB به سریال FT232 را مشاهده می کنید که قابلیت عملکرد در ولتاژ ۳,۳ ولت را دارد.

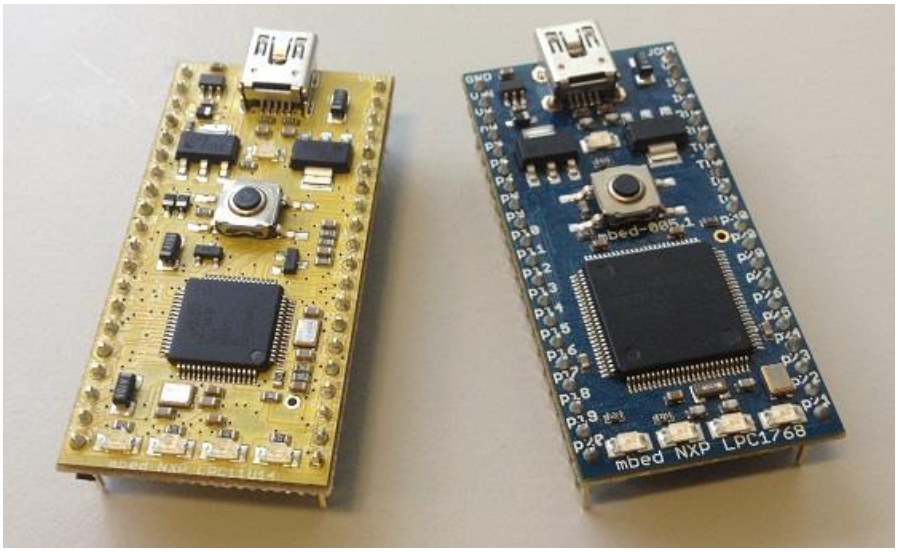


۳- **با استفاده از رابط USB در برخی میکروکنترلرها :** با استفاده از پورت USB به صورت مستقیم و بدون واسطه میتوان میکروکنترلرهای ARM که دارای واحد USB می باشند (مانند LPC1768) را پروگرام نمود. به این صورت که پس از اتصال پایه های +D و -D به پایه های مربوطه به میکرو ، به صورت اتوماتیک میکرو شبیه یک فلش توسط کامپیوتر شناسایی می شود و میتوان فایل Hex برنامه دلخواه را مستقیم داخل آن Copy نمود.



برد راه اندازی میکروکنترلرهایARM

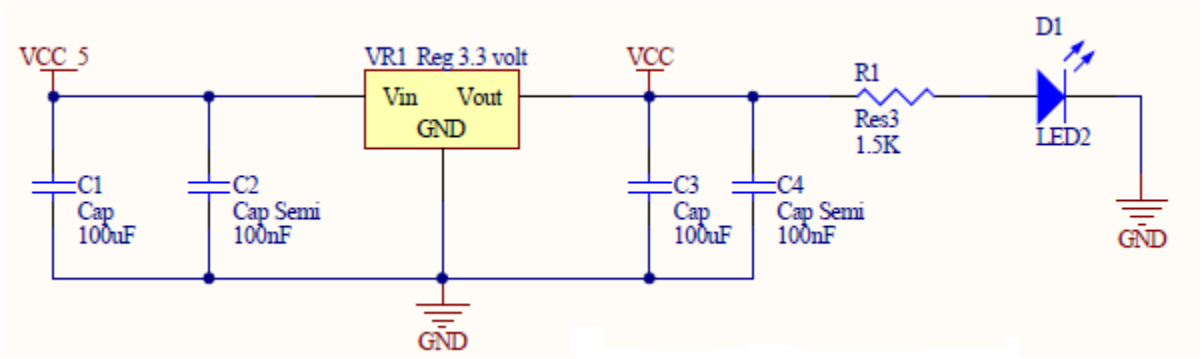
به علت اینکه این میکروکنترلرها فقط در بسته بندی smd وجود دارند ، نمیتوان آن ها را روی برد بورد به صورت مستقیم استفاده کرد. به همین علت معمولا برای کار کردن روی برد بورد از یک برد راه انداز استفاده می گردد. برد راه انداز تمامی سخت افزار لازم برای راه اندازی یک میکرو از قبیل رگولاتور ، خازن ، مقاومت ، سلف ، کریستال و ... را در خود دارد. شکل زیر برد راه اندازی برای دو میکروکنترلر نمونه ۶۴ و ۱۰۰ پایه را نشان می دهد.



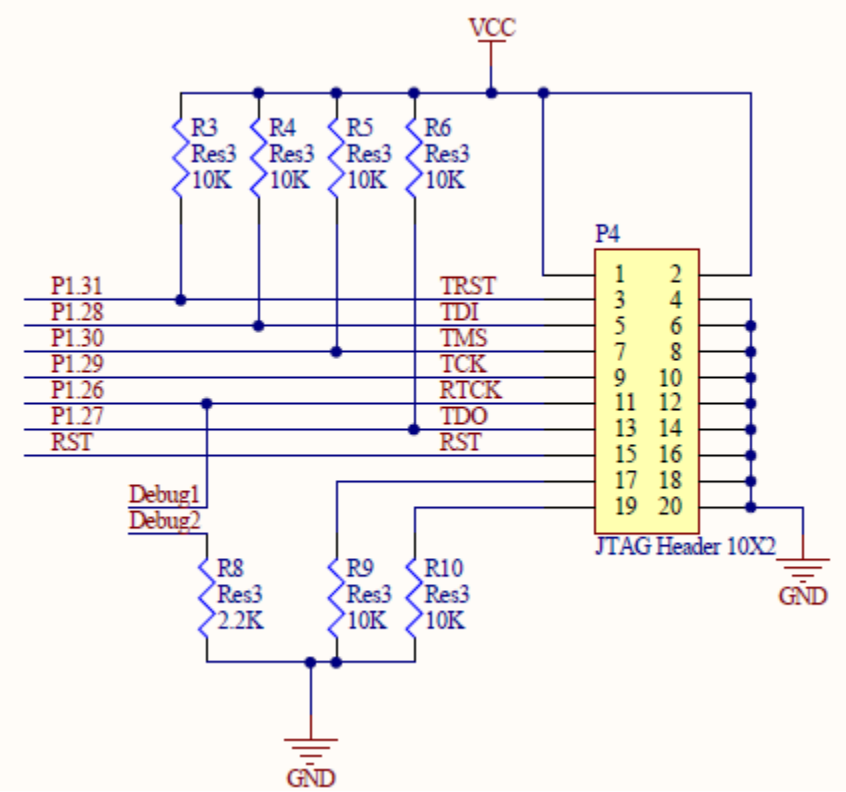
طراحی برد راه انداز برای میکروکنترلرLPC2138

میخواهیم بوردی طراحی کنیم که تمامی امکانات مورد نیاز ما برای کارکردن با این میکرو را فراهم کند. چنین بوردی از قسمت های مختلفی تشکیل شده است و دارای ویژگی های زیر است:

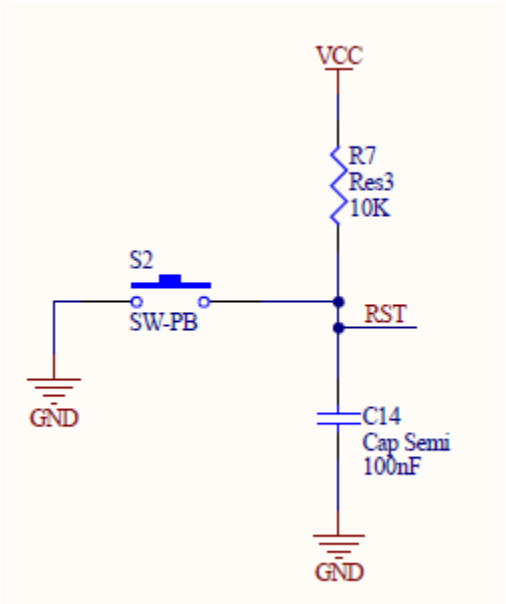
۱- طراحی ورودی و رگولاتور 3.3V: برای این منظور از یک ورودی آداپتور ، یک کلید ، رگولاتور ۳،۳ ولت و چند خازن به صورت مدار زیر استفاده می شود. از یک LED به منظور نشان دادن برقراری تغذیه استفاده شده است.



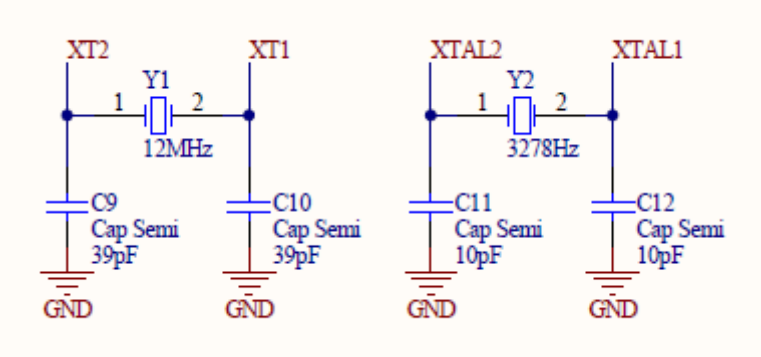
۲- **طراحی رابط JTAG** : برای پروگرام کردن این میکروکنترلر از استاندارد JTAG به صورت مدار زیر استفاده می گردد.



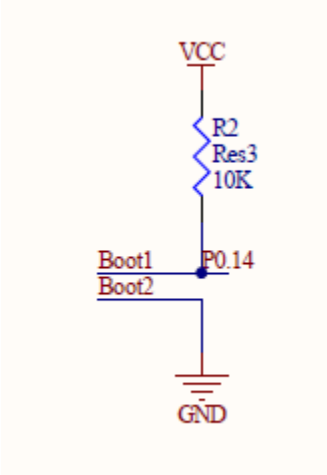
۳- **طراحی مدار Power On Reset** : برای اینکه میکروکنترلر در هنگام روشن شدن به صورت خودکار ریست شود و اینکه بتوان آن را به صورت دستی ریست نمود از مدار استاندارد زیر استفاده می گردد.



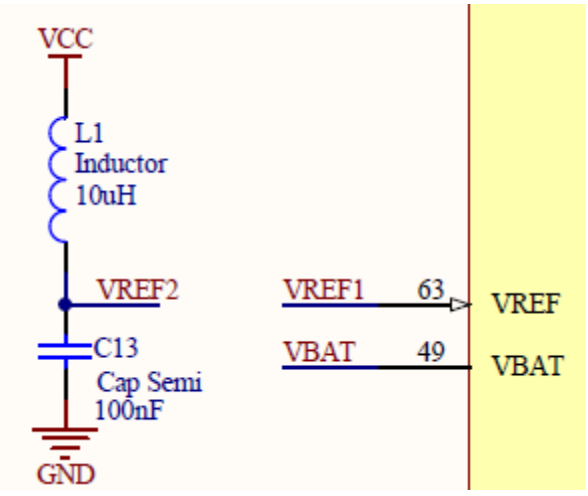
۴- **طراحی کریستال ها :** به منظور راه اندازی این میکرو از یک کریستال 12MHz و یک کریستال ساعت برای واحد RTC به صورت مدار زیر استفاده می شود.



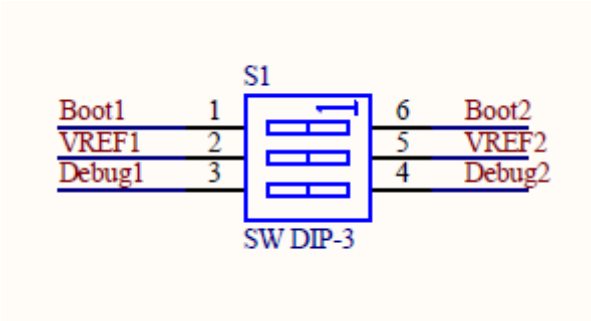
۵- طراحی مدار پروگرام از طریق UART0 : برای پروگرام بوسیله UART0 میبایست بتوانیم پایه P0.14 را در زمان پروگرام به زمین متصل کنیم. بنابراین مداری به صورت زیر خواهیم داشت.



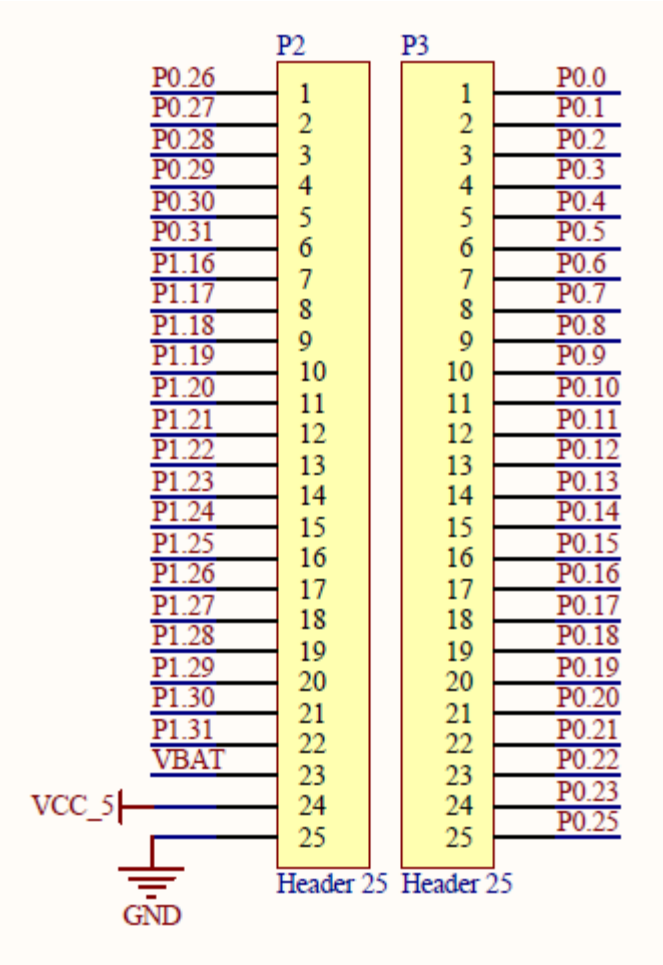
۶- اتصال پایه VREF و تمهیدات کاهش نویز : برای پایه VREF (ولتاژ مرجع واحد ADC) از یک سلف و خازن با مقادیر استاندارد به منظور کاهش نویز استفاده می کنیم. برای تغذیه واحد RTC ، پایه VBATرا به صورت شکل زیر به یک پین هدر متصل می کنیم تا بتوانیم در زمان دلخواه آن را به VCC داخل مدار یا به ولتاژی از بیرون مدار متصل نماییم.



۷ -قراردادن دیپ سوئیچ : به منظور کنترل روی قطع و وصل کردن حالت های Boot و Debug و همچنین اتصال یا عدم اتصال ولتاژ VREF از یک دیپ سوئیچ سه تایی به صورت شکل زیر استفاده می کنیم.

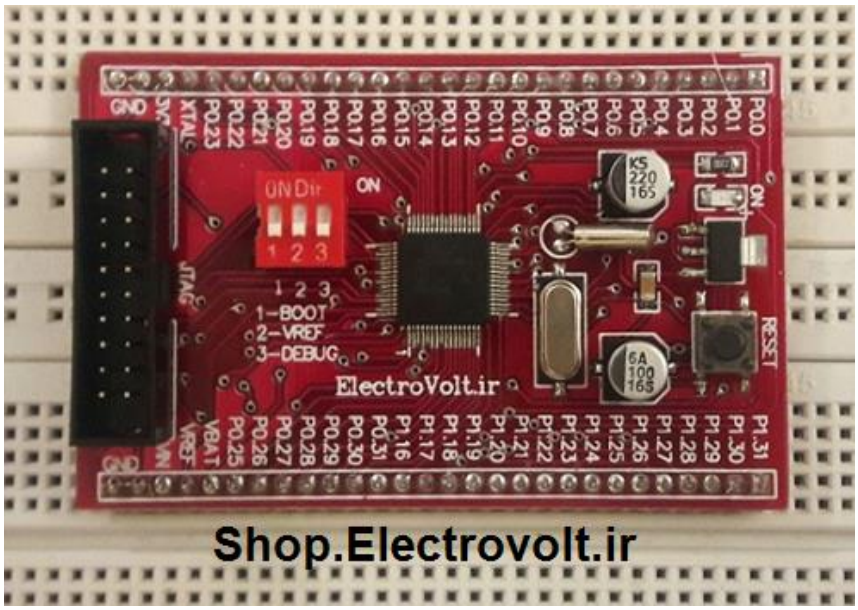


۸ – قرار دادن پین هدر : برای تمامی پایه های میکرو هدر در نظر می گیریم تا بتوان آن را روی برد بورد نصب کرد.



طراحی و ساخت PCB

بعد از طراحی شماتیک مدار بورد راه انداز نوبت به طراحی و ساخت فیبر مدارچاپی می رسد که با استفاده از نرم افزار Altium Designer این کار صورت می گیرد. شکل زیر مدار طراحی شده نهایی توسط الکترو ولت را نشان می دهد.



کامپایلرها و مفسرهای ARM

برای میکروکنترلرهای ARM کامپایلرها و مفسرهای مختلفی ارائه شده است. زبان برنامه نویسی اغلب این کامپایلرها C و C++است.هر کدام از این کامپایلرها ویژگی ها و امکانات خاصی ارائه می دهند که در جدول زیر مهمترین آن ها را مشاهده می کنید.

امکانات	IAR	Cross Work	Keil	Win Arm	Flow Code	ADS
برنامه نویسی اسمبلی	Ok	Ok	Ok	--	گرافیکی	Ok
برنامه نویسی C	Ok	Ok	Ok	Ok	--	Ok
برنامه نویسی C++	Ok	--	Ok	Ok	--	Ok
امکان شبیه سازی	Ok	--	Ok	--	Ok	Ok
پشتیبانی از هسته‌ها	همه	Arm7	همه	Arm7	Arm7	همه
منابع آموزشی	متوسط	کم	متوسط	متوسط	متوسط	متوسط
ویرایشگر	قوی	ساده	قوی	قوی-متن باز	ساده	قوی

از میان کامپایلرهای بالا به دلایل زیر کامپایلر قدرتمند Keil انتخاب می شود:

- دارای محیطی حرفه ای ، ویرایشگر قوی و کاربر پسند نسبت به سایرین
- پشتیبانی از تمامی میکروکنترلرهای ARM و امکانات جانبی آن ها
- دارای سیستم شبیه سازی با قابلیت شبیه سازی هسته و ادوات جانبی
- دارای کد هگز خروجی بهینه و با حجم کمتر نسبت به سایر کامپایلرها

فصل ۵ – شروع به کار با نرم افزارهای KEIL و Proteus

مقدمه

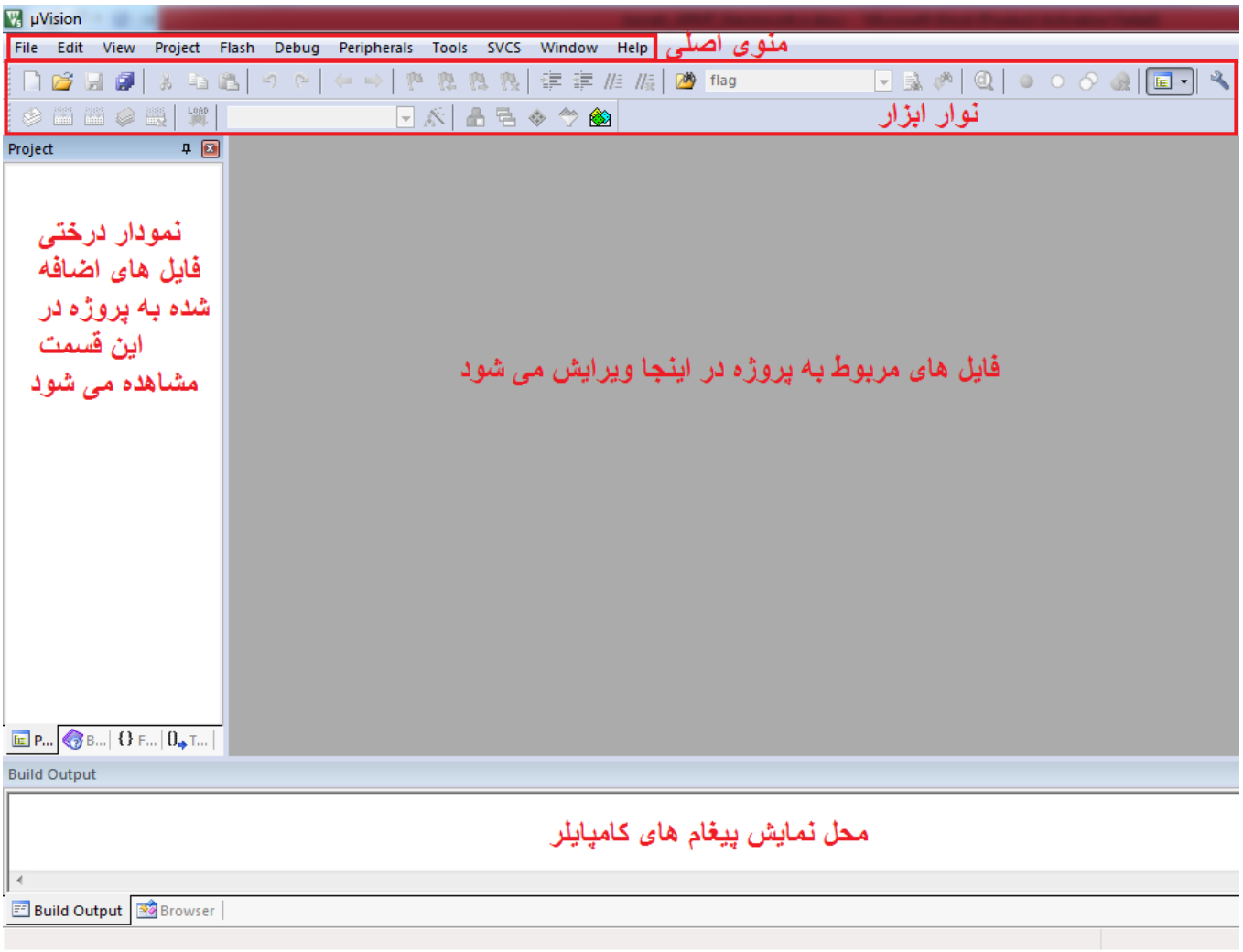
میکروکنترلرهای ARM سری LPC21xx یکی از راحت ترین میکروکنترلرها از نظر برنامه نویسی و سخت افزار مورد نیاز می باشند که طیف وسیعی از پروژه ها را نیز پوشش می دهند. ضمن اینکه جزو معدود میکروکنترلرهای ARM هستند که در نرم افزار پروتئوس وجود دارند و میتوان آن ها را همانند میکروکنترلرهای AVR در پروتئوس شبیه سازی کرد. بنابراین هدف اصلی کار با این میکرو کاملا آموزشی است زیرا برنامه نویسی ساده تر و قابلیت شبیه سازی در نرم افزار پروتئوس را دارند.

دانلود و نصب نرم افزار Keil 5.15

نرم افزار Keil یک کامپایلر ، شبیه ساز و پروگرام کننده همه میکروکنترلرها و تراشه های مبتنی بر ARM می باشد که امتیاز آن نیز برای خود شرکت ARM می باشد. به علت کامل تر بودن نسخه 5.15 نسبت به نسخه های پیشین و برطرف شدن برخی از باگ ها ، از این نسخه از نرم افزار برای برنامه نویسی کلیه میکرو کنترلرها استفاده خواهیم کرد. برای دانلود و آموزش نحوه نصب به وبسایت الکترو ولت بخش دانلود نرم افزارها مراجعه نمایید.

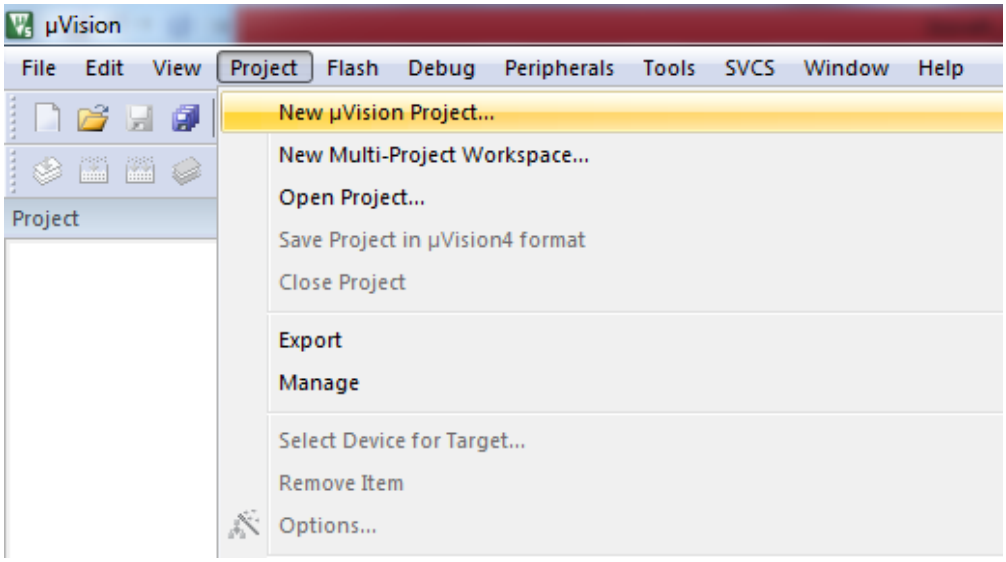
شروع به کار با نرم افزار KEIL

بعد از نصب و فعالسازی نرم افزار ، برنامه به صورت زیر باز می شود. همیشه بعد از باز کردن کامپایلر ، آخرین پروژه انجام شده باز می شود که میتوانید آن را از مسیر Project و با کلیک روی گزینه Close Project ببندید.

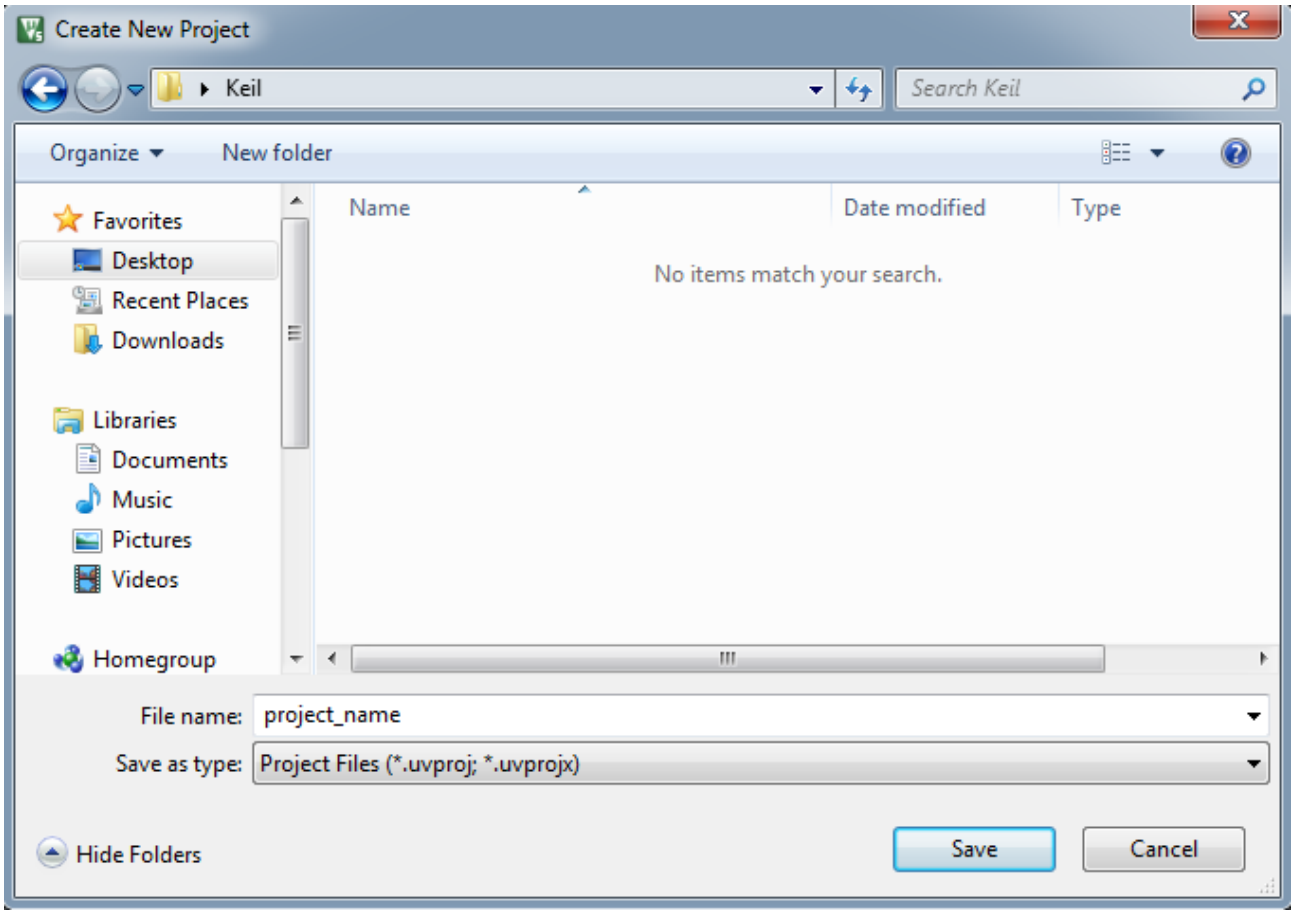


ایجاد یک پروژه جدید

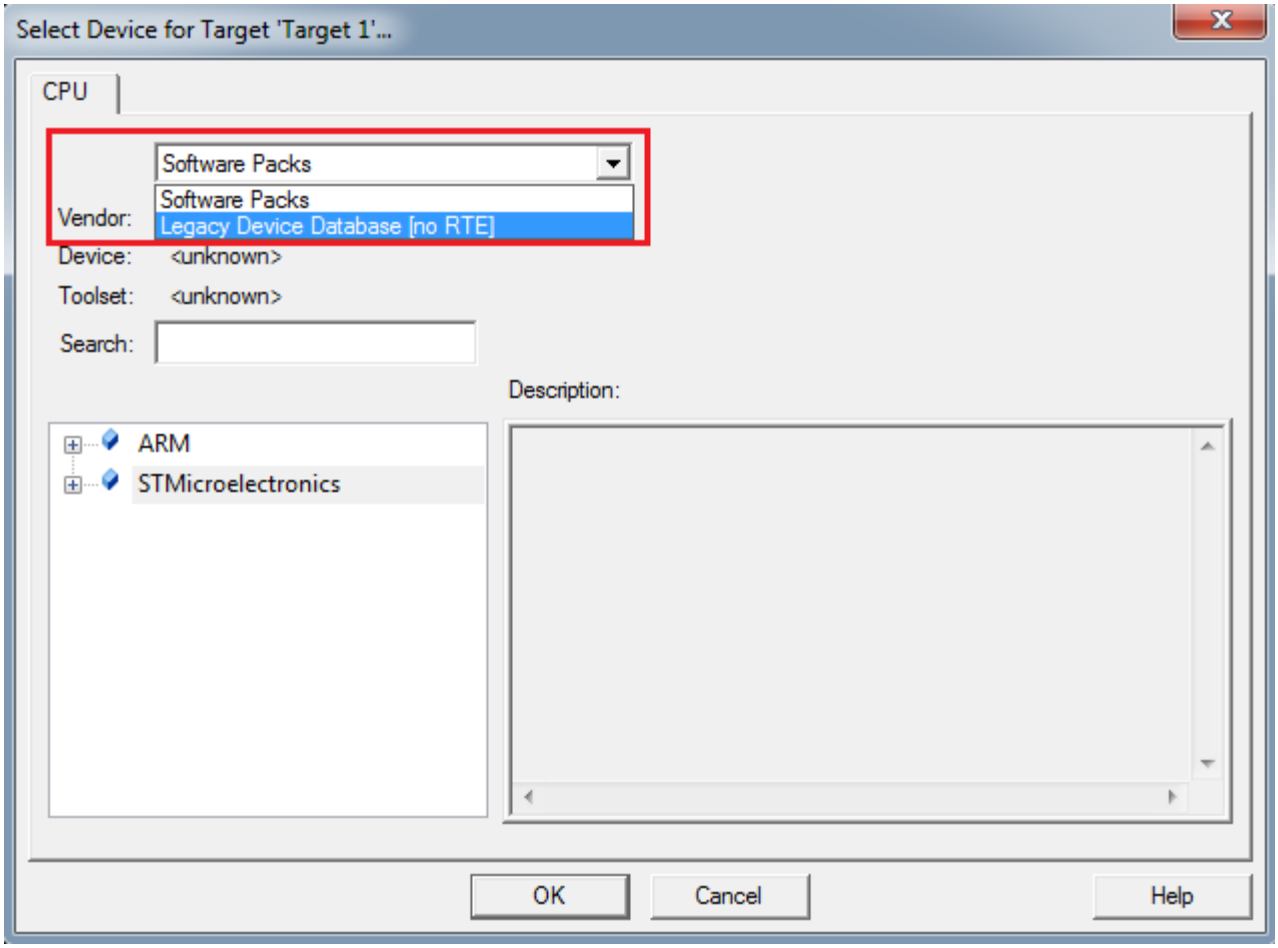
از منوی Project روی گزینه New Uvision Project کلیک کنید. با کلیک روی این گزینه از شما می خواهد تا یک مسیر برای ذخیره پروژه انتخاب نمایید.



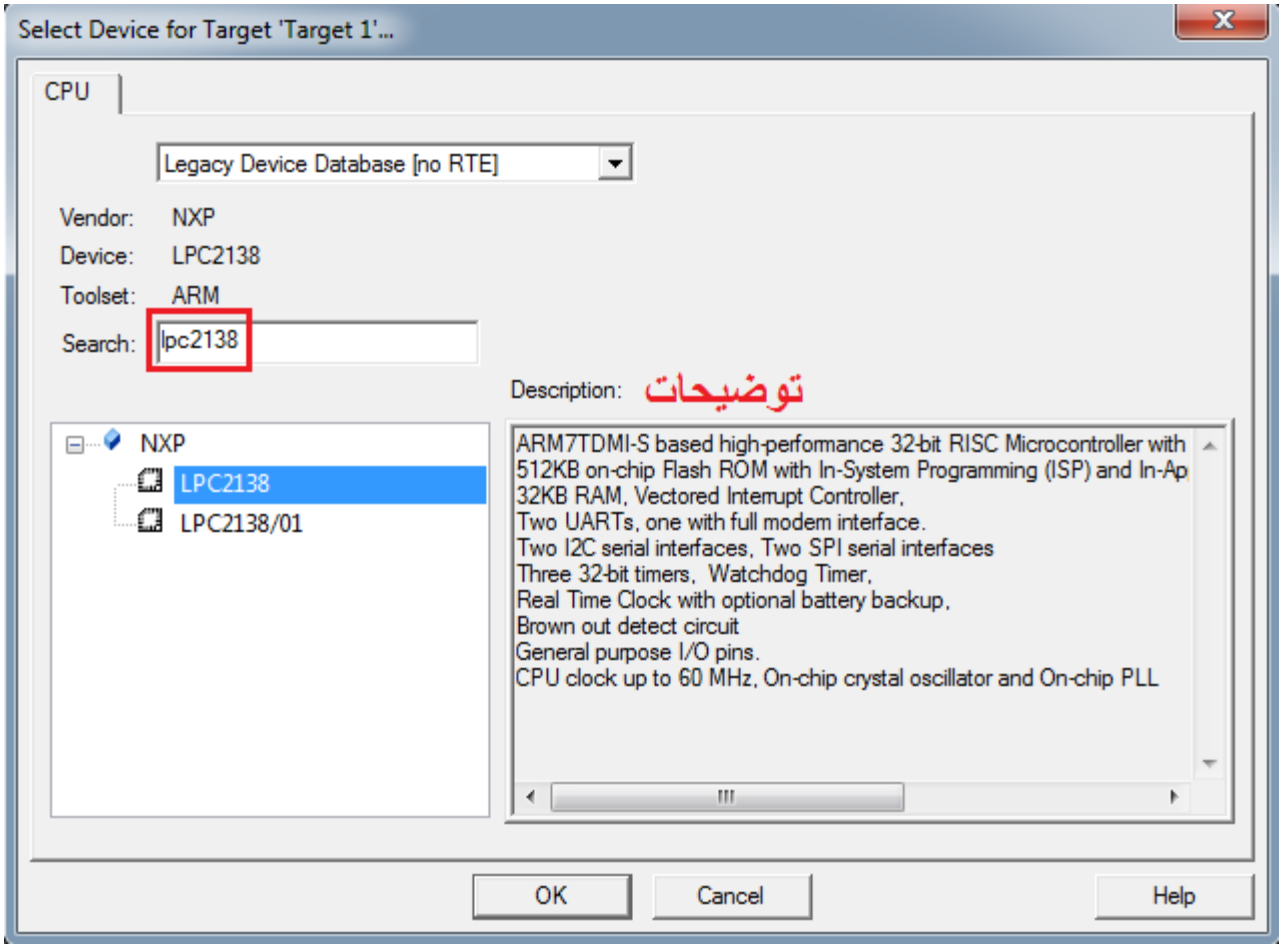
به علت جداسازی و در کنار هم بودن تمام پروژه آدرس پروژه جدید را در یک پوشه خالی انتخاب نمایید. سپس نام فایل اصلی پروژه را انتخاب نموده و روی گزینه Save کلیک کنید. با این کار فایل اصلی پروژه با پسوند uvproj ذخیره می شود.



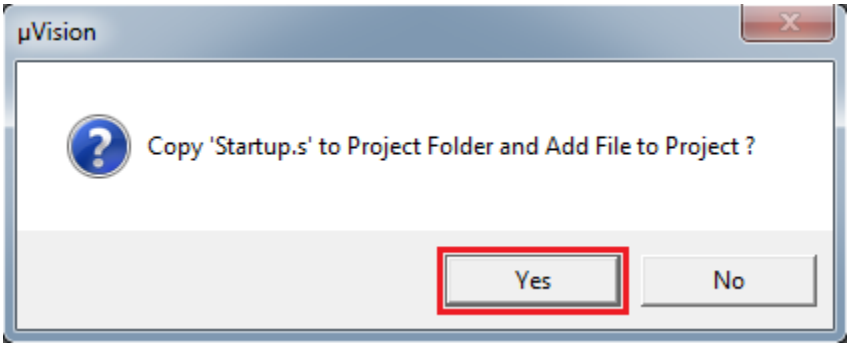
در مرحله بعد پنجره ی Select Device باز می شود که نوع میکرو مورد نظر را باید در آن انتخاب نمود. همانطور که مشاهده می کنید زمانی که بخش Vendor (فروشنده) روی Software Packs قرار داشته باشد تعداد کمی از هسته های ARM قابل مشاهده می باشد. برای مشاهده دیگر هسته های ARM ی که آنها را نصب کرده اید ، مطابق شکل زیر بخش Vendor را روی Lagancy Device Database قرار دهید.



سپس میتوانید نام قطعه مورد نظر خود را در قسمت Search وارد کنید. یا اینکه میتوانید در پنجره زیر آن روی شرکت NXPکلیک کنید و سپس LPC2138 را انتخاب نمایید. همانطور که مشاهده می کنید با انتخاب هر میکرو مشخصات آن در سمت راست پنجره نمایش داده می شود. بعد از انتخاب روی گزینه OK کلیک کنید.



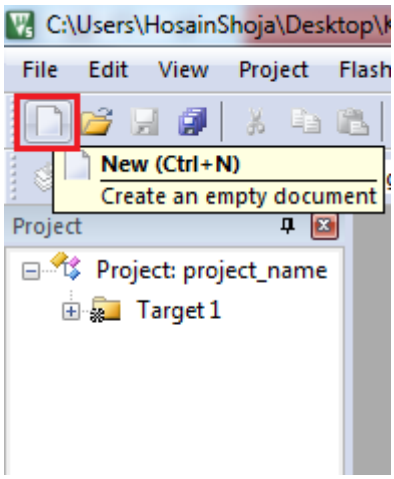
در نهایت از کاربر میخواهد که فایل مربوط به راه اندازی میکرو (Sturtup.s) که همیشه در پروژه وجود دارد و به همراه فایل های دیگر کامپایل می شود را به پروژه به صورت اتوماتیک اضافه نماید. با کلیک کردن بر روی گزینه Yes کار ایجاد پروژه جدید تمام می شود.



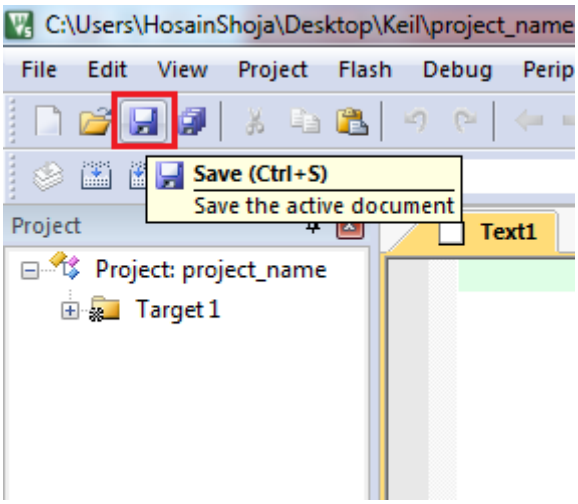
ایجاد یک فایل در ویرایشگرKEIL

بعد از ساختن پروژه همانطور که در شکل زیر مشاهده می کنید ، شاخه ای با نام Target 1ایجاد می شود که درون آن فقط فایل Startup.s وجود دارد. برای نوشتن برنامه به زبان C ، باید یک فایل با پسوند c اضافه نماییم. برای این منظور

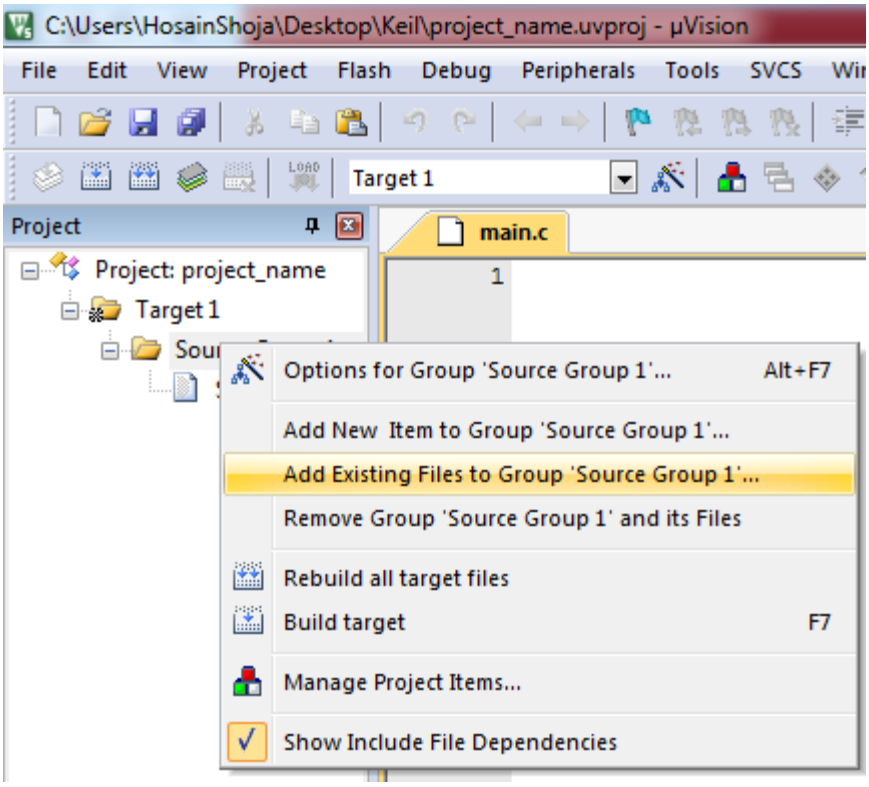
از منوی File گزینه New را کلیک می کنیم و یا از نوار ابزار روی تصویر New کلیک می کنیم (همچنین میتوانیم با زدن Ctrl+N نیز این کار را انجام دهیم).



بعد از اینکه فایل جدید ایجاد شد باید آن را از منوی File یا از دکمه Save روی نوار ابزار (و یا Ctrl+S) ذخیره کرد. حتما دقت کنید که فایل مورد نظر را با اسم دلخواه و با پسوند .c (مثلا main.c) و در پوشه اصلی پروژه ذخیره نمایید.

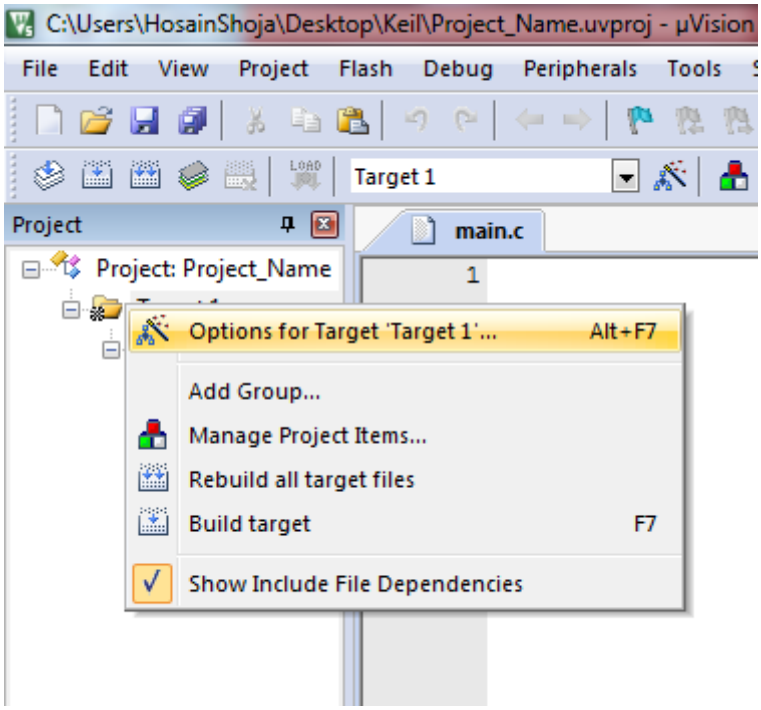


بعد از ذخیره کردن می بایست فایل مورد نظر را به پروژه اضافه نمود. برای اضافه کردن فایل جدید به پروژه ، از بخش Projectروی Source Group کلیک راست می کنیم و از قسمت Add Existing Files To Group ، فایل main.c را به پروژه اضافه می کنیم.

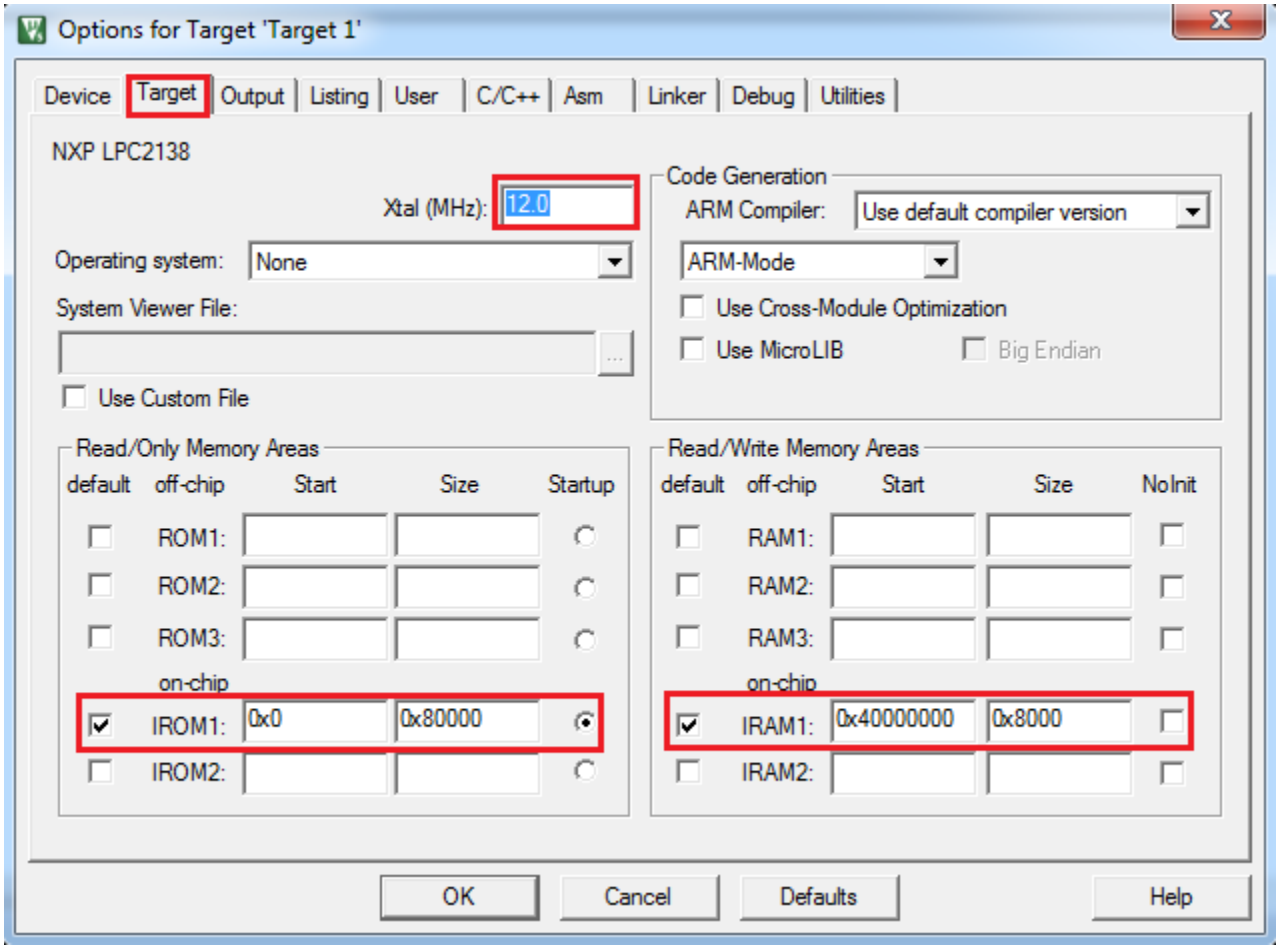


انجام تنظیمات پروژه

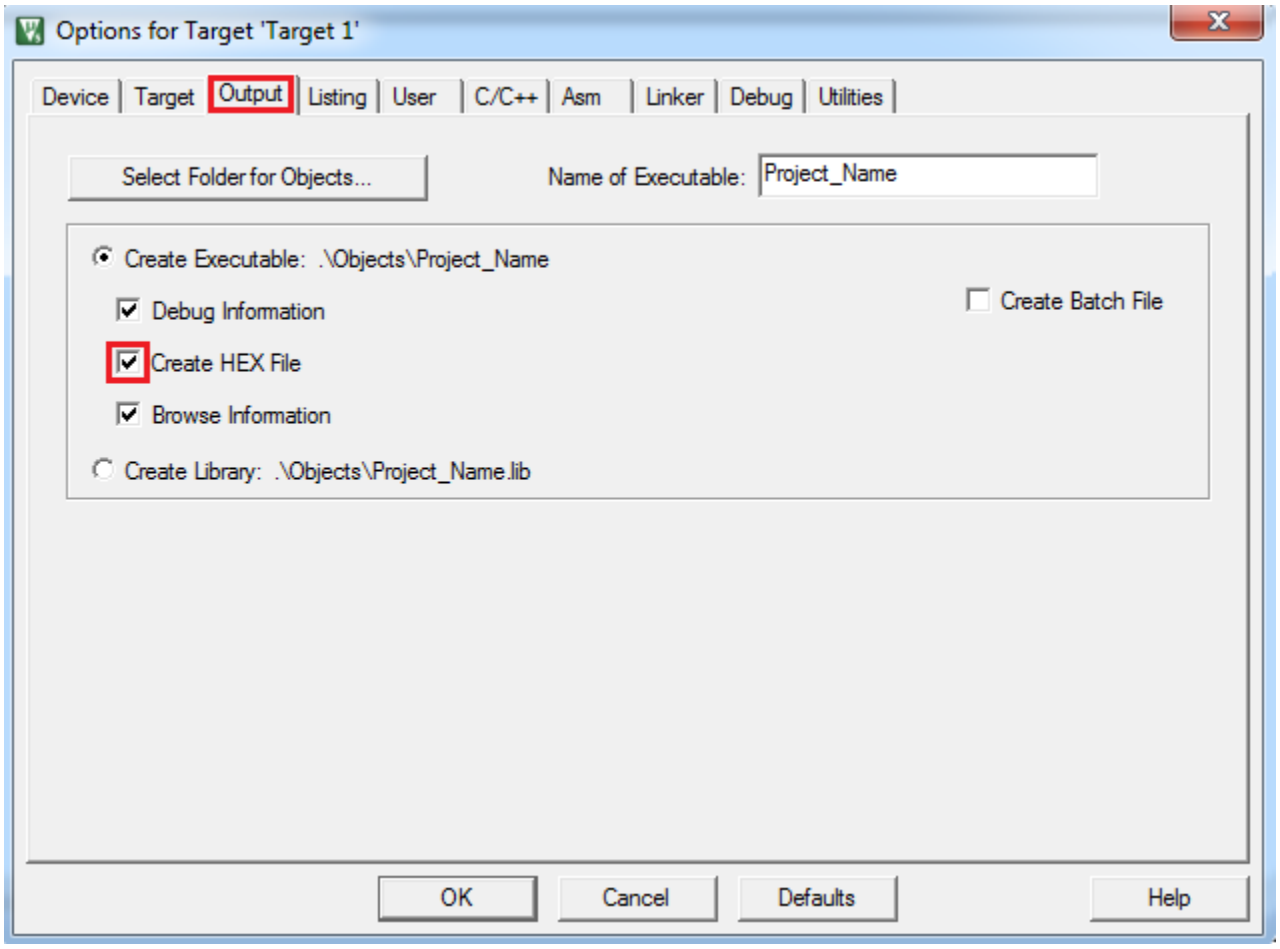
برای صحیح کامپایل شدن پروژه باید تنظیمات پروژه را انجام داد. برای این منظور از قسمت Project و با کلیک راست روی Target 1 و زدن Options For Target (یا از روی نوار ابزار) پنجره تنظیمات را باز می کنیم.



در صفحه تنظیمات و در سربرگ Target باید تنظیمات مربوطه را به صورت شکل زیر انجام داد.



سپس در مرحله بعد به سربرگ Output رفته و تیک گزینه Create Hex File را می زنیم.



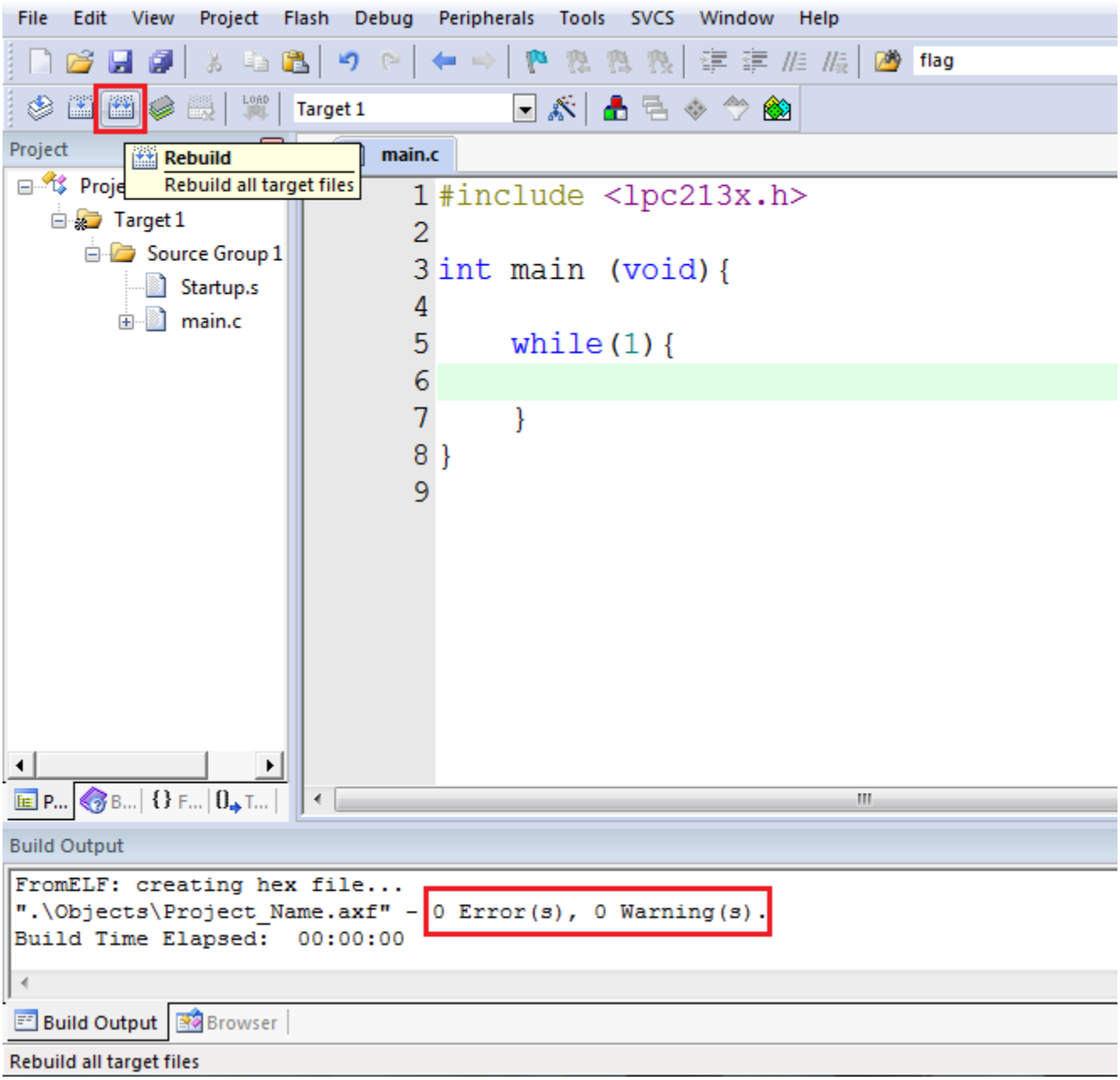
ساختار برنامه نویسی ARM در نرم افزارKEIL

برنامه مورد نظر در نرم افزار Keil درون فایلی که اضافه کردیم (به نام main.c) نوشته می شود. همانند برنامه نویسی که در میکروکنترلرهای AVR داشتیم ، در میکروکنترلرهای ARM نیز ساختار برنامه به همان صورت می باشد با این تفاوت که هدر فایلی که به پروژه اضافه می کنیم، LPC213x.h نام دارد و تابع main در نرم افزار Keil باید خروجی int داشته باشد. بنابراین ساختار برنامه نویسی به صورت زیر خواهد بود.

```
#include <lpc213x.h>

int main (void){

  while(1){

  }

}
```

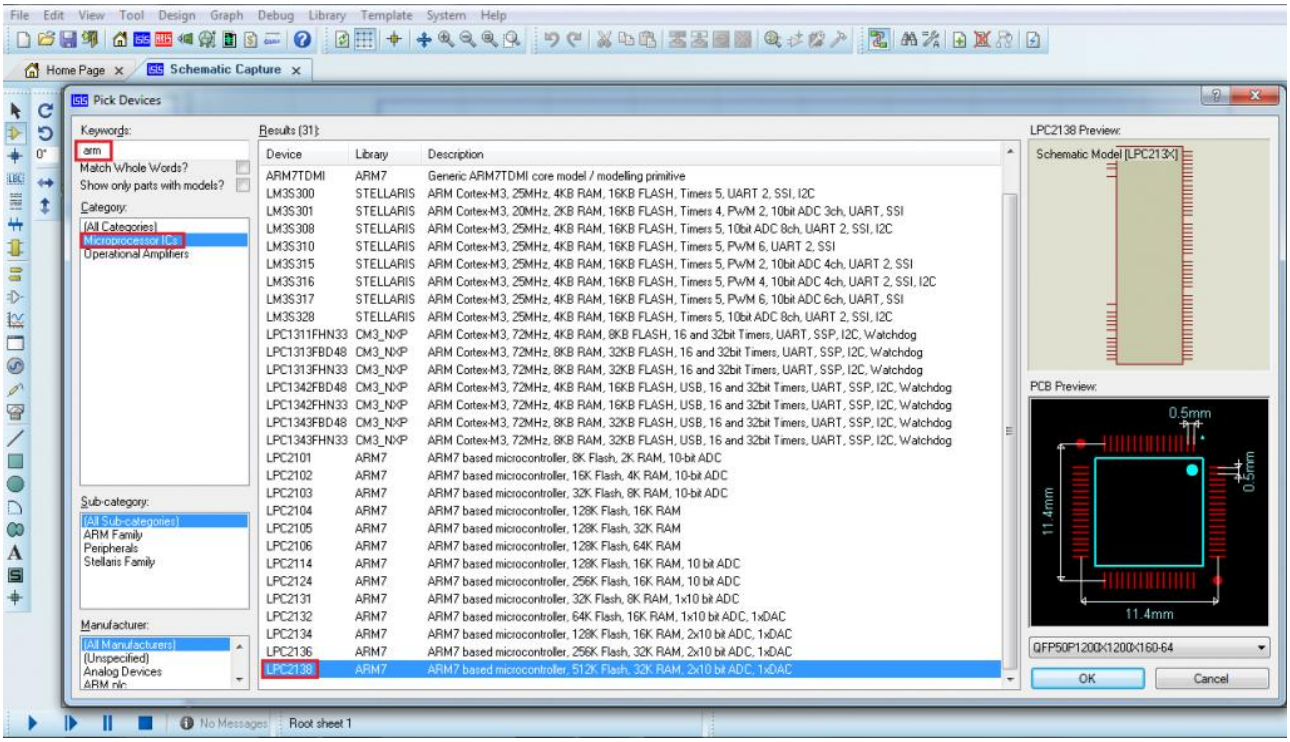
کامپایل کردن پروژه و ساخت فایل خروجی

بعد از اینکه برنامه کامل شد ، برای کامپایل و ساخت فایل های خروجی ، باید از منوی Project روی گزینه Rebuild All Target Fileکلیک کنید. همچنین میتوانید این کار را با زدن دکمه مشخص شده در شکل زیر انجام دهید. بعد از کامل شدن پروسه در قسمت پیغام ها ، خطا ها و اخطار های مورد نظر نمایش داده خواهد شد و در صورت 0 بودن خطا ها یعنی کامپایل پروژه با موفقیت انجام شده و فایل های خروجی (از جمله فایل Hex) ساخته شده است.

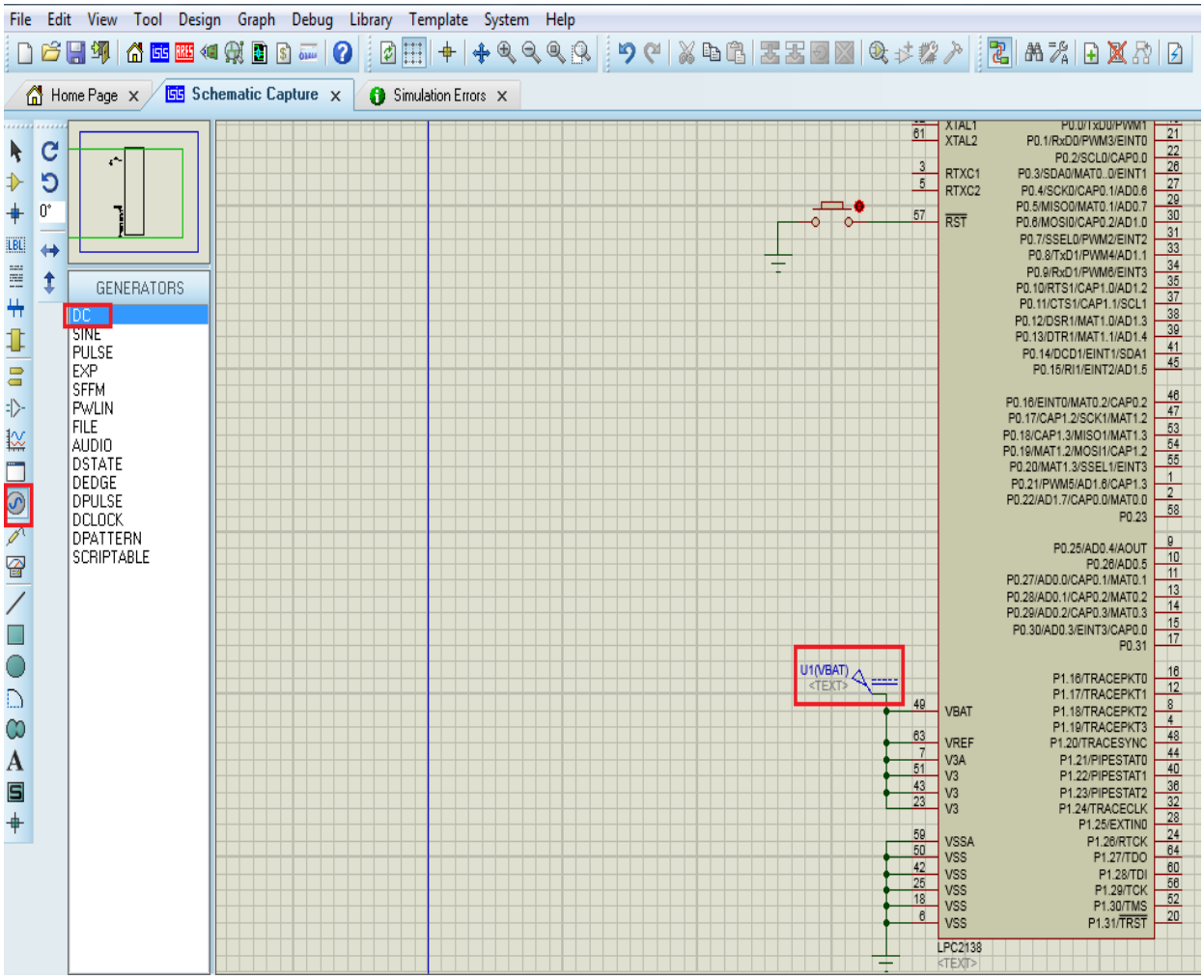


اضافه کردن فایل خروجی به شبیه سازی پروتئوس

برای شبیه سازی میکروکنترلرهای ARM میتوان از خود نرم افزار Keil استفاده کرد. اما این شبیه سازی به صورت بسیار محدود صورت می گیرد. به طوری که تنها میتوان تغییرات پایه ها و رجیستر ها را در زمان کوتاه مشاهده کرد. اما نرم افزار قدرتمند پروتئوس میتواند برخی از میکروکنترلرهای ARM7 و CortexM3 را در مدار به صورت شبیه سازی نزدیک به واقعیت انجام دهد. برای اینکه بدانید نرم افزار پروتئوس از چه میکروکنترلرهای ARM پشتیبانی می کند ، درون این نرم افزار رفته و در قسمت انتخاب قطعه عبارت arm را تایپ کنید و از کتابخانه های بوجود آمده روی Microprocessor ICsکلیک کنید. شکل زیر لیست این میکروکنترلرها را نشان می دهد.



بعد از انتخاب میکروکنترلرLPC2138 ، بهتر است از یک دکمه برای ریست استفاده کنیم. بنابراین با تایپ button قطعه را به پروتئوس اضافه می کنیم. سپس در نرم افزار پروتئوس به صورت زیر اتصال قطعات را تکمیل می کنیم.



نکته ۱ : چون پروتئوس صرفا یک نرم افزار شبیه سازی است میتوان از قرار دادن کریستال و خازن ها و مقاومت های پایه های ریست و کریستال به جهت تسریع در شبیه سازی ، صرفنظر کرد.

نکته ۲ : از آن جایی که ولتاژ عملکرد آی سی ۳،۳ ولت است ، برای تغذیه آی سی از Generator در نرم افزار پروتئوس استفاده کردیم. بعد از جایگذاری این قطعه روی آن دابل کلیک کرده و ولتاژ آن را روی 3.3 ولت قرار می دهیم.

در پایان روی میکرو LPC2138 دابل کلیک می کنیم و فرکانس کاری میکرو را روی 12Mhz قرار می دهیم. سپس از قسمت انتخاب فایل میتوانیم فایل Hex مورد نظرمان را برای شبیه سازی اضافه کنیم.

فصل ۶ – آموزش برنامه ریزی و راه اندازی میکروکنترلرLPC2138

مقدمه

همانطور که متوجه شدید، میکروکنترلرهای ARM7 از نظر برنامه نویسی کمی از میکروکنترلرهای AVR دشوارتر هستند. در میکروکنترلرهای AVR درون نرم افزار CodeVision ابزاری به نام کدویزارد وجود داشت که بخشی از کدهای مورد نیاز برنامه نویسی را به صورت اتوماتیک و تنها با تنظیمات ساده ای تولید می کرد. اما در میکروکنترلرهای ARM7 در هیچ کدام از نرم افزارها چنین قابلیت ویژه ای وجود ندارد. بنابراین تمامی برنامه نویسی و تنظیمات رجیسترها به صورت دستی و توسط برنامه نویس انجام می گیرد. در نتیجه این مسئله لزوم یادگیری عملکرد سخت افزار و درک عمیق عملکرد آی سی را بیش از پیش نمایان می کند. در این بخش از آموزش به بررسی مهمترین واحد میکرو یعنی واحد GPIO (ورودی/خروجی همه منظوره) می پردازیم.

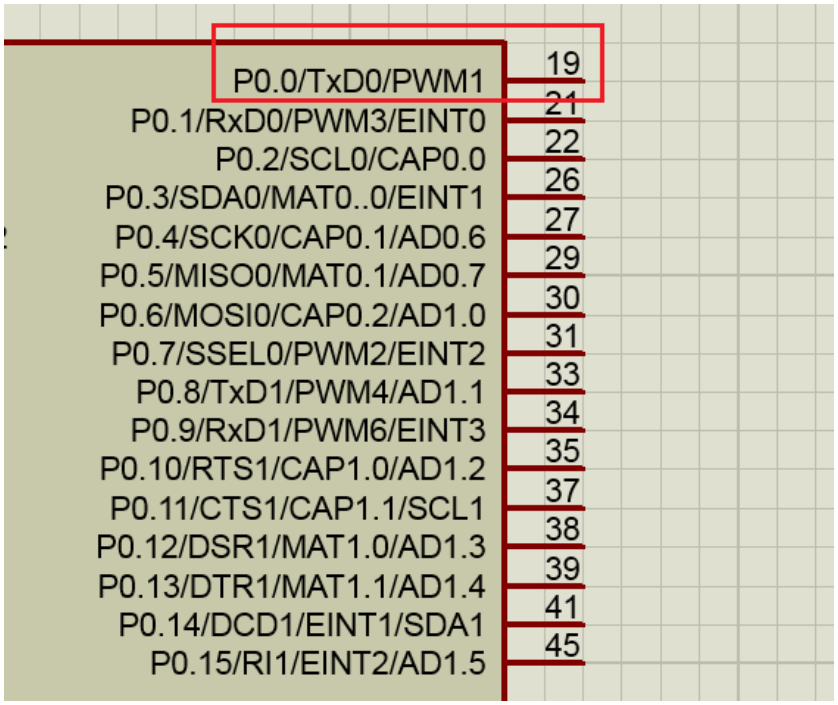
تعیین عملکرد پورت ها در میکروکنترلرLPC2138

همانطور که در بخش چهارم آموزش به آن اشاره کردیم ، میکروکنترلر LPC2138 دارای دو پورت P0 و P1 می باشد. هر کدام از این پورت ها ۳۲ بیتی هستند اما دسترسی به برخی از بیت ها به صورت سخت افزاری میسر نیست .

بیت های در دسترس عبارتند از : کلیه بیت های پورت P0 به جز P0.24 و نصف پورت P1 از P1.16 تا P1.31 در دسترس هستند .

اما بیت هایی که در دسترس هستند هر کدام دارای چندین عملکرد متفاوت می باشد .در میکروکنترلرهای AVR زمانی که از پایه ها به صورت ورودی/خروجی استفاده می شد یا به صورت واحدهای جانبی دیگر استفاده می شد ، پایه به صورت

اتوماتیک تغییر وضعیت می داد. اما در میکروکنترلرهای ARM7 باید دقیقا مشخص کنیم از کدام ویژگی یک پایه استفاده می شود. بنابراین برای تنظیم عملکرد پورت ها رجیستری به نام PINSEL تعبیه شده است. رجیستر PINSEL مشخص می کند که هر پایه در چه عملکردی استفاده می شود. شکل زیر عملکردهای مختلف پایه P0.0 در نرم افزار پروتئوس را نشان می دهد. این پایه در یکی از حالت های : ورودی/خروجی P0.0 ، خروجی واحد UART و خروجی اول واحد PWM مورد استفاده قرار می گیرد.



بنابراین در هر میکروکنترلر ARM7 به تعداد پایه های در دسترس رجیستر PINSELوجود دارد که در LPC2138 تعداد این رجیسترها ۳ عدد می باشد. در جدول زیر عملکرد این رجیسترها را مشاهده می کنید.

نام رجیستر	توضیح عملکرد	دسترسی به رجیستر	مقدار پیش فرض
PINSEL0	رجیستر تنظیم عملکرد پایه های P0.0 تاP0.15	**خواندن/نوشتن**	0x00000000
PINSEL1	رجیستر تنظیم عملکرد پایه های P0.16 تاP0.31	**خواندن/نوشتن**	0x00000000
PINSEL2	رجیستر تنظیم عملکرد پایه های P1.16 تاP1.31	**خواندن/نوشتن**	به جدول ۳۹ صفحه ۶۳ دیتاشیت مراجعه شود

نکته ۱ : تمام پایه های پورت P0 و P1 در حالت پیش فرض روی **GPIO** (واحد ورودی/خروجی همه منظوره) هستند. یعنی همواره همه پایه ها به عنوان عملکرد ورودی/خروجی استفاده می شوند مگر اینکه تنظیمات پیش فرض آنها یعنی محتویات رجیستر PINSEL را تغییر دهیم.

نکته۲ : هر دو بیت از رجیستر PINSEL عملکرد یک پایه را مشخص می کند برای مثال بیت های ۰ و ۱ از رجیستر PINSEL0 عملکرد پایه P0.0 را مشخص می کند ، بیت های ۲ و ۳ از رجیستر PINSEL0 عملکرد پایه P0.1 را مشخص می کند و .... اگر عملکرد هر پایه را به صورت مجزا نگاه کنیم ، مشاهده می شود هر پایه بین حداکثر ۴ عملکرد متفاوت دارد. برای مثال P0.0 می تواند در سه عملکرد مختلف GPIO ، TXD0 ، PWM1 باشد. در نتیجه مقدار دهی به دو بیت ۰ و ۱ از رجیستر PINSEL0 ، به صورت جدول زیر مشخص می گردد.

نام پایه	مقدار آن در رجیستر PINSEL0	توضیح عملکرد	مقدار ریست
P0.0	00	GPIO Port 0.0	0
	01	TXD0 (UART0)	
	10	PWM1	
	11	RESERVED	

نکته ۳ : بیت RESERVED به معنای رزرو بودن آن حالت است و خواندن/نوشتن در بیت های رزرو مجاز نمی باشد.

نکته ۴ : به علت اینکه رجیسترها ۳۲ بیتی هستند ، در اکثر اوقات به صورت hexadecimal (مبنای ۱۶) آن ها را مقدار دهی می کنیم.

مثال ۱ : فرض کنید میخواهیم از پایه P0.0 در حالت PWM1 استفاده کنیم در این صورت باید بیت اول رجیستر PINSEL0را مقدار ۱ بدهیم ، یعنی داریم:

```
PINSEL0 = 0x00000002;
```

تذکر : برای بقیه پایه ها نیز به همین صورت مقدار دهی به رجیسترهای PINSEL در حالتی که از پایه به غیر از GPIO استفاده می شود ، ضروری است. عملکرد کلی پایه ها را میتوانید در **UM10120** (**راهنمای کاربر LPC213x**) صفحات ۵۹ تا ۶۳ مشاهده نمایید.

مثال ۲ : می خواهیم P0.27 را روی حالت Match 0.1 قرار دهیم. به UserManual مراجعه کرده و در صفحه ۶۲ مشاهده می کنیم که باید بیت های ۲۲ و ۲۳ رجیستر PINSEL1 را ۱ کنیم. در نتیجه داریم:

```
PINSEL1=0x00C00000;
```

نکته مهم : مقدار دهی به یک رجیستر ۳۲ بیتی کمی دشوار به نظر می رسد. برای راحتی مقدار دهی در رجیسترها میتوان از عملگرهای بیتی Or و Shift استفاده کرد .یعنی برای مثال ۲ داریم:

```
PINSEL1 |= (3<<22);
```

توضیح : رجیستر ۳۲ بیتی PINSEL1 در عبارت فوق با عدد ۳ در مبنای دسیمال (۱۱ باینری) که ۲۲ تا شیفت به سمت چپ یافته است ، Or می شود. نتیجه این عملگر Or باعث می شود تنها بیت های ۲۲ و ۲۳ از رجیستر PINSEL1 مقدار ۱ بگیرند و بقیه **دست نخورده** باقی می مانند. این شیوه مقدار دهی بسیار رایج و پر کاربرد است چرا که علاوه بر امکان ویژه مقدار دهی یک یا چند بیت بدون دست خوردن بقیه بیت ها ، سبب خوانا تر شدن برنامه می شود.

راه اندازی پورت ها به صورت GPIO در میکروکنترلر LPC2138

بعد از اینکه وظیفه هر کدام از پایه ها مشخص شد ، چنانچه از پایه ای به صورت عملکرد GPIO استفاده کنیم ، باید برای تنظیم ورودی یا خروجی بودن از رجیستر **IODIR**، برای نوشتن در حالت خروجی بودن پورت از رجیسترهای **IOSET** و **IOCLR** و نیز برای خواندن منطق پورت در حالت ورودی بودن از رجیستر **IOPIN** استفاده می شود. به طور خلاصه:

- رجیستر **IODIR** (مخفف input output direction) : برای تنظیم جهت پورت (ورودی یا خروجی بودن)
- رجیستر **IOSET** (مخفف input output set) : برای تغییر منطق پورت در حالت خروجی
- رجیستر **IOCLR** (مخفف input output clear) : برای تغییر منطق پورت در حالت خروجی
- رجیستر **IOPIN** (مخفف input output pin) : برای خواندن منطق پورت در حالت ورودی

نکته ۱ : از هر دو رجیستر IOSET و IOCLR به منظور تغییر منطق پورت در حالت خروجی استفاده می شود. وجود دو رجیستر برای تعیین منطق پورت ها در میکروکنترلرهای ARM نسبت به میکروکنترلرهای AVR (که یک رجیستر PORTX داشتند) ، باعث راحتی برنامه نویسی می گردد. عملکرد این دو رجیستر به این صورت است که : اگر بیتی در IOSET برابر ۱ شود ، منطق خروجی پایه متناظر با آن ۱ می شود و اگر بیتی در IOCLR برابر ۱ شود ، منطق خروجی پایه متناظر با آن برابر ۰ می شود. بیت های صفری که در رجیسترهای IOSET و IOCLR وجود دارند ، بی تاثیر هستند.

نکته ۲ : زمانی که میکروکنترلر ARM7 دارای یک پورت باشد (مانند سری LPC210x) رجیسترهای فوق به همین صورت (IODIR,IOSET,IOPIN) در برنامه استفاده می شود اما زمانی که میکروکنترلر دارای پورت های بیشتری باشد (مانند سری های LPC213x و LPC236x) بعد از IO در نام همه رجیسترهای فوق عدد پورت مورد نظر وارد می شود. شکل زیر این موضوع را نشان می دهد.

Generic Name	IODIR	IOSET	IOPIN	IOCLR
Access	Read/Write	Read/Write	Read only	Write only
Reset Value	0x00000000	0x00000000	NA	0x00000000
PORT0 Address & Name	0xE0028008 IO0DIR	0xE0028004 IO0SET	0xE0028000 IO0PIN	0xE002800C IO0CLR
PORT1 Address & Name	0xE0028018 IO1DIR	0XE0028014 IO1SET	0XE0028010 IO1PIN	0XE002801C IO1CLR
PORT2 Address & Name	0XE0028028 IO2DIR	0XE0028024 IO2SET	0XE0028020 IO2PIN	0XE002802C IO2CLR
PORT3 Address & Name	0XE0028038 IO3DIR	0XE0028034 IO3SET	0XE0028030 IO3PIN	0XE002803C IO3CLR
Description	Port Direction Control Resister Individually controls the direction of each port pin	Port Output Set Register Writing one produces high at the port pin. It controls the state of output pins	Regardless of pin direction we can read the current status of corresponding port pin	Port Output Clear register, writing one produces low at port pin and clears the corresponding port pin

مثال : می خواهیم پورت P1.3 را در میکروکنترلر LPC2138 به عنوان خروجی تنظیم و منطق ۱ را در خروجی قرار دهیم:

```
IO1DIR = 0x00000008;

IO1SET = 0x00000008;
```

یا می توانیم دو خط فوق را به صورت معادل صحیح تر زیر (با عملگر بیتی Or و Shift) بنویسیم:

```
IO1DIR |= (1<<3);

IO1SET |= (1<<3);
```

مراحل کلی انجام پروژه با میکروکنترلرARM

به طور کلی وقتی که قرار است پروژه ای با میکروکنترلرهای ARM انجام دهید ، بعد از مشخص شدن **هدف پروژه** و **صرفه اقتصادی** آن مراحل زیر به وجود می آید:

۱. **طراحی سخت افزار :** در این مرحله می بایست بر اساس هدف پروژه و شرایط مکانی به کارگیری پروژه ، نوع و مقدار تک تک المان های سخت افزار مورد نیاز طراحی و روی کاغذ یا در نرم افزار Proteus آورده شود.

۲. **طراحی نرم افزار :** در این مرحله ابتدا الگوریتم یا فلوچارت مورد نیاز در کاغذ رسم و سپس برنامه نویسی مورد نظر بر اساس آن در نرم افزار KEIL نوشته می شود.

۳. **شبیه سازی :** قبل از پیاده سازی عملی ، تست صحت عملکرد مدار در این مرحله توسط نرم افزار Proteus و برنامه نوشته شده در KEIL صورت می گیرد.

۴. **طراحی و ساخت برد :** طراحی PCB مدار مورد نظر در این مرحله و معمولا توسط نرم افزار Altium Designer صورت می گیرد. سپس فایل برد طراحی شده برای ساخت معمولا به یکی از شرکت های ساخت PCB داده می شود .

۵. **تست و عیب یابی :** با وصل منبع تغذیه به مدار و پروگرام کردن میکروکنترلر ، تست و عیب یابی برد مورد نظر در این مرحله صورت می گیرد.

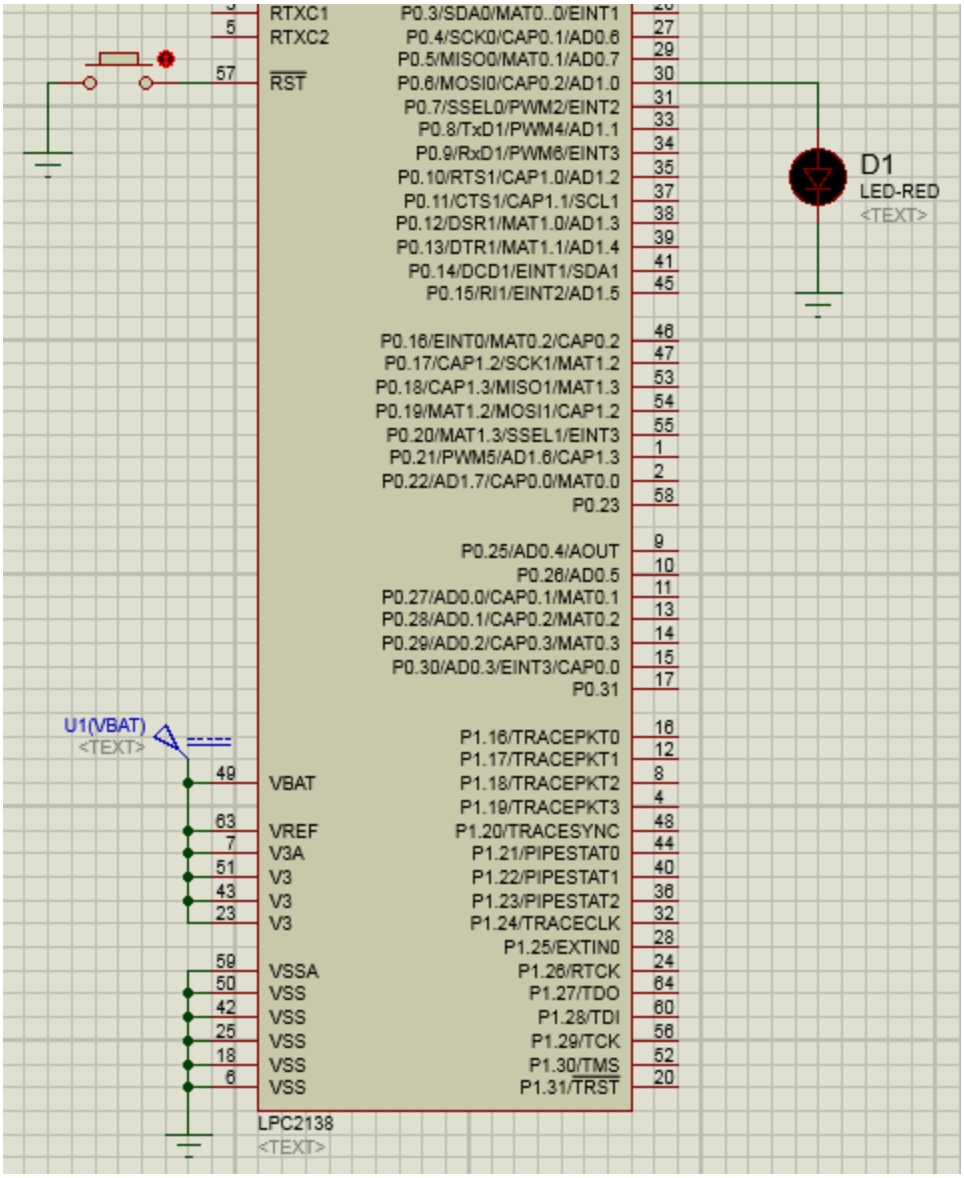
۶. **پشتیبانی و ارتقا :** بعد از ارائه به مشتری ، پشتیبانی پروژه ، ارتقای برنامه نویسی و بهبود عملکرد مدار در این مرحله می باشد.

تذکر : پیاده سازی میکروکنترلرهای ARM روی برد بورد بدون وجود PCB مناسب امکان پذیر نیست. در نتیجه باید اقدام به طراحی و ساخت برد دلخواه نمود و یا به جای طراحی و ساخت برد توسط خودمان ، میتوان از برد های آماده موجود در بازار استفاده کرد.

مثال عملی شماره ۱ : برنامه ای بنویسید که LED موجود روی P0.6 را با سرعت قابل دیدن به صورت چشمک زن روشن و خاموش کند. سپس آن را در نرم افزار Proteus شبیه سازی کرده و پس از اطمینان از عملکرد صحیح برنامه روی میکروکنترلر LPC2138 پیاده سازی نمایید.

حل:

مرحله اول : طراحی سخت افزار



مرحله دوم : طراحی نرم افزار

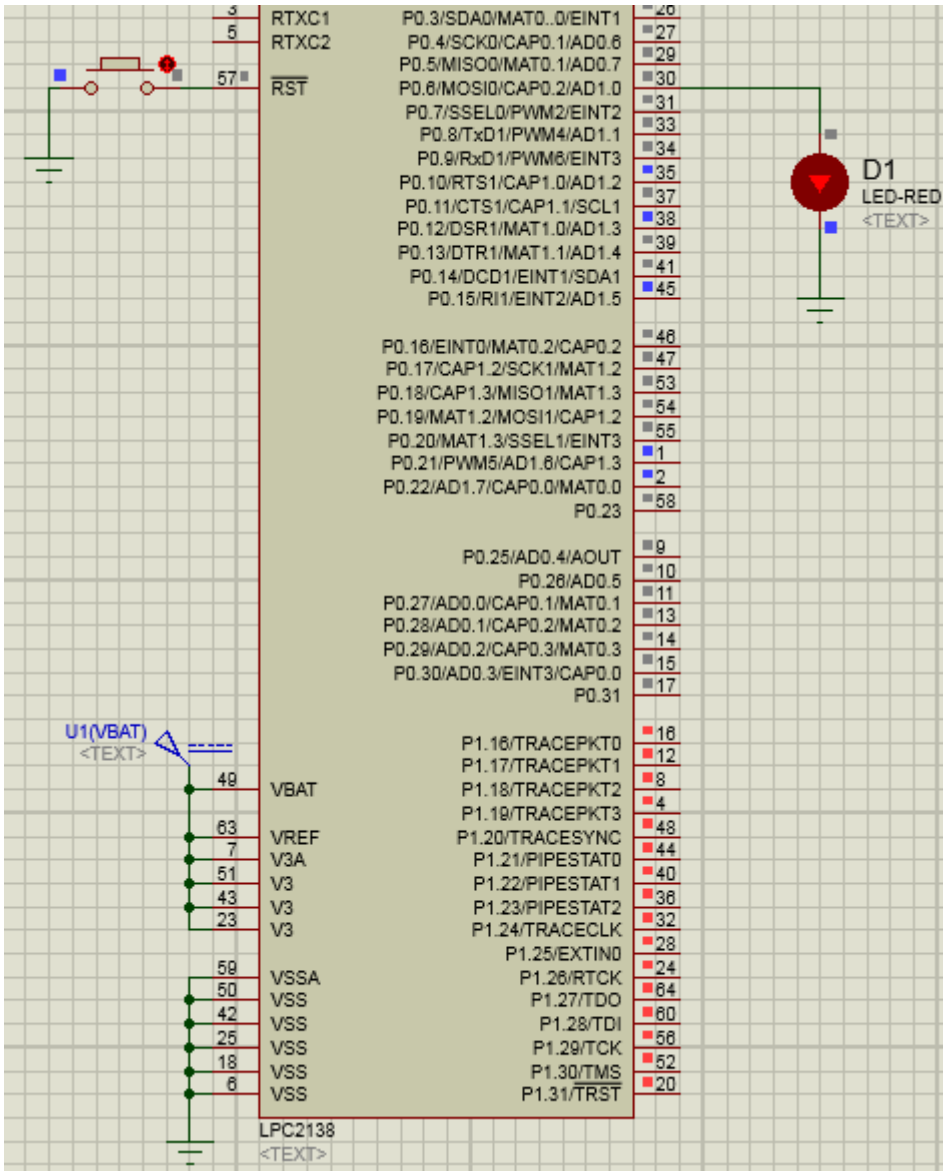
بعد از ساخت پروژه جدید و انجام تنظیمات به همان صورت گفته شده در نرم افزار KEIL ، برنامه خواسته شده را به صورت زیر طراحی و کد نویسی می کنیم:

```
#include <lpc213x.h>

void delay(void){
   unsigned int i=1000000;
   while(i--);
}

int main (void){

   IO0DIR |= (1<<6);

   while(1){

      IO0SET |= (1<<6);
      delay();
      IO0CLR |= (1<<6);
      delay();

   }
}
```

توضیح برنامه : در ابتدای برنامه به منظور ایجاد تاخیر یک تابع delay خودمان تعریف کردیم که این تابع ورودی و خروجی ندارد و درون آن یک حلقه while وجود دارد که دائما متغیر i تعریف شده را کم می کند تا زمانی که متغیر 0 شود و سپس برنامه از تابع بیرون می آید. در تابع اصلی برنامه ابتدا جهت پایه P0.6 را خروجی کردیم و سپس در حلقه اصلی برنامه ابتدا پایه را SET (منطق ۱) می کنیم و بعد از تاخیر پایه را CLEAR (منطق ۰) می کنیم و بعد از تاخیر مجدد برنامه تمام می شود.

نکته : در برنامه نویسی ARM و کامپایلر KEIL هدر فایلی شبیه delay.h که در AVR و کامپایلر Codevision بود وجود ندارد تا آن را به برنامه اضافه کرده و تابع delay به اندازه میکروثانیه یا میلی ثانیه داشته باشیم. بنابراین مشابه چنین تابعی خودمان در برنامه تعریف می کنیم.

مرحله سوم : شبیه سازی



نکته: سرعت اجرای برنامه در نرم افزار پروتئوس از واقعیت کندتر است.

مرحله چهارم : پیاده سازی

برای پیاده سازی این مثال از برد راه انداز LPC2138 استفاده می کنیم. این برد راه انداز را در فصل قبلی طراحی و ساخته بودیم. برای استفاده از این بورد راه انداز به دو عدد برد بورد نیاز داریم.

لینک خرید آنلاین بورد LPC2138 از فروشگاه الکترو ولت

به طور کلی میتوان گفت زمانی که از این برد راه انداز استفاده می کنیم ، برای پیاده سازی یک پروژه دو مرحله زیر وجود دارد:

- **بستن مدار مورد نظر روی برد برد**
- **پروگرام کردن میکروکنترلر**

ابتدا برد راه انداز را وسط دو عدد برد برد برد قرار داده و سپس مدار مثال ۱ را می بندیم. همانطور که در بخش چهارم به آن اشاره کردیم برای پروگرام کردن میکروکنترلرهای ARM سه روش مختلف وجود دارد : JTAG ، UART و برخی از میکرو ها با USB . اما چون میکروکنترلر LPC2138 از USB پشتیبانی نمی کند و استفاده از رابط JTAG نیازمند تهیه پروگرامر JLINK می باشد ، استفاده از پورت UART بهترین گزینه پیش رو است. بنابراین در پیاده سازی این پروژه از ارتباط سریال UART برای پروگرام کردن و به طبع آن از ماژول USB to Serial و نرم افزار FlashMagic استفاده خواهیم کرد.

نکته : قبل از اتصال ماژول USB to Serial به کامپیوتر نیاز به نصب درایور آن دارید. در صورتی که از ماژول PL2303 استفاده می کنید میتوانید درایور آن را از لینک زیر دریافت و اقدام به نصب نمایید. (طبق گفته سازنده در تمامی ویندوزها قابل نصب است)

PL2303_Prolific_DriverInstaller_v1_12_0

نکته : ولتاژ عملکرد آی سی ۳,۳ ولت است که باید توسط رگولاتور به آن اعمال شود اما پایه های ورودی/خروجی میکرو توانایی تحمل ولتاژ ۵ ولت به صورت لحظه ای را دارند.

تذکر مهم : ماژول PL2303 دارای ولتاژ عملکرد ۵ ولت می باشد ، در نتیجه به علت اینکه میکروکنترلرهای ARM دارای ولتاژ عملکرد ۳,۳ ولت می باشد ، بهتر است از ماژول FT232 که توانایی عملکرد در هر دو حالت ۳,۳ ولت و ۵ ولت را دارد برای پروگرام نمودن استفاده کرد.

لینک خرید آنلاین ماژول FT232 از فروشگاه الکترو ولت

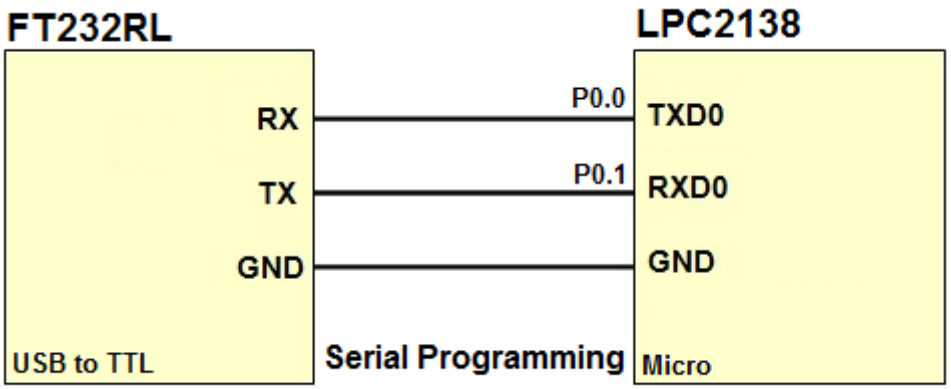
همچنین آخرین نسخه نرم افزار Flash Magic را از لینک روبرو دریافت و آن را نصب نمایید. **FlashMagicTool**

پروگرام کردن برد راه انداز LPC2138 به روش سریال

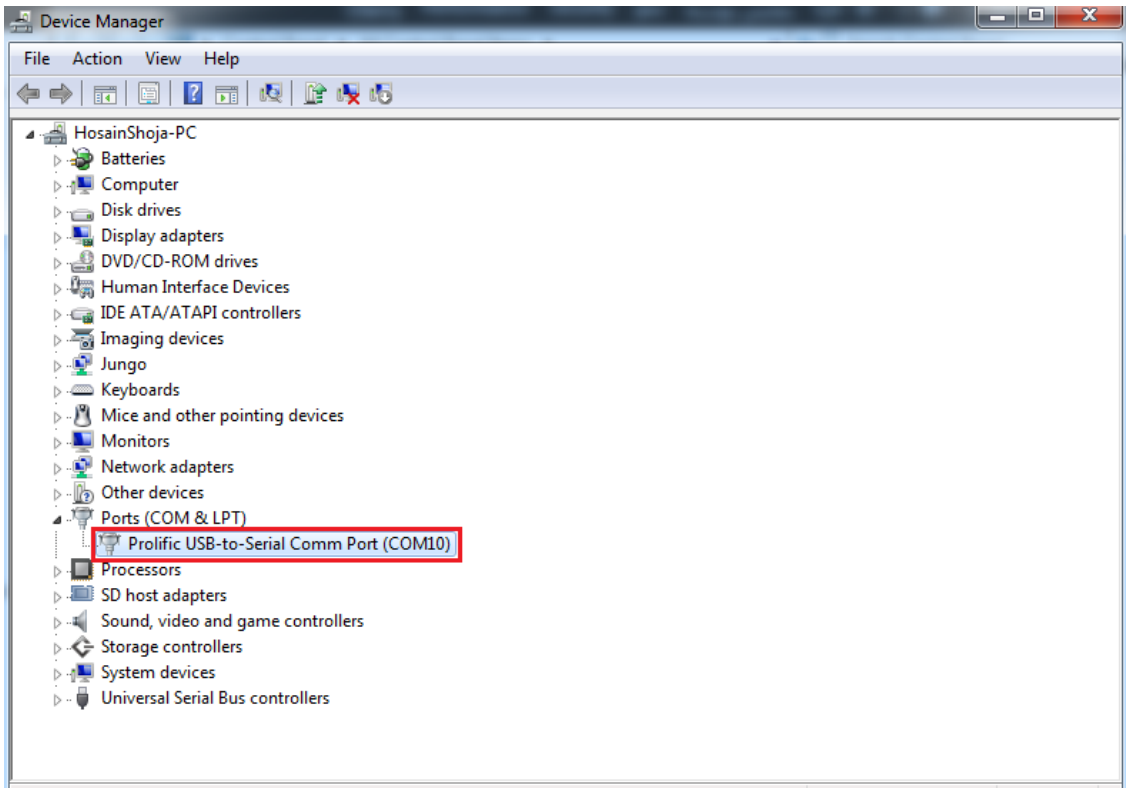
پروگرام کردن LPC2138 به روش سریال را میتوان به ترتیب به پنج مرحله زیر تقسیم کرد:

۱. اتصال ماژول USB to Serial به پورت UART0 میکرو
۲. اتصال ماژول USB to Serial به پورت USB کامپیوتر
۳. برقراری تغذیه برد راه انداز و بردن میکرو به حالت پروگرام سریال (بوت سریال)
۴. استفاده از نرم افزار Flash Magic جهت پروگرام کردن
۵. تغییر حالت میکرو کنترلر از بوت سریال به حالت کار معمولی

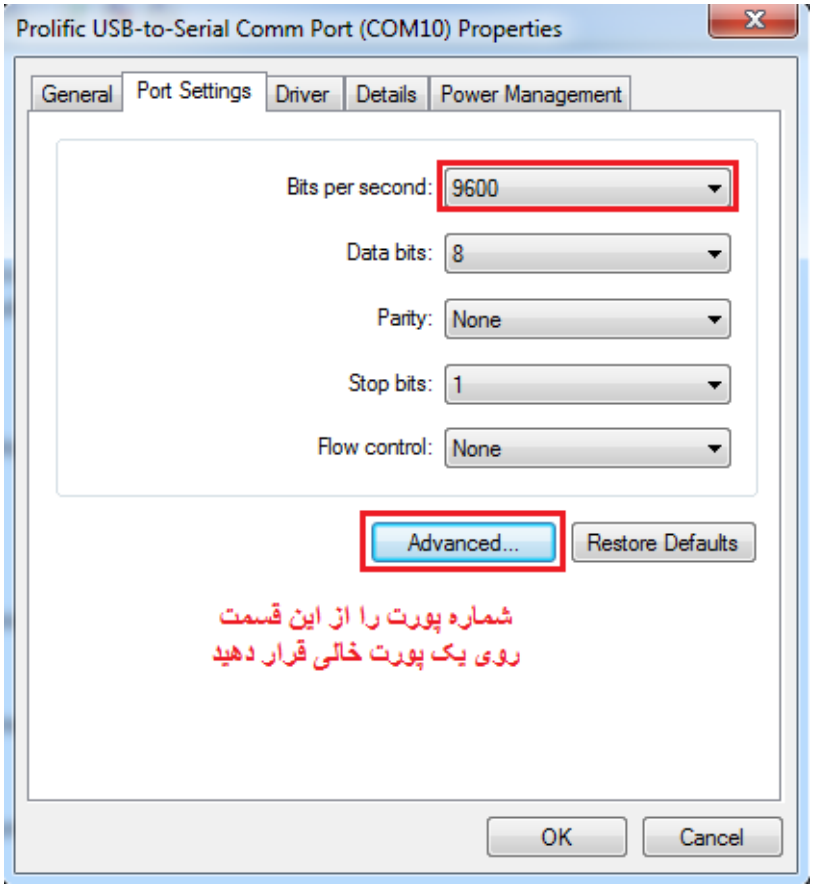
۱- اتصال ماژول USB to Serial به پورت UART0 میکرو : ماژول FT232 را طوری وصل می کنیم که پایه **Rx** ماژول به پایه **P0.0** و **Tx** ماژول به پایه **P0.1** متصل شده باشد. شکل زیر نحوه اتصال ماژول FT232 به پورت سریال جهت پروگرام کردن را نشان می دهد.



۲- اتصال ماژول USB to Serial به پورت USB کامپیوتر : پس از اتصال ماژول FT232 به USB کامپیوتر به Control Panel ویندوز خود رفته و در قسمت Device Manager و سپس در قسمت Ports(Com&LPT) پورت سریال مجازی که ماژول دارد را شناسایی کنید. همانطور که در شکل زیر مشاهده می کنید ، پورتی که در اینجا تنظیم شده است COM10 می باشد.

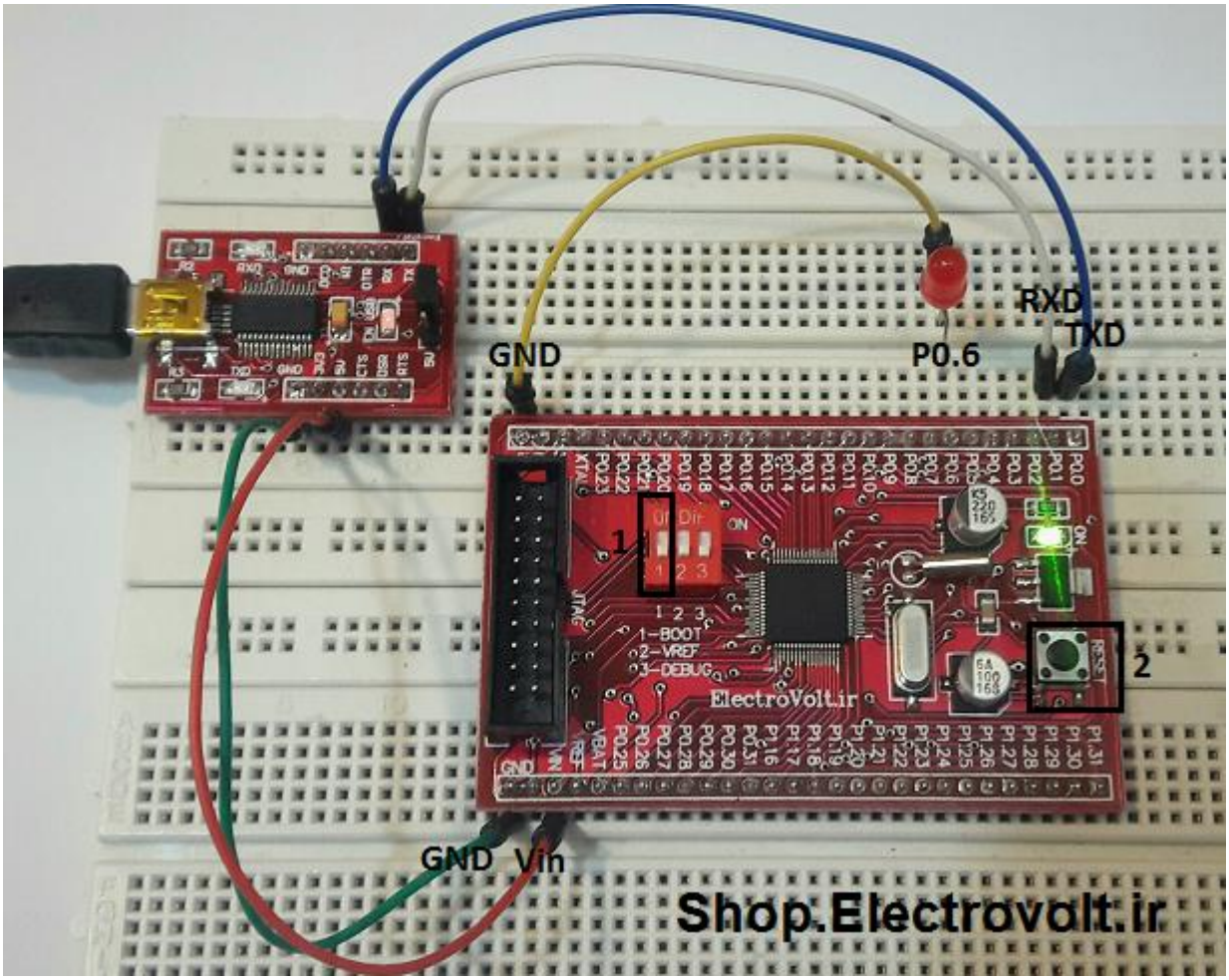


همچنین میتوانید با دوبار کلیک روی گزینه قرمز رنگ فوق وارد Properties شوید و در آنجا تنظیمات پورت سریال را تغییر دهید. برای این منظور در پنجره Propertiesوارد سربرگ Port Setting شده و تنظیمات آن را همانند زیر تغییر دهید.

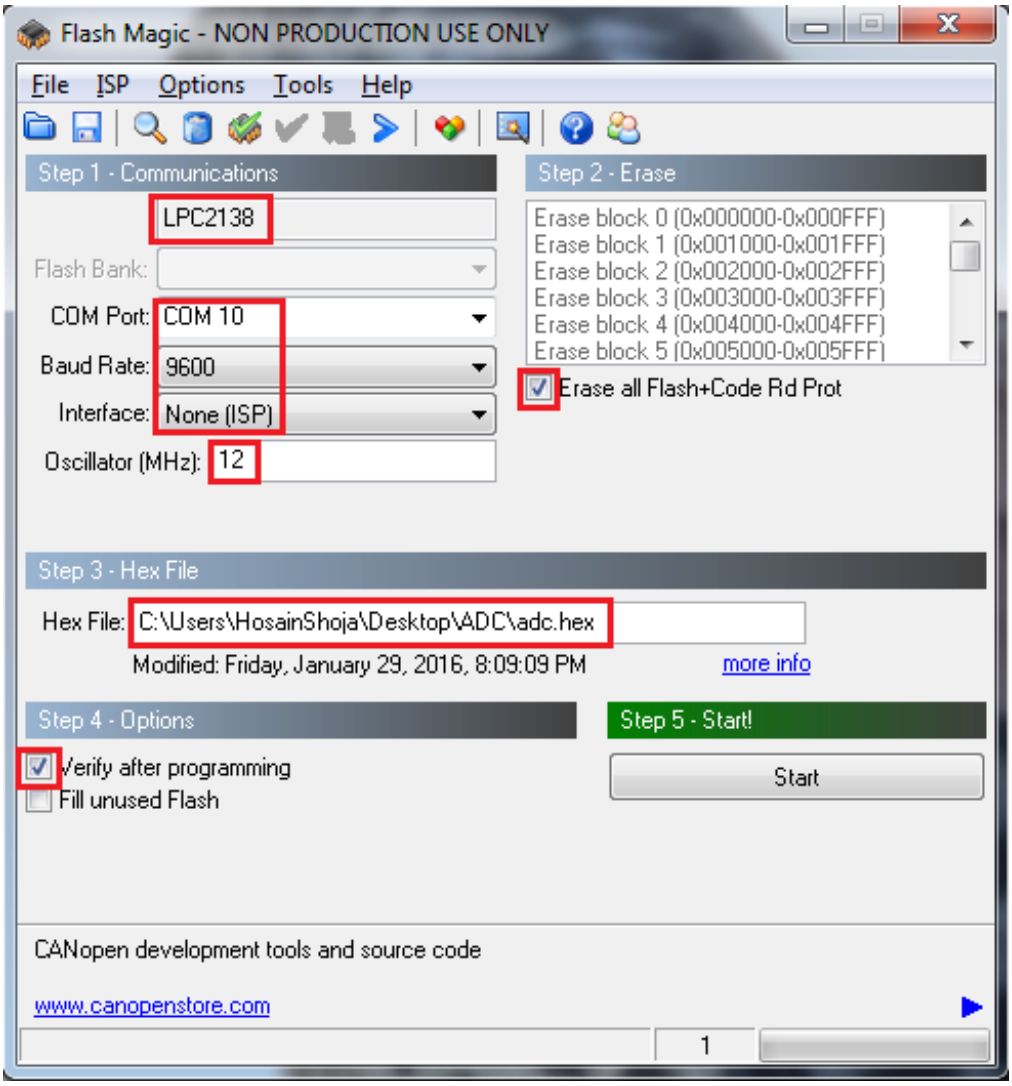


همچنین میتوانید با زدن گزینه Advanced شماره پورت سریال را از COM10 به هر شماره دیگری تغییر دهید. توجه کنید که شماره پورتی که انتخاب می کنید حتما خالی باشد یعنی توسط دستگاه یا نرم افزارهای دیگر اشغال نشده باشد.

۳- برقراری تغذیه برد راه انداز و بردن میکرو به حالت پروگرام سریال (بوت سریال) : زمانی که می خواهیم برنامه مورد نظر را روی میکرو پروگرام کنیم باید ابتدا میکرو کنترلر را به بوت سریال ببریم. بوت سریال یعنی حالتی که میکرو در آن برنامه را از پورت UART0 می پذیرد و پروگرام می شود. به منظور بردن میکروکنترلر LPC2138 به بوت سریال کافی است ابتدا پایه P0.14 را زمین کرده و سپس میکرو را ریست کنیم. در برد راه انداز ، این پایه توسط یک دیپ سوئیچ (DIP Switch) به زمین متصل خواهد شد. بنابراین با فعال کردن سوئیچ شماره ۱ و سپس ریست کردن میکرو به بوت سریال می رویم. شکل زیر مراحل کار را نشان می دهد. همانطور که مشاهده می کنید در این شکل از پایه +5 ولت ماژول FT232RL برای تغذیه بورد راه انداز و نیز پروگرام کردن میکرو استفاده شده است.

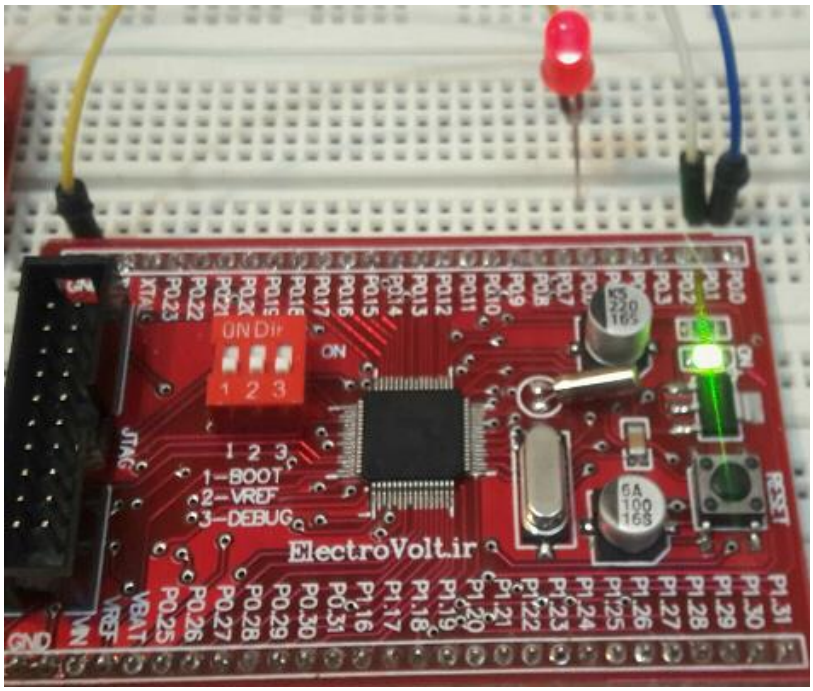


۴- **استفاده از نرم افزار Flash Magic جهت پروگرام کردن** : در این نرم افزار ابتدا تنظیمات Step1 و Step2 را مطابق شکل زیر انجام می دهیم. در قسمت Com Port باید شماره پورتی را که ماژول به آن متصل شده است را تنظیم کنیم. سپس در Step3 مسیر فایل hex ساخته شده توسط برنامه Keil را می دهیم. در Step4 نیز مطابق شکل تیک Verifyرا می زنیم و در نهایت Start را کلیک می کنیم. در صورتی که همه نکات گفته شده رعایت شده باشد برنامه به خوبی پروگرام می شود.



۵- تغییر حالت میکرو کنترلر از بوت سریال به حالت کار معمولی : بعد از پروگرام شدن میکروکنترلر ابتدا دیپ سوئیچ را از بوت سریال **خارج** می کنیم (پایه P0.14 از منطق 0 خارج می شود) سپس با فشار دادن دکمه ریست ، میکرو ریست شده و به حالت کار عادی باز میگردد و اجرای برنامه مورد نظر آغاز می شود.

پیاده سازی مثال عملی شماره ۱ با میکروکنترلرLPC2138



پروگرام کردن برد راه انداز LPC2138 با JLINK

JLINK و JTRACE نام دیباگرهای میکروکنترلرهای ARM است که توانایی پروگرام کردن میکرو با سرعت بالا و قابلیت عیب یابی میکرو از طریق پورت JTAG را دارد. این پروگرامرها ساخت شرکت آلمانی Segger می باشد که در مدل های مختلفی تولید شده است. در جدول شکل زیر تمامی این مدل ها را مشاهده می کنید که از سایت segger.com استخراج شده است.

Model	Price[4]	Download Speed into RAM[6]	Download Into Flash[1]	GDB Server	Unlimited Flash Breakpoints	J-Flash	RDI / RDDI	Ethernet
J-Link PRO	798 EUR 998 USD	3.0 MByte/sec	✓	✓	✓	✓	✓	✓
J-Link ULTRA+[2]	598 EUR 748 USD	3.0 MByte/sec	✓	✓	✓	✓	✓	✗
J-Link PLUS	498 EUR 598 USD	1.0 MByte/sec	✓	✓	✓	✓	✓	✗
J-Link BASE	298 EUR 378 USD	1.0 MByte/sec	✓	✓	✗	✗	✗	✗
J-Link EDU[3]	42 EUR 60 USD	1.0 MByte/sec	✓	✓	✓	✗	✗	✗
Debug probes with built-in Trace memory								
J-Trace PRO Cortex-M	1,390 EUR 1,748 USD	3.0 MByte/sec	✓	✓	✓	✓	✓	✓
J-Trace Cortex-M	995 EUR 1248 USD	3.0 MByte/sec	✓	✓	✓	✓	✓	✗
J-Trace ARM[5]	995 EUR 1248 USD	1.0 MByte/sec	✓	✓	✓	✓	✓	✗

✓ Included ✗ Not included

[1] For use from within an IDE

[2] Best value: Includes all software enhancements and maximum performance

[3] Educational use

[4] Price does not include VAT or Sales Tax which may be applicable in some regions

[5] ARM 7/9 cores only

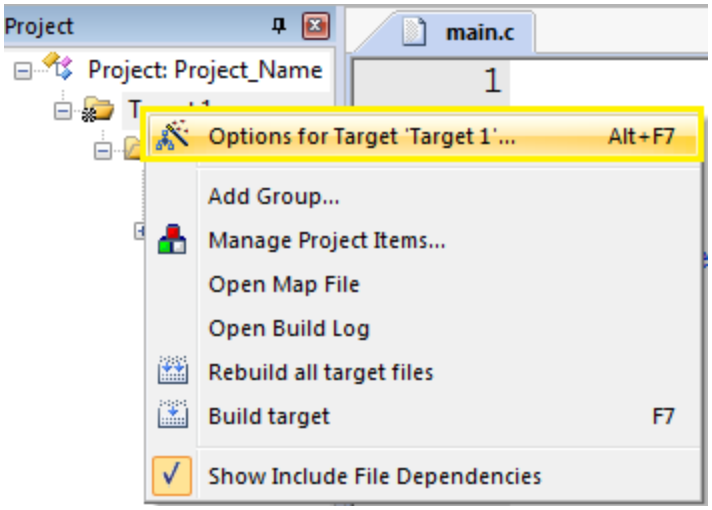
[6] The download speeds listed here are the peak download speeds that can be achieved by the particular J-Link model. The actual download speed may be lower as it depends on various factors, such as, but not limited to: The selected debug interface & speed, the CPU core and its operating frequency, other devices in the JTAG chain in case JTAG is used as target interface.

اما به خاطر قیمت این دیباگر ها مدل های مختلف چینی و ایرانی از JLINK در بازار وجود دارند که کپی شده از نسخه اصلی هستند و قیمت و کیفیت پایین تری دارند. در شکل زیر دو نمونه JLINK V8 شرکت های ECA و کویر الکترونیک را مشاهده می کنید.

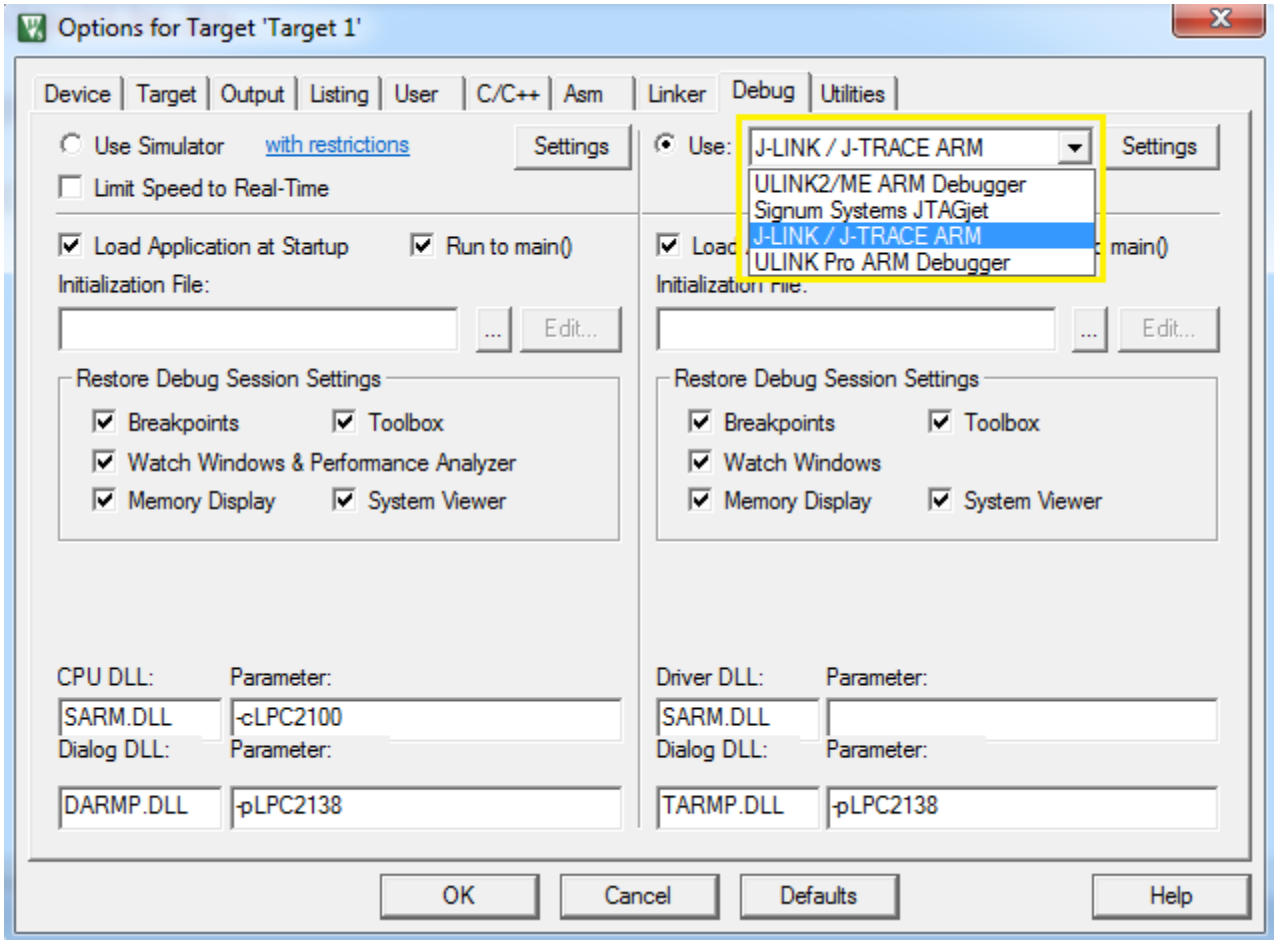
با استفاده از هر کدام از این JLINK ها که خریداری شود میتوان میکروهای ARM را به دو روش پروگرام و دیباگ نمود. روش اول با استفاده از خود نرم افزار KEIL است و روش دوم با استفاده از نرم افزار J-Flash شرکت Segger می باشد.

راهنمای تنظیمات JLINK در نرم افزار KEIL

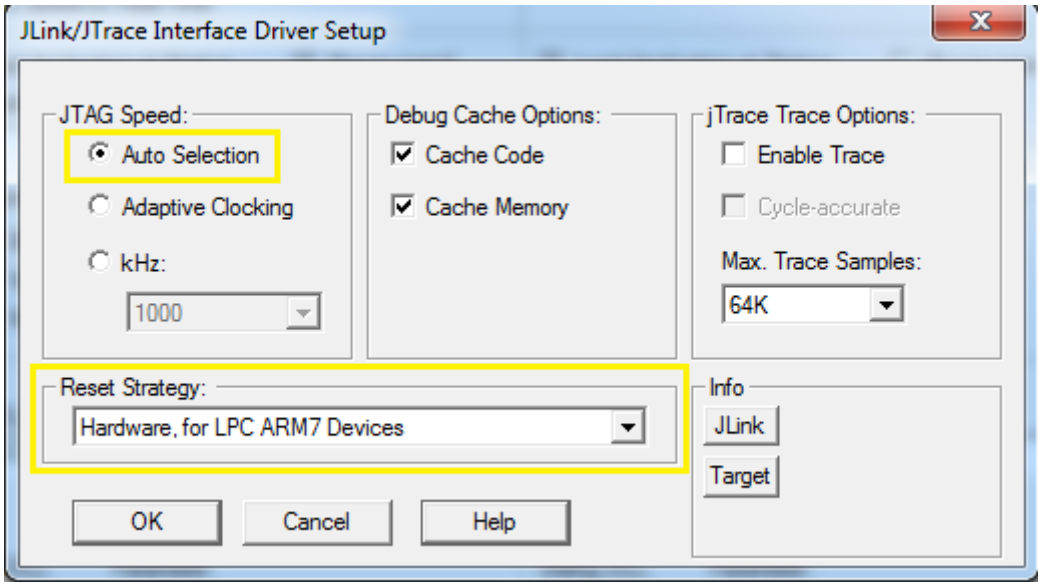
برای پروگرام کردن ابتدا باید تنظیمات آن را در KEIL انجام دهید. ابتدا JLINK را به کامپیوتر متصل کرده و سپس مراحل زیر را در KEIL انجام دهید.



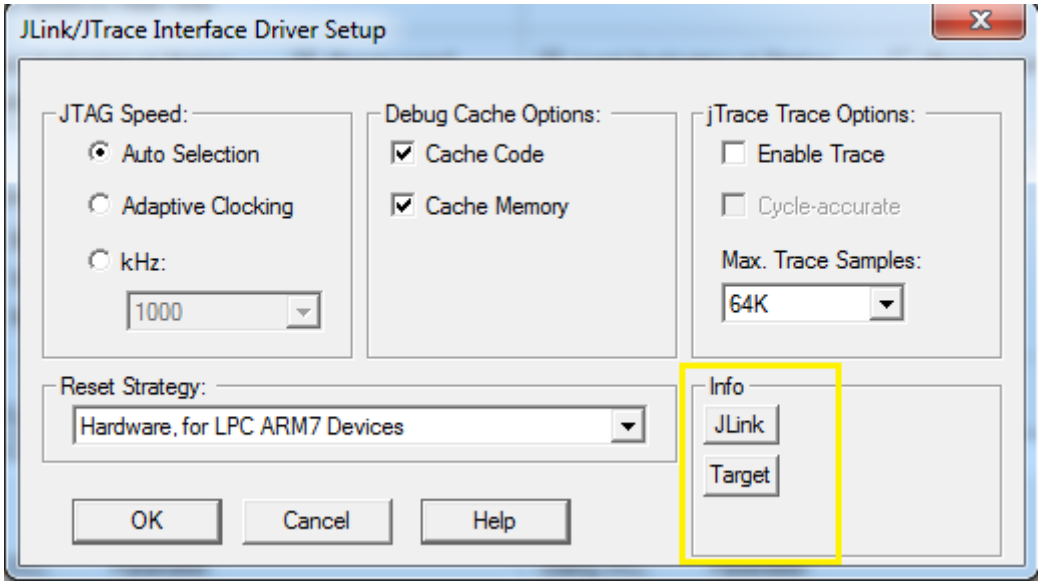
در پنجره Option به سربرگ Debug رفته و JLINK/J-TRACE ARM را انتخاب کنید:



بعد از انتخاب گزینه قبل روی Setting کلیک کنید و مطابق شکل زیر تنظیمات را انجام دهید

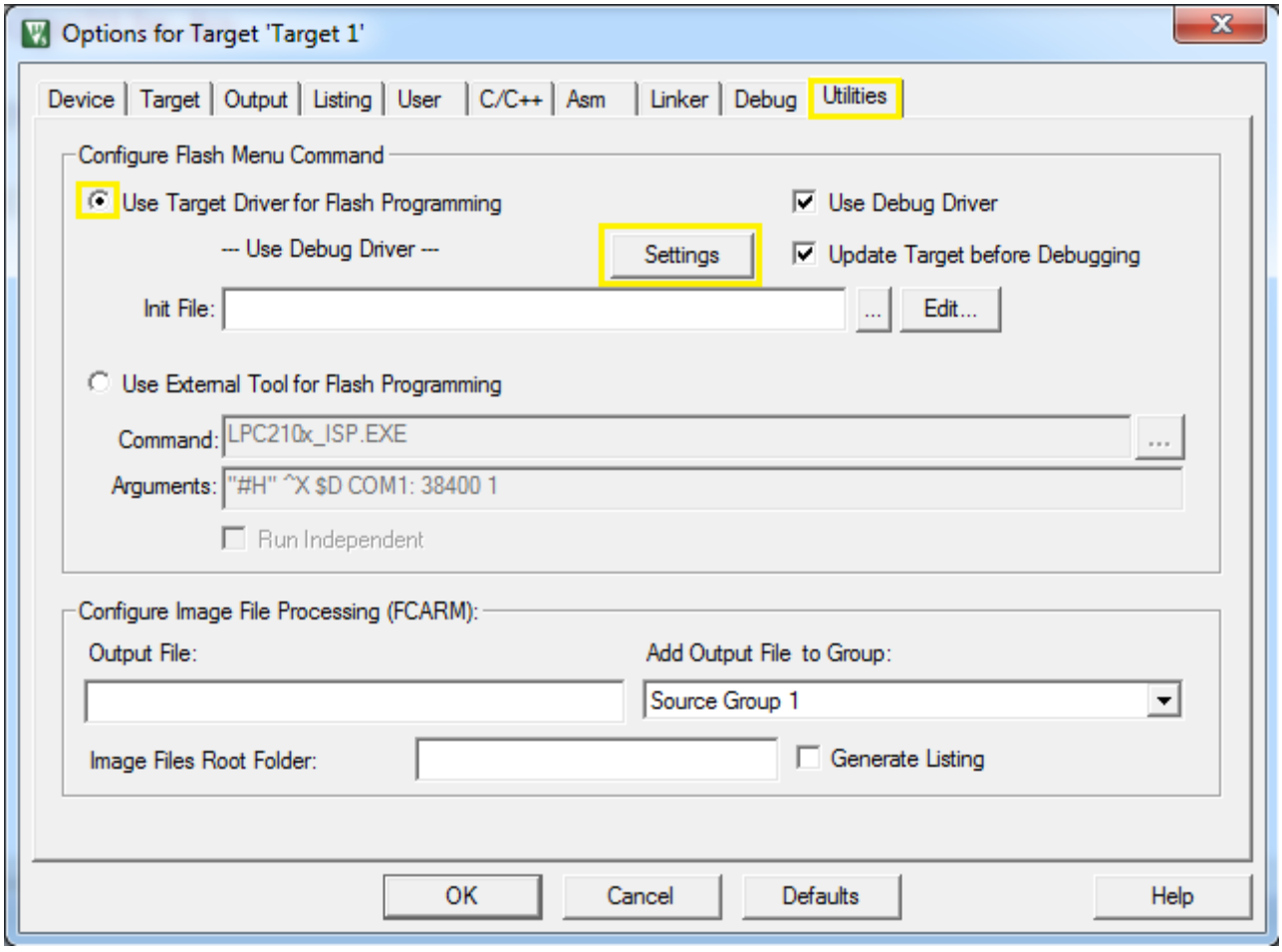


همچنین میتوانید با کلیک بر روی Jlink مشخصات دیباگر خود را مشاهده نمایید.

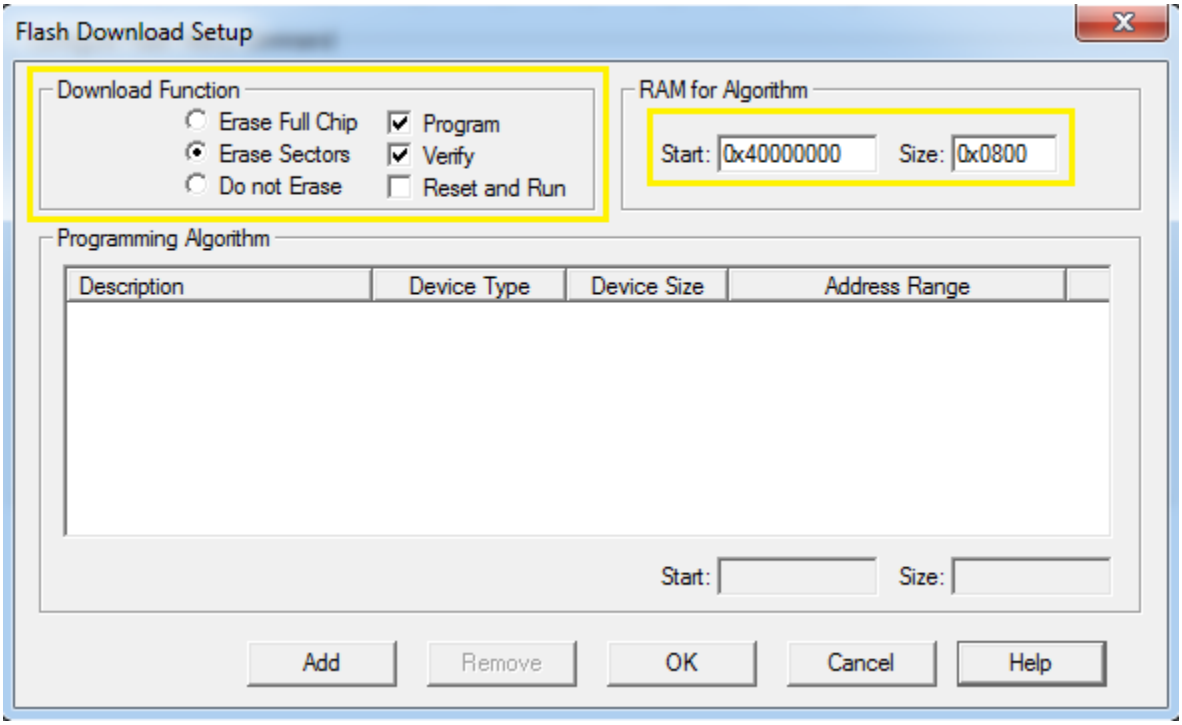


با کلیک بر روی OK این مرحله به اتمام می رسد.

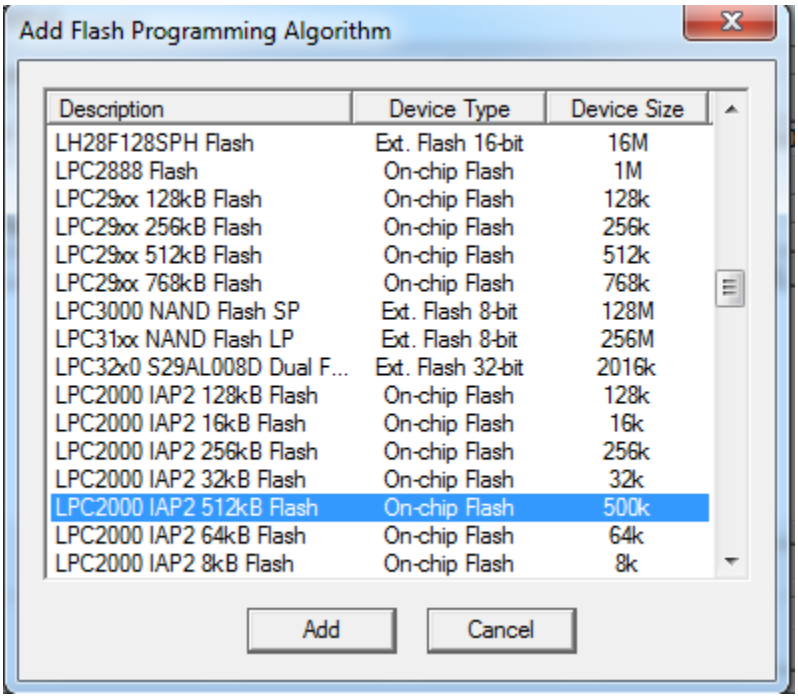
سپس به سربرگ Utilities رفته و مطابق شکل زیر تنظیمات آن را انجام دهید:



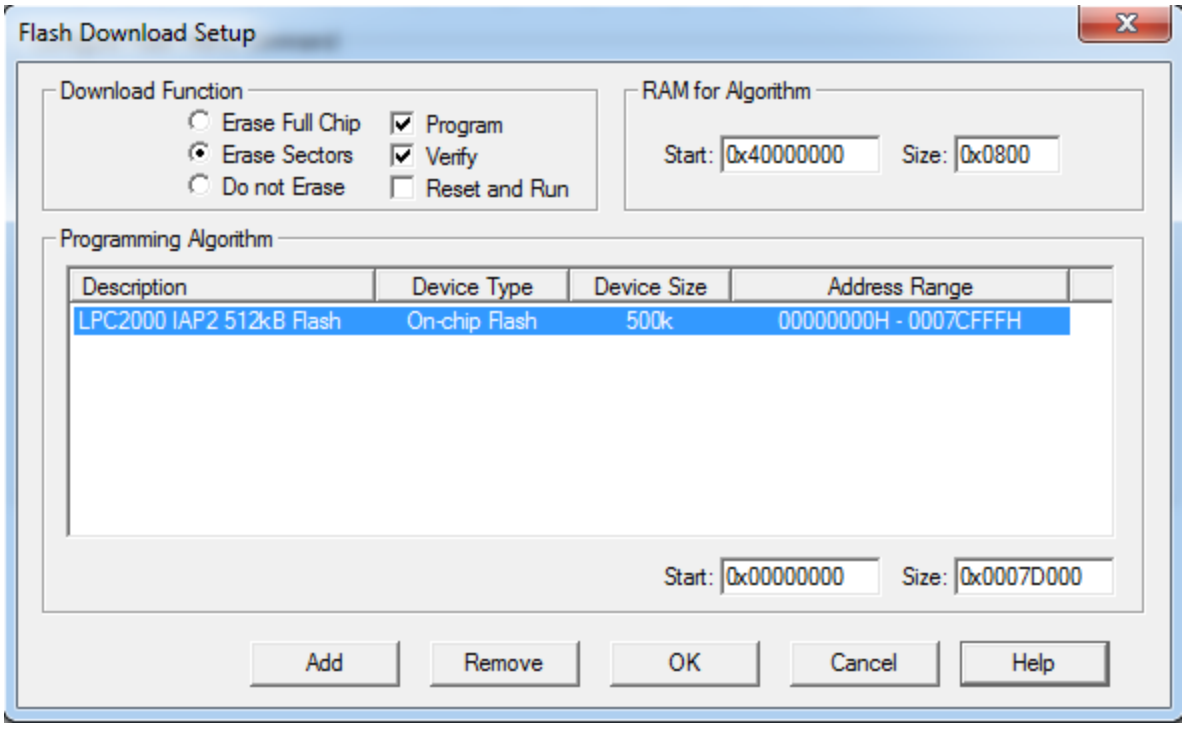
سپس روی Setting کلیک کرده و تنظیمات RAM و Flash را انجام دهید.



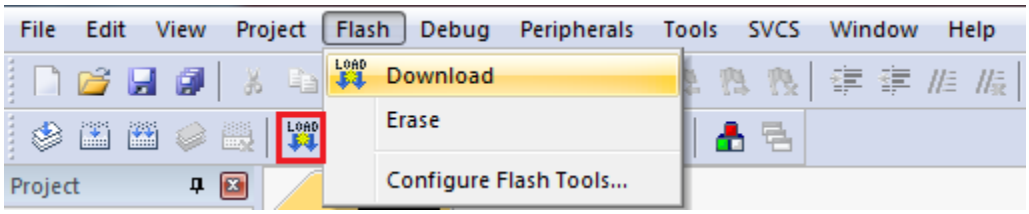
روی Add کلیک کرده و با توجه به نوع میکرو کنترلر مورد نظر سایز حافظه Flash را انتخاب نمایید. برای مثال جهت استفاده برای LPC2138 روی گزینه زیر قرار دهید و سپس Ok نمایید.



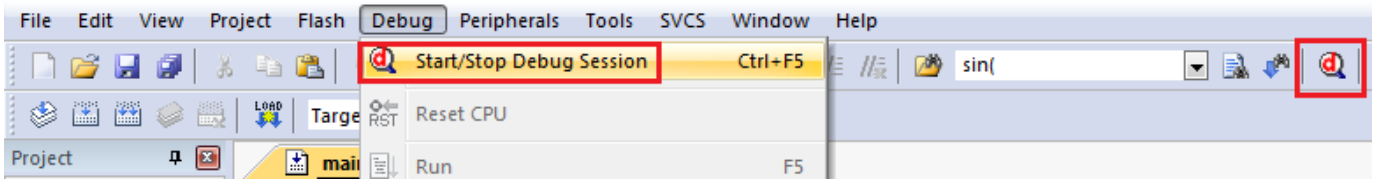
با قرار گرفتن مقدار حافظه Flash تنظیمات به پایان می رسد.



برای پروگرام کردن میکرو بعد از اتصال JTAG و فعال کردن جامپر مربوطه و ریست کردن میکرو از منوی Flash گزینه Download را کلیک کنید و یا روی نوار ابزار این کار را انجام دهید.



جهت استفاده از دیباگر از منوی Debug روی Start/Stop Debuging یا روی نوار ابزار کلیک نمایید.



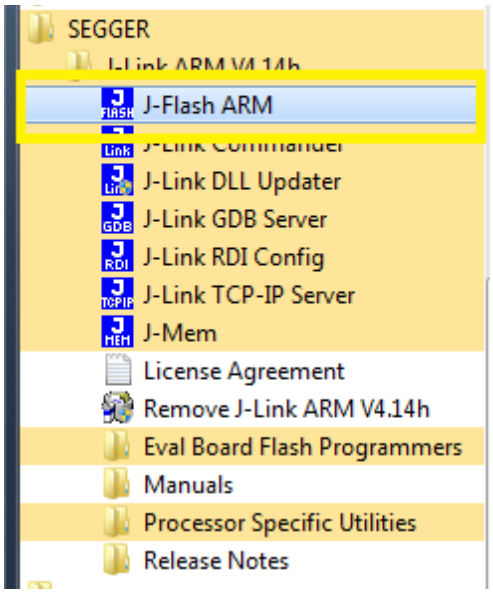
حال شما میتوانید با استفاده از دکمه های زیر برنامه خود را دیباگ کنید:

راهنمای تنظیمات JLINK در نرم افزار J-Flash

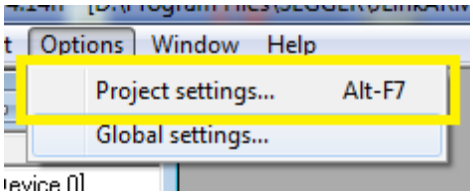
ابتدا به لینک زیر رفته و آخرین نسخه نرم افزارهای شرکت Segger که شامل J-Flash درایورها و نرم افزارهای کاربردی دیگری نیز می باشد را دانلود و سپس نصب نمایید:

https://www.segger.com/jlink-software.html

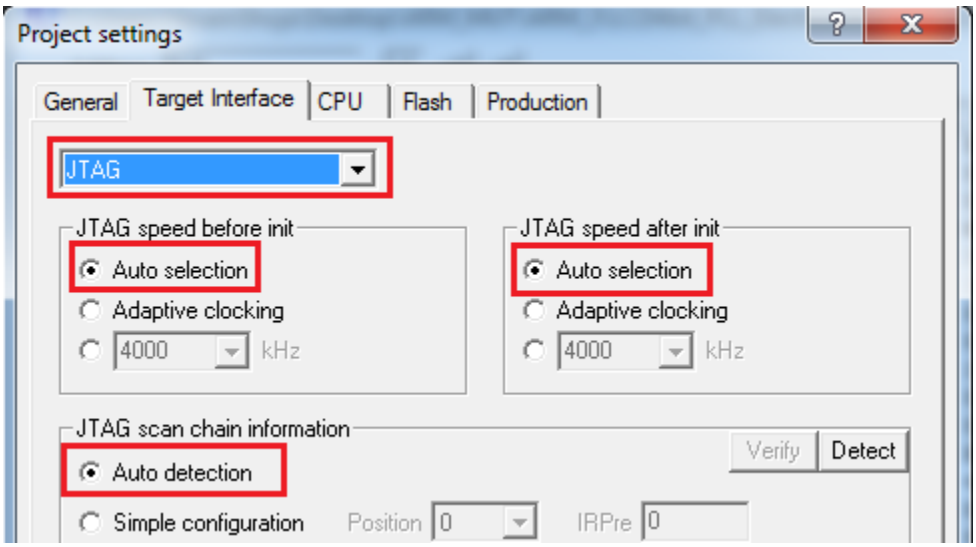
با نصب این نرم افزارها درایورهای JLINK نیز نصب می شود و با اتصال پروگرامر به کامپیوتر به طور خودکار شناخته می شود. بعد از نصب از نرم افزارهای متعددی وجود دارد که J-Flash برای پروگرام کردن می باشد. آن را از منوی Start باز نمایید.



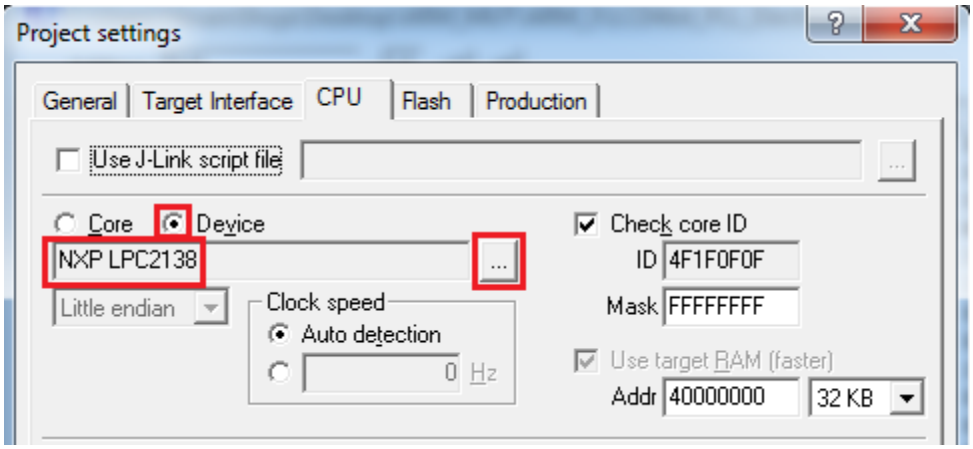
بعد از باز کردن برنامه از منوی Options گزینه ی Project Setting را انتخاب نمایید.



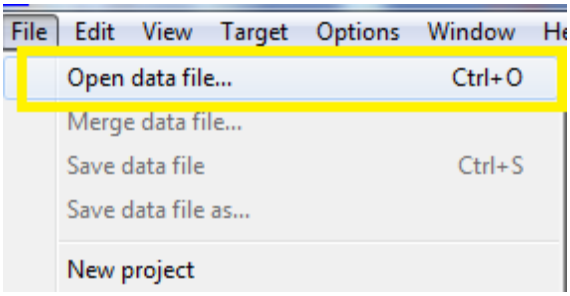
در سربرگ Target Interface تنظیمات زیر را انجام دهید.



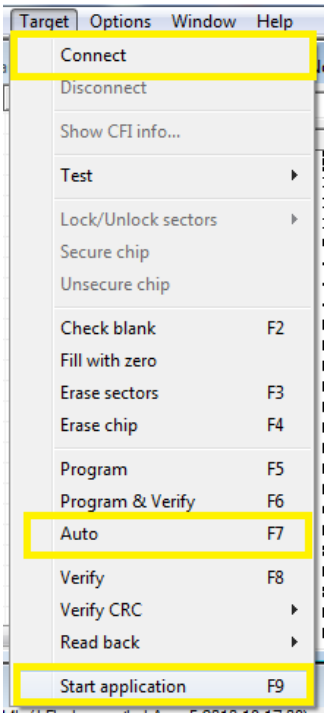
در سربرگ CPU روی Device انتخاب کرده و نام میکرو مورد نظر خود را انتخاب نمایید. در پایان Ok کنید.



جهت باز کردن فایل Hex مورد نظر از منوی File گزینه Open data file را انتخاب کنید و فایل باینری یا هگز خود را انتخاب نمایید.



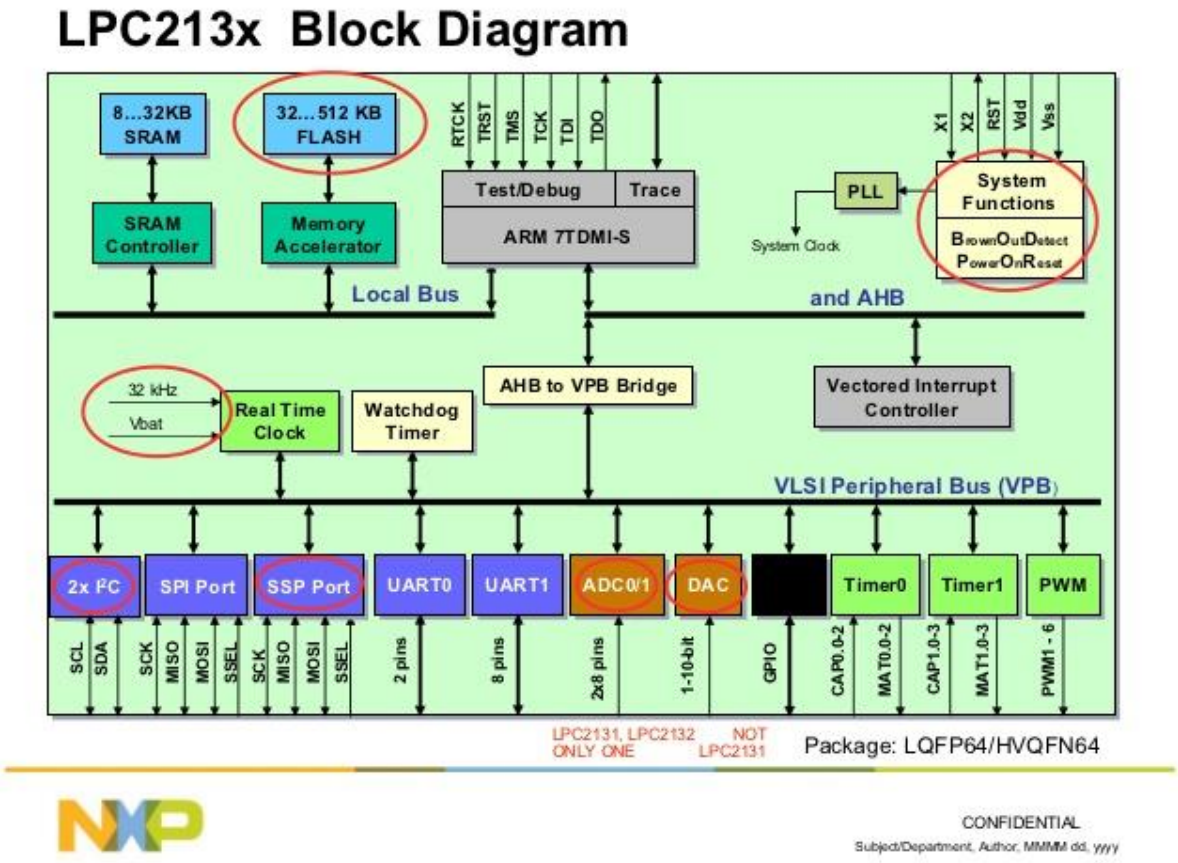
حال با استفاده از گزینه Connect در منوی Target ابتدا ارتباط بین میکرو و کامپیوتر را برقرار نمایید. در صورت صحیح بودن همه موارد اتصال برقرار شده و میتوان با استفاده از گزینه Auto میکرو را پروگرام نمود. همچنین بعد از پروگرام شدن با زدن گزینه Start Application برنامه اجرا می شود.



فصل ۷ – آموزش واحد کنترل سیستم و راه اندازی PLL

مقدمه

یکی از مهمترین مباحث در راه اندازی میکروکنترلرهای ARM کنترل توان مصرفی و نیز کلاک کاری میکرو (سرعت پردازش CPU) است. در میکروکنترلرهای AVR مبحثی به نام فیوز بیت وجود داشت که وظیفه های کنترلی از جمله کنترل کلاک میکرو را برعهده داشت. اما تنظیمات توان و کلاک میکرو در میکروکنترلرهای ARM7 ، درون واحدی به نام کنترل سیستم (System Control) صورت می گیرد. علت این مسئله رابطه تنگاتنگ میان سرعت کلاک و توان مصرفی آی سی است. هر چه CPU دارای فرکانس بالاتری باشد ، سرعت اجرای دستورات افزایش یافته و به طبع آن مصرف توان افزایش می یابد. اما به منظور کاهش توان مصرفی و مدیریت روی کلاک میکرو در میکروکنترلرهای ARM7 ، معمولا با راه اندازی واحد PLL ، CPU را در سرعت بالاتر (AHB) قرار داده و بقیه واحد ها را در سرعت کمتر (APB) راه اندازی می کنند.



معرفی واحد کنترل سیستم

این واحد یکی از مهمترین واحدهای کنترل و مدیریت کلی روی عملکرد آی سی است. در این واحد مدیریت کلی روی مصرف توان ، تولید و پخش کلاک به واحدهای دیگر ، تنظیمات وقفه خارجی ، منابع ریست و نیز امنیت و اطلاعات سیستم وجود دارد .رجیسترهای تنظیمات واحد کنترل سیستم را در شکل زیر به طور کامل مشاهده می کنید.

Table 11. Summary of system control registers

Name	Description	Access	Reset value[1]	Address
External Interrupts				
EXTINT	External Interrupt Flag Register	R/W	0	0xE01F C140
INTWAKE	External Interrupt Wakeup Register	R/W	0	0xE01F C144
EXTMODE	External Interrupt Flag register	R/W	0	0xE01F C148
EXTPOLAR	External Interrupt Wakeup Register	R/W	0	0xE01F C14C
Memory Mapping Control				
MEMMAP	Memory Mapping Control	R/W	0	0xE01F C040
Phase Locked Loop				
PLLCON	PLL Control Register	R/W	0	0xE01F C080
PLLCFG	PLL Configuration Register	R/W	0	0xE01F C084
PLLSTAT	PLL Status Register	RO	0	0xE01F C088
PLLFEED	PLL Feed Register	WO	NA	0xE01F C08C
Power Control				
PCON	Power Control Register	R/W	0	0xE01F C0C0
PCONP	Power Control for Peripherals	R/W	0x03BE	0xE01F C0C4
APB Divider				
APBDIV	APB Divider Control	R/W	0	0xE01F C100
Reset				
RSID	Reset Source Identification Register	R/W	0	0xE01F C180
Code Security/Debugging				
CSPR	Code Security Protection Register	RO	o	0xE01F C184
Syscon Miscellaneous Registers[2]				
SCS	System Controls and Status	R/W	0	0xE01F C1A0

[1] Reset value reflects the data stored in used bits only. It does not include reserved bits content.

[2] Available in LPC213x/01devices only.

در این بخش از آموزش تنها رجیسترهای PLL ، Power Control و APB Divider که از اهمیت بسیار زیادی برخوردار هستند معرفی و تشریح می شود. در عمل بیشتر از این رجیسترها استفاده می گردد. برای اطلاعات بیشتر میتوانید به **UM10120** مراجعه نمایید.

رجیسترهای کنترل توان و نحوه مدیریت توان

کنترل توان مصرفی میکروکنترلر (واحد Power Control) یکی از واحدهای مهم است چرا که در اکثر پروژه ها سعی داریم کمترین توان ممکن مصرف شود تا سیستم بهینه گردد. برای کنترل توان مصرفی دو راهکار وجود دارد:

- حالت های کاهش مصرف توان (رجیستر PCON) : شامل دو حالت Idel Mode وPower-Down Mode
- کنترل توان مصرفی واحدهای جانبی (رجیستر PCONP) : قطع کردن تغذیه واحدهای جانبی بی کار

رجیستر PCON : (صفحه ۴۱ از UM10120)

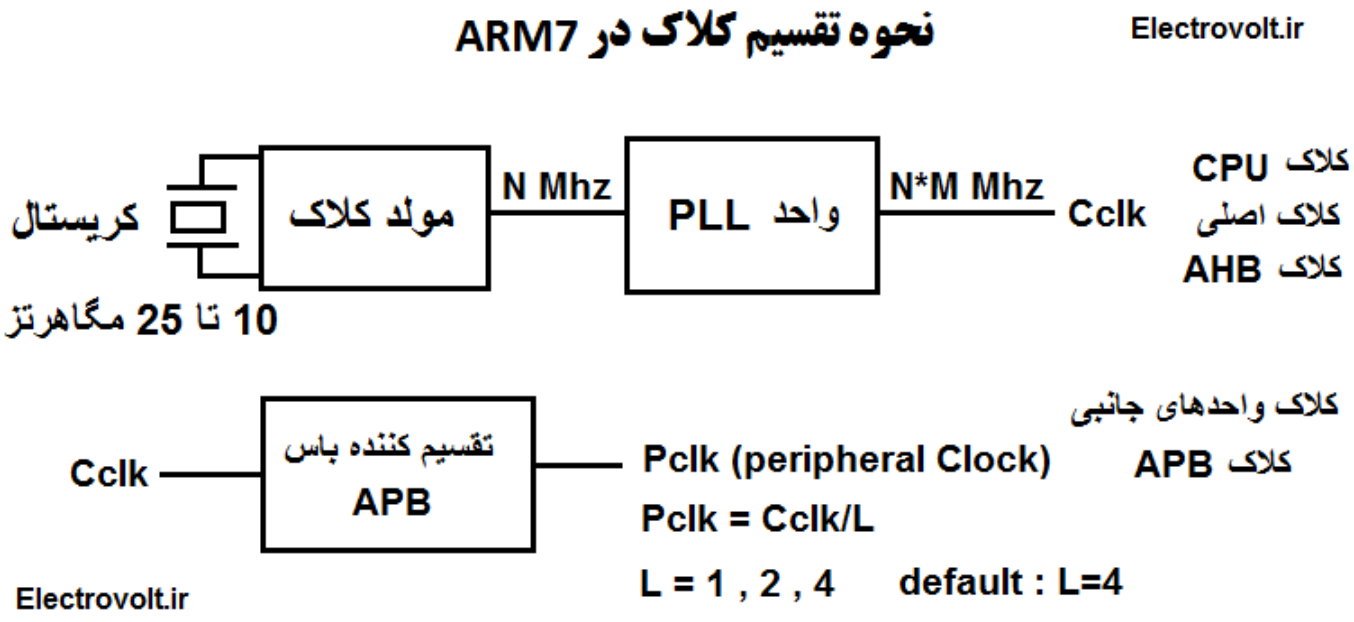
توسط این رجیستر میتوان میکروکنترلر را به حالت های کم مصرف برد. در حالت Idle Mode ، کلاک CPU از کار افتاده و CPU به خواب می رود در حالی که دیگر واحدهای جانبی به کار خود مشغول هستند. در حالت Power-Down Mode، کلاک تمام آی سی از کار افتاده و کل میکرو به خواب می رود. برای فعال کردن حالت Idle Modeدر هر کجای برنامه کافی است بیت صفرام از رجیستر PCON را ۱ کنیم (PCON |= 1) و نیز برای فعال کردن-Power Down Modeدر هر کجای برنامه کافی است بیت اول از رجیستر PCON را ۱ کنیم .(PCON |= 2) برای غیر فعال کردن این حالت ها دو راه وجود دارد. راه اول زمانی است که بیت های مربوط به هر یک از حالت ها ، در هر کجای برنامه صفر شود. در غیر این صورت به محض فعال شدن هر یک از حالت های فوق با آمدن اولین وقفه خارجی میکرو کنترلر از خواب بیدار شده و به حالت کار نرمال بر می گردد.

رجیستر PCONP : (صفحه ۴۲ از UM10120)

تغذیه تمامی واحد های جانبی (I2C ، ADC، SPI، RTC و ...) توسط این رجیستر کنترل می شود. به طوری که در حالت نرمال تغذیه تمامی واحدهای جانبی وصل بوده و تمامی واحدهای جانبی آماده به کار هستند. اما فعال بودن تغذیه هر واحد هر چند که در یک پروژه از آن استفاده نشده باشد و آن واحد بیکار باشد ، باعث می شود جریانی کشیده شود و توان مصرفی بالا می رود. بنابراین در یک پروژه حرفه ای باید واحدهای بیکار شناسایی شود و تغذیه آن واحد توسط رجیستر PCONP به کلی قطع گردد.

نحوه عملکرد واحد PLL و کنترل کلاک سیستم

همانطور که در بخش معماری میکروکنترلرهای ARM7 به آن اشاره کردیم ، یک باس پرسرعت AHB و یک باس کم سرعت APB و یک واحد PLL وجود دارد. اما چگونه این واحدها عمل می کنند. شکل زیر این مسئله را نشان می دهد.



مثال : برای مثال زمانی که از کریستال ۱۲ مگاهرتز استفاده کنیم و واحد PLL غیر فعال باشد (N=12 و M=1) در نتیجه کلاک اصلی یا Cclk برابر همان ۱۲ مگاهرتز می شود. تقسیم کننده باس APB یا APB Divider نیز در حالت پیش فرض روی ۴ است. بنابراین Cclk برابر ۱۲ مگاهرتز و Pclk برابر ۳ مگاهرتز می شود.

نکته ۱ : کلاک اصلی در کلیه میکروکنترلرهای ARM7 حداکثر می تواند ۷۲ مگاهرتز باشد که البته برای میکروکنترلر LPC21XX نهایتا ۶۰ مگاهرتز است.

نکته ۲ : برای هر چه بهتر اجرا شدن برنامه و هنگ نکردن سیستم بهتر است کلاک CPU را تا جای ممکن افزایش داد.

نکته ۳ : کلاک تمامی واحدهای جانبی مانند UART ، SPI، ADC به باس کم سرعت APB متصل هستند.

نکته ۴ : در حالت پیش فرض PLL غیر فعال بوده و تقسیم کننده APB روی ۴ است. (M=1 و L=4)

تنظیم تقسیم کننده APB با استفاده از رجیستر APBDIV

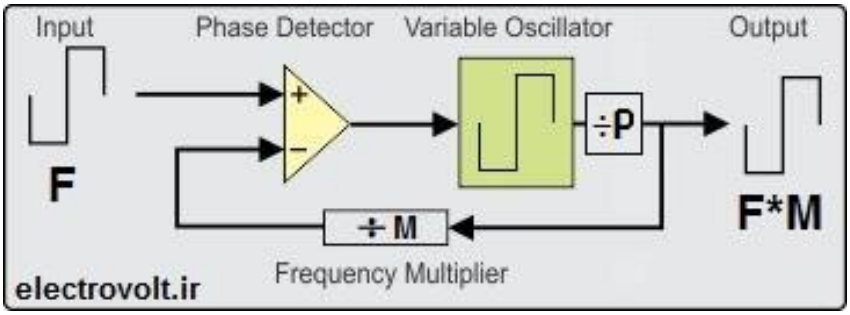
کلاک پر سرعت یا AHB ، توسط یک تقسیم کننده کلاک که به آن پل AHB به APB نیز گویند (AHB To APB Bridge) به کلاک کم سرعت تبدیل می شود. کلاک پرسرعت به واحدهای پر سرعت نظیر CPU ، Memoryمتصل است و کلاک کم سرعت به واحدهای جانبی سرعت پایین نظیر ADC ، I2c، SPIمتصل می باشد. این تقسیم کننده فقط میتواند به یکی از صورت های (۱÷ ، ۲÷ و ۴ ÷) عمل کند. دو بیت اول رجیستر APBDIV، به صورت جدول زیر این ضریب تقسیم را مشخص می کند.

VPBDIV=0x00;	Pclk=Cclk/4
VPBDIV=0x01;	Pclk=Cclk
VPBDIV=0x02;	Pclk=Cclk/2
VPBDIV=0x03;	RESERVED

توجه : نوشتن و خواندن بر روی بیت رزرو شده امکان پذیر نمی باشد.

راه اندازی واحد PLL

تنها زمانی میتوانیم از واحد PLL استفاده کنیم که از کریستال بین ۱۰ تا ۲۵ مگاهرتز استفاده کرده باشیم. توسط این واحد میتوان فرکانس Cclk را نهایتا تا ۶۰ مگاهرتز افزایش داد. درون این واحد یک آشکار ساز فاز (Phase Detector) ، یک اسیلاتور کنترل شونده با جریان (Current Controled Oscillator) و یک تقسیم کننده وجود دارد. شکل زیر مدار داخلی واحد PLL را نشان می دهد.



اسیلاتور کنترل شونده با جریان (CCO) فقط می تواند در محدوده فرکانسی بین ۱۵۶ تا ۳۲۰ مگاهرتز کار کند. بنابراین یک تقسیم کننده دیگر بعد از بلوک Variable Oscillator در شکل فوق وجود دارد که این فرکانس را در محدوده مجاز بین ۱۰ تا ۶۰ مگاهرتز نگه می دارد. در نتیجه برای فعالسازی این واحد ابتدا مقادیری را که نیاز داریم به صورت زیر تعریف می کنیم:

- **Lock Time** : مدت زمانی است که طول می کشد تا اسیلاتور PLL روی فرکانس مورد نظر قفل شود. فرکانس خروجی واحد PLL کم کم بالا می آید تا به فرکانس مورد نظر برسد .بعد از گذشت این زمان ، مجاز به استفاده از واحد PLL هستیم.

- **ضریب PLL (عدد M یا MSEL) :** عدد M ، عدد صحیحی است که مقدار ضرب کنندگی واحد PLL را مشخص می کند. این عدد قابلیت این را دارد که بین ۱ تا ۳۲ باشد اما در عمل در میکروکنترلرهای سری LPC21XX (چون نهایتا ۶۰ مگاهرتز بیشتر نمی تواند باشد) این عدد صحیح بین ۱ تا ۶ می باشد.

- **ضریب تقسیم (عدد P یا PSEL) :** این عدد ، ضریب تقسیم خروجی CCO می باشد. از آن جایی که CCO در محدوده ۱۵۶ تا ۳۲۰ مگاهرتز کار می کند ولی خروجی بین ۱۰ تا ۶۰ مگاهرتز است ، نیاز به این تقسیم کننده وجود دارد که فرکانس خروجی را در محدوده مجاز نگه دارد.

نکته ۱ : برای استفاده از واحد PLL ، باید ابتدا تنظیمات مربوطه انجام شود ، سپس واحد PLL فعال شود و در نهایت مدت زمانی طول می کشد تا فرکانس خروجی ، روی فرکانس مورد نظر قفل شود و واحد PLL به میکرو اعمال شود.

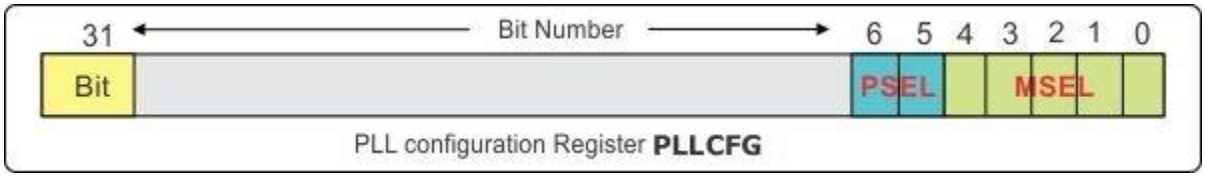
نکته ۲ : به ازای هر مگاهرتز افزایش کلاک در واحد PLL ، تقریبا به اندازه ۰,۵ میلی آمپر افزایش جریان مصرفی خواهیم داشت.

مراحل راه اندازی واحد PLL :

۱. تنظیم MSEL و PSEL در رجیستر PLLCFG
۲. اعمال تنظیمات با استفاده از رجیستر PLLFEED
۳. روشن کردن تغذیه واحد PLL با استفاده از رجیستر PLLCON
۴. اعمال تنظیمات با استفاده از رجیستر PLLFEED
۵. انتظار برای راه افتادن واحد PLL و قفل شدن روی فرکانس مورد نظر (PLLSTATS)
۶. اتصال واحد PLL در مدار به عنوان اسیلاتور با استفاده از رجیستر PLLCON
۷. اعمال تنظیمات با استفاده از رجیستر PLLFEED

معرفی رجیستر PLLCFG :

در این رجیستر تنظیمات MSEL و PSEL صورت می گیرد. نحوه قرار گرفتن بیت های مربوط را در شکل زیر مشاهده می کنید.



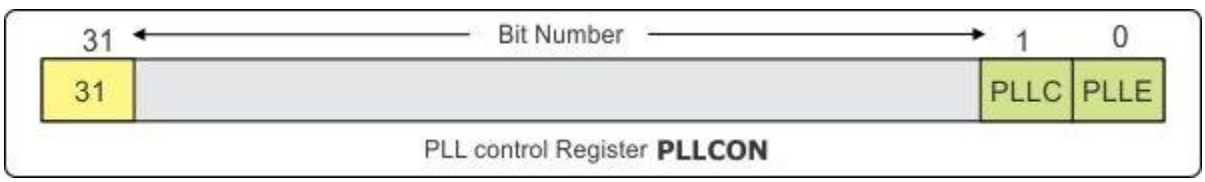
تنظیمات رجیستر PLLCFG :

در حالتی که از کریستال ۱۲ مگاهرتز استفاده نماییم ، تنظیمات PSEL و MSEL برای رسیدن به فرکانس Cclk مورد نظر به صورت جدول زیر می باشد.

فرکانس کریستال اسیلاتور F_{osc} = 12MHz Electrovolt.ir						
CCLK	M	P	PSEL	MSEL	PLLCFG(bin)	PLLCFG(hex)
12	1	8	11	00000	110 0000	60
24	2	4	10	00001	100 0001	41
36	3	4	10	00010	100 0010	42
48	4	2	01	00011	010 0011	23
60	5	2	01	00100	010 0100	24

معرفی رجیستر PLLCON :

در این رجیستر دو بیت PLLE و PLLC به صورت شکل زیر وجود دارد. بیت PLLE برای PLL Enableیعنی روشن/خاموش کردن تغذیه واحد PLL می باشد. بیت PLLC برای PLL Connectیعنی اتصال/عدم اتصال واحد PLL به مدار به عنوان اسیلاتور می باشد.



تابع راه اندازی واحد PLL :

تابعی که برای راه اندازی واحد PLL در میکروکنترلر LPC2138 تعریف می کنیم ، دارای دو ورودی M و P می باشد. کافی است طبق جدول معرفی شده اعداد M و P مناسب را در هنگام فراخوانی به این تابع بدهیم و بعد از آن واحد PLL راه اندازی می شود.

```
void pll_init(unsigned char msel, unsigned char psel){

 PLLCFG|=msel;
 PLLCFG|=(psel<<5);

 PLLFEED=0xAA;
 PLLFEED=0x55;

 PLLCON|=1;

 PLLFEED=0xAA;
 PLLFEED=0x55;

 while((PLLSTAT & (1<<10)) == 0);

 PLLCON|=(1<<1);

 PLLFEED=0xAA;
 PLLFEED=0x55;
}
```

توضیح : به علت اینکه ۴ بیت اول رجیستر PLLCFG عدد MSEL را مشخص می کند ، این عدد را مستقیما از ورودی تابع گرفته و درون رجیستر قرار دادیم. همچنین برای تنظیم PSEL در رجیستر PLLCFG ، باید ۵ شیفت به آن اعمال

شود. دو خط بعدی که عینا سه بار در تابع آمده است مربوط به اعمال تنظیمات با استفاده از رجیستر PLLFEED می باشد. تنظیمات بقیه رجیستر ها درون تابع نیز با توجه به UM10120 بدست آمده است.

نکته مهم : نحوه استفاده از تابع اهمیت زیادی دارد و تنظیم رجیسترهای صورت گرفته در درون تابع از اهمیت کمتری برخوردار هستند. چرا که ما تابع را به همین صورت کپی کرده و از آن استفاده می کنیم.

مثال : برای مثال می خواهیم کلاک CPU ، حداکثر کلاک ممکن یعنی ۶۰ مگاهرتز باشد .برای تنظیم واحد PLL طبق جدول باید (M=5 و P=2) باشد. در نتیجه داریم:

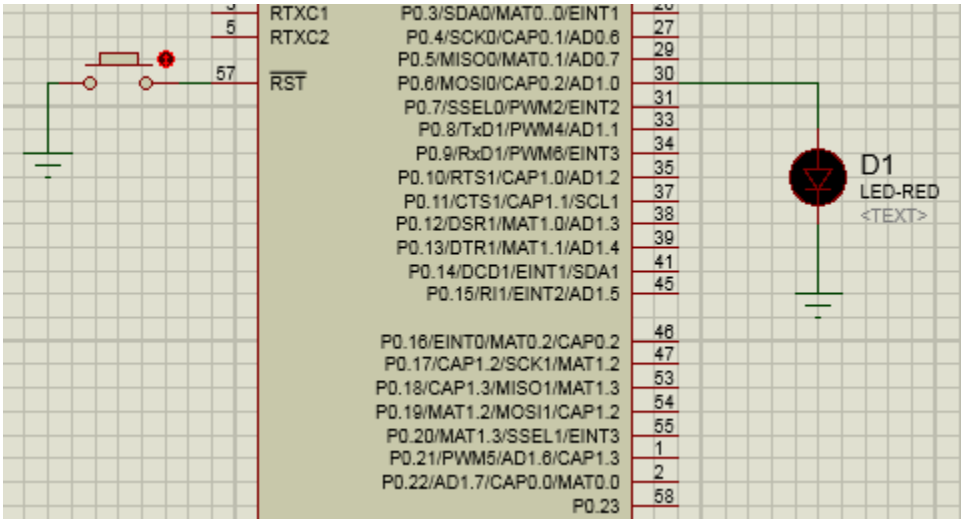
```
pll_init(4,1);
```

توضیحات تکمیلی : با مراجعه به جدول مشاهده می شود که در آرگومان اول ورودی تابع ، همیشه M-1 قرار می گیرد و آرگومان دوم تابع نیز عدد ۳ یا ۲ یا ۱ بسته به عدد M و فرکانس مورد نیاز تعیین می گردد.

مثال عملی شماره ۲ : برنامه LED چشمک زن را با استفاده از واحد PLL و در فرکانس 60Mhz مجددا بنویسید. سپس در نرم افزار پروتئوس شبیه سازی کرده و تفاوت آن با حالت بدون PLL را بررسی نمایید.

حل :

مرحله اول : طراحی سخت افزار



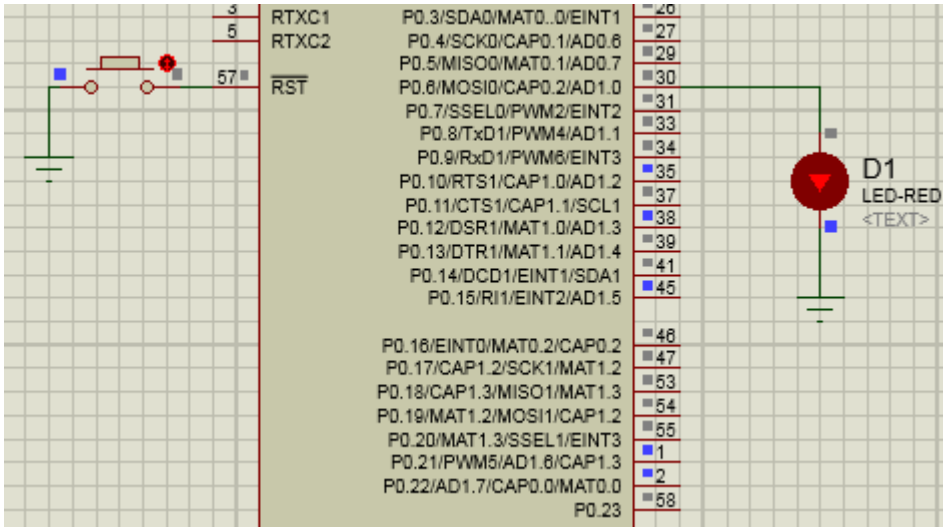
مرحله دوم : طراحی نرم افزار

در این مرحله با اضافه کردن تابع pll_init به قبل از تابع Main و فراخوانی تابع در ابتدای تابع Main برنامه را به صورت زیر تکمیل می کنیم.

```
#include <lpc213x.h>


void delay (void){
        unsigned int i = 1000000;
        while(i--);
}


void pll_init(unsigned char msel, unsigned char psel){
PLLCFG|=msel;
PLLCFG|=(psel<<5);
PLLFEED=0xAA;
PLLFEED=0x55;
PLLCON|=1;
PLLFEED=0xAA;
PLLFEED=0x55;
while((PLLSTAT & (1<<10)) == 0);
PLLCON|=(1<<1);
PLLFEED=0xAA;
PLLFEED=0x55;
}


int main (void){
        pll_init(5,2);
        IO0DIR |= (1<<6);
```

```
        while(1){
                IO0SET |= (1<<6);
                delay();
                IO0CLR |= (1<<6);
                delay();
        }
}
```

مرحله سوم : شبیه سازی



نتیجه : سرعت چشمک زدن LED در حالت استفاده از واحد PLL روی ۶۰ مگاهرتز نسبت به حالتی که واحد PLL وجود نداشت (۱۲ مگاهرتز) ، ۵ برابر افزایش می یابد.

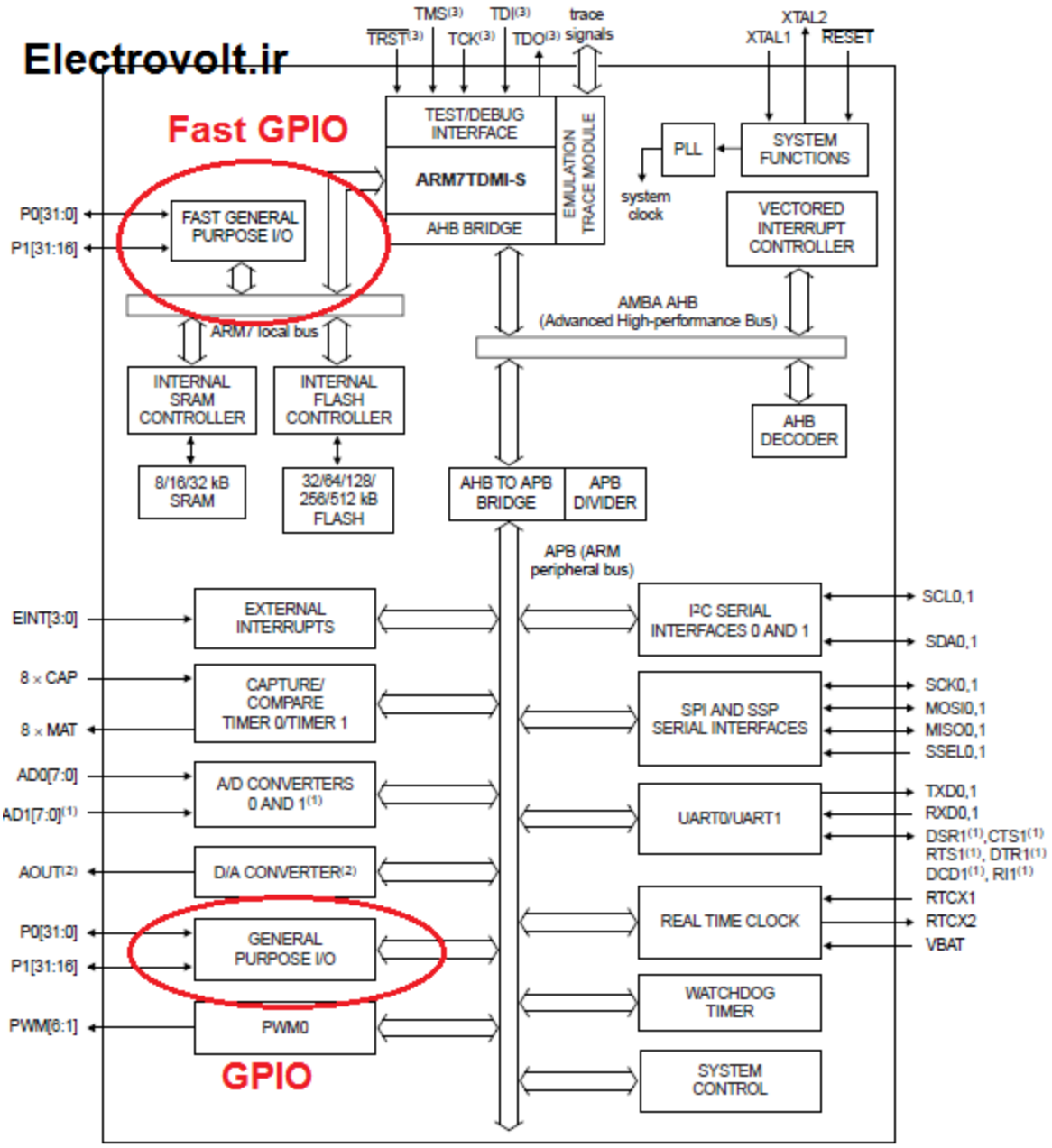
فصل ۸ – آموزش کار با پورت ها و راه اندازی وسایل جانبی

مقدمه

مهمترین تفاوت میان برنامه نویسی ARM7 و AVR یکی عدم وجود ابزاری همانند کدویزارد است که بخش مهم و رجیستری کدها را برای ما تولید کند و دیگری عدم وجود هدر فایل هایی نظیر delay.h ، lcd.h و ... در کامپایلر KEIL است. در این فصل و فصل های بعدی توابع و هدر فایل هایی خواهیم ساخت که با استفاده از آن ها میتوان همانند برنامه نویسی که در AVR وجود داشت ، برای ARM7 نیز مشابه آنها استفاده کرد.

واحد GPIO سریع

این واحد یکی از ویژگی های بهبود یافته در پورت های میکروکنترلرهای ARM7 شرکت NXP می باشد. همان پورت های در دسترسی که به صورت GPIO استفاده می شوند، میتوانند به صورت سریعتر عمل کنند. یعنی سرعت تغییر آنها از ۰ به ۱ و بالعکس در این واحد سریعتر می باشد. رجیسترهای تنظیمات این واحد یک F در ابتدای نام رجیسترهای GPIO تفاوت دارد. در حقیقت Fast GPIO و GPIO در عمل دو واحد مجزا هستند که هر دو از پورت های یکسان استفاده می کنند. شکل زیر تفاوت میان واحد ورودی/خروجی معمولی را با سریع نشان می دهد. واحد Fast GPIO به باس پرسرعت وصل است و نهایت سرعتی که دارد ، سرعت AHB است اما واحد GPIO به APB متصل است و بسته به ضریب تقسیم کننده APB معمولا سرعت کمتری دارد.



نتیجه گیری : برای استفاده از واحد Fast GPIO در پروژه هایی که به سرعت تغییر بالای پورت ها نیاز داریم ، به جای رجیسترهایی که داشتیم از FIODIR ، FIOPIN، FIOCLR و FIOSET استفاده می کنیم.

نکته مهم : در صورتی که بیشتر از یک IO در میکروکنترلر داشته باشیم ، که برای میکروکنترلر LPC2138 دو واحد داریم ، بعد از FIO عدد 0 یا 1 گذاشته می شود .در نتیجه رجیسترهای فوق به صورت FIOXCLR ، FIOXPIN، FIOXDIR و FIOXSET در می آید که به جای X در میکروکنترلرLPC2138 ، عدد 0 یا 1 بسته به پورت انتخابی قرار می گیرد.

تعریف توابع تاخیر نسبی درARM7

همانطور که در بخش قبل گفتیم و دیدید ، در کامپایلر KEIL هدر فایلی برای delay وجود ندارد. اما می خواهیم خودمان با استفاده از دانش کلاک سیستم ، به طور تقریبی این delay ها را ایجاد کنیم. با اضافه کردن چند خط زیر به ابتدای برنامه میتوان از توابع delay_s ، delay_ms و delay_us در پروژه ها استفاده نمود.

نکته : این توابع از دقت کاملا نسبی برخوردار هستند و از آنها به حسب نیاز در راه اندازی کلید ، صفحه کلید و ... میتوان استفاده نمود و برای داشتن تاخیر دقیق باید از واحد Timer میکرو استفاده کرد.

```
#define CCLK 12000000


void delay_s (unsigned int x)
{
x=x*CCLK;
while(x--);
}
void delay_ms(unsigned int x)
{
x=x*(CCLK/1000);
while(x--);
}
void delay_us(unsigned int x)
{
x=x*(CCLK/1000000);
while(x--);
}
```

توضیح : در ابتدا مقدار کلاک کاری CPU را تعریف می کنیم. (اگر از PLL استفاده کردیم باید مقدار دقیق CCLK را مشخص کنیم) سپس سه تابع تعریف کردیم و آنها را delay_s ، delay_ms و delay_us تعریف کردیم. با فرض اینکه هر یک واحد کم شدن متغیر x ، یک سیکل کلاک طول می کشد ، میتوان گفت هر ثانیه تقریبا 12000000کلاک طول می کشد.

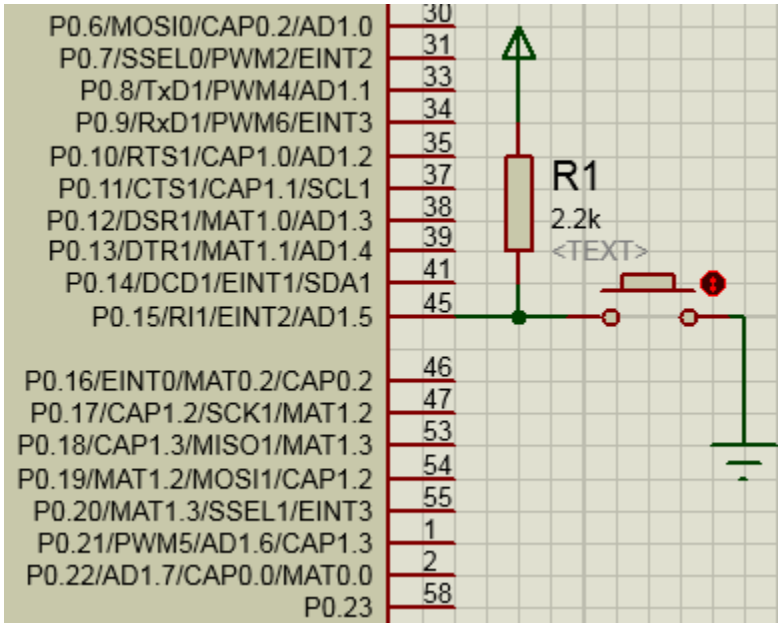
نکته : در عمل نمیتوان دقیقا مشخص کرد که حلقه while برای اجرا چند سیکل کلاک طول می کشد اما معمولا بین ۱ تا ۴ سیکل برای آن میتوان در نظر گرفت. در نتیجه تخمین توابع تعریف شده فوق ، در خوشبینانه ترین حالت هستند و در پروژه های مختلف بسته به نیاز میتوانید خودتان آن ها را کم یا زیاد کنید.

راه اندازی کلید در LPC2138

راه اندازی کلید به عنوان ورودی در میکروکنترلرهای ARM کاملا شبیه میکروکنترلرهای AVR است با این تفاوت که به جای استفاده از رجیستر PINX از رجیستر IOXPIN استفاده می شود و دسترسی بیتی به رجیسترها در کامپایلر KEIL وجود ندارد.

نکته : دسترسی به بیت های رجیسترهای پورت در کامپایلر KEIL همانند میکروکنترلرهای AVR به صورت نقطه ای ممکن نیست (مثلا نمیتوان نوشت IO0PIN.0 غلط است)

راهکار خواندن از رجیستر PIN :



فرض کنید که کلیدی به صورت **پول آپ** شده به یکی از پورت های میکرو متصل است. برای اینکه کلید زده شده است یا خیر باید منطق پورت را خواند. برای خواندن منطق پورت از رجیستر IOPIN استفاده می شود. چون دسترسی به یک بیت از این رجیستر وجود ندارد برای فهمیدن منطق یک بیت از این رجیستر از عملگر & (AND بیتی) استفاده می کنیم. عدد ۱ را به اندازه بیت مورد نظر شیفت داده و آن را با رجیستر IOPIN به صورت بیتی AND می کنیم.

برای مثال : برای خواندن منطق کلیدی که به صورت پول آپ به پورت P0.15 وصل شده است ، میتوان نوشت:

```
if( ( IO0PIN & (1<<15) ) == 0 ) );
```

نکته ۱ : بهترین مقدار برای مقاومت پول آپ در میکروکنترلرهای ARM به علت اینکه تغذیه ۳,۳ ولت دارند ، 2.2K یا 3.3k است.

نکته ۲ : پول آپ داخلی در میکروکنترلرهای ARM7 شرکت NXP وجود ندارد.

کلید نوع ۱ :

```
if( ( IO0PIN & (1<<15) ) == 0 ) )
{
دستورات مربوط به بعد از زدن کلید
delay_ms(200);
}
```

توضیح : به محض فشار دادن کلید توسط کاربر شرط if برقرار شده و دستورات مورد نظر اجرا می شود سپس به علت ایجاد تاخیر زیاد توسط تابع delay_ms (در اینجا ۲۰۰ میلی ثانیه) با این کار احتمال اینکه زمانی که برنامه در حلقه whileبه if می رسد و شرط برقرار باشد ، کاهش می یابد . مزیت این کلید این است که در صورتی که کاربر کلید را فشار داده و نگه دارد تقریبا در هر ۲۰۰ میلی ثانیه یکبار کار مورد نظر صورت می گیرد . عیب این روش نیز این است که هنوز احتمال دارد که زمانی که یکبار کلید زده شود ، دوبار کار مورد نظر انجام شود.

کلید نوع ۲ :

```
if( ( IO0PIN & (1<<15) ) == 0 ) )
{
delay_ms(20);
while(( IO0PIN & (1<<15)) == 0 ));
دستورات مربوط به بعد از زدن کلید
}
```

توضیح : به محض فشار دادن کلید توسط کاربر شرط if برقرار شده و برنامه به مدت ۲۰ میلی ثانیه صبر می کند تا منطق کلید ثابت شود و از منطقه bounce عبور کند سپس توسط حلقه while با همان شرط برقراری کلید در این

مرحله برنامه تا زمانی که کلید توسط کاربر فشرده شده است در حلقه گیر می کند و هیچ کاری انجام نمی دهد . به محض اینکه کاربر دست خود را بر می دارد ، شرط برقرار نبوده و خط بعدی یعنی دستورات مربوطه اجرا می شود . مزیت این روش این است که در هر بار فشردن کلید برنامه تنها یکبار اجرا می شود . معایب این روش این است که تا زمانی که کاربر کلید را نگه داشته اتفاقی نمی افتد و به محض رها کردن کلید کار مورد نظر انجام می شود.

کلید نوع ۳ :

```
if((( IO0PIN & (1<<15)) == 0 ) && (flag==0)) {
flag=1; start=!start; }
else if (( IO0PIN & (1<<15) ) == 0 ) flag=0;
if(start){
دستورات مربوط به بعد از زدن کلید
}
```

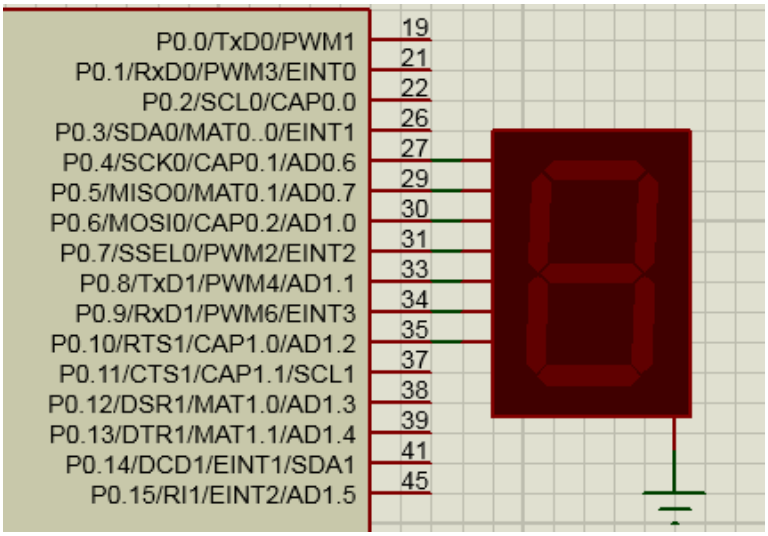
توضیح : این کلید به صورت start/stop عمل می کند یعنی بار اولی که کاربر کلید را فشار می دهد دستورات مربوط به بعد از زدن کلید دائما اجرا می شود تا زمانی که کاربر دست خود را از روی کلید رها کرده و دوباره کلید را فشار دهد ، در این صورت دستورات دیگر اجرا نمی شود. دو متغیر از نوع bit با نام های flag و start با مقدار اولیه ۰ برای این کلید باید تعریف شود . زمانی که کاربر برای اولین بار کلید را فشار می دهد شرط if برقرار شده و flag=1 و start=1 می شود. در این صورت شرط if دوم برقرار بوده و دستورات مربوطه با هر بار چرخش برنامه درون حلقه نامتناهی while یکبار اجرا می شود . زمانی که کاربر دست خود را از روی کلید بر می دارد و منطق ۱ وارد میکرو می شود flag=0 شده و برنامه دوباره آماده این می شود که کاربر برای بار دوم کلید را فشار دهد . زمانی که کاربر بار دوم کلید را می فشارد start=0 شده و دستورات مربوطه اجرا نخواهد شد سپس با برداشته شدن دست کاربر از روی کلید، همه چیز به حالت اول بر میگردد . این کلید طوری نوشته شده است که bounce در آن کمترین تاثیر مخرب ممکن را دارد.

مثال عملی شماره ۳ : با استفاده از یک کلید و ۸ عدد LED متصل به میکروکنترلر LPC2138 برنامه ای بنویسید که با هر بار فشار دادن کلید ، شمارنده های حلقوی و جانسون اجرا شود.

لینک دانلود سورس مثال عملی شماره ۳

راه اندازی سون سگمنت

یکی از نمایشگرهای پرکاربرد سون سگمنت است که می توان توسط آن اعداد و برخی از حروف ها را نشان داد. روش های متفاوت و مختلفی برای راه اندازی سون سگمنت وجود دارد که ساده ترین آن استفاده از مداری به شکل زیر است.



تذکر : در نرم افزار proteus بیت هشتم سون سگمنت که پایه digit می باشد، وجود ندارد.

برای نشان دادن اعداد روی سون سگمنت کافی است پایه مربوطه را پس از خروجی کردن یک کنیم برای مثال برای نشان دادن عدد یک باید سگمنت های b و c روشن شود ، در نتیجه باید P0.4 و P0.5 را 1 نمود. بنابراین دستور زیر عدد یک را روی سون سگمنت نمایش می دهد.

IO0SET |= (3<<4);

برای بقیه اعداد نیز این کار را تکرار کرده و دستور مورد نظر برای آن را می یابیم. سپس تمامی این مقادیر را درون یک آرایه ریخته تا بتوان راحت تر از آن استفاده نمود.

برای بدست آوردن راحت تر این مقادیر میتوان ابتدا فرض نمود که سون سگمنت به پایه های P0.0 تا P0.6 میکرو متصل شده باشد. سپس با این فرض مقادیر را بدست آورده و در نهایت به مقداری که مورد نیاز است شیفت اعمال شود.

با فرض فوق آرایه زیر برای سون سگمنت کاتد مشترک بدست می آید :

unsigned char seg[10]={0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f};

هر عددی که بخواهیم روی سون سگمنت نمایش دهیم کافی است داخل [ ] قرار دهیم و به اندازه مورد نیاز شیفت دهیم تا به کد سون سگمنت تبدیل شود. یعنی :

IO0SET |= (seg[i]<<4);

همانطور که مشاهده می کنید در این روش علاوه بر تعریف یک آرایه و اشغال شدن شش پایه از میکرو ، مشکل دیگری نیز وجود دارد و آن روشنایی کم سگمنت ها در هنگام پیاده سازی مدار است. این موضوع به علت جریان دهی کم پایه های

میکرو می باشد. چرا که برای روشنایی مناسب هر سگمنت به ۲۰ میلی آمپر جریان نیاز دارد در حالی که میکرو توانایی تامین ۱۰ میلی آمپر جریان را دارد.

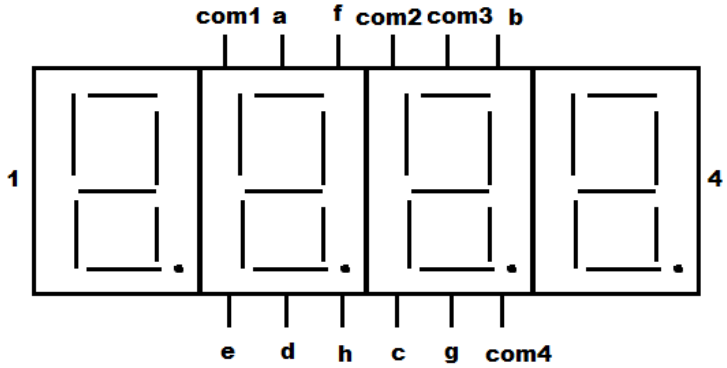
مثال عملی شماره ۴ : با استفاده از یک سون سگمنت متصل به میکروکنترلر LPC2138 برنامه ای بنویسید که اعداد ۰ تا ۹ را به ترتیب روی آن نشان دهد.

لینک دانلود سورس مثال عملی شماره ۴

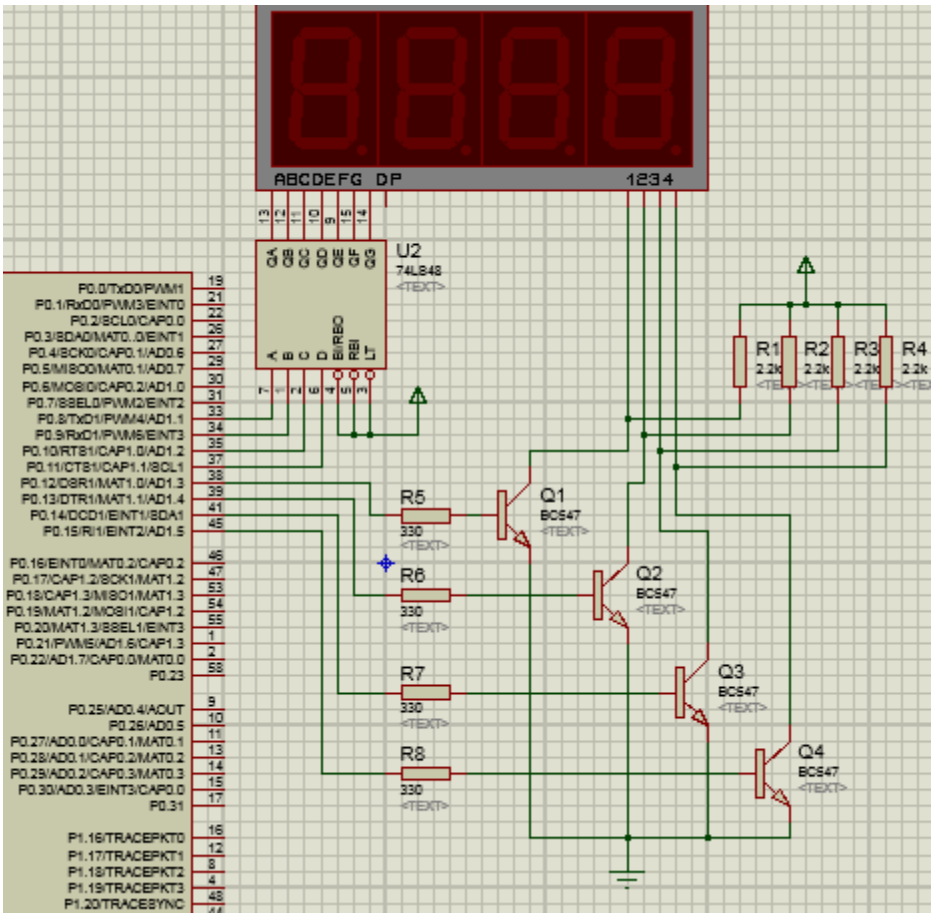
راه اندازی سون سگمنت مالتی پلکس

زمانی که تعداد بیشتری سون سگمنت در پروژه مورد نیاز باشد ، بهتر است از سون سگمنت های مالتی پلکس شده استفاده نمود. تنها تفاوت این نوع سون سگمنت در این است که ۸ بیت دیتا (a ، b، c و...) برای همه سگمنت ها با هم یکی شده است (مشترک هستند). در شکل زیر یک سون سگمنت مالتی پلکس چهار تایی را مشاهده می کنید.

پایه های سون سگمنت مالتی پلکس شده ۴ تایی را در شکل زیر مشاهده می کنید. در صورت اتصال پایه های Com سون سگمنت مربوطه روشن می شود.



بنابراین مدار مورد نظر در این طراحی به صورت زیر است. وجود ترانزیستور npn در این طراحی به علت جریان بالای روشن شدن سون سگمنت است. زیرا در هر سون سگمنت روشن جریانی در حدود ۱۰۰ میلی آمپر به زمین وارد می شود در حالی که هر پایه میکرو توانایی عبور ۱۰ میلی آمپر جریان را دارد. همچنین در این طراحی از آی سی ۷۴۴۸ برای ساخت کدهای سون سگمنت استفاده شده است. با استفاده از این آی سی علاوه بر این که نیازی به تعریف آرایه کدهای سون سگمنت نداریم ، به علت تغذیه ۵ ولت آی سی ، سون سگمنت ها نور مطلوبی خواهند داشت.



مثال عملی شماره ۵ : با استفاده از سون سگمنت مالتی پلکس شده چهارتایی و میکروکنترلر LPC2138 برنامه یک شمارنده از 0000 تا 9999 را بنویسید. (بدون ۷۴۴۸)

مثال عملی شماره ۶ : به مثال قبلی یک آی سی ۷۴۴۸ و یک کلید اضافه نمایید سپس برنامه را طوری تغییر دهید که با فشردن کلید شمارش آغاز شود و با فشردن مجدد کلید شمارش متوقف شود.

لینک دانلود سورس مثال عملی شماره ۵ و ۶

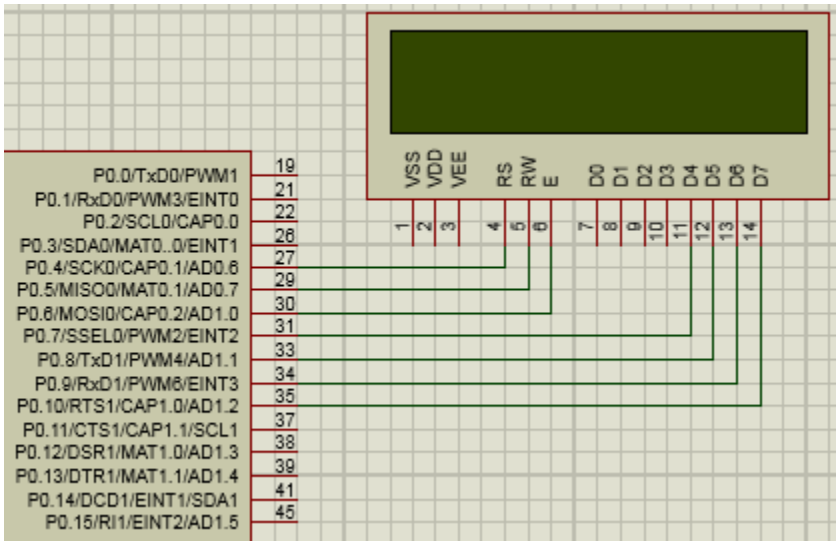
ال سی دی های کارکتری اغلب دارای ۱۴ پایه هستند که ۸ پایه برای انتقال اطلاعات ۳ پایه برای کنترل نرم افزاری یک پایه برای کنترل سخت افزاری و دو پایه تغذیه می باشد در جدول زیر اطلاعات پایه ها را مشاهده می کنید :

پایه های LCD

پایه	نام	عملکرد
1	VSS	زمین
2	VCC	5V+
3	VEE	کنترل درخشندگی (می توانید با یک مقاومت ۱ کیلو آن را زمین کنید)
4	RS	اگر این پایه ۰ باشد اطلاعات روی DB0-DB7 به عنوان فرمان و اگر ۱ باشد به عنوان کاراکتر پذیرفته می شود
5	R/W	اگر این پایه ۰ باشد LCD برای نوشتن آماده می شود و اگر ۱ باشد برای خواندن آماده می شود
6	E	فعال سازی LCD که با یک لبه پایین رونده می باشد
7	DB0	دیتای 0
8	DB1	دیتای 1
9	DB2	دیتای 2
10	DB3	دیتای 3
11	DB4	دیتای 4
12	DB5	دیتای 5
13	DB6	دیتای 6
14	DB7	دیتای 7
15	A	5V+ از پایه ۱۵ و ۱۶ برای روشن کردن LED پس زمینه استفاده می شود
16	K	زمین

LCD ها را میتوان با ۸ خط دیتا یا ۴ خط دیتا راه اندازی کرد اما معمولا آن را با ۴ خط دیتا راه اندازی می کنند که در این نوع راه اندازی تنها پایه های RS، R/W، E و D4 تا D7 به پورت دلخواه از میکرو متصل می شود. تنها تفاوت راه اندازی با ۴ خط دیتا در این است که داده های ۸ بیتی LCD به جای یکبار، در دو مرحله ۴ بیتی ارسال می شوند. مزیت

راه اندازی با ۴ خط دیتا در این است که اتصال LCD به میکروکنترلر تنها ۷ پایه از میکرو را اشغال می کند. این ۷ خط دیتا را میتوان به هر پایه دلخواهی از میکرو متصل نمود اما بهتر است از پورت های کنار هم استفاده نمود. در اینجا ما LCDرا به صورت شکل زیر متصل کردیم.



نحوه استفاده از LCD کاراکتری :

تعریف این توابع کاملا شبیه به توابع در AVR هستند با این تفاوت که کامپایلر KEIL هیچگونه تنظیمات و هدر فایلی در این خصوص ندارد و هدر فایلی که به پروژه اضافه می کنیم توسط خودمان نوشته شده است.

برای اضافه کردن قابلیت استفاده از LCD کاراکتری به برنامه کافی است فایل lcd.h و نیز فایل lcd.c را بعد از دانلود در کنار فایل اصلی پروژه (main.c) در پوشه اصلی پروژه کپی کرد و در ابتدای برنامه به صورت زیر این هدر فایل را به برنامه اضافه کرد.

```
#include "lcd.h"
```

اگر به درون دو فایل موجود یعنی lcd.h و lcd.c نگاهی بیاندازیم ، مشاهده می کنیم که در lcd.h تعاریف پایه های اتصال LCD به میکرو و نیز اعلان توابع وجود دارد. در درون فایل lcd.c نیز بدنه همان توابع اعلان شده در lcd.h وجود دارد.

تذکر : در صورتی که بخواهید LCD را به صورتی به جز شکل نشان داده شده در بالا متصل کنید ، یعنی بخواهید از پایه های دیگر میکرو برای راه اندازی LCD استفاده نمایید ، باید عدد پایه های تعریف شده در هدر فایل lcd.h را از پایه های موجود به پایه های دلخواه تغییر دهید.

توابع کار با LCD کاراکتری در ARM :

1- تابع راه اندازی LCD کاراکتری (بدون ورودی – بدون خروجی) :

lcd_init();

این تابع LCD را راه اندازی می کند و همیشه قبل از حلقه while نوشته می شود.

2- تابع پاک کردن تمام LCD (بدون ورودی – بدون خروجی) :

lcd_clear();

این تابع lcd را پاک کرده و مکان نما را در سطر و ستون صفر قرار می دهد.

3- تابع رفتن به ستون x و سطر y ام (با دو ورودی – بدون خروجی) :

lcd_gotoxy(x , y);

تابع فوق مکان نمای ال سی دی را در سطر x و ستون y قرار می دهد و باید به جای x و y عدد سطر و ستون مورد نظر جا گذاری شود. مثال :

lcd_gotoxy(8,1);

4- تابع چاپ یک کاراکتر (یک ورودی – بدون خروجی) :

lcd_putchar(' کاراکتر ');

مثال :

lcd_putchar('A');

5- تابع چاپ یک رشته یا یک متغیر رشته ای (یک ورودی – بدون خروجی) :

lcd_print(" رشته ");

lcd_print(نام متغیر رشته ای);

مثال :

lcd_print("IRAN1395");

lcd_print(str);

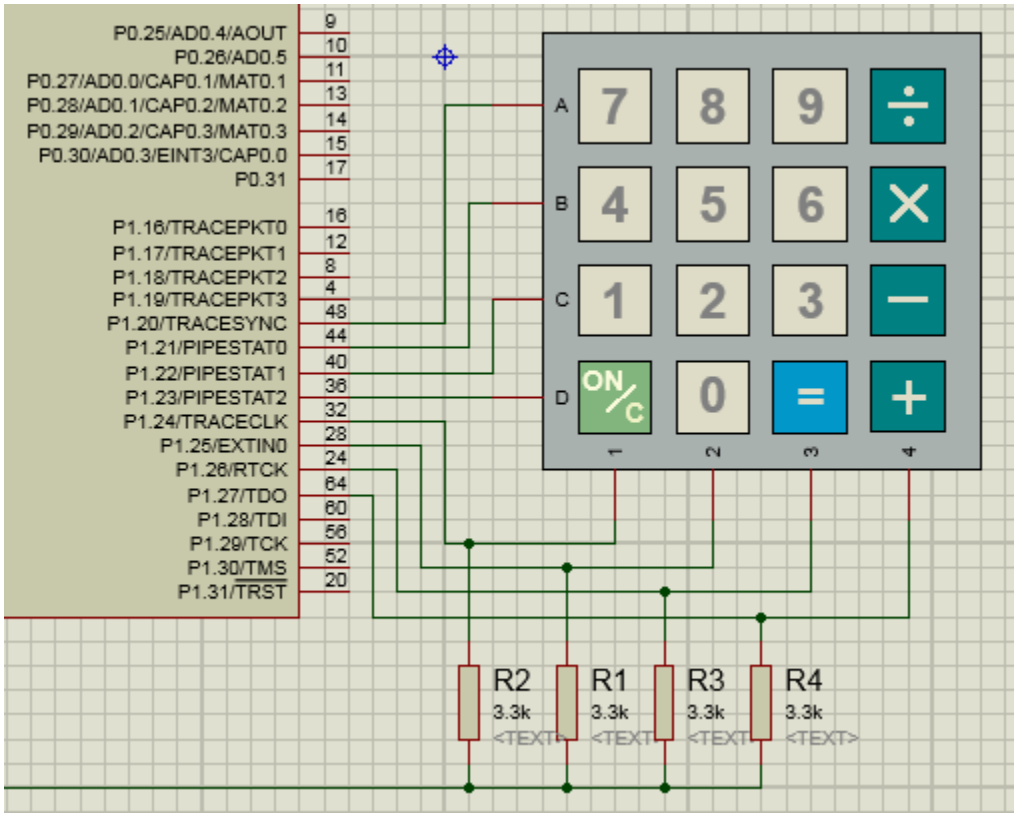
مثال عملی شماره ۷ : یک LCD کاراکتری ۲ در ۱۶ را به صورت ۴ بیتی به میکروکنترلر LPC2138 متصل کرده و برنامه ای بنویسید که یک عدد را روی LCD شمارش کند. با استفاده از راه اندازی واحد PLL سرعت شمارش را افزایش دهید و تغییرات را مشاهده نمایید.

مثال عملی شماره ۸ : برنامه مثال قبل را برای وقتی که LCD به صورت ۸ بیتی به میکروکنترلر متصل می شود بازنویسی کنید.

لینک دانلود سورس مثال عملی شماره ۷ و ۸

راه اندازی صفحه کلید در LPC2138

راه اندازی انواع صفحه کلیدها با استفاده از میکروکنترلرهای ARM7 کاملا شبیه به AVR می باشد با تفاوت اینکه مقاومت پول آپ داخلی وجود ندارد و نیز در برنامه برای مقدار دهی رجیسترهای واحد GPIO ، دسترسی با عملگر Shift و AND میسر است. برای مثال می خواهیم صفحه کلید ۴ در ۴ را به میکرو متصل کنیم. ۴ سطر و ۴ ستون را به هر پایه دلخواه (ترجیحا کنار هم) متصل می کنیم و سپس با استفاده از مقاومت های 3.3K ، ستون های صفحه کلید را پول آپ میکنیم. همانطور که در شکل زیر مشاهده می کنید ، در اینجا ما صفحه کلید را به پایه های P1.20 تا P1.27 متصل کردیم.



تابع خواندن از صفحه کلید:

```
char keyboard (void)
{
 char code[16]={'7','8','9','/','4','5','6','*','1','2','3','-','c','0','=','+'};
 int i=0,j=0;
 for(i=0;i<4;i++)
 {
  IO1CLR|=(1<<(20+i));
  delay_ms(2);
    if((IO1PIN & 0x00f00000)!=0x00f00000)
    {
     for(j=0;j<4;j++)
      if((IO1PIN & (1<<(j+24)))!=(1<<(j+24)))
     return code[i*4+j];
    delay_ms(2);
   }
 IO1SET|=(1<<(20+i));
}
return 20;
}
```

توضیح : یک آرایه به نام code برای اصلاح اعداد صفحه کلید در نظر گرفته شده است .در صورتی که این آرایه نباشد با زدن مثلا کلید ۷ عدد ۰ نمایش داده می شود ! بنابراین به جای اینکه i*4+j به خروجی فرستاده شود ، code[i*j+j] به خروجی فرستاده می شود تا بدین وسیله اعداد اصلاح شود. حلقه for اول وظیفه 0 کردن یکی یکی سطر ها را برعهده دارد. اگر کلیدی توسط کاربر فشار داده شده باشد ، رجیستر در بیت های بین ۲۴ تا ۲۷ از رجیستر IO0PIN تغییر ایجاد می شود. حلقه if اولی تشخیص می دهد که آیا یکی از ستون ها 0 است یا خیر. حلقه for دومی تشخیص می دهد که کدام ستون زده شده است. در آن ستونی که کاربر کلید را فشار داده است ، شرط if دوم برابر شده و داخل آن می شود. در

درون if دوم هم، code[i*j+j] به خروجی فرستاده می شود. تابع keyboard در صورتی که هیچ کلیدی زده نشده باشد مقدار ۲۰ را بر می گرداند.

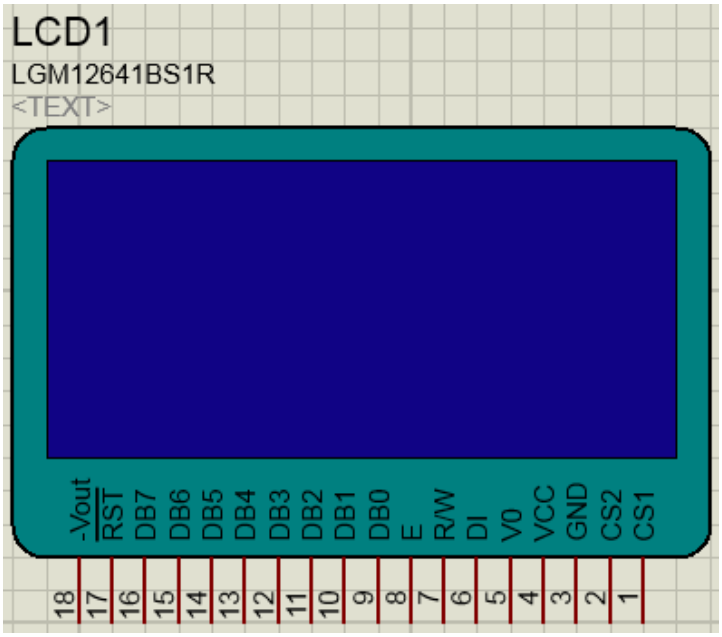
مثال عملی شماره ۹ : با اتصال یک صفحه کلید ۴ در ۴ و یک LCD کاراکتری ۲ در ۱۶ به میکروکنترلر LPC2138 برنامه ای بنویسید که با زدن هر کلید کاراکتر مربوط به آن را روی LCD نمایش دهد.

لینک دانلود سورس مثال عملی شماره ۹

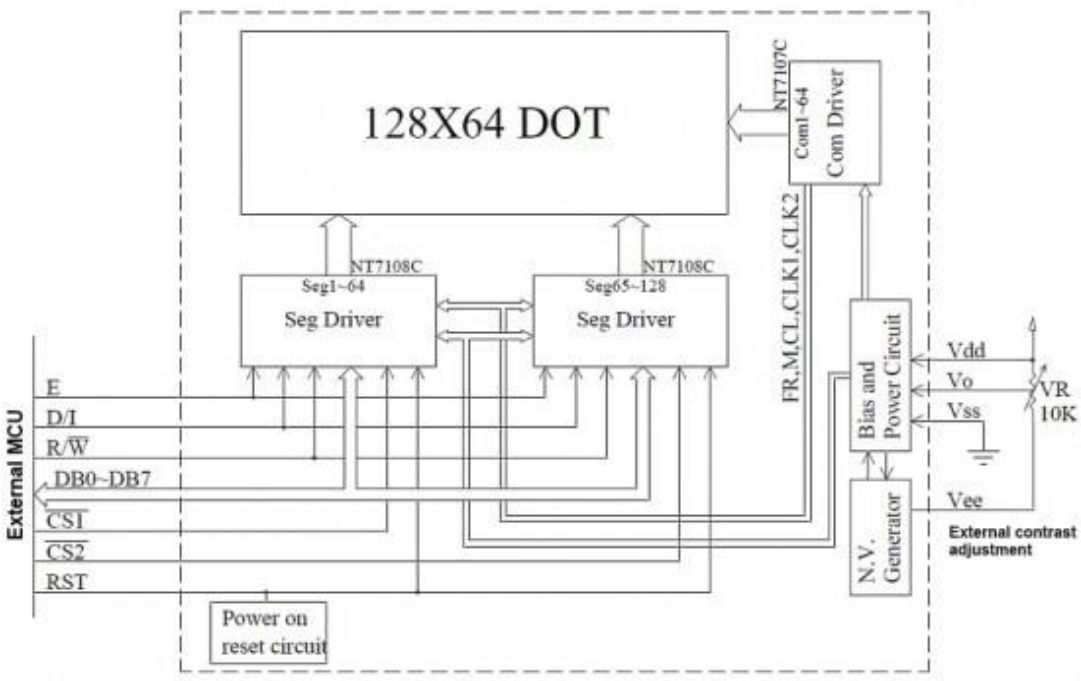
راه اندازی LCD گرافیکی

نوع دیگری از LCD ها وجود دارند که به آنها Graphic LCD یا GLCD گفته می شود. این LCD ها نسبت به LCD های کاراکتری دارای ابعاد بزرگتری هستند و برخلاف LCD های کاراکتری که تنها کاراکترها را نشان می دهند ، GLCD ها با روشن و خاموش کردن نقاط کوچک ، هر نوع تصویر سیاه و سفیدی را میتوانند نمایش دهند. از مزایای دیگر این LCD ها میتوان به تفکیک پذیری بالاتر و امکان ساخت منو و واسط کاربری را اشاره کرد. GLCD ها سایز و انواع متفاوتی دارند که شکل زیر GLCD با کنترلر KS0108 در ابعاد 128*64 را مشاهده می کنید.

در نرم افزار پروتئوس این GLCD را با تایپ کردن LGM126 میتوان به پروژه اضافه نمود.



در شکل زیر بلوک دیاگرام LCD گرافیکی که در این آموزش از آن استفاده شده است را مشاهده می کنید. این GLCD که دارای ۱۲۸*۶۴ پیکسل سیاه و سفید است ، دارای دو عدد کنترلر KS0108B می باشد.



پایه های این GLCD و عملکرد هر پایه به صورت زیر می باشد. (برخی از پایه ها در پروتئوس وجود ندارد)

شماره پین	نام	عملکرد
۱	VSS	زمین
۲	VDD	تغذیه
۳	V0	کنتراست نمایشگر
٤	D/I	Data/Instruction
٦	E	در لبه ی پایین رونده داده Latch شده و در سطح یک امکان خواندن وجود دارد.
۷-۱٤	DB[7:0]	داده
۱۵	CS1	انتخاب چیپ ۱
۱٦	CS2	انتخاب چیپ ۲
۱۷	Reset	LCD خاموش شده و رجیستر خط صفر می شود.
۱۸	VEE	ولتاژ منفی
۱۹	A+	آند LED پشت زمینه
۲۰	A-	کاتد LED پشت زمینه

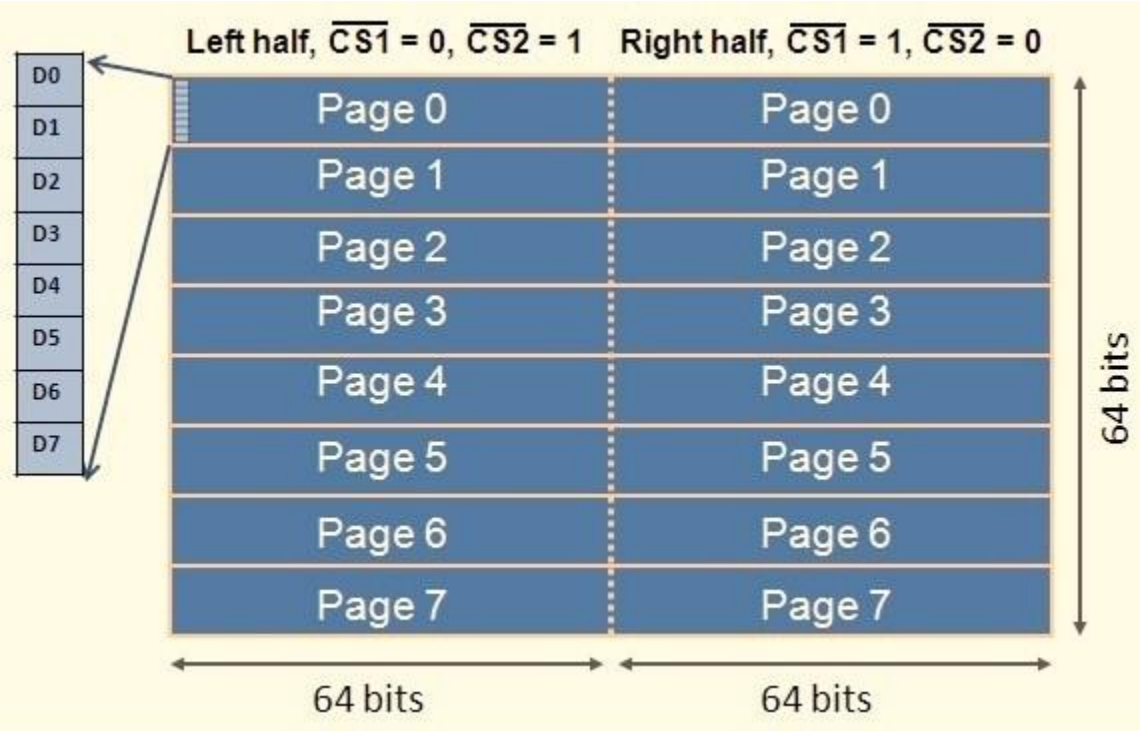
این GLCD دو کنترلر جداگانه دارد ، به طوری که نیمه سمت چپ از LCD با یک کنترلر (در زمانی که CS1=1 و CS2=0) و نیمه دیگر آن با کنترلر دیگر (در زمانی که CS2=1 و CS1=0) راه اندازی می شود.

سه رجیستر X ، Y و Z برای راحتی در این GLCD تعبیه شده است که به صورت زیر تعریف می شود :

رجیستر X : هر ۸ سطر یک صفحه (page) نامیده می شود. بنابراین تعداد 8 صفحه در هر چیپ وجود دارد. رجیستر X مشخص می کند که کدام صفحه فعال و آماده به کار می باشد. آدرس اولین صفحه 0XB8 و آدرس آخرین صفحه 0XBF می باشد.

رجیستر Y : این رجیستر مشخص می کند که کدام ستون از 64 ستون هر چیپ فعال است. آدرس اولین ستون 0X40 و آخرین ستون 0X7F می باشد.

رجیستر Z : این رجیستر مشخص می کند که هر صفحه از کدام ستون شروع شود. در نتیجه با تغییر این رجیستر میتوان به صورت افقی تصویر را حرکت داد. آدرس این رجیستر از 0XC0 تا 0XFF متغیر می باشد.

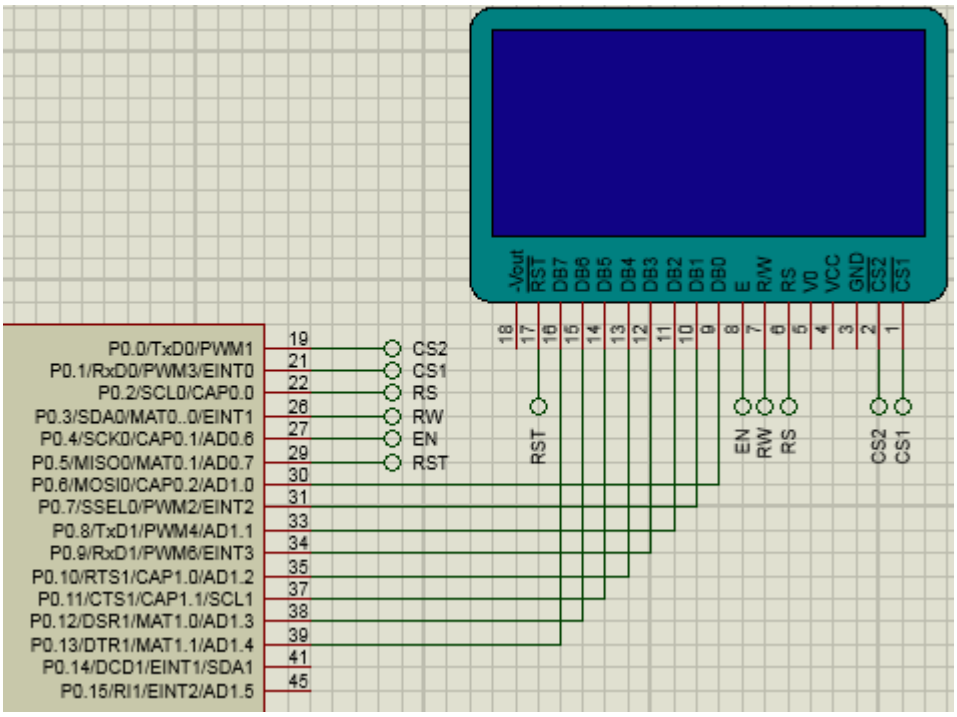


برای راه اندازی و استفاده از رجیسترها دستورات زیر مورد استفاده قرار می گیرند.

Instruction	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Function
Display on/off	L	L	L	L	H	H	H	H	H	L/H	Controls the display on or off. Internal status and display RAM data is not affected. L:OFF, H:ON
Set address (Y address)	L	L	L	H	Y address (0-63)						Sets the Y address in the Y address counter.
Set page (X address)	L	L	H	L	H	H	H	Page (0-7)			Sets the X address at the X address register.
Display Start line (Z address)	L	L	H	H	Display start line (0-63)						Indicates the display data RAM displayed at the top of the screen.
Status read	L	H	Bus y	L	On/ Off	Reset	L	L	L	L	Read status. BUSY L: Ready H: In operation ON/OFF L: Display ON H: Display OFF RESET L: Normal H: Reset
Write display data	H	L	Write data								Writes data (DB0: 7) into display data RAM. After writing instruction, Y address is increased by 1 automatically.
Read display data	H	H	Read data								Reads data (DB0: 7) from display data RAM to the data bus.

راه اندازی LCD گرافیکی :

برای راه اندازی LCD گرافیکی ابتدا مدار را به صورت شکل زیر به میکرو متصل می کنیم.



برای اضافه کردن قابلیت استفاده از LCD گرافیکی به برنامه کافی است فایل glcd.h و نیز فایل glcd.c را در کنار فایل اصلی پروژه (main.c) در پوشه اصلی پروژه کپی کرد و در ابتدای برنامه به صورت زیر این هدر فایل را به برنامه اضافه کرد.

```
#include "glcd.h"
```

اگر به درون دو فایل موجود یعنی glcd.h و glcd.c نگاهی بیاندازیم ، مشاهده می کنیم که در glcd.h تعاریف پایه های اتصال LCD به میکرو و نیز اعلان توابع وجود دارد. در درون فایل glcd.c نیز بدنه همان توابع اعلان شده در glcd.h وجود دارد.

تذکر : در صورتی که بخواهید LCD را به صورتی به جز شکل نشان داده شده در بالا متصل کنید ، یعنی بخواهید از پایه های دیگر میکرو برای راه اندازی LCD استفاده نمایید ، باید عدد پایه های تعریف شده در هدر فایل glcd.h را از پایه های موجود به پایه های دلخواه تغییر دهید.

توابع کار با GLCD در ARM :

1- تابع راه اندازی LCD کاراکتری (بدون ورودی – بدون خروجی) :

glcd_init();

این تابع LCD را راه اندازی می کند و همیشه قبل از حلقه while نوشته می شود.

2- تابع پاک کردن تمام LCD (بدون ورودی – بدون خروجی) :

glcd_clear();

این تابع lcd را پاک کرده و مکان نما را در سطر و ستون صفر قرار می دهد.

3- تابع رفتن به صفحه x (یک ورودی – بدون خروجی) :

glcd_gotox (x);

تابع فوق مکان نمایش ال سی دی را به صفحه (Page) که به جای x قرار می گیرد ، جابجا می کند.

4- تابع رفتن به ستون y (یک ورودی – بدون خروجی) :

glcd_gotoy (y);

تابع فوق مکان نمایش ال سی دی را به ستون (Column) که به جای y قرار می گیرد ، جابجا می کند.

5- تابع رفتن به خط z (یک ورودی – بدون خروجی) :

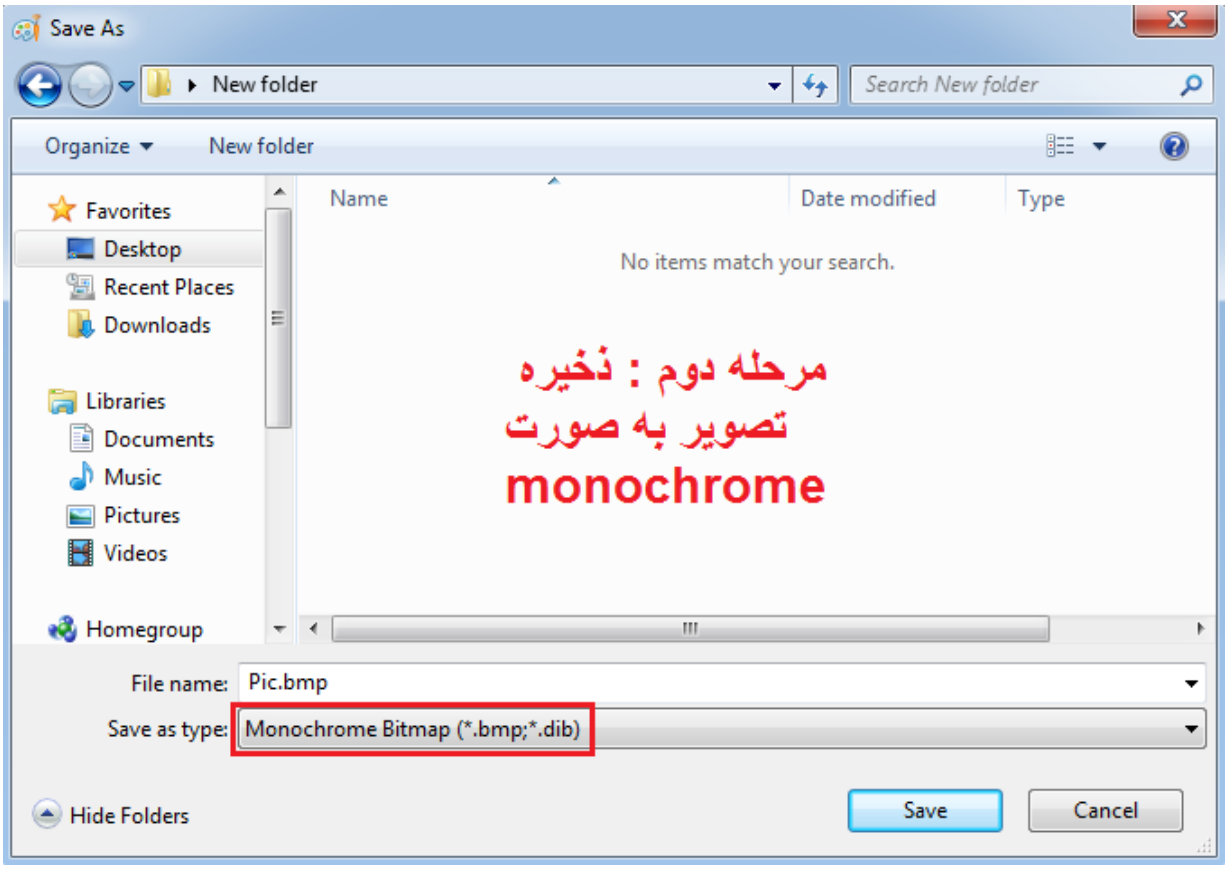
glcd_gotoz (z);

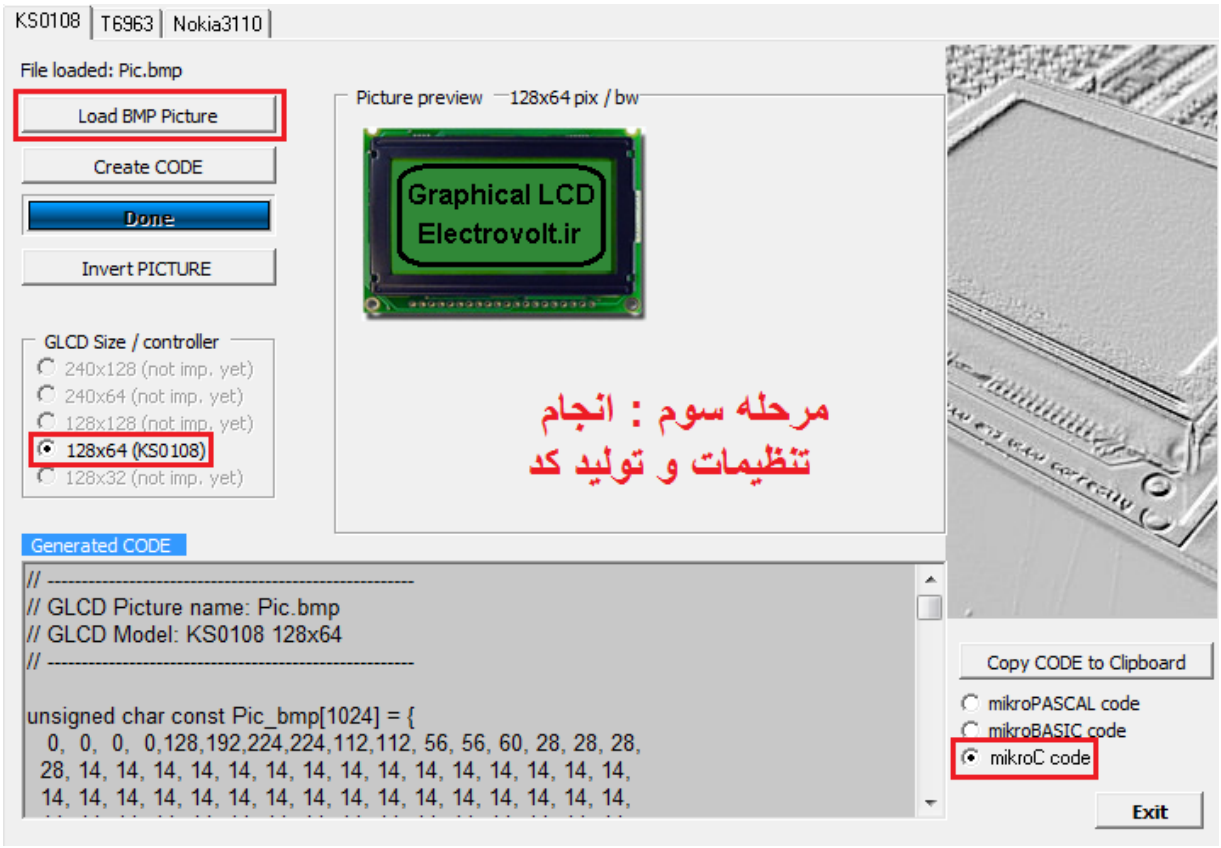
تابع فوق مکان نمایش ال سی دی را به خط (Line) که به جای z قرار می گیرد ، جابجا می کند.

6- تابع چاپ یک تصویر روی LCD گرافیکی (یک ورودی – بدون خروجی) :

glcd_display(*image);

این تابع میتواند تصویری سیاه سفید (monochrome) به ابعاد ۱۲۸*۶۴ را روی LCD گرافیکی نمایش دهد به طوری که ابتدا تصویر در نرم افزارهای دیگر نظیر Paint ویندوز ساخته شده و با فرمت bitmap ذخیره می گردد. سپس این تصویر توسط نرم افزارهای دیگر نظیر GLCD editor به کدهای هگز تبدیل می شود و در یک آرایه ای از نوع char قرار می گیرد. سپس در ورودی تابع glcd_display می بایست به این آرایه اشاره کرد تا روی LCD به نمایش درآید. مراحل ساخت و تبدیل تصویر را در شکل های زیر مشاهده می کنید.





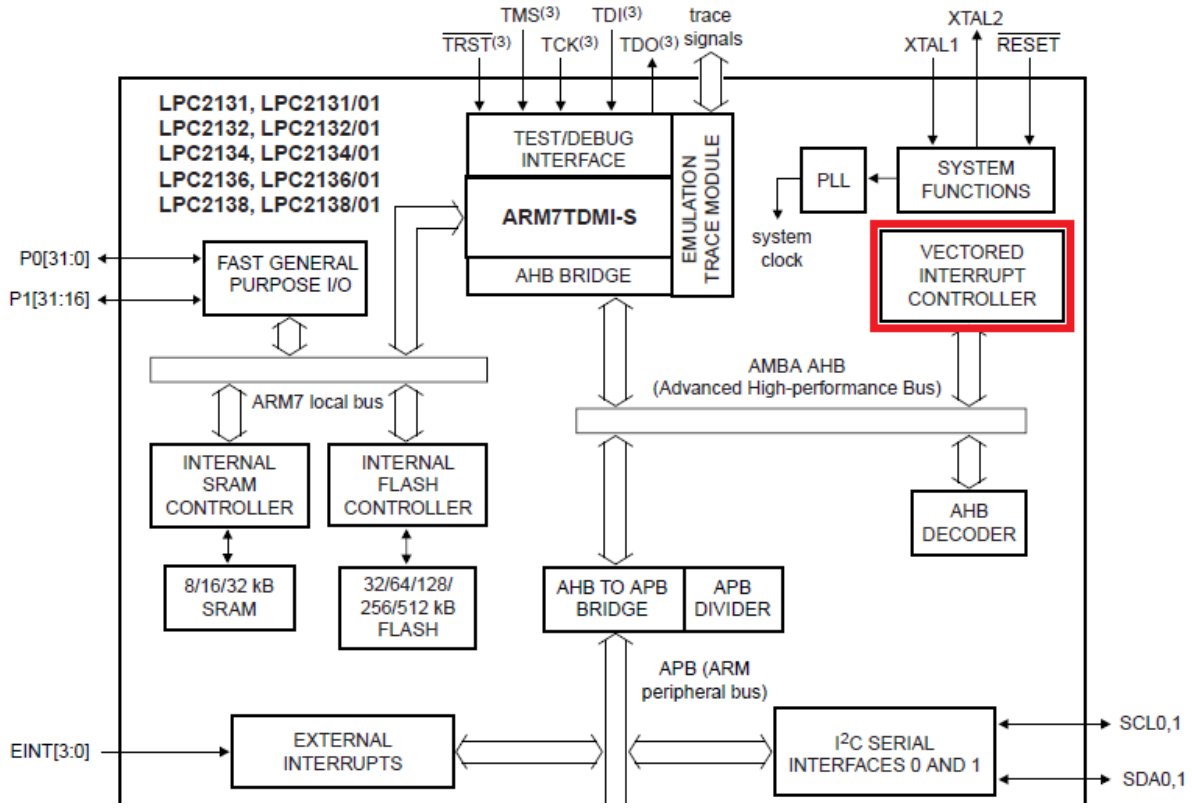
مثال عملی شماره ۱۰ : یک LCD گرافیکی ۶۴ در ۱۲۸ را به میکروکنترلر LPC2138 متصل کرده و برنامه ای بنویسید که یک تصویر دلخواه را روی آن نمایش دهد.

لینک دانلود سورس مثال عملی شماره ۱۰

فصل ۹ – راه اندازی واحد وقفه برداری (VIC)

مقدمه

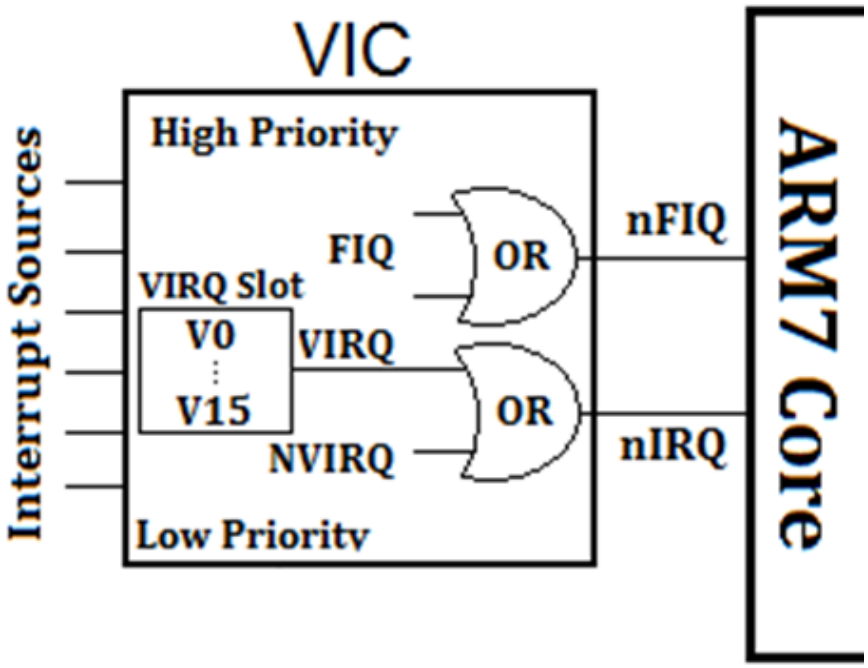
وقفه (interrupt) یک سیگنال است که از طرف سخت افزار یا نرم افزار به منظور توجه فوری تولید شده و به پردازنده مرکزی (CPU) وارد می شود. با تولید سیگنال وقفه ، CPU باید به سخت افزار یا نرم افزاری که نیاز به توجه فوری دارد پاسخ دهد. بنابراین CPU در اولین فرصت ممکن با ذخیره سازی وضعیت فعلی خود ، به سخت افزار یا نرم افزار مورد نظر سرویس دهی می کند و بعد از سرویس دهی مجددا به وضعیت ذخیره شده قبلی خود باز می گردد. مدیریت و کنترل سیگنال های وقفه یعنی این که سیگنال وقفه از کدام سخت افزار یا نرم افزار آمده است ، اولویت بندی در انجام سیگنال های وقفه چگونه است ، نحوه اتصال سیگنال وقفه چگونه است و ... همه بر عهده واحد کنترل وقفه برداری (VIC) است. محل قرار گرفتن این واحد درون LPC213X را در شکل زیر مشاهده می کنید.



معرفی واحد VIC

این واحد Vectored Interrupt Control به معنای واحد کنترل وقفه برداری می باشد. این واحد به تعداد ۳۲ وقفه ورودی را از منابع وقفه دریافت کرده و در دو گروه آن ها را دسته بندی می نماید. دسته اول IRQ (مخفف Interrupt ReQuest) و دسته دوم FIQ (مخفف Fast Interrupt reQuest) نام دارند. سیگنال های وقفه ای که در گروه

FIQ قرار می گیرند دارای بالاترین اولویت هستند. دسته IRQ خود به دو بخش برداری (Vectored) و غیر برداری (Non-Vectored) تقسیم بندی می شود. سیگنال های وقفه ای که در بخش برداری هستند دارای اولویت متوسط و سیگنال های غیر برداری دارای پایین ترین اولویت می باشند. شکل زیر اولویت بندی وقفه ها در میکروکنترلرهای ARM7 را نشان می دهد. همانطور که مشاهده می کنید CPU ابتدا به سیگنال دسته با اولویت بالاتر پاسخ می دهد و سیگنال های وقفه ورودی واحد VIC در هر دسته با یکدیگر OR می شوند.



تعریف ISR : مخفف Interrupt Service Routine به معنای روتین (روال) پاسخ دهی به وقفه می باشد. همانطور که از نام آن مشخص است زمانی که یک سیگنال وقفه رخ می دهد کارهایی را که CPU باید برای پاسخ دهی به آن واحد انجام می دهد ISR گویند.

تفاوت وقفه برداری و غیر برداری : برداری در اصطلاح به معنای این است که CPU زمانی که وقفه رخ می دهد از آدرس ISR مورد نظر و منبع وقفه آگاه است اما در وقفه های غیر برداری CPU اطلاعی از آدرس ISR و منبع وقفه ندارد. برای حالت برداری یک جدول (VIRQ Slot) درون VIC وجود دارد که کاربر میتواند منبع وقفه و آدرس ISR آن را به دلخواه خود تغییر دهد. به عبارت دیگر تفاوت بین VIRQ (وقفه برداری) و NVIRQ (وقفه غیر برداری) در این است که برای هر یک از روتین های وقفه برداری میتوان تابع ISR متفاوتی تعریف کرد و اولویت انجام شدن آنها را نیز مشخص کرد اما برای همه وقفه های غیر برداری تنها یک تابع روتین وقفه میتوان تعریف کرد و تعیین اولویت برای آن امکان پذیر نیست و دارای کمترین اولویت در انجام آن می باشد.

نکته : برای دسته وقفه برداری میتوان تعداد ۱۶ وقفه را تعریف کرد. این تعریف توسط کاربر و درون اسلات وقفه برداری (VIRQ Slot) صورت می گیرد به طوری که بردار 0 ام (V0) دارای بیشترین اولویت و بردار 15 ام (V15) دارای کمترین اولویت می باشد.

منابع وقفه در LPC213X

هر یک از واحدهای جانبی (Peripherals) توسط یک خط به واحد VIC متصل هستند که سیگنال وقفه را تولید و به عنوان منبع وقفه برای VIC ارسال می کنند. منابع وقفه (interrupt sources) در میکروکنترلرهای سری LPC213X به ۲۲ بلاک (Block) وقفه تقسیم بندی می شود که هر بلاک وقفه شامل حداقل یک پرچم (Flag) وقفه می باشد که در صفحه ۷۳ از UM10120 تمامی این منابع را میتوان مشاهده کرد.

رجیسترهای واحد VIC

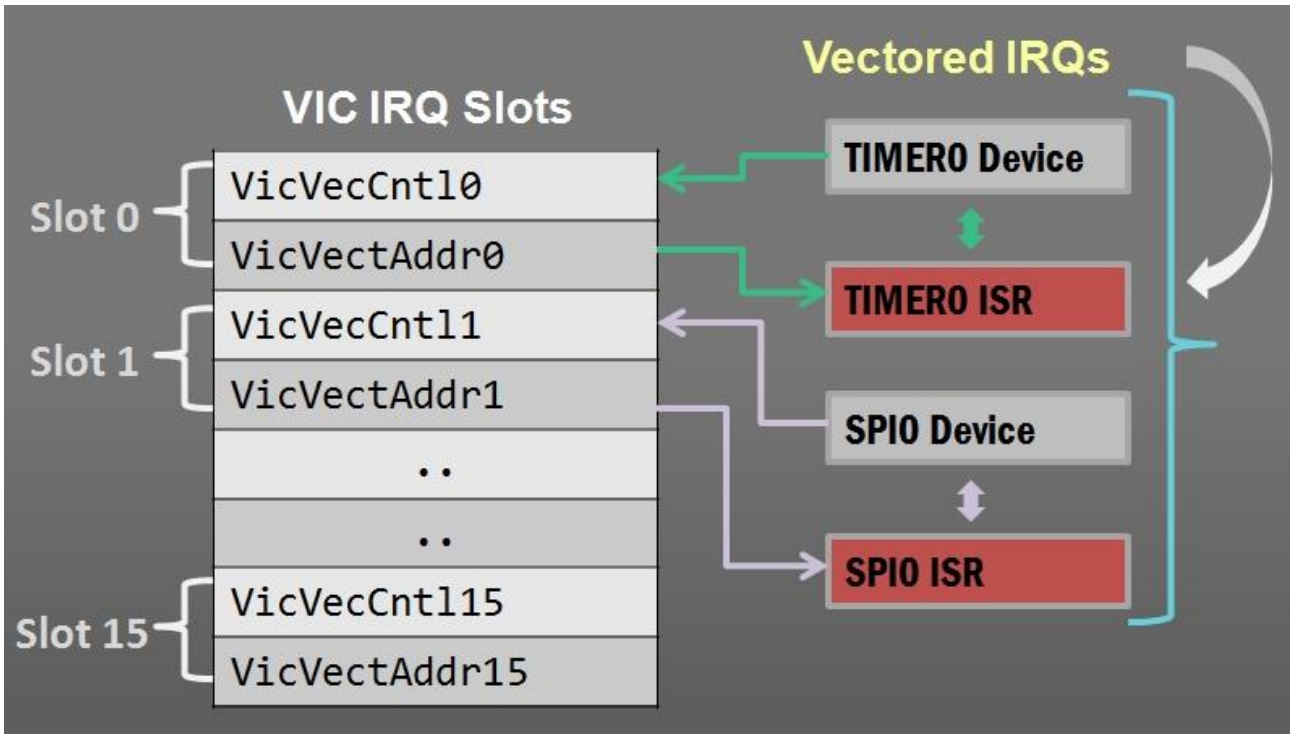
در صفحه ۶۵ و ۶۶ از UM10120 جدول و توضیحات کاملی از تمامی رجیسترهای موجود در این واحد را مشاهده می کنید که به صورت جدول شکل زیر خلاصه شده است :

نام رجیستر	توضیحات عملکرد
VICIntSelect	FIQ (1) یا IRQ (0) بودن وقفه را تعیین می کند
VICIntEnable	فعال (1) یا غیر فعال (0) کردن وقفه را تعیین می کند
VICVectAddrX	آدرس تابع روتین وقفه برداری شماره X را مشخص می کند (X از 0 تا 15)
VICVectCntlX	منبع وقفه برداری شماره X را مشخص می کند (X از 0 تا 15)
VICVectAddr	آدرس برگشت به برنامه اصلی بعد از اتمام وقفه برداری را مشخص می کند
VICDefVectAddr	آدرس تابع روتین وقفه غیر برداری را مشخص می کند (NVIRQ)

در شکل زیر جزئیات مربوط به هر یک از بیت های رجیسترهای VICIntSelect و VICIntEnable را مشاهده می کنید. در حالت پیش فرض همه بیت های این رجیسترها صفر هستند. در صورتی که بخواهیم هر یک از وقفه های مربوط به یکی از ۲۲ واحدهای جانبی مشخص شده را FIQ کنیم باید بیت متناظر آن در رجیستر VICIntSelect را 1 نماییم. همچنین برای فعال سازی هر یک از این وقفه ها باید بیت متناظر آن در VICIntEnable را 1 نماییم.

Bit	31	30	29	28	27	26	25	24
Symbol	-	-	-	-	-	-	-	-
Bit	23	22	21	20	19	18	17	16
Symbol	-	-	AD1	BOD	I2C1	AD0	EINT3	EINT2
Bit	15	14	13	12	11	10	9	8
Symbol	EINT1	EINT0	RTC	PLL	SPI1/SSP	SPI0	I2C0	PWM0
Bit	7	6	5	4	3	2	1	0
Symbol	UART1	UART0	TIMER1	TIMER0	ARMCore1	ARMCore0	-	WDT

همچنین با مقدار دهی رجیسترهای VICVectCntlX و VICVectAddrX که در آن X یک عدد صحیح از 0 تا 15 است ، میتوان اسلات مربوط به وقفه برداری دلخواه را به یکی از ۳۲ منبع موجود متصل کرد.برای مثال فرض کنید می خواهیم از وقفه های TIMER0 و SPI0 استفاده کنیم به طوری که وقفه تایمر اولویت بالاتری داشته باشد. بنابراین باید اسلات 0 ام که اولویت بالاتری دارد را به TIMER0 و اسلات 1 ام را به SPI0 تخصیص دهیم. شکل زیر این موضوع را نشان می دهد.



در نتیجه برای راه اندازی وقفه در مثال فوق باید خطوط زیر را بنویسیم :

```
VICIntSelect = 0x00000000;

VICIntEnable = (1<<4) | (1<<10);

VICVectCntl0 = 0x20 | 4;

VICVectCntl1 = 0x20 | 10;
```

```
VICVectAddr0 = (unsigned) TIMER_ISR;
VICVectAddr1 = (unsigned) SPI0_ISR;
```

در حالت کلی برای راه اندازی اسلات شماره X و تخصیص آن به منبع وقفه شماره Y و روتین وقفه با نام دلخواه myISR به صورت زیر عمل می کنیم :

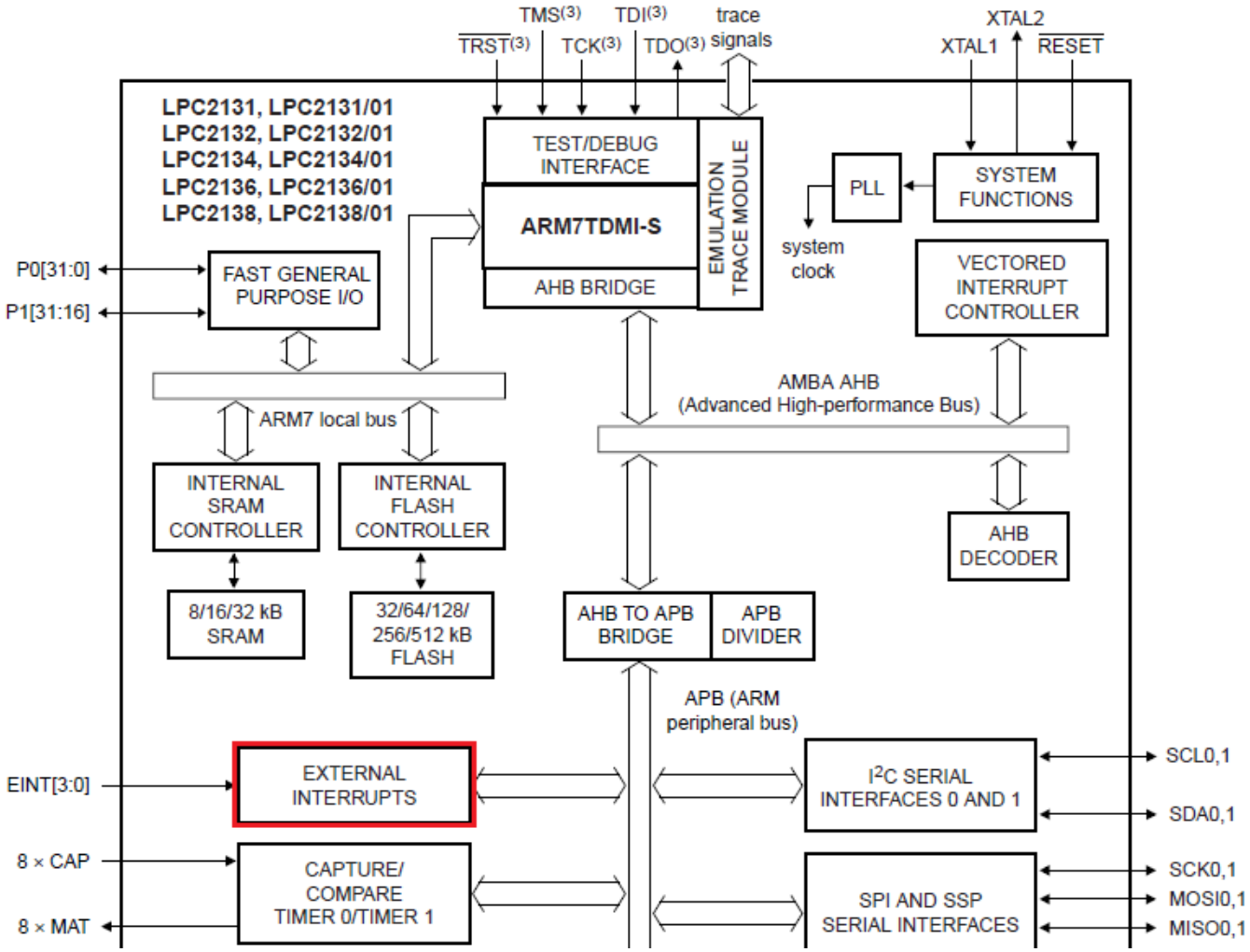
```
VICIntEnable  |= (1<<Y) ;
VICVectCntlX   = (1<<5) | Y ;
VICVectAddrX  = (unsigned) myISR;
```

روتین وقفه (ISR) تابعی است که در هنگام رخ دادن وقفه CPU آن را اجرا می کند. این تابع در ARM7 میتواند نام دلخواهی داشته باشد و حتما در ابتدای آن __IRQ یا __FIQ وجود دارد. همچنین در انتهای تابع روتین وقفه نیز باید آدرس برگشت را صفر کرد. برای مثال :

```
__irq void myISR ( void ) {
کارهایی که در زمان وقفه باید انجام گیرد
VICVectAddr = 0;
}
```

راه اندازی واحد وقفه خارجی (External Interrupt)

این واحد میتواند وقفه را روی پایه های خارجی میکرو که به صورت EINTx هستند ، فعال نماید. با فعالسازی این واحد ، پایه های EINTx به صورت ورودی تنظیم شده و در زمان رخداد سیگنال وقفه خارجی ، میکرو به تابع روتین وقفه پرش می کند و برنامه نوشته شده در آن را اجرا می کند. در میکروکنترلرهای سری LPC213X تعداد ۴ پایه وقفه خارجی وجود دارد. در شکل زیر واحد وقفه خارجی و پایه های آن را مشاهده می کنید.



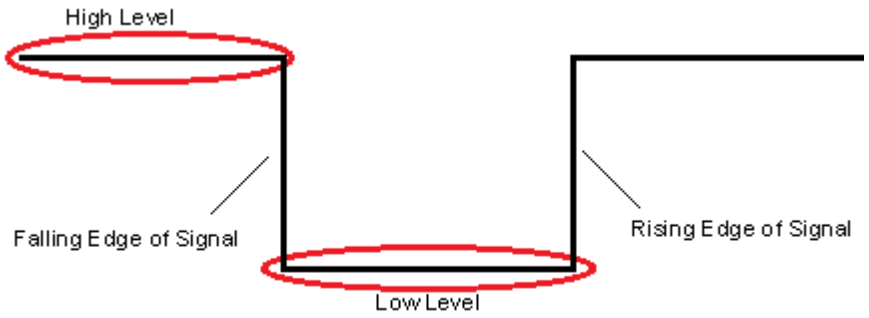
در جدول زیر وقفه های خارجی و پایه های مربوط به آن را مشاهده می کنید :

Name	EXTINT0	EXTINT1	EXTINT2	EXTINT3
Pins	P0.1	P0.3	P0.7	P0.9
	P0.16	P0.14	P0.15	P0.20
	P1.25	-	-	P0.30

وقفه خارجی در یکی از رخدادهای زیر اتفاق می افتد :

- لبه بالارونده پالس ورودی
- لبه پایین رونده پالس ورودی
- سطح منطقی ۱ پالس ورودی
- سطح منطقی ۰ پالس ورودی

شکل زیر لبه ها و سطوح مختلف را در یک سیگنال ورودی نمونه نشان می دهد :



رجیسترهای واحد وقفه خارجی

این رجیسترها تنظیمات واحد وقفه خارجی می باشد که به ترتیب بیت های 0 تا 3 آنها مربوط به EXINT0 تا EXTINT3 می باشد.

رجیستر EXTINT (مخفف External Interrupt Flag Register)

این رجیستر شامل پرچم (Flag) برای وقفه های خارجی 0 تا 3 می باشد. به طوری که با فعال بودن حالت وقفه خارجی روی پایه مورد نظر در رجیستر PINSEL ، با 1 کردن بیت مربوط به وقفه مورد نظر در این رجیستر فعال می شود.

رجیستر EXTMOD (مخفف External Interrupt Mode Register)

این رجیستر حالت عملکرد وقفه خارجی را روی EXTINT0..3 مشخص می کند. در صورتی که بیت مربوطه 0 باشد ، وقفه خارجی حساس به سطح (Level Sensitive) و در صورت 1 بودن آن حساس به لبه (Edge Sensitive) می باشد.

رجیستر EXTPOLAR (مخفف External Interrupt Polar Register)

این رجیستر بسته به تنظیم رجیستر قبلی پلاریته یا وضعیت رخداد وقفه را مشخص می کند. در صورتی که بیت مربوط به EXTINT0...3 در این رجیستر 0 باشد یعنی وقفه در سطح 0 یا لبه پایین رونده (بسته به رجیستر قبلی) می باشد. در صورتی که بیت مربوطه در این رجیستر 1 باشد ، یعنی وقفه در سطح 1 یا لبه بالارونده (بسته به رجیستر قبل) می باشد.

مراحل راه اندازی وقفه خارجی

۱. تنظیم رجیستر PINSEL
۲. تنظیم رجیسترهای EXTMOD و EXTPOLAR
۳. تنظیمات وقفه خارجی در واحد VIC
۴. فعالسازی عمومی وقفه
۵. نوشتن تابع سابروتین وقفه

تابع راه اندازی وقفه خارجی 0 ام روی P0.1

```
void extint0_init(void)
{
 //extint0 falling edge
 EXTMODE =1;
 EXTPOLAR =0;
 PINSEL0 |= (3<<2);
 VICIntSelect &= ~(1<<14);
 VICIntEnable |= (1<<14);
 VICVectCntl0 = 0x20 | 14;
 VICVectAddr0 = (unsigned) extint_isr;
 EXTINT &= ~(1<<0);
}
```

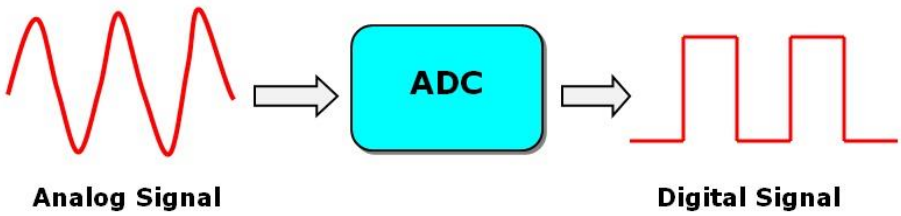
مثال عملی شماره ۱۱ : با اتصال یک LED به پورت دلخواه میکروکنترلر LPC2138 و اتصال یک کلید به یکی از پایه های وقفه آن برنامه ای بنویسید که با زدن کلید ، LED در تابع سابروتین وقفه یکبار روشن و خاموش شود.

لینک دانلود سورس مثال عملی شماره ۱۱

فصل ۱۰ – راه اندازی واحد مبدل آنالوگ به دیجیتال(ADC)

مقدمه

همانطور که میدانید تمامی عملیات ها در میکروکنترلرها به صورت دیجیتال انجام می شود. برای تبدیل انواع سیگنال های الکترونیکی آنالوگ نظیر سیگنال خروجی بسیاری از سنسورها ، از این واحد استفاده می شود. این واحد ولتاژ ورودی آنالوگ را دریافت کرده و به کد دیجیتال در خروجی تبدیل می کند. خروجی دیجیتال درون یک رجیستر ذخیره می شود.



معرفی واحد مبدلADC

واحد ADC در میکروکنترلر LPC2138 دو واحد مجزا از هم می باشد که هر یک ۸ کانال ورودی مختلف نیز دارند. در نتیجه مجموعا ۱۶ ورودی آنالوگ وجود دارد. در یک زمان فقط یکی از کانال ها میتواند به کد دیجیتال در خروجی تبدیل شود. این واحد دارای مشخصات زیر می باشد:

ولتاژ مرجع (VREF) : بین ۰ تا VDDA (حداکثر 3.3v) می تواند باشد.

دقت (رزولوشن) : فقط ۱۰ بیت می تواند باشد

حداکثر نرخ نمونه برداری : 4.5Msps (چهار و نیم میلیون نمونه در ثانیه)

تعداد واحدهای ADC در LPC2138 : دو واحد مجزا از هم

تعداد کانال های ورودی آنالوگ هر واحد : 8 کانال

$$\text{ضریب تفکیک} = \frac{Input\ Range}{Resolution} = \frac{3.3V}{2^{10}} = 3.23\,\text{mV}$$

پایه های واحد ADC

Name	Pin	Description	Value
VDDA	VDDA	Analog Power	3.3v
VSSA	VSSA	Analog Ground	0v
VREF	VREF	Reference Voltage	0-3.3v
AD0.0	P0.27	ADC0 Channel0	0-VREF
AD0.1	P0.28	ADC0 Channel1	0-VREF
AD0.2	P0.29	ADC0 Channel2	0-VREF
AD0.3	P0.30	ADC0 Channel3	0-VREF
AD0.4	P0.25	ADC0 Channel4	0-VREF
AD0.5	P0.26	ADC0 Channel5	0-VREF
AD0.6	P0.4	ADC0 Channel6	0-VREF
AD0.7	P0.5	ADC0 Channel7	0-VREF
AD1.0	P0.6	ADC1 Channel0	0-VREF
AD1.1	P0.8	ADC1 Channel1	0-VREF
AD1.2	P0.10	ADC1 Channel2	0-VREF
AD1.3	P0.12	ADC1 Channel3	0-VREF
AD1.4	P0.13	ADC1 Channel4	0-VREF
AD1.5	P0.15	ADC1 Channel5	0-VREF
AD1.6	P0.21	ADC1 Channel6	0-VREF
AD1.7	P0.22	ADC1 Channel7	0-VREF

رجیسترهای مربوط به واحد ADC

رجیسترهای ADXCR برای کنترل واحد و رجیسترهای ADXDR برای دیتای خروجی دیجیتال هستند که در آن X میتواند ۰ یا ۱ بسته به شماره واحد ADC مورد استفاده باشد.

رجیستر کنترل ADXCR

یک رجیستر ۳۲ بیتی برای تنظیمات واحد ADC شماره X می باشد که در LPC2138 عدد X می تواند ۰ یا ۱ باشد.

8 بیت اول این رجیستر مشخص کننده شماره کانالی است که می خواهیم ولتاژ ورودی آنالوگ را از آن بخوانیم. هر بیتی که ۱ شود ، ورودی آنالوگ متناظر با آن وارد واحد ADC می شود.

8 بیت بعدی آن عددی قرار می گیرد که کلاک واحد ADC (نرخ نمونه برداری) را مشخص می کند. این کلاک می تواند حداکثر 4.5Mhz باشد اما استاندارد نمونه برداری برای کارهای معمولی مورد استفاده بین ۱۰۰ تا ۳۰۰ کیلوهرتز می باشد .اگر عددی که در این ۸ بیت قرار می گیرد را Y بنامیم ، به صورت زیر مشخص می شود :

$$Y = \frac{P_{clk}}{ADC_{clk}} - 1 \quad , \quad P_{clk} = \frac{C_{clk}}{N}$$

برای مثال فرض کنید می خواهیم واحد ADC را با نرخ 100KSPSراه اندازی نماییم .در صورتی که از کریستال 12Mhz استفاده نماییم و تقسیم کننده APB روی N=4 باشد. داریم :

$$Y = \frac{3M}{100K} - 1 = 30 - 1 = 29$$

زمانی که میکروکنترلر شروع به کار می کند ، واحد ADC در حالت غیر فعال است (حالت Power Down است). بعد از مشخص کردن شماره کانال و مقدار کلاک واحد باید بیت ۲۱ از رجیستر ADXCR را ۱ کرد تا واحد ADC فعال شود.

سه بیت ۲۴ ، ۲۵ و ۲۶ از این رجیستر مربوط به وضعیت تبدیل می باشد. به طوری که

زمانی که این سه بیت به صورت 001 باشد ، تبدیل آغاز می گردد. بنابراین به محض 1 شدن بیت ۲۴ ام عملیات تبدیل آغاز می گردد.

بعد از انجام شدن یک تبدیل ، هر سه بیت باید 0 شود تا تبدیل قطع شده و واحد ADC برای تبدیل بعدی آماده گردد.

رجیستر دیتا ADXDR

یک رجیستر ۳۲ بیتی است که نتیجه تبدیل زمانی که بیت ۳۱ ام آن ۱ شود ، در بیت های ۶ تا ۱۵ آن قرار می گیرد.

بیت 31ام زمانی که تبدیل به پایان رسیده باشد و نتیجه آن در بیت های ۶ تا ۱۵ قرار گرفته باشد ، به طور خودکار 1 می شود. بنابراین باید تا زمانی که این بیت 0 است ، صبر کنیم.

توابع راه اندازی ADC

یک تابع تعریف می کنیم که هنگام نیاز به راه اندازی ADC از آن استفاده کنیم . (initialization) همچنین یک تابع هم برای خواندن (read) از کانال مورد نظر تعریف می کنیم که همانند برنامه نویسی در AVR از آن استفاده نماییم.

1- توابع adc0_init و adc1_init :

این دو تابع که به صورت زیر تعریف می شوند ، تنظیمات لازم برای رجیستر ADXCR را اعمال می نمایند و در صورت نیاز از آن ها در برنامه استفاده می گردد .ورودی این توابع عدد Y می باشد.

```
void adc0_init(unsigned char CLKDIV)
 {
 AD0CR=(CLKDIV<<8)|(1<<21);
 AD0CR&=0xFFFFFF00;
 }
void adc1_init(unsigned char CLKDIV)
 {
 AD1CR=(CLKDIV<<8)|(1<<21);
 AD1CR&=0xFFFFFF00;
 }
```

توضیح برنامه : در خط اول هر تابع ، تنظیمات مربوط به عدد Y و نیز فعالسازی واحد ADC صورت می گیرد و در خط دوم هر تابع ، برنامه کانال های واحد ADC را 0 می کند (یعنی هیچ کانالی انتخاب نشود)

2- توابع read_adc0 و read_adc1 :

این دو تابع که به صورت زیر تعریف می شوند، در ورودی خود عدد مربوط به کانال را دریافت و بعد از اتمام عملیات تبدیل، عدد دیجیتال را به خروجی تابع می فرستد.

```
unsigned short read_adc0(unsigned char channel)
 {
```

```
switch (channel)
{
case 0:
PINSEL1 |= (1<<22);
break;
case 1:
PINSEL1 |= (1<<24);
break;
case 2:
PINSEL1 |= (1<<26);
break;
case 3:
PINSEL1 |= (1<<28);
break;
case 4:
PINSEL1 |= (1<<18);
break;
case 5:
PINSEL1 |= (1<<20);
break;
case 6:
PINSEL0 |= (3<<8);
break;
case 7:
PINSEL0 |= (3<<10);
break;
}
AD0CR|=(1<<24)|(1<<channel);
```

```
  while((AD0DR & 0x80000000) == 0);
AD0CR&=0xF8FFFFFF;
return ((AD0DR>>6) & 0x3FF);
  }
unsigned short read_adc1(unsigned char channel)
  {
  switch (channel)
  {
  case 0:
  PINSEL0 |= (3<<12);
  break;
  case 1:
  PINSEL0 |= (3<<16);
  break;
  case 2:
  PINSEL0 |= (3<<20);
  break;
  case 3:
  PINSEL0 |= (3<<24);
  break;
  case 4:
  PINSEL0 |= (3<<26);
  break;
  case 5:
  PINSEL0 |= (3<<30);
  break;
  case 6:
  PINSEL1 |= (2<<10);
```

```
 break;
 case 7:
 PINSEL1 |= (1<<12);
 break;
 }
AD1CR|=(1<<24)|(1<<channel);
 while((AD1DR & 0x80000000) == 0);
AD1CR&=0xF8FFFFFF;
return ((AD1DR>>6) & 0x3FF);
 }
```

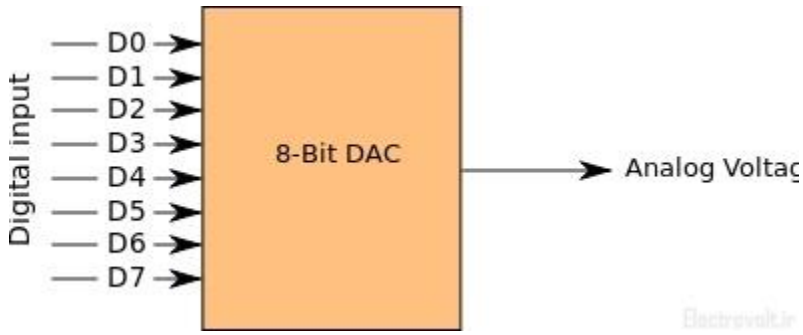
توضیح برنامه : در ابتدای هر تابع ، بسته به ورودی تابع ، رجیستر PINSEL مربوط به پایه کانال مورد نظر ، روی حالت ADC قرار می گیرد. سپس کانال مورد نظر و شروع تبدیل به رجیستر ADXCR اعمال می گردد. سپس برنامه منتظر میماند تا تبدیل به اتمام برسد. سپس بیت تبدیل در رجیستر ADXCR برابر 0 می گردد. در نهایت مقدار بیتهای بین ۶ تا ۱۵ رجیستر ADXDR به خروجی تابع برمی گردد.

مثال عملی شماره ۱۲ : با استفاده از یک LCD کاراکتری ۲ در ۱۶ و یک مقاومت متغیر که به میکروکنترلر LPC2138 متصل شده است برنامه ای بنویسید که ولتاژ آنالوگ ایجاد شده توسط مقاومت متغیر را روی LCD نمایش دهد.

لینک دانلود سورس مثال عملی شماره ۱۲

فصل ۱۱ – راه اندازی واحد مبدل دیجیتال به آنالوگ (DAC)

این واحد عملیاتی برعکس مبدل ADC را انجام می دهد. یعنی یک عدد دیجیتال ۱۰ بیتی به صورت ۰ و ۱ را درون یک رجیستر دریافت کرده و متناسب با آن عدد ولتاژ آنالوگ بین ۰ تا VREF در خروجی تولید می کند. شکل زیر یک مبدل DAC نمونه ۸ بیتی را نشان می دهد. در میکروکنترلرهای ARM7 از جمله LPC2138 این واحد ۱۰ بیتی است.



این واحد در میکروکنترلرهای سری LPC213x دارای ویژگی های زیر است:

- مبدل دیجیتال به آنالوگ با دقت ۱۰ بیت
- دارای معماری به صورت آرایه های مقاومتی
- دارای بافر پایه خروجی
- دارای حالت کم مصرف
- قابلیت تنظیم سرعت و توان پایه خروجی

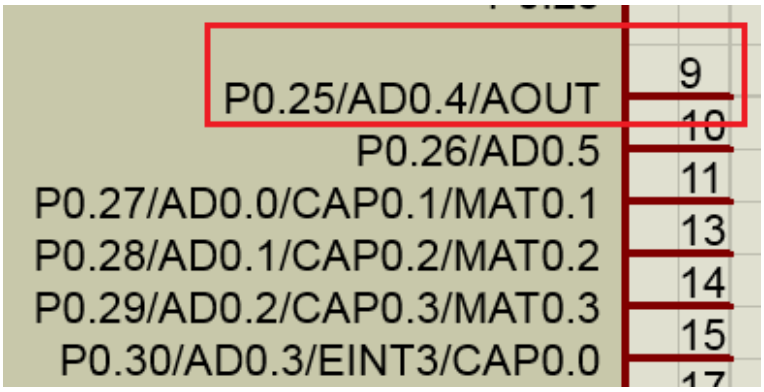
نام رجیستر ورودی دیجیتال : DACR

پایه خروجی آنالوگ : پایهAOUT

ولتاژ مرجع واحد DAC : ولتاژ پایهVREF

تغذیه واحد DAC : پایه های VSSA وVDDA

در شکل زیر پایه خروجی این واحد را مشاهده می کنید. (P0.25)



رجیستر DACR

مخفف Digital to Analog Control Registerمی باشد. یک رجیستر ۳۲ بیتی است که فقط بیت های زیر از این رجیستر استفاده می شود و بقیه بیت ها بلا استفاده (RESERVE) هستند.

بیت های ۶ تا ۱۵ : دیتای دیجیتال ورودی (۱۰ بیت)

بیت ۱۶ : تنظیم سرعت و توان پایه خروجی

در صورتی که این بیت ۰ باشد حداکثر زمان تبدیل ۱ usو حداکثر جریان پایه خروجی 700mA می شود. در صورتی که این بیت ۱ باشد حداکثر زمان تبدیل 2.5us و حداکثر جریان پایه خروجی 350ma می شود.

توابع راه اندازی واحد DAC

برای استفاده از واحد مبدل DAC دو تابع به صورت زیر تعریف می کنیم.

1- تابع dac_init :

با فراخوانی این تابع ، پایه AOUT به عنوان خروجی واحد DAC تنظیم می شود.

```
void dac_init(void)
{
 PINSEL1|=0x00080000;
}
```

2- تابع dac_convert :

با فراخوانی این تابع ، متناسب با مقدار ورودی تابع ولتاژی آنالوگ روی پایه AOUT تولید می شود. درون این تابع تنظیمات رجیستر DACR انجام می گیرد.

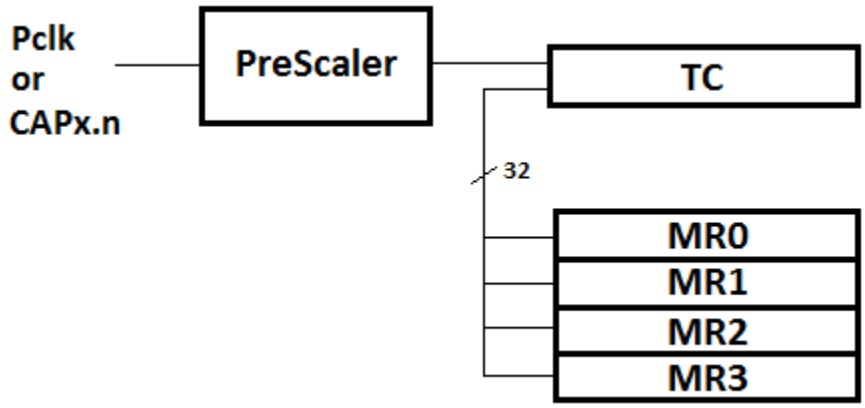
```
void dac_convert(unsigned short value)
{
 DACR=(value<<6)|(1<<16);
}
```

مثال عملی شماره ۱۳ : با استفاده از میکروکنترلر LPC2138 و واحد DAC برنامه ای بنویسید که یک موج سینوسی با دامنه ۰ تا ۳,۳ ولت روی پایه Aout ایجاد کند.

لینک دانلود سورس مثال عملی شماره ۱۳

فصل ۱۲ – راه اندازی واحد تایمر/کانتر Timer/Counter

این واحد در میکروکنترلرهای ARM7 شامل یک شمارنده ۳۲ بیتی است که در دو حالت تایمری و کانتری از آن استفاده می گردد. شکل زیر بلوک دیاگرام ساده شده این واحد را نشان می دهد.



در میکروکنترلر LPC2138 دو واحد تایمر/کانتر ۳۲ بیتی وجود دارد که در هر یک از آنها یک رجیستر شمارنده اصلی یا TC وجود دارد که از 0 تا 2^{32} -1 تغییر کرده و سپس سر ریز می شود. یعنی بعد از پر شدن به حالت اولیه 0 برمیگردد و مجددا به صورت افزایشی ادامه می یابد. یک Prescaler یا پیش تقسیم کننده نیز وجود دارد که کلاک ورودی واحد Timer را تقسیم بر یک عدد صحیح می کند. تعداد ۴ رجیستر تطبیق (Match Register) وجود دارد که همواره با عدد رجیستر TC مقایسه می شود و در صورت برابر بودن با هر یک از آنها در برنامه وقفه تطبیق اتفاق می افتد.

نکته مهم : کلاک ورودی در حالت تایمری از Pclk و در حالت کانتری از پایه CAPx.n تامین می شود.

نکته : در میکروکنترلرهای ARM7 به علت استفاده زیاد از موج PWM ، یک واحد مجزا برای تولید PWM در نظر گرفته شده است.

نتیجه : از واحد تایمر/کانتر فقط برای تولید زمان خاص (تایمر) یا برای شمارش (کانتر) استفاده می شود.

در این واحد ۴ رجیستر ۳۲ بیتی (MR0 تا MR3) برای مقایسه در نظر گرفته شده است که قابلیت تولید وضعیت های زیر در شرایط تطبیق را دارد:

-ادامه شمارش تایمر و تولید وقفه در زمان تطبیق

-متوقف کردن شمارش تایمر و تولید وقفه اختیاری در زمان تطبیق

-ریست کردن شمارش تایمر و تولید وقفه اختیاری در زمان تطبیق

همچنین تا ۴ پایه خروجی (MATx.n) برای زمان تطبیق در نظر گرفته شده است که قابلیت های زیر را دارد:

-خروجی در زمان تطبیق Low می شود.

-خروجی در زمان تطبیق High می شود.

-خروجی در زمان تطبیق تغییر وضعیت می دهد.(Toggle)

-خروجی در زمان تطبیق هیچ تغییری نمی کند.

پایه های مربوط به واحد تایمر/کانتر

در میکروکنترلرهای ARM7 چندین خروجی با نام MATX.n وجود دارد که هر یک از آنها به طور مجزا میتواند زمانی که شرایط تطبیق بین رجیستر تایمر و یکی از رجیسترهای مقایسه اتفاق افتاد ، تغییر وضعیت بدهد. همچنین چندین ورودی با نام CAPX.n وجود دارد که هر یک از آنها در زمان رخداد پالس ورودی یک واحد به رجیستر شمارنده اضافه می کند.

	Match0	Macth1	Match2	Match3	Capture0	Capture1	Capture2	Capture3
Timer0	P0.3 P0.22	P0.5 P0.27	P0.16 P0.28	P0.29	P0.2 P0.22 P0.30	P0.4 P0.27	P0.6 P0.16 P0.28	P0.29
Timer1	P0.12	P0.13	P0.17 P0.19	P0.18 P0.20	P0.10	P0.11 P0.19	P0.17	P0.18 P0.21

کلاک واحد تایمر/کانتر

در حالت تایمری کلاک این واحد Pclk و در حالت کانتری کلاک واحد از پایه های CAPX.nتامین می شود که در آن X یکی از اعداد ۰ یا ۱ و n یکی از اعداد 0,1,2,3می باشد.

یک پیش تقسیم کننده (pre Scaler) درون واحد تایمر/کانتر وجود دارد که میتوان با استفاده از آن کلاک این واحد را قبل از ورود به سیستم تقسیم کرد .این پیش تقسیم کننده به علت ۳۲ بیتی بودن از ۰ تا ۱-۲ ^{32}می تواند تقسیم کند.

رجیسترهای واحد تایمر/کانتر

رجیستر TXTC (مخفف Timer/Counter) : رجیستر تایمر شماره X است که عدد لحظه ای تایمر را نشان می دهد.

رجیستر TXTCR (مخفف Timer Control Register) : رجیستر تنظیمات تایمر/کانتر شماره X است. بیت ۰ ام آن اگر ۱ باشد، تایمر فعال Enable می شود و بیت ۱ ام آن اگر ۱ باشد تایمر Reset می شود.

رجیستر TXPR (مخفف Prescaler Register) : این رجیستر مربوط به پیش تقسیم کننده است. در صورتی که ۰ باشد، عدد تقسیم برابر ۱ است. به طور کلی عددی که میخواهیم تقسیم شود منهای ۱ باید در آن قرار گیرد. بنابراین پیش تقسیم کننده به اندازی PR+1 تقسیم می کند.

رجیسترهای TXMR0 تا TXMR3 (مخفف Match Register) : رجیسترهای مقایسه می باشند. مقدار رجیستر TXTCهر لحظه با همه این رجیسترها مقایسه می شود و به محض برابر شدن مقدار تایمر میتواند Reset یا Stopشود یا وارد تابع وقفه شود.

رجیستر TXMCR (مخفف Match Control Register) : این رجیستر تنظیمات مربوط به Reset ، Stopیا وقفه قرار گرفتن رجیسترهای TXMR0 تا TXMR3 در زمان برابری را مشخص می کند. در این رجیستر برای هر ۴ رجیستر TXMR0 تاTXMR3 ، به تعداد ۳ بیت برای تنظیم وقفه ، Reset یا Stop شدن وجود دارد (مجموعا ۱۲ بیت). بنابراین سه بیت اول این رجیستر برای MR0 ، سه بیت دوم برایMR1 ، ...سه بیت چهارم برای MR3 می باشد.

رجیستر TXEMR (مخفف External Match Register) : این رجیستر برای فعالسازی و تنظیمات چگونگی عملکرد پایه های خروجی MATX.n در زمان تطبیق است.

رجیستر TXCTCR (مخفف Count Control Register) : دو بیت اول این رجیستر تنظیمات حالت تایمری/کانتری بودن واحد را مشخص می کند. دو بیت بعدی آن تنظیم اینکه کانتر روی کدام پایه عمل می کند.

مراحل راه اندازی واحد تایمر/کانتر در حالت تایمری

۱. تنظیم کلاک واحد توسطTxPR
۲. تنظیم TxMR0..3 در صورت نیاز
۳. تنظیم TxMCR متناسب با مرحله قبل

۴. تنظیم رجیستر PINSEL برای پایه های خروجی MATx.n

۵. انجام تنظیمات واحد VIC در صورت نیاز به وقفه تطبیق

۶. ریست کردن رجیستر تایمر به منظور آماده شدن به شروع کار

۷. فعال کردن تایمر توسط رجیستر TxTCR

تابع راه اندازی واحد تایمر/کانتر

برای راحتی توابع timer0_init و timer1_init را تعریف کرده و مراحل را اندازی را در این توابع انجام می دهیم. برای راه اندازی LPC2138 به علت وجود دو واحد مجزا تایمر/کانتر دو تابع تعریف می کنیم.

```
#define PRESCALE0  1
#define PRESCALE1  1
#define CCLK 12000000
#define PCLK 3000000
void Timer0_init( void ){
T0TC=0;
T0PR=PRESCALE0-1;
T0MR0=PCLK;
T0MCR=3;
T0TCR=2;
T0TCR=1;
}
void Timer1_init( void ){
T1TC=0;
T1PR=PRESCALE1-1;
T1MR0=PCLK;
T1MCR=3;
T1TCR=2;
```

```
T1TCR=1;
}
```

مثال عملی شماره ۱۴ : با استفاده از واحد تایمر میکروکنترلر LPC2138 و اتصال یک LCD کاراکتری به آن برنامه ای بنویسید که با هر بار آمدن تعداد PCLK/3 کلاک ، یک واحد به عدد روی LCD اضافه شود.

لینک دانلود سورس مثال عملی شماره ۱۴

استفاده از واحد تایمر/کانتر در حالت کانتری

در حالت کانتری، باید به یکی از پایه های CAPX.n به عنوان ورودی کلاک واحد تایمر/کانتر ، پالسی اعمال شود. مراحل راه اندازی در این حالت به صورت زیر است:

۱. تنظیم حالت رخداد پالس ورودی در رجیستر TXCTCR
۲. تنظیم عدد تطبیق در رجیسترهای TXMR0..3 و حالت تطبیق در TXMCR
۳. تنظیم رجیستر PINSELX برای CAPX.n مورد نظر
۴. انجام تنظیمات واحد VIC در صورت نیاز به وقفه تطبیق
۵. ریست کردن رجیستر تایمر به منظور آماده شدن به شروع کار
۶. فعال کردن کانتر توسط رجیستر TXTCR

نحوه تنظیم رجیستر TXCTCR : برای دو بیت اول (بیت ۰ و ۱) این رجیستر ۴ حالت زیر قابل استفاده است:

00: حالت کار تایمری (با لبه بالارونده کلاک PCLK/PRESCALE کار می کند)

01: حالت کار کانتری با لبه بالا رونده یکی از پایه های مشخص شده در بیت ۲ و ۳ همین رجیستر

10: حالت کار کانتری با لبه پایین رونده یکی از پایه های مشخص شده در بیت ۲ و ۳ همین رجیستر

11: حالت کار کانتری با هر دو لبه بالا رونده و پایین رونده یکی از پایه های مشخص شده در بیت ۲ و ۳ همین رجیستر

برای دو بیت دوم (بیت ۲ و ۳) این رجیستر نیز وقتی دوبیت قبلی ۰۰ نباشد چهار حالت زیر قابل استفاده است:

00: تنظیم CAP0.0 به عنوان ورودی TIMER0 و CAP1.0 به عنوان ورودیTIMER1

01: تنظیم CAP0.1 به عنوان ورودی TIMER0 و CAP1.1 به عنوان ورودیTIMER1

10: تنظیم CAP0.2 به عنوان ورودی TIMER0 و CAP1.2 به عنوان ورودیTIMER1

11: تنظیم CAP0.3 به عنوان ورودی TIMER0 و CAP1.3 به عنوان ورودیTIMER1

برای مثال در صورتی که کانتر در لبه بالا رونده عمل کند و ورودی CAP0.2 مورد نظر باشد رجیستر TXCTCR روی ۱۰۰۱یعنی ۹ تنظیم می شود.

تابع راه اندازی کانتر توسط TIMER0 رویCAP0.2

طبق مراحل راه اندازی عمل می کنیم. ابتدا رجیستر PINSEL1 مربوط به پایه P0.28 را تنظیم می کنیم.

```
void counter0_init( void ) {
PINSEL1=(1<<25);
T0CTCR=9;
T0TC=0;
T0TCR=1;
}
```

مثال عملی شماره ۱۵ : با اتصال یک کلید به ورودی واحد تایمر/کانتر میکروکنترلر LPC2138 و یک LCD کاراکتری ۲ در ۱۶ برنامه ای بنویسید که با زدن هر بار کلید یک واحد به عدد روی نمایشگر اضافه شود.

لینک دانلود سورس مثال عملی شماره ۱۵

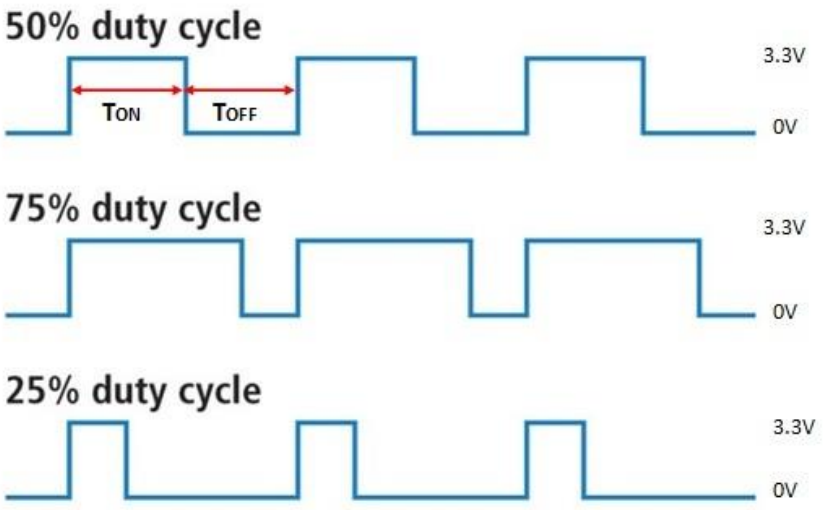
فصل ۱۳ – راه اندازی واحد PWM

مقدمه

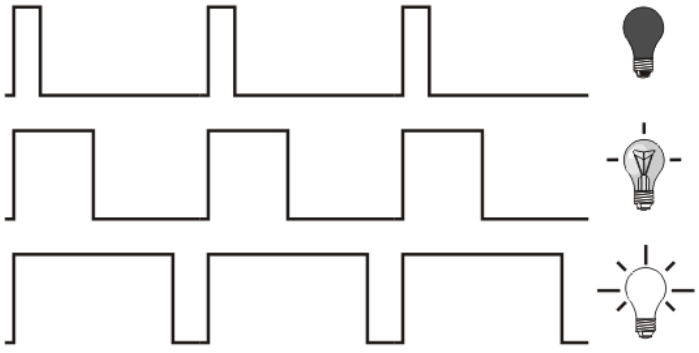
مدولاسیون عرض پالس (Pulse Width Modulation) به معنای تولید یک پالس مربعی متناوب با فرکانس ثابت اما پهنای متغیر می باشد. منظور از پهنای پالس متغیر ، دیوتی سایکل (Duty Cycle) است که به صورت فرمول زیر تعریف می شود که در آن TON زمان ۱ بودن سیگنال و TOFF زمان ۰ بودن سیگنال در یک دوره تناوب می باشد.

$$Duty\ Cycle = \frac{TON}{TON + TOFF}$$

$$Duty\ Cycle\ (\%) = \frac{TON}{TON + TOFF}\ X\ 100$$



هر چه دیوتی سایکل بیشتر باشد سیگنال توان بیشتری را به همراه دارد. بنابراین از این خاصیت میتوان در کاربردهایی نظیر کنترل سرعت موتورها ، کنترل نور لامپ و ... استفاده کرد. شکل زیر این موضوع را نشان می دهد.

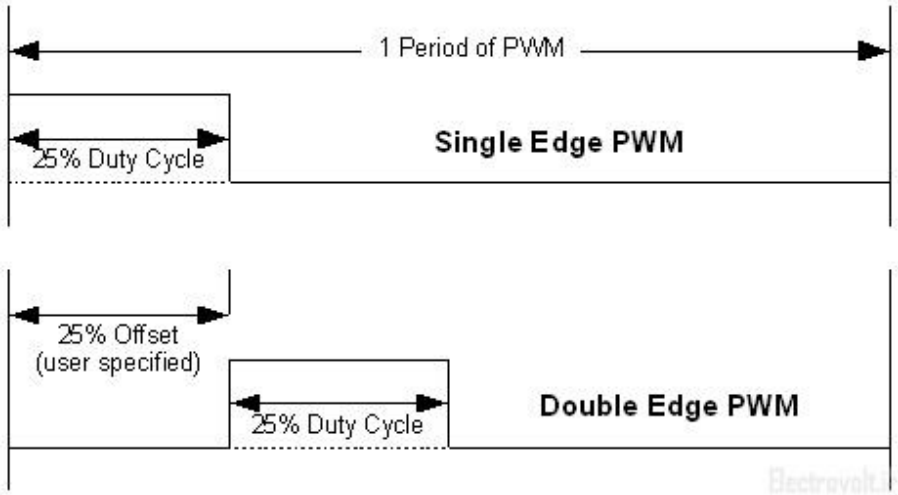


معرفی واحد PWM

در میکروکنترلرهای ARM7 این واحد مجزا از واحد تایمر/کانتر است و متشکل از یک شمارنده ۳۲ بیتی و یک پیش تقسیم کننده ۳۲ بیتی و تعدادی رجیستر می باشد. ورودی پیش تقسیم کننده Pclk و خروجی پیش تقسیم کننده کلاک اصلی این واحد می باشد. این واحد در میکروکنترلر LPC2138 دارای ۶ کانال خروجی مجزا می باشد که توانایی تولید ۶ پالس PWM تک لبه یا ۳ پالس PWM دو لبه یا ترکیبی از این دو حالت با فرکانس یکسان اما سیکل وظیفه (Duty Cycle) متفاوت را دارد. PWM در ARM7 دارای هفت رجیستر تطابق است (Match Register) که رجیستر صفر یعنی MR0 برای تعیین دوره تناوب سینگال استفاده می شود و بقیه رجیسترها تطابق یعنی MR1 تا MR6 برای تعیین طول پالس استفاده می شود. برخلاف میکروکنترلرهای AVR که دارای ۳ حالت کاری مختلف برای تولید پالس PWM بودند، در میکروکنترلرهای ARM7 تنها دو حالت کاری زیر وجود دارد:

۱. **PWM تک لبه (single edge)** : در این حالت یک لبه پایین رونده درون یک سیکل از پالس PWM قرار گرفته است.

۲. **PWM دو لبه (double edge)** : در این حالت هر دو لبه پایین رونده و بالارونده درون یک سیکل از پالس PWMقرار گرفته است.



نتیجه : حالت PWM دو لبه به علت کنترل بیشتر روی لبه ها، دارای دقت بیشتر و عملکرد قوی تر می باشد.

پایه های واحد PWM :

به تعداد ۶ پایه (کانال) خروجی واحد PWM است که از PWM1 تا PWM6 نامگذاری شده است.

Name	PWM1	PWM2	PWM3	PWM4	PWM5	PWM6
Pin	P0.0	P0.7	P0.1	P0.8	P0.21	P0.9

رجیسترهای واحدPWM

رجیستر PWMTCR (مخفف PWM Timer Control Register) : این رجیستر به منظور کنترل عملکرد واحد PWMبه کار می رود.تنظیمات بیت های این رجیستر به صورت زیر است:

بیت ۰ : زمانی که این بیت برابر ۱ قرار گیرد ، پیش تقسیم کننده و شمارنده واحد PWM فعال می شود.

بیت ۱ : زمانی که این بیت برابر ۱ قرار گیرد ، پیش تقسیم کننده و شمارنده واحد PWM ریست می گردد.

بیت ۳ : زمانی که این بیت برابر ۱ قرار گیرد ، حالت تطبیق PWM فعال می گردد. (فعالساز PWM)

رجیستر PWMPR (مخفف PWM Prescale Register) : رجیستر پیش تقسیم کننده کلاک واحد PWM می باشد که به اندازه PWMPR+1 فرکانس Pclk را تقسیم می کند.

رجیستر PWMPCR (مخفف PWM Control Register) : این رجیستر فعال کننده خروجی های واحد PWM و مشخص کننده حالت یک لبه یا دو لبه بودن عملکرد این واحد می باشد. بیت های ۲ تا ۶ از این رجیستر به ترتیب مربوط به تک لبه یا دو لبه بودن خروجی های PWM2 تا PWM6 می باشد. در صورت ۰ بودن هر بیت ، حالت تک لبه و در صورت ۱ بودن هر بیت ، حالت دو لبه انتخاب می گردد. همچنین بیت های ۹ تا ۱۴ از این رجیستر مربوط به فعال کردن هر یک از خروجی های PWM1 تا PWM6 می باشد. در صورت ۱ شدن هر یک از این بیت ها خروجی مربوط به آن در واحد PWM ، فعال می شود.

رجیستر PWMMCR (مخفف PWM Match Control Register) : این رجیستر تنظیمات مربوط به Reset ، Stopیا وقفه قرار گرفتن رجیسترهای PWMMR0تا PWMMR6 در زمان برابری (تطبیق) را مشخص می کند. در این رجیستر برای هر ۷ رجیستر PWMMR0 تاPWMMR6 ، به تعداد ۳ بیت برای تنظیم وقفه ، Resetیا Stop شدن وجود دارد (مجموعا ۲۱ بیت). بنابراین سه بیت اول این رجیستر برایMR0 ، سه بیت دوم برایMR1 ، ...سه بیت هفتم برای MR6 می باشد.

رجیسترهای PWMMR0 تا PWMMR6 (مخفف PWM Match Register) : کلیه این رجیسترها به منظور تطابق هستند به طوری که رجیستر PWMMR0 برای تعیین دوره تناوب سینگال (فرکانس (PWM استفاده می شود و بقیه رجیسترها تطابق یعنی PWMMR1 تا PWMMR6 برای تعیین طول پالس (Duty Cycle) استفاده می شود.

نکته : عرض پالس خروجی کانال های ۱ تا ۶ (یعنی مقدار رجیسترهای PWMMR1 تا PWMMR6) تنها می تواند عددی بین ۰ تا مقدار رجیستر PWMMR0 را به خود بگیرد.

محاسبه فرکانس PWM : فرکانس واحد PWM همواره ثابت بوده و توسط رابطه زیر بدست می آید:

$$F_{pwm} = \frac{P_{clk}}{(PWMPR+1)(PWMMR0+1)}$$

مراحل راه اندازی واحدPWM (تک لبه)

۱. تنظیم رجیستر PINSELX متناسب با کانال خروجی PWM مورد نیاز
۲. تنظیم رجیسترهای PWMPR (پیش تقسیم کننده) و PWMMR0 (تطبیق ۰) برای فرکانس مورد نیاز
۳. تنظیم رجیستر PWMMRX (که X بین ۱ تا ۶ است) برای داشتن DutyCycle مورد نظر
۴. تنظیم چگونگی عملکرد PWM در هنگام تطبیق (و فعالسازی وقفه) در رجیسترPWMMCR
۵. فعالسازی و تنظیمات نهایی PWM در رجیستر PWMTCR وPWMPCR

توابع راه اندازی واحدPWM

1- توابع pwmx_init :

در این توابع که x عددی بین ۱ تا ۶ می باشد ، واحد PWM روی کانال x فعالسازی می گردد. این توابع به شرح زیر می باشند.

```
void pwm1_init(void)
{
  PWMPR=0;
  PWMPCR|=(1<<9);
  PWMMR0=0x000000FF;
  PWMMCR=0;
  PWMTCR=0x00000002;
  PWMTCR=0x00000009;
  PINSEL0|=(1<<1);
}
void pwm2_init(void)
{
  PWMPR=0;
  PWMPCR|=(1<<10);
  PWMMR0=0x000000FF;
  PWMMCR=0;
  PWMTCR=0x00000002;
  PWMTCR=0x00000009;
  PINSEL0|=(1<<15);
}
void pwm3_init(void)
{
  PWMPR=0;
  PWMPCR|=(1<<11);
  PWMMR0=0x000000FF;
  PWMMCR=0;
  PWMTCR=0x00000002;
  PWMTCR=0x00000009;
```

```
  PINSEL0|=(1<<3);
}
void pwm4_init(void)
{
  PWMPR=0;
  PWMPCR|=(1<<12);
  PWMMR0=0x000000FF;
  PWMMCR=0;
  PWMTCR=0x00000002;
  PWMTCR=0x00000009;
  PINSEL0|=(1<<17);
}
void pwm5_init(void)
{
  PWMPR=0;
  PWMPCR|=(1<<13);
  PWMMR0=0x000000FF;
  PWMMCR=0;
  PWMTCR=0x00000002;
  PWMTCR=0x00000009;
  PINSEL1|=(1<<10);
}
void pwm6_init(void)
{
  PWMPR=0;
  PWMPCR|=(1<<14);
  PWMMR0=0x000000FF;
  PWMMCR=0;
```

```
 PWMTCR=0x00000002;
 PWMTCR=0x00000009;
 PINSEL0|=(1<<19);
}
```

توضیح برنامه : همانطور که در هر تابع مشاهده می کنید ، تنها تنظیم رجیستر PWMPCR و PINSEL برای هر کانال PWMXمتفاوت است. بقیه توابع مربوط به مراحل راه اندازی گفته شده می باشند.

2- توابع pwmx_out :

این توابع رجیسترهای لچ و تطبیق مربوط به هر کانال را مقداردهی می کنند. ورودی این تابع عددی بین ۰ تا PWMMR0می تواند باشد.

```
void pwm1_out(unsigned int value)
{
 PWMMR1=value;
 PWMLER=2;
 }
void pwm2_out(unsigned int value)
{
 PWMMR2=value;
 PWMLER=4;
 }
void pwm3_out(unsigned int value)
{
 PWMMR3=value;
 PWMLER=8;
 }
```

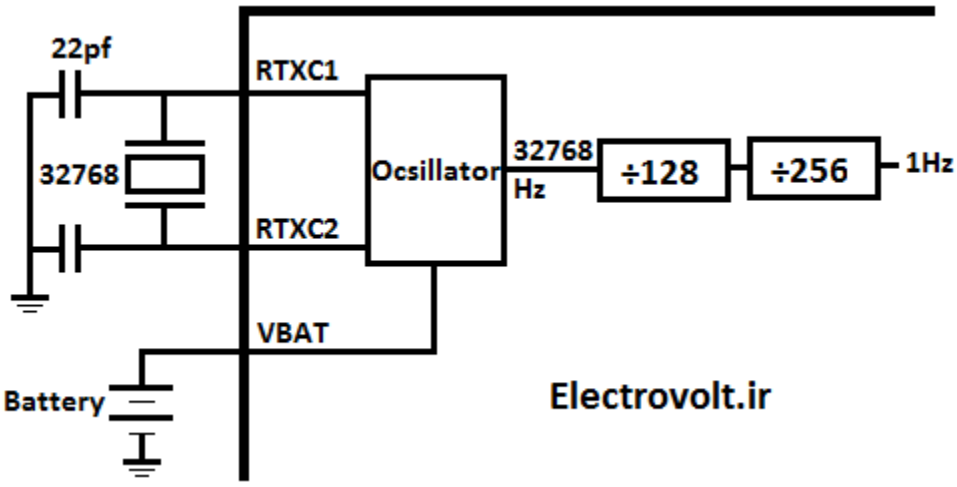
```
void pwm4_out(unsigned int value)
{
 PWMMR4=value;
 PWMLER=16;
 }
void pwm5_out(unsigned int value)
{
 PWMMR5=value;
 PWMLER=32;
 }
void pwm6_out(unsigned int value)
{
 PWMMR6=value;
 PWMLER=64;
 }
```

مثال عملی شماره ۱۶ : با راه اندازی واحد PWM به صورت تک لبه روی یکی از پایه های خروجی PWM در میکروکنترلر LPC2138 برنامه ای بنویسید که PWM متغیر بین ۰ تا ۱۰۰ درصد را ایجاد کند.

لینک دانلود سورس مثال عملی شماره ۱۶

فصل ۱۴ – راه اندازی واحدRTC

این واحد یک ساعت و تقویم دقیق را درون میکرو راه اندازی می کند که توانایی محاسبه گذشت ثانیه ، دقیقه ، ساعت ، روز ، هفته ، ماه و سال را دارد. همچنین این واحد قابلیت تنظیم آلارم را نیز دارد. برای استفاده از این واحد یک کریستال ۳۲۷۶۸ هرتز به پایه های RTCX1 و RTCX2 متصل شده و پایه تغذیه واحد RTC یعنی VBAT نیز به ولتاژ ۱,۸ تا ۳,۳ ولت متصل می گردد. در شکل زیر نحوه اتصال این واحد و نحوه تولید یک ثانیه دقیق را مشاهده می کنید.



رجیسترهای واحد RTC

رجیستر کنترل کلاک CCR (مخفف Clock Control Register)

بیت 0 و 1 این رجیستر تنظیمات اصلی واحد RTC را مشخص می کند. به طوری که با 1 شدن بیت 0 ام آن کلاک واحد RTC فعال می گردد (ClkEN) و با 1 شدن بیت 1 ام این رجیستر واحد RTC ریست می شود و تا زمانی که این بیت 1 باشد ریست باقی خواهد ماند.

بیت 4 ام این رجیستر منبع کلاک واحد RTC را مشخص می کند به طوری که اگر این بیت 0 باشد کلاک واحد RTC از کلاک سیستم تامین می شود و اگر این بیت 1 باشد کلاک واحد RTC از کریستال خارجی و اسیلاتور RTC تامین می گردد.

نکته : در این واحد قابلیتی فراهم شده است که با تنظیم رجیسترهای PREINT و PREFRAC بتوان از فرکانس اصلی سیستم برای واحد RTC نیز استفاده کرد. در صورت استفاده از کریستال ساعت خارجی به آن رجیسترها کاری نداریم.

رجیسترهای تنظیم ساعت و تاریخ کنونی

نام رجیستر	وظیفه رجیستر	محدوده مقدار
SEC	ثانیه	0-59
MIN	دقیقه	0-59
HOUR	ساعت	0-23
DOM	روز از ماه	1-31
DOW	روز از هفته	0-6
DOY	روز از سال	1-365(366)
MONTH	ماه	1-12
YEAR	سال	0-4095

رجیسترهای تنظیم آلارم در یک زمان خاص

نام رجیستر	وظیفه رجیستر	محدوده مقدار
ALSEC	ثانیه	0-59
ALMIN	دقیقه	0-59
ALHOUR	ساعت	0-23
ALDOM	روز از ماه	1-31
ALDOW	روز از هفته	0-6
ALDOY	روز از سال	1-365(366)
ALMON	ماه	1-12
ALYEAR	سال	0-4095

رجیستر فعالسازی وقفه افزایشی CIIR (مخفف Counter Increment Interrupt Register)

این رجیستر قابلیت تولید وقفه در هر بار افزایش یکی از مقادیر زمانی را جدول زیر را دارد. با یک شدن بیت مربوطه وقفه روی آن فعال می شود و با صفر شدن وقفه غیر فعال می باشد.

شماره بیت	نام بیت	وظیفه بیت
0	IMSEC	فعالسازی وقفه در هر ثانیه
1	IMMIN	فعالسازی وقفه در هر دقیقه
2	IMHOUR	فعالسازی وقفه در هر ساعت
3	IMDOM	فعالسازی وقفه در هر روز از ماه
4	IMDOW	فعالسازی وقفه در هر روز از هفته
5	IMDOY	فعالسازی وقفه در هر روز از سال
6	IMMON	فعالسازی وقفه در هر ماه
7	IMYEAR	فعالسازی وقفه در هر سال

رجیستر تعیین مکان وقفه ILR (مخفف Interrupt Location Register)

این رجیستر دارای دو بیت برای تنظیمات اینکه کدام بلاک وقفه را تولید می کند ، می باشد. بیت 0 ام این رجیستر زمانی که وقفه افزایشی در رجیستر CIIR رخ دهد برابر 1 می شود. بیت 1 ام این رجیستر زمانی که رجیسترهای آلارم وقفه تولید کنند برابر 1 می شود. نوشتن 1 در هر یک از این بیت ها باعث پاک شدن بیت مورد نظر می شود.

مراحل راه اندازی RTC

۱. **تنظیم رجیسترهای ILR و CCR**
۲. **تنظیم وقفه در واحد VIC**
۳. **تنظیم وقفه در رجیستر CIIR**
۴. **تنظیم ساعت و تاریخ (در صورت تنظیم نبودن اولیه)**

تابع راه اندازی واحد RTC

```
void rtc_init (void)

{

  ILR=1;
```

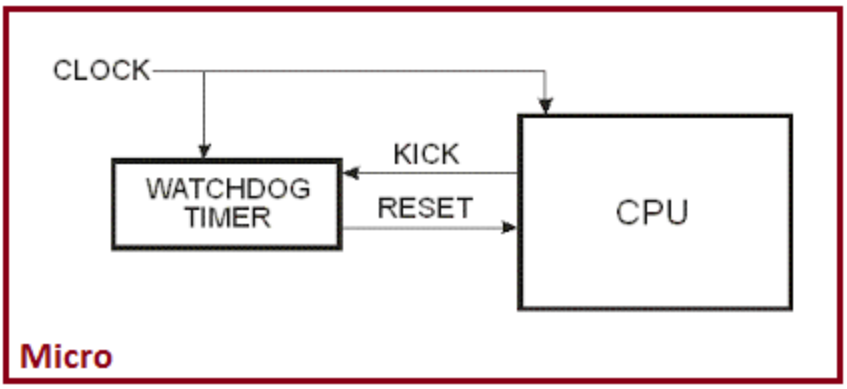
```
 CCR=0x02;
 CIIR=0x01;
 CCR=0x11;
}
```

مثال عملی شماره ۱۷ : با راه اندازی واحد RTC در میکروکنترلر LPC2138 و اتصال یک LCD کاراکتری ۲ در ۱۶ به آن برنامه ای بنویسید که ساعت و تقویم را روی LCD نمایش دهد.

لینک دانلود سورس مثال عملی شماره ۱۷

فصل ۱۵ – راه اندازی تایمر سگ نگهبان Whatchdog

سگ نگهبان نام یک نوع تایمر در میکروکنترلرها می باشد که وظیفه ریست کردن میکرو در هنگام هنگ یا توقف برنامه را دارد. در میکروکنترلرهای ARM7 نیز این قابلیت وجود دارد که تایمر سگ نگهبان بین ۱۶,۱ میلی ثانیه تا ۲,۱ ثانیه تنظیم شود. سپس در صورتی که در این مدت تنظیم شده ، تایمر سگ نگهبان ریست نشود ، باعث سر ریز شدن تایمر می شود ، و در نتیجه تایمر به صورت خودکار میکرو را ریست می کند. شکل زیر بلوک دیاگرام این واحد را نشان می دهد.



رجیسترهای واحد Watchdog

رجیستر تنظیم حالت عملکرد WDMOD (مخفف Watchdog Mode Register)

در صورت 1 بودن بیت 0 ام این رجیستر ، واحد تایمر سگ نگهبان فعال می شود.

در صورت 1 بودن بیت 1 ام این رجیستر ، قابلیت ریست شدن واحد Watchdog فعال می گردد.

بیت 2 ام این رجیستر مربوط به ریست کننده شمارنده تایمر می باشد و در طول برنامه مداوم باید صفر شود.

رجیستر تنظیم زمان WDTC (مخفف Watchdog Timer Constant Register)

درون این رجیستر یک مقدار ۳۲ بیتی قرار می گیرد که زمان سر ریز شدن تایمر سگ نگهبان را تنظیم می کند. این زمان بر حسب ثانیه از رابطه زیر بدست می آید:

$$WD_{time} = \frac{4 \times T_{WDTC}}{P_{CLK}}$$

که در آن T_{WDTC} مقدار ۳۲ بیتی رجیستر WDTC است که حداقل میتواند FF باشد.

رجیستر اعمال تنظیمات WDFEED (مخفف Watchdog Feed Register)

هر تغییری که در رجیسترهای واحد سگ نگهبان صورت می گیرید با اعمال دو خط زیر به واحد اعمال می شود :

```
WDFEED=0xAA;
WDFEED=0x55;
```

رجیستر مقدار تایمر WDTV (مخفف Watchdog Timer Value Register)

این رجیستر مقدار کنونی شمارنده تایمر سگ نگهبان را نشان می دهد که فقط میتوان این مقدار ۳۲ بیتی را خواند.

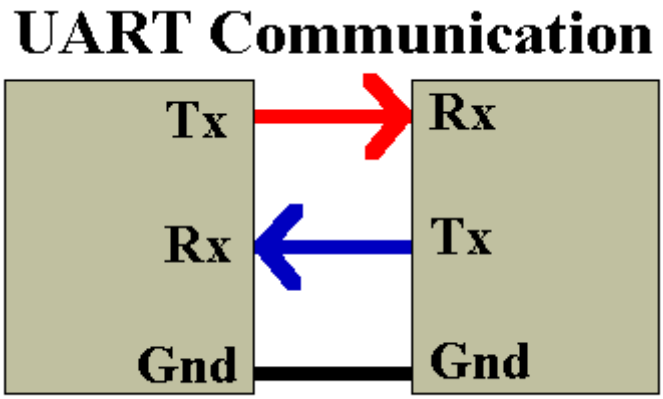
مثال عملی شماره ۱۸ : با راه اندازی واحد Watchdog در میکروکنترلر LPC2138 و اتصال یک LCD کاراکتری ۲ در ۱۶ به آن برنامه ای بنویسید که عددی روی LCD شمارش شده و در حین شمارش توسط واحد سگ نگهبان ریست شود.

لینک دانلود سورس مثال عملی شماره ۱۸

فصل ۱۶ – راه اندازی واحد UART

مقدمه

یکی از مهمترین ارتباطات سریال در میکروکنترلرهای ARM7 واحد UART (مخفف Universal Asynchronous Reciever and Transmitter به معنای فرستنده و گیرنده آسنکرون جهانی) می باشد. هر ارتباط UART میتواند به یکی از دو صورت Half-Duplex (نیم دوطرفه) یا Full-Duplex (تمام دوطرفه) باشد. در این ارتباط حداقل سه اتصال RX (Receive Data) و TX (Transmitt Data) و زمین (GND) وجود دارد. شکل زیر نحوه سیم بندی این ارتباط به صورت تمام دوطرفه را نشان می دهد.



راه اندازی واحد UART در LPC2138

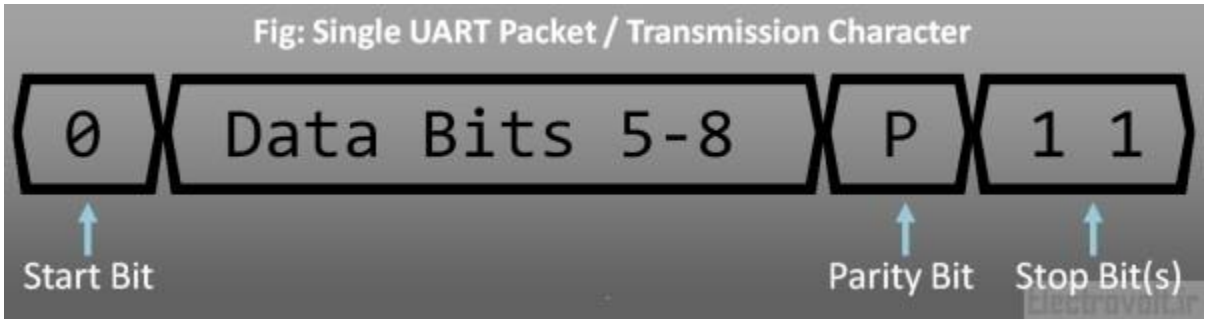
این واحد ارتباط سریال توسط پروتکل UART را فراهم می آورد. در این میکروکنترلر دو واحد UART0 و UART1 وجود دارد. برای UART0 دو پایه RXD0 و TXD0 وجود دارد. از UART0 میتوان برای پروگرام کردن میکرو به صورت بوت سریال نیز استفاده کرد. اما واحد UART1 علاوه بر پایه های RXD1 و TXD1 دارای پایه های دیگری همچون RTS1 ، CTS1 ، DSR1 ، DTR1و RI1 می باشد که برای کنترل جریان در شبکه ها (Flow Control) و ارتباط با تجهیزات دارای ترمینال همچون پرینترها و مودم ها به کار گرفته می شود. در جدول زیر پایه های اصلی این دو واحد و شماره پورت آنها را مشاهده می کنید.

RXD	TXD	
P0.1	P0.0	UART0
P0.9	P0.8	UART1

رجیسترهای واحدUART

رجیستر UxLCR (مخفف UART Line Control Register)

یک رجیستر ۸ بیتی می باشد که تنظیمات دیتای ارسالی را مشخص می کند. دیتای ارسالی با عنوان فریم (Frame) شناخته می شود که در شکل زیر آن را مشاهده می کنید.



بیت ۰ و ۱ : این دو بیت با هم به صورت زیر طول دیتای ارسالی را مشخص می کند:

00 : 5 bit

01 : 6 bit

10 : 7 bit

11 : 8 bit

بیت ۲ : این بیت تعداد Stop بیت ها را مشخص می کند. در حالت دیفالت این بیت ۰ بوده و تعداد Stop بیت ها ۱ می باشد. در صورتی که این بیت ۱ شود تعداد Stop بیت ها ۲ می شود.

بیت ۳ : در صورتی که این بیت ۰ باشد No parity و در صورتی که ۱ باشد یک بیت Parity در فریم ارسالی وجود خواهد داشت.

بیت ۴ و ۵ : این دو بیت تنظیم وضعیت بیت Parity در صورت وجود را به صورت زیر مشخص می کند:

00 : Odd Parity

01 : Even Parity

10 : parity=1

11: parity=0

توضیح : در حالت Odd ابتدا تعداد بیت های ۱ دیتا را می شمارد و در صورت فرد بودن Parity=1و در غیر این صورت Parity=0می شود. در حالت Even نیز ابتدا تعداد بیت های ۱ دیتا را می شمارد و در صورت زوج بودن Parity=1 و در غیر این صورت Parity=0 می شود. در دو حالت بعدی بیت Parity همیشه ثابت است.

بیت ۶ : این بیت که Break Control نام دارد ، کنترل جریان نرم افزاری ایجاد می کند. به طوری که در صورت ۱ شدن این بیت ارسال متوقف شده و پایه TXD منطق ۰ می گیرد.

بیت ۷ : این بیت اجازه انجام تقسیم فرکانس در رجیسترهای UxDLL و UxDLM را می دهد .بنابراین برای تنظیم Baud Rateابتدا این بیت ۱ می شود و بعد از مقداردهی به رجیسترها باید مجددا این بیت ۰ شود.

رجیسترهای UxDLL و UxDLM (مخفف UART Divisor Latch LSB و UART Divisor Latch MSB)

این رجیسترها تقسیم کننده هستند که به منظور ساخت Baud Rate دلخواه تنظیم می شوند. هر دو این رجیسترها ۸ بیتی هستند و درون آنها عدد ۱۶ بیتی زیر قرار می گیرد:

$$X = \frac{P_{CLK}}{(16 \times BaudRate)}$$

رجیستر UxRBR (مخفف UART Receiver Buffer Register)

یک رجیستر ۸ بیتی فقط خواندنی (Read Only) می باشد که دیتای مورد نظر بعد از دریافت درون آن قرار می گیرد.

رجیستر UxTHR (مخفف UART Transmit Holding Register)

یک رجیستر ۸ بیتی فقط نوشتنی (Write Only) می باشد که دیتای مورد نظر جهت ارسال درون آن قرار می گیرد.

رجیستر UxLSR (مخفف UART Line Status Register)

این رجیستر ۸ بیتی فقط خواندنی است و از هر بیت آن به منظور خواندن وضعیت قسمت های مختلف واحد UART به شرح زیر استفاده می شود.

بیت ۰ : زمانی که این بیت ۱ شود به معنای تمام شدن دریافت کاراکتر توسط واحد UART می باشد و بعد از آن میتوان از رجیستر UxRBR دیتای وارد شده را خواند.

بیت ۵ : زمانی که این بیت ۱ شود به معنای مشغول بودن واحد UART در ارسال کاراکتر می باشد و به محض ۰ شدن آن میتوان دیتا ارسالی را درون رجیستر UxTHR ریخت.

رجیستر UxIER (مخفف UART Interrupt Enable Register)

یک رجیستر ۸ بیتی است که به منظور فعالسازی انواع وقفه ها در واحد UART استفاده می شود.

بیت 0 ام این رجیستر به منظور فعالسازی وقفه دریافت کاراکتر و بیت 1 ام این رجیستر به منظور فعالسازی وقفه ارسال کاراکتر می باشد.

مراحل راه اندازی UART0 وUART1

۱. انجام تنظیمات Frame در رجیستر UxLCR و ۱ کردن بیت ۷ آن
۲. تعیین نرخ ارسال/دریافت (Baud Rate) در رجیستر UxDLL وUxDLM
۳. باز گرداندن بیت ۷ ام رجیستر UxLCR به حالت اولیه خود یعنی 0
۴. فعال/غیر فعال کردن وقفه در واحد VIC و در رجیستر UxIER
۵. تعیین عملکرد پایه های RXD و TXD میکرو در رجیسترPINSEL

توابع راه اندازی UART0 وUART1 (بدون وقفه)

```
void uart0_init(unsigned short baud)
{
  unsigned short div;
  div = (unsigned short)(PCLK/(baud*16));
  U0LCR = 0x83;
```

```
 U0DLL = (div);
 U0DLM = (div>>8);
 U0LCR &= ~(1<<7);
 PINSEL0 = (1<<3)|1;
}
void uart1_init(unsigned short baud)
{
 unsigned short div;
 div = (unsigned short)(PCLK/(baud*16));
 U1LCR = 0x83;
 U1DLL = (div);
 U1DLM = (div>>8);
 U1LCR &= ~(1<<7);
 PINSEL0 = (1<<16)|(1<<18);
}
```

توابع ارسال و دریافت در ارتباط سریال UART (بدون وقفه)

این توابع برای ارسال یک کاراکتر یا دریافت یک کاراکتر مورد استفاده قرار می گیرند.

```
int U0Write(int data)
{
 while( (U0LSR & (1<<5))== 0);
 return (U0THR = data);
}
int U0Read(void)
{
 while( (U0LSR & (1<<0))==0);
 return U0RBR;
```

```
}
int U1Write(int data)
{
 while( (U1LSR & (1<<5))== 0);
 return (U1THR = data);
}
int U1Read(void)
{
 while( (U1LSR & (1<<0))==0);
 return U1RBR;
}
```

مثال عملی شماره ۱۹ : با راه اندازی واحد UART در میکروکنترلر LPC2138 و اتصال آن به کامپیوتر برنامه ای بنویسید که با دریافت هر کاراکتر توسط میکرو ، همان کاراکتر مجددا به سمت کامپیوتر ارسال شود.

لینک دانلود سورس مثال عملی شماره ۱۹

فعالسازی وقفه دریافت UART : برای فعال سازی وقفه دریافت کافی است ابتدا واحد VIC را برای UART شماره X تنظیم کرده و سپس بیت صفرام رجیستر UxIER را برابر 1 کنیم. همچنین یک تابع سابروتین وقفه به برنامه اضافه می شود که کافی است درون آن یک رشته با اندازه دلخواه تعریف کنیم و مقدار آن را برابر رجیستر UxRBR قرار دهیم. در نتیجه با آمدن هر وقفه ، یک کاراکتر دریافت و درون رشته مورد نظر ذخیره می شود.

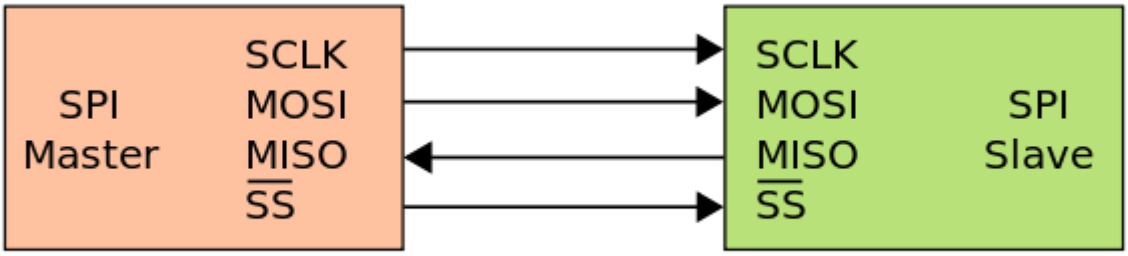
مثال عملی شماره ۲۰ : با راه اندازی واحد UART و فعالسازی وقفه دریافت در میکروکنترلر LPC2138 برنامه ای بنویسید که با دریافت هر کاراکتر توسط میکرو آن را به سمت کامپیوتر ارسال نماید.

لینک دانلود سورس مثال عملی شماره ۲۰

فصل ۱۷ – راه اندازی واحد SPI

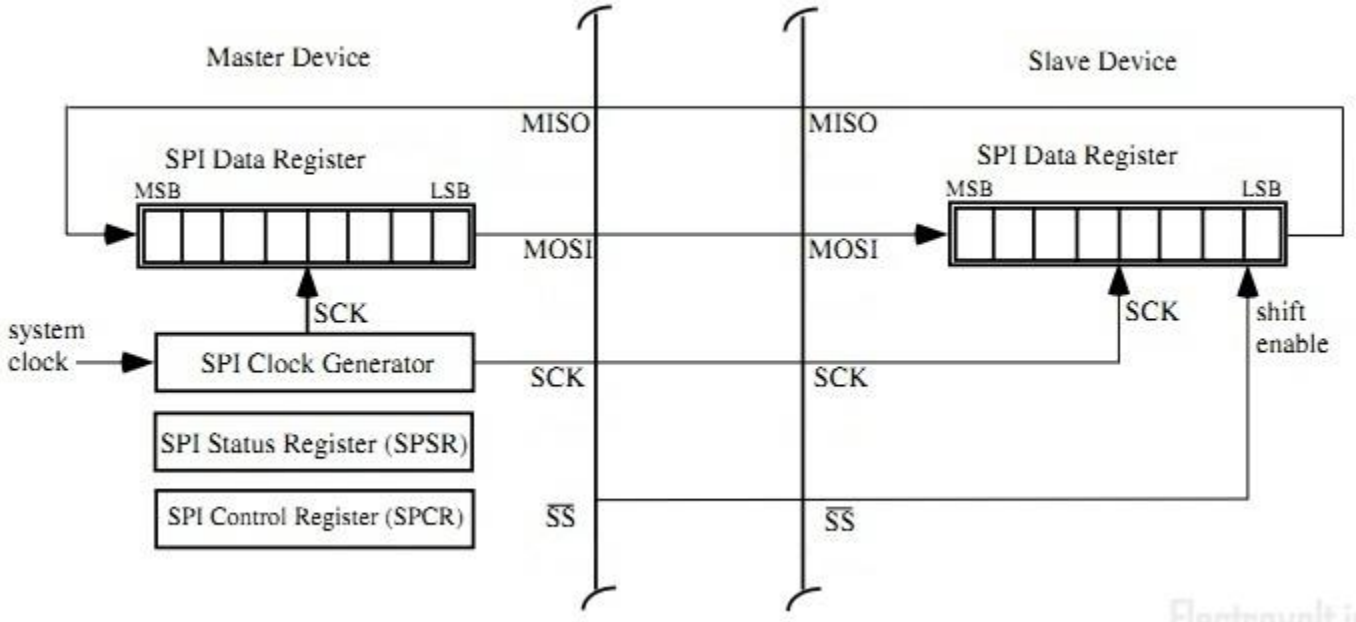
مقدمه

این واحد ارتباط سریال توسط پروتکل SPI (مخفف Serial Pheripheral Interface) را فراهم می آورد. پروتکل SPI یک ارتباط دوطرفه با سرعت بسیار بالا می باشد که بین Master و Slave در فواصل کوتاه برقرار می شود. شکل زیر نحوه سیم بندی این ارتباط را نشان می دهد.



راه اندازی واحد SPI در LPC2138

در میکروکنترلر LPC2138 دو واحد SPI0 و SSP/SPI1 وجود دارد. این رابط های SPI هر یک ارتباط دو طرفه Full Duplex، به صورت سنکرون ، با سرعت حداکثر 30Mhz و قابلیت ۸ تا ۱۶ بیت در هر ارسال و دریافتی را فراهم می آورد. شکل زیر نحوه ارتباط بین Master و Slave توسط این پروتکل را نشان می دهد.



همانطور که مشاهده می کنید ، واحد SPI دارای ۴ پایه به نام های زیر می باشد:

SCK : Serial Clock

MISO : Master in Slave out

MOSI : Master out Slave in

SSEL : slave select

در جدول زیر پایه های مربوط به این واحد در میکروکنترلر LPC2138 مشاهده می کنید.

SSEL	MOSI	MISO	SCK	
P0.7	P0.6	P0.5	P0.4	**SPI0**
P0.20	P0.19	P0.18	P0.17	**SPI1**

نکته مهم : نام رجیسترهای SPI0 و SSP/SPI1 با یکدیگر تفاوت دارد در نتیجه برنامه ای جداگانه برای راه اندازی هر یک باید نوشته شود که در این آموزش SPI0 راه اندازی می شود و برای SPI1 نیز با کمی تفاوت میتواند مورد استفاده قرار گیرد.

رجیسترهای واحدSPI0

رجیستر S0SPCR (مخفف SPI0 Serial Peripheral Control Register)

این رجیستر مربوط به تنظیمات کنترلی واحد SPI می باشد:

بیت ۲ (BitEnable) : در صورتی که این بیت ۰ باشد در هربار ارسال/دریافت ۸ بیت جابجا می شود اما وقتی این بیت ۱ باشد در هربار ارسال/دریافت بین ۸ تا ۱۱بیت جابجا می شود.

بیت ۳ (CPHA) : در صورتی که این بیت ۰ باشد دیتا در زمان رسیدن در لبه بالارونده SCK خوانده می شود و در صورتی که ۱ باشد در لبه پایین رونده SCK خوانده می شود.

بیت ۴ (CPOL) : این بیت تعیین کننده Polarity سیگنال SCK در حالتی است که دیتا رد و بدل نمی شود به طوری که وقتی این بیت ۰ باشد سیگنال SCK در حالت عادی High و در صورتی که این بیت ۱ باشد سیگنال SCK در حالت عادی Low است.

بیت ۵ (MSTR) : تعیین Master یا Slave بودن عملکرد واحد SPI است (سیگنال SCK همیشه توسط Master تولید می گردد) به طوری که در صورت ۰ بودن این بیت Slave و در صورت 1بودن این بیت Master می باشد.

بیت ۶ (LSBF) : این بیت جهت ارسال بیت های دیتا را مشخص می کند به طوری که در صورت ۰ بودن این بیت جهت ارسال از با ارزشترین بیت (MSB First) تا کم ارزشترین بیت می باشد و در صورت ۱ بودن این بیت جهت ارسال از کم ارزشترین بیت (LSB First) تا با ارزشترین بیت می باشد.

بیت های ۸ تا ۱۱ (BITS) : زمانی که بیت دوم همین رجیستر ۱ باشد میتوان به صورت زیر مقدار دیتای جابجا شده در هر بار انتقال را مشخص کرد.

1000 : 8Bit

1001 : 9Bit

1010 : 10Bit

1011 : 11Bit

1100 : 12Bit

1101 : 13Bit

1110 : 14Bit

1111 : 15Bit

0000 : 16Bit(default)

رجیستر S0SPSR (مخفف SPI0 Serial Peripheral Status Register)

این رجیستر ۸ بیتی فقط خواندنی (Read Only) می باشد و وضعیت های زیر را نشان می دهد:

بیت ۳ (ABRT) : زمانی که این بیت ۱ شود نشان دهنده در دسترس نبودن Slave می باشد(Slave Abort)

بیت ۴ (MODF) : زمانی که این بیت ۱ شود نشان دهنده خطا در تنظیمات SxSPCR است.

بیت ۵ (ROVR) : زمانی که این بیت ۱ شود نشان دهنده سرریز شدن در هنگام دریافت (خواندن) دیتا است (Read Overrun)

بیت ۶ (WCOL) : زمانی که این بیت ۱ شود نشان دهنده تصادم در هنگام نوشتن (ارسال) دیتا است (Write Collision)

بیت ۷ (SPIF) : زمانی که این بیت ۱ شود نشان دهنده پایان یافتن صحیح ارسال می باشد. این بیت در پایان هر ارسال صحیح یکبار ۱ می شود.(SPI Transfer Complete Flag) .

رجیستر S0SPDR (مخفف SPI0 Serial Peripheral Data Register)

این رجیستر مربوط به دیتایی است که ارسال/دریافت می شود. در حالت عادی در هر مرحله ۸ بیت ارسال می شود و در صورتی که بیت ۲ رجیستر SxSPCR تنظیم شده باشد ، به تعداد تنظیم شده در بیت های ۸ تا ۱۱ رجیستر SxSPCR ارسال/دریافت خواهیم داشت. در هر کلاک SCK یک بیت جابجا می شود تا اینکه انتقال پایان یابد.

رجیستر S0SPCCR (مخفف SPI0 Clock Counter Register)

این رجیستر تنظیمات فرکانس SCK در حالت Master را مشخص می کند. اگر مقداری که درون این رجیستر ریخته می شود را X بنامیم ، X همیشه زوج و بزرگتر مساوی ۸ باید باشد. با داشتن Pclk و قرار دادن فرکانس مورد نظر Fspi مقدار Xبه صورت زیر بدست می آید:

$$X = \frac{P_{CLK}}{F_{spi}}$$

مراحل راه اندازی SPI0

۱. انجام تنظیمات S0SPCR (برای Master و Slave)

۲. انجام تنظیمات S0SPCCR (فقط برای Master)

۳. انجام تنظیمات پایه ها در رجیستر PINSEL

۴. فعال سازی Slave توسط Master (فقط برای Master)

تابع راه اندازی واحد SPI0 درMaster

در میکرو کنترلر Master در صورتی که فرض کنیم Pclk=3Mhz و Fspi=100Khz داریم:

```
void spi0_master_init(void)
{
 S0SPCCR=0x1E;//Fspi
 S0SPCR=0x20;//master8bit
 PINSEL0|=0x00005500;
 ssel_init;
 ssel_1;
}
```

تذکر : از هر کدام از پایه های میکرو میتوان به عنوان SSEL برای انتخاب Slave ها استفاده نمود. در خطوط انتهایی تابع فوق باید پایه SSEL خروجی شود و سپس برابر ۱ شود تا Slave غیر فعال گردد. برای مثال میتوان نوشت:

```
IO0DIR|=(1<<8);//ssel_init
IO0SET|=(1<<8);//ssel_1
```

تابع راه اندازی واحد SPI0 درSlave

در صورتی که یک LPC2138 دیگر به صورت Slave با Master در ارتباط باشد ، از تابع زیر برای راه اندازی آن استفاده می گردد:

```
void spi_slave_init(void)
{
 S0SPCR =0x00;//slave8bit
 PINSEL0|=0x00005500;
}
```

تابع انتقال دیتا در واحدSPI0

تابعی که یک انتقال در میکروکنترلر Master یا میکروکنترلر Slave (در صورت وجود) انجام می دهد به صورت زیر می باشد. این تابع یک ۸ بیتی را در ورودی خود دریافت کرده و در ۸ کلاک از SCK آن را بین Master و Slave جابجا می کند .در نهایت تابع ، ۸ بیت دریافت شده را به خروجی برمیگرداند.

```
unsigned char spi0_master(unsigned char data)
{
 ssel_0;
 S0SPDR = data;
 while((S0SPSR&0x80)==0);
 ssel_1;
 return S0SPDR;
}

unsigned char spi0_slave(unsigned char data)
{
 S0SPDR=data;
 while((S0SPSR&0x80)==0);
 return S0SPDR;
}
```

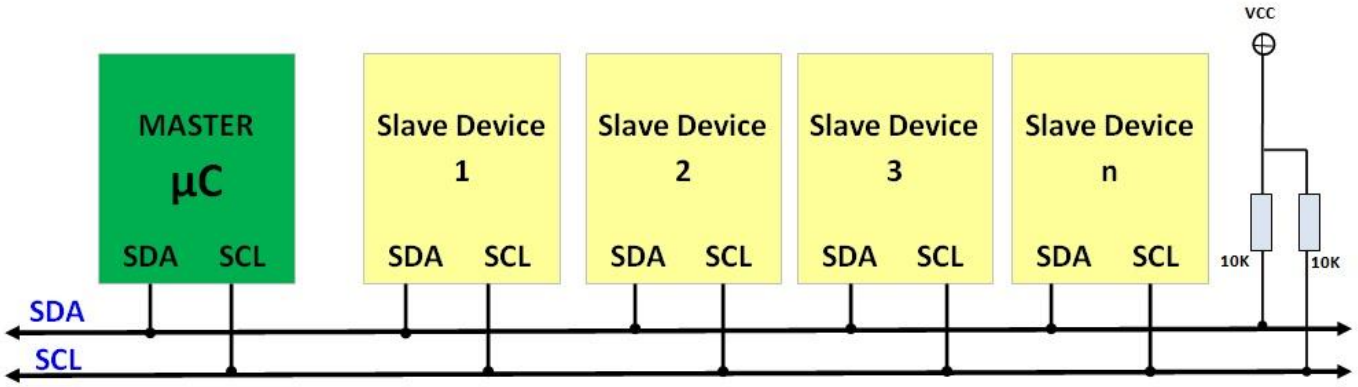
مثال عملی شماره ۲۱ : دو میکرو LPC2138 را توسط پورت SPI به یکدیگر متصل نمایید. برنامه ای بنویسید که هر یک از میکروکنترلرها به دیگری کاراکتر دلخواهی ارسال کند. میکروکنترلر دیگر کاراکتر را بررسی کرده و در صورتی که کاراکتر صحیح دریافت شده باشد یک LED را روشن کند.

لینک دانلود سورس مثال عملی شماره ۲۱

فصل ۱۸ – راه اندازی واحد I2C

مقدمه

یکی از پروتکل های ارتباطی سریال پرکاربرد I2C یا پروتکل دو سیمه می باشد. در این پروتکل از دو سیم SCL (مخفف Serial Clock) و SDA (مخفف Serial Data) استفاده می شود. این دو سیم باید با یک مقاومت 2.2K پول آپ شود. حداکثر سرعت در این پروتکل 400Khz است. در این پروتکل یک وسیله Master و حداکثر 127 وسیله Slave وجود دارد که همگی SDA ها و SCL های آنها به هم متصل است. شکل زیر نحوه ارتباط توسط این پروتکل را نشان می دهد.



Typical I²C Device Connection

راه اندازی I2C سخت افزاری در LPC2138

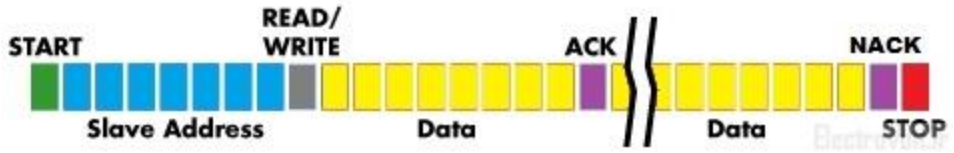
میکروکنترلر LPC2138 دارای دو واحد I2C0 و I2C1 سخت افزاری مجزا می باشد که هر یک میتواند Master یا Slaveباشد. پایه های SDA و SCL مربوط به واحد I2C را در جدول زیر مشاهده می کنید.

SDA	SCL	
P0.3	P0.2	**I2C0**
P0.14	P0.11	**I2C1**

توضیحات پروتکل I2C

قالب دیتا در پروتکل I2C دارای ۴ وضعیت مختلف است:

وضعیت Start ، وضعیت آدرس ، وضعیت دیتا و وضعیت Stop .



در وضعیت عادی باس High می باشد تا زمانی که شروع به ارسال/دریافت شود و SDA و SCLبرابر ۰ شوند (وضعیت Start) سپس ۷ بیت برای تعیین آدرس به همراه یک بیت تعیین وضعیت خواندن یا نوشتن (R/W) روی باس قرار می گیرد (وضعیت آدرس). سپس دیتای ۸ بیتی مورد نظر روی باس قرار می گیرد (وضعیت دیتا) و در نهایت باس به حالت اولیه خود باز می گردد (وضعیت Stop) در وضعیت های آدرس و دیتا یک بیت به نام ACK (مخففAcknowledge) از طرف گیرنده به فرستنده ارسال می شود که نشان دهنده وجود گیرنده و نیز صحیح دریافت شدن دیتا می باشد.

با توجه به حالت های عملکرد Master/Slave و اینکه هر یک میتواند دیتا ارسال/دریافت نماید چهار حالت زیر بوجود می آید:

۱. **حالت Master Transmitter** : زمانی که Master به Slave دیتا ارسال می کند.
۲. **حالت Master Receiver** : زمانی که Master از Slave دیتا دریافت می کند.
۳. **حالت Slave Transmitter** : زمانی که Slave به Master دیتا ارسال می کند.
۴. **حالت Slave Receiver** : زمانی که Slave از Master دیتا دریافت می کند.

رجیسترهای واحدI2C

رجیستر I2CxCONSET (مخفف I2C Control Set Register)

این رجیستر تنظیمات کلی واحد I2C را به صورت زیر مشخص می کند:

بیت ۲ (AA) : در صورت ۱ کردن این بیت یک وضعیت Acknowledge ایجاد می شود.

بیت ۳ (SI) : بیت وقفه می باشد که در صورت هر گونه تغییر در وضعیت باس I2C این بیت ۱ می شود.

بیت ۴ (STO) : در صورت ۱ کردن این بیت یک وضعیت Stop در حالت Master روی باس I2C ایجاد می شود.

بیت ۵ (STA) : در صورت ۱ کردن این بیت یک وضعیت Start در حالت Master روی باس I2C ایجاد می شود.

بیت ۶ (I2EN) : در صورت ۱ کردن این بیت واحد I2C فعال می شود.

نکته : برای غیر فعال کردن بیت های رجیستر CONSET باید از رجیستر CONCLR استفاده کرد. صفر کردن بیت ها تاثیری ندارند.

رجیستر I2CxCONCLR (مخفف I2C Control Clear Register)

بیت ۲ (AAC) : در صورت ۱ کردن این بیت ، AAدر رجیستر I2CxCONSET پاک می شود. در نتیجه وضعیت ACK پاک می شود.

بیت ۳ (SIC) : در صورت ۱ کردن این بیت ، SIدر رجیستر I2CxCONSET پاک می شود. در نتیجه وقفه غیر فعال می گردد.

بیت ۵ (STAC) : در صورت ۱ کردن این بیت ، STAدر رجیستر I2CxCONSET پاک می شود. در نتیجه وضعیت Startپاک می شود.

بیت ۶ (I2ENC) : در صورت ۱ کردن این بیت ، I2CENدر رجیستر I2CxCONSET پاک می شود. در نتیجه واحد I2Cغیر فعال می شود.

رجیستر I2CxSTAT (مخفف I2C Status Register)

بیت های ۳ تا ۷ از این رجیستر وضعیت باس I2C را مشخص می کند به طوری که برای هر یک از حالت های چهارگانه عملکرد سیستم یک جدول وجود دارد. این جدول ها را به طور کامل در صفحه ۱۷۴ تا ۱۷۷ از UM10120 مشاهده می کنید. برای مثال در حالت Master Transmitter وضعیت های زیر را داریم:

0X08: یک وضعیت Start ارسال شده است.

0X10: یک وضعیت Start تکرار شده است.

0X18: آدرس Slave و بیت R/W ارسال شده است و ACK دریافت شده است.

0X20: آدرس Slave و بیت R/W ارسال شده است اما ACK دریافت نشده است.

0X28: دیتای روی I2CxDAT ارسال شده است و ACK دریافت شده است.

0X30: دیتای روی I2CxDAT ارسال شده است اما ACK دریافت نشده است.

0X38: آدرس Slave ، بیت R/W یا دیتای روی I2CxDAT از بین رفته است.

رجیستر I2CxADR (مخفف I2C Slave Address Register)

در صورت ۱ کردن بیت ۰ این رجیستر General Call (اعلان عمومی) در حالت Slave فعال می گردد. در بیت های ۱ تا ۷ از این رجیستر آدرس Slave مورد نظر که عددی بین ۰ تا ۱۲۷ است ، قرار می گیرد.

رجیستر I2CxDAT (مخفف I2C Data Register)

دیتای ۸ بیتی جهت ارسال درون این رجیستر قرار می گیرد.

رجیستر های I2CxSCLL و I2CxSCLH (مخفف I2C SCL High و I2C SCL Low)

این رجیسترها فرکانس SCL در حالت Master را به صورت زیر مشخص می کند به طوری که عدد مورد نظر برای فرکانس SCL حداکثر 400Khz می باشد.

$$F_{SCL} = \frac{P_{CLK}}{I2CxSCLL + I2CxSCLH}$$

مراحل راه اندازی واحدI2C

۱. تنظیم رجیسترهای کنترلی I2CxCONSET و I2CxCONCLR (در Master و Slave)

۲. تنظیم سرعت SCL در رجیسترهای I2CxSCLL و I2CxSCLH (فقط در Master)

۳. تنظیم پورت پایه های SCL و SDA در رجیستر PINSEL (در Master و Slave)

۴. تشخیص حالت عملکرد مورد نیاز و قرار دادن Switch روی رجیستر I2CxSTAT (در Master و Slave)

۵. استفاده از رجیسترهای I2CxADR و I2CxDAT برای آدرس و دیتا (در Master و Slave)

توابع راه اندازی I2C در حالتMaster

```
void i2c0_master_init (void ) {
 PINSEL0|=(1<<4)|(1<<6);
 I2C0SCLL=50;
 I2C0SCLH=50;
 I2C0CONCLR=0xFF;
 I2C0CONSET=0x40;
 I2C0CONSET=0x20;
}

void i2c1_master_init (void ) {
 PINSEL0|=(3<<22)|(3<<28);
 I2C1SCLL=50;
 I2C1SCLH=50;
 I2C1CONCLR=0xFF;
 I2C1CONSET=0x40;
 I2C1CONSET=0x20;
}
```

راه اندازی I2C در حالتSlave

```
void i2c0_slave_init( void ){
PINSEL0|=(1<<4)|(1<<6);
I2C0CONCLR=0xFF;
I2C0CONSET=0x44;
I2C0ADR=0x02;
```

```
}

void i2c1_slave_init( void ){
PINSEL0|=(3<<22)|(3<<28);
I2C0CONCLR=0xFF;
I2C0CONSET=0x44;
I2C0ADR=0x02;
}
```

نمونه تابع Switch برای حالتMaster Transmitt

```
switch(I2C0STAT)
  {
  case 0x08://start condition
    I2C0CONCLR=0x02;
    I2C0DAT=0x02;
    I2C0CONCLR=0x08;
  break;

  case 0x18://ack ok
    I2C0DAT='H';//sample data
    I2C0CONCLR=0x08;
  break;

  case 0x20://not ack
    I2C0DAT=0x20;
    I2C0CONCLR=0x08;
  break;
```

```
case 0x28://data ack ok
  I2C0CONSET=0x10;
  I2C0CONCLR=0x08;
  I2C0CONCLR=0x10;
}
```

مثال عملی شماره ۲۲ : دو میکرو LPC2138 را توسط پورت I2C به یکدیگر متصل نمایید. برنامه ای بنویسید که هر یک از میکروکنترلرها به دیگری کاراکتر دلخواهی ارسال کند. میکروکنترلر دیگر کاراکتر را بررسی کرده و در صورتی که کاراکتر صحیح دریافت شده باشد یک LED را روشن کند.

لینک دانلود سورس مثال عملی شماره ۲۲

راه اندازی I2C نرم افزاری در LPC2138

همانطور که مشاهده می کنید راه اندازی I2C سخت افزاری به علت رجیسترهای زیاد و وضعیت های مختلف پیچیدگی خاص خود را دارد. از آن جایی که معمولا از میکروکنترلر در حالت Master استفاده می شود ، میتوان از I2C نرم افزاری به جای I2C سخت افزاری استفاده نمود. در I2C نرم افزاری میتوان هر دو پایه دلخواه را به عنوان SDA و SCL انتخاب کرد. با اضافه کردن هدر فایل i2c.h به برنامه میتوان با Slave توسط پروتکل I2C ارتباط برقرار کرد. با اضافه کردن این هدر فایل از توابع زیر در برنامه استفاده می شود :

1. void i2c_start(void)
2. void i2c_stop(void)
3. unsigned char i2c_write(unsigned char data)
4. unsigned char i2c_read(unsigned char ack)

مثال عملی شماره ۲۳ : با استفاده از میکروکنترلر LPC2138 و AT24C512 که یک آی سی EEPROM توسط پروتکل I2c می باشد ، برنامه ای بنویسید که یک عبارت رشته ای را ابتدا درون EEPROM ریخته و سپس خوانده و روی LCD نشان دهد.

لینک دانلود سورس مثال عملی شماره ۲۳

پایان

امیدوارم مباحث مطرح شده مورد استفاده و توجه شما قرار گرفته باشد. برای شروع به کار عملی با میکروکنترلرهای ARM و خرید قطعات مورد نیاز به فروشگاه الکترو ولت به آدرس Shop.Electrovolt.ir مراجعه نمایید. همچنین برای دانلود سورس شبیه سازی کلیه مثال های این جزوه ، به آدرس زیر مراجعه کنید. با تشکر شجاع داودی

لینک خرید و دانلود آنلاین بسته آموزش کاربردی میکروکنترلرهای ARM7