به نام خدا

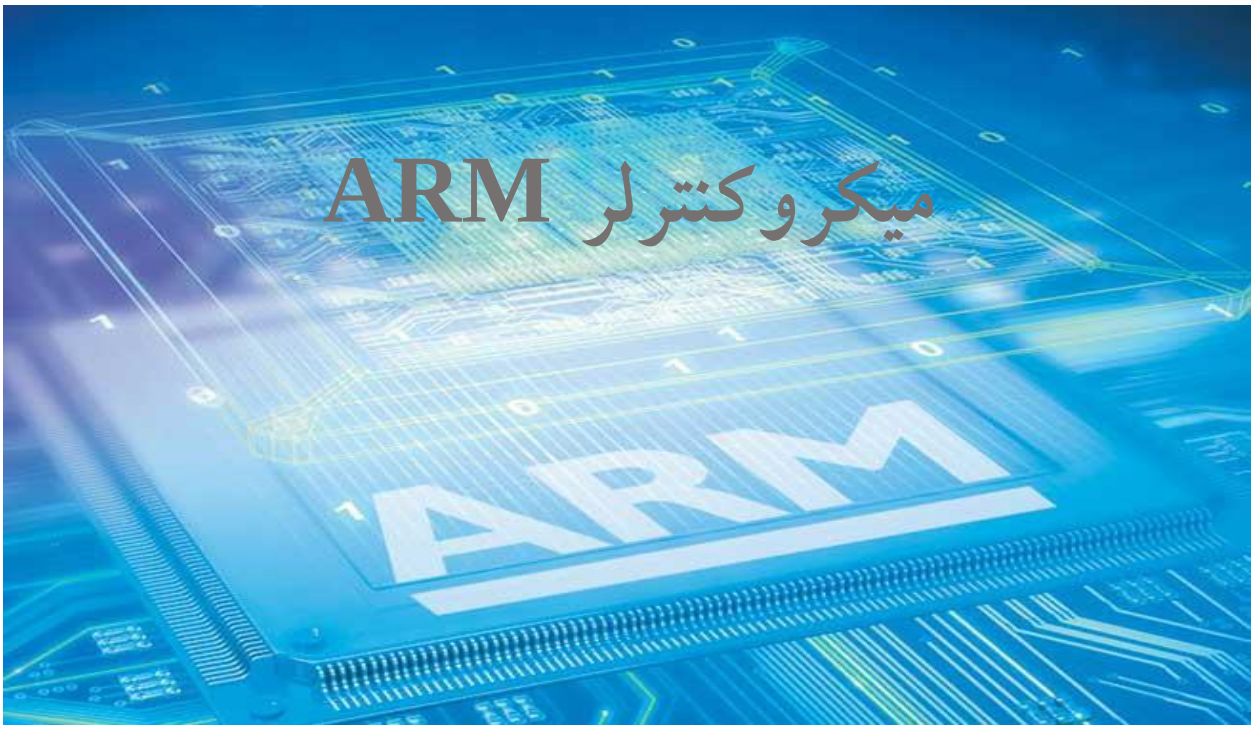
جزوه اصول میکروکامپیوتر

مدرس : مهندس علی‌اکبر شیخی

بهار ۹۹

Email address: Aliakbar.sheikhy@yahoo.com

LinkedIn address: https://www.linkedin.com/feed/?trk=404_page



فصل اول

میکروکنترلر یک مدار مجتمع یا چیپ الکترونیکی (IC) است که دارای ROM،CPU،RAM و تعدادی ورودی خروجی قابل برنامه ریزی است. میکروکنترلر در واقع یک میکروکامپیوتر هستند که برای مصارف خاصی برنامه ریزی می شوند. میکرو کنترلرها در انواع مختلف و برای مصارف مختلفی تولید می شوند.

کاربردهای میکروکنترلر

تصاویر زیر چند نمونه از کابردهای میکروکنترلرها را نشان می دهد.

بیشترین کاربرد میکروکنترلر در سیستم هایی از قبیل:

- رباتیک
- اتوماسیون خانه
- اتوماسیون های صنعتی
- درایور موتور

و ... است.

ARM چیست ؟

ARM که از معماری RISC تبعیت می‌کند، درRISC ها، طراحی براساس دستورالعمل‌های ساده و در عین حال قدرتمند صورت گرفته است! که تنها در یک سیکل کاری پردازنده و در سرعت کلاک (فرکانس کاری پردازنده) بالا اجرا می‌شوند.

برای ساخت پردازنده های ۳۲ بیتی و ٦٤ بیتی استفاده می‌شود و توسط کمپانی ARM Holdings توسعه داده شده است.

میکروکنترلرهایی که از هسته‌ی ARM در آن‌ها استفاده شود؛ میکروکنترلرهای ARM گفته می‌شود . این پردازنده ها برای کاربردهای قابل حمل (Portable) بهینه سازی شده‌اند به صورتی که مصرف توان آن‌ها بسیار کم است و می‌توان آن‌ها را توسط باتری برای مدت زیادی روشن نگه داشت به عنوان نمونه می‌توان گوشی‌های موبایل را نام برد که در آن‌ها از این هسته پردازشی استفاده می‌شود.

دلایل استفاده از ARM

به دلیل قیمت ارزان، سرعت بسیار زیاد و توان مصرفی پایین این پردازنده‌ها؛ اکثر سیستم‌های نهفته (مثل میکروکنترلرها ، موبایل و تبلت و در کل سیستم‌هایی با حجم کوچک و امکانات بالا) از این پردازنده استفاده می کنند. آمار نشان می‌دهد که از میان میکروکنترلرهای معروف دنیا ARM بیشتربن کاربرد را دارد و آینده از آن میکروکنترلرهای ۳۲ بیتی است.

معروفترین هسته پردازنده ARM، ARM7 می باشد که یکی از رایج ترین هسته های پردازشی موجود می باشد.

بعد از ARM 7 به ترتیب ARM9 و ARM10 و ARM11 قرار دارند.

انواع هسته های پردازنده سری ARM7 :

۱. ARM7TDMI رایج ترین هسته پردازنده ۳۲ بیتی با معماری RISK می باشد.
۲. ARM7TDMI-S این هسته نسخه قابل سنتز ARM7TDMI است.
۳. ARM72OT این هسته علاوه بر ویژگی های هسته های بالا داری حافظه CASH و بخش مدیریت حافظه می باشد.
٤. ARM7EJ-5 این هسته برخی از قابلیت های پیشرفته DSP را در خود دارد و برای کارهای پردازش سیگنال مناسب می باشد.

پردازنده های ARM از سیستم PIPELINE برای پردازش استفاده می‌کنند منظور از این سیستم این است که پردازنده دارای سه مد کاری برای اجرای یک دستور است :

- FETCH یا بازخوانی اطلاعات از حافظه کد
- DECODE یا رمزگشایی اطلاعات نوشته شده
- EXECUTE یا اجرای برنامه در پردازنده های قدیمی تر

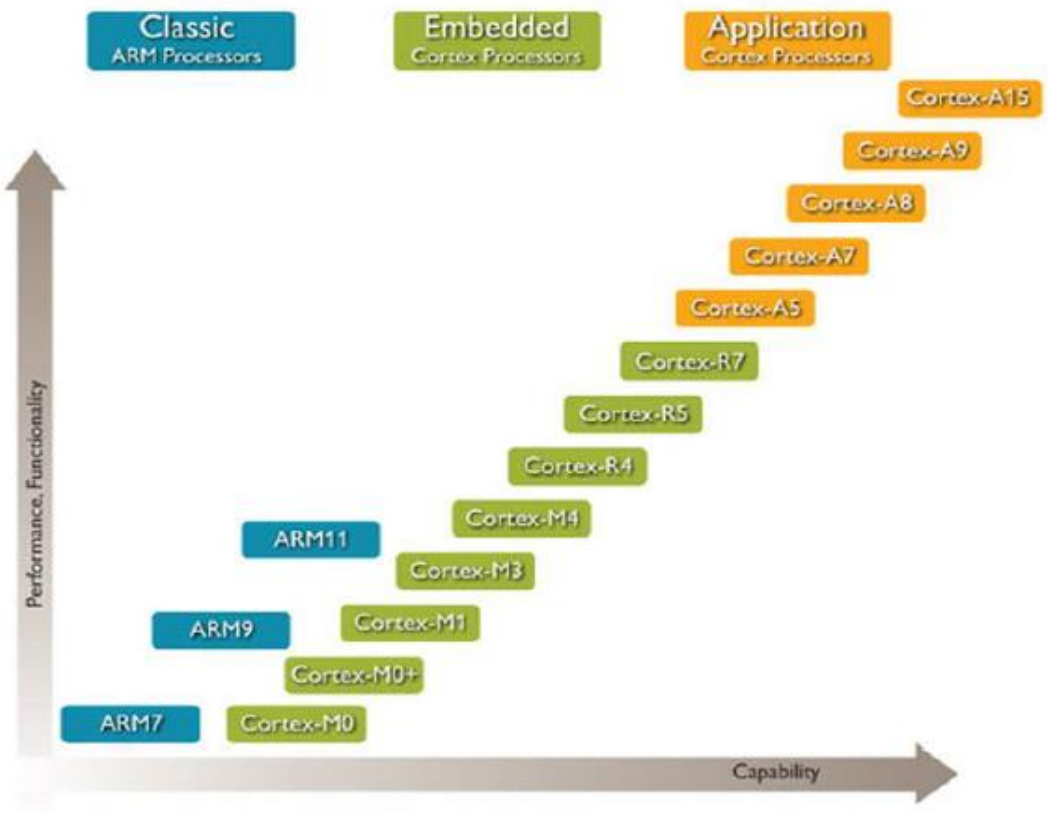
در سیکل اول دستور اول FETCH می‌شود، در سیکل دوم دستور اول DECODE می‌شود دستور دوم FETCH می‌شود. در سیکل سوم دستور اول EXECUTE دستور دوم DECODE می‌شود و دستور سوم FETCH می شود. این نوع سیستم ۳ STAGE PIPELINE است.

در پردازنده‌های ARM بالاتر مانند ARM9 سیستم پردازش۵ STAGE PIPELINE می‌باشد که عملیات خواندن و نوشتن از حافظه ها نیز جزء این عملیات قرار گرفته در ۱۰ ARM سیستم پردازش به صورت ٦ STAGE PIPELINE است.

انتخاب میکرو کنترلر :

شرکت‌های مختلفی میکروکنترلرهای بر مبنای پروسسور ARM می‌سازند مانند : Samsung ,atmel , Philips St-micro , Motorola و کمپانی های دیگر ما از میان این شرکت‌ها میکروکنترلرهای ساخت Philips رو که از تولید شرکت NXP است به دلایل زیر انتخاب کردیم :

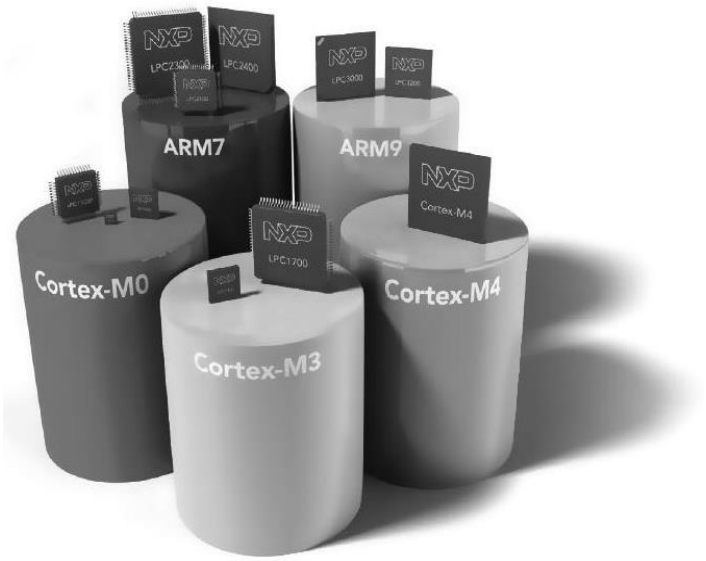
قطعات سری LPC2000 یکی از متنوع‌ترین خانواده‌های میکروکنترلرهای با هسته‌ی ARM7 هستند و قطعات این سری، در مقایسه با AT91SAM قیمت کمتری دارند. مثلاً قیمت LPC2101 حدود ۲ دلار است که این مقدار از خیلی از میکروکنترلرهای ۸ بیتی مثل(ATmega16) کمتر است. شکل زیر دسته بندی میکروهای ARM را نشان می‌دهد.



اجرای برنامه از حافظه‌ی فلش بسیار سریعتر است. بدلیل دسترسی ۱۲۸ بیتی به حافظه‌ی فلش و وجود واحد شتاب‌دهنده‌ی حافظه(MAM) ، قطعات LPC2000 می‌توانند در مُد ARM با حداکثر سرعت ٦٠ تا ٧٥ MHz به حافظه‌ی فلش دسترسی داشته باشند؛ در حالیکه که SAM7 ها با سرعتی کمتر از نصف این مقدار کد برنامه را اجرا می‌کنند. علاوه براین، در مقایسه با سایر میکروهای با هسته ی ARM7 ، فرکانس کاری میکروکنترلرهای LPC2000 نسبتاً بالاست (٦٠ تا ٧٠ مگاهرتز در LPC2000 ها در مقایسه با ٥٥ مگاهرتر در sam راه‌اندازی

Peripheralهای قطعات LPC2000 ساده‌تر و اکثر سخت‌افزارهای جنبی به شکلی طراحی شده‌اند که لازمه رجیسترهای کمتری تنظیم شوند و بسیاری از آن‌ها را می‌توانید به حالت پیش‌فرض رها کنید.

قطعات LPC دارای تعداد I/O (پایه ورودی/خروجی) بیشتری هستند. مثلاً قطعه‌ی LPC2132 که یک قطعه‌ی ٦٤ پایه است ٤٧ پایه‌ی GPIO دارد درحالیکه قطعه مشابه ٦٤ پایه ای AT91SAM7S64 ، دارای ٣٢ خط I/O است.



مستندات و نمونه برنامه‌های ارائه شده توسط NXP برای LPCها کامل‌تر و غنی از ATmel برای SAM7هاست.

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 نوشته شده برای یک تراشه را بدون نیاز به تغییر ، توسط تراشه‌های 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 ٤٠٠Mhz کار می کند، با کارایی پردازنده PENTIUM2 با فرکانس ٣٠٠ Mhz تقریبا برابر است اما مصرف توان ARM مثلا یک پنجاهم پردازنده پنتیوم است.
- قابلیت استفاده از سیستم عامل‌هایی نظیر Linux، Windows CE، Android و... که به صورت رایگان و متن باز در دسترس است.

سری های پرکاربرد میکروکنترلرهای ARM شرکتNXP

در این قسمت به معرفی خصوصیات و ویژگی‌های اصلی برخی از سری های پرکاربرد می‌پردازیم.

سری LPC21xx : این میکروکنترلرها بر اساس هسته ١٦ یا ٣٢ بیتی ARM7TDMI-S با حداکثر سرعت ٧٢ مگاهرتز، به همراه یک حافظه فلش سرعت بالا بین ٨ تا ٥١٢ 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 ١٠ بیتی با سرعت نمونه برداری ٤/٥ MSPS
- یک واحد مبدل DAC با دقت ١٠ 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، LPC213x و LPC214x 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.

فصل دوم

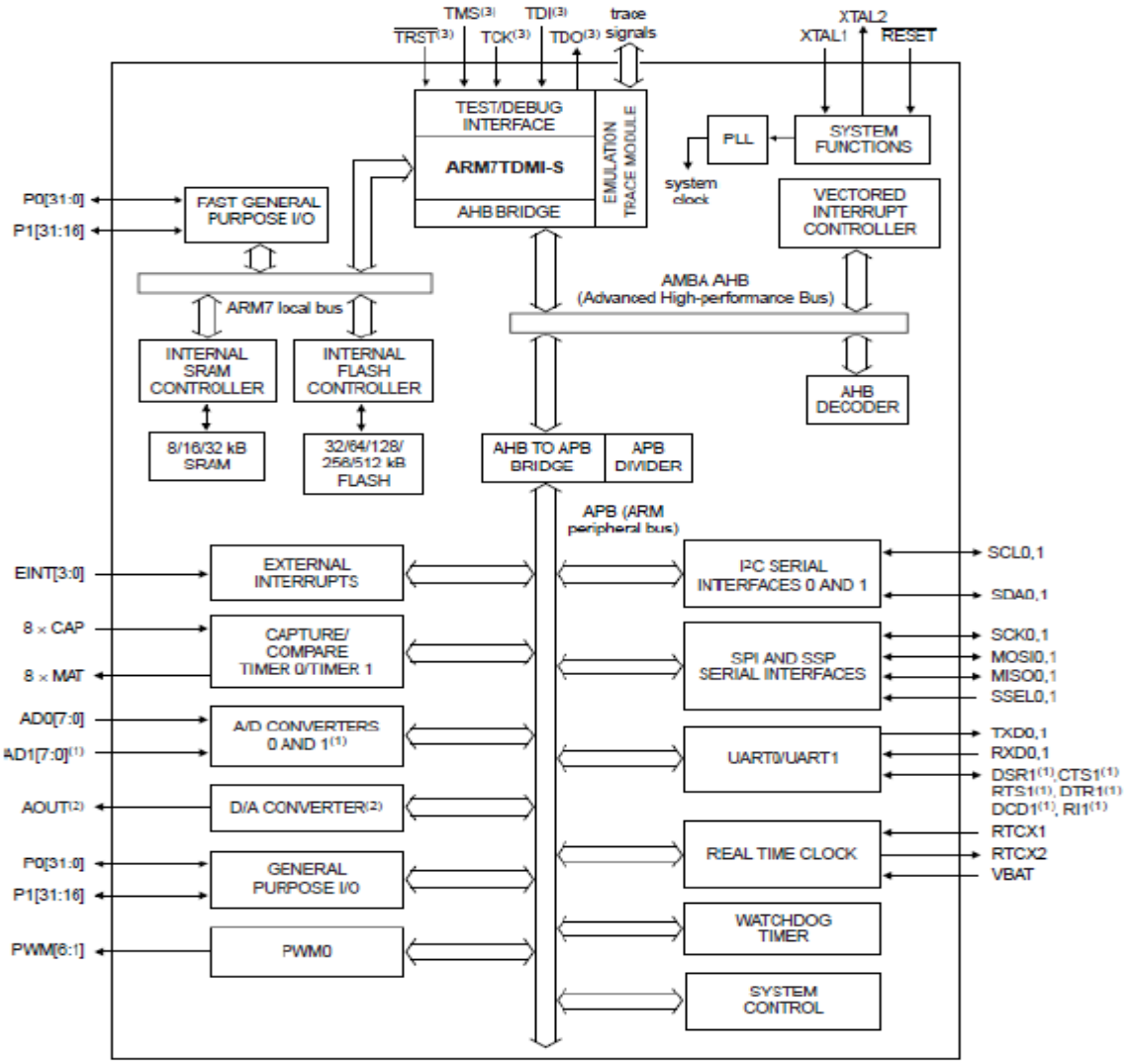
معرفی ویژگی های میکروکنترلرهای سری LPC213X

- یک میکروکنترلر ARM7TDMI به صورت ۳۲ / ۱٦ بیتی در بسته بندی کوچک ٦٤ پایه
- دارای حافظه SRAM داخلی ۸ / ۱٦ / ۳۲ کیلوبایت
- دارای حافظه FLASH داخلی ۳۲ / ٦٤ / ۱۲۸ / ۲٥٦ / ٥۱۲ کیلوبایت
- دارای امکان ذخیره سریع در حافظه به صورت ۱۲۸ بیتی در فرکانس ٦۰ 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 ٤۰۰ میلی ثانیه
- توانایی پروگرام شدن ۲٥٦ 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 ۲/٤٤ 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 ٦ خروجی
- واحد تایمرWatchdog
- واحد RTC با مصرف توان پایین
- دارای واحدهای سریال UART ، I2C، SPIوSSP
- دارای واحد کنترل وقفه برداری و با اولویت بندی قابل تنظیم و آدرس برداری
- دارای ٤۷ پایه ورودی/خروجی که قابلیت تحمل ٥ ولت را دارند
- دارای ۹ پایه به منظور وقفه خارجی به صورت حساس به لبه یا پالس
- دارای حداکثر سرعت کلاک ٦۰ 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 واحدRTC 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 می‌دهد.



همانطور که در شکل فوق مشاهده می‌کنید، 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 ASBمتصل 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 ٤ 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.

منابع تولید کلاک در LPC213X

۱. منبع کلاک خارجی : اتصال کلاک آماده به پایه XTAL1 و آزاد گذاشتن پایه XTAL2
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 Pull Up 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 میکروکنترلرهایARM7 معماری سنتی است اما در عوض یک معماری باس بهبود یافته AMBA اضافه شده است.
- منابع کلاک در AVR می‌توانست به صورت‌های 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 پیچیدگی‌های 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 علاوه‌بر استفاده از رابط JTAG برای پروگرام کردن از رابط سریال UART و به صورت‌های ISP و IAP استفاده می‌شود.
- 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 واحدهای جانبی نظیر USART و I2C در میکروکنترلرهای AVR حداکثر یک بود اما در میکروکنترلرهایARM7 تعداد آن‌ها حداکثر دو می‌باشد.
- 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 میکروکنترلرهایARM7 دو 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.

مقایسه بین میکروکنترلرهای سری LPC213X

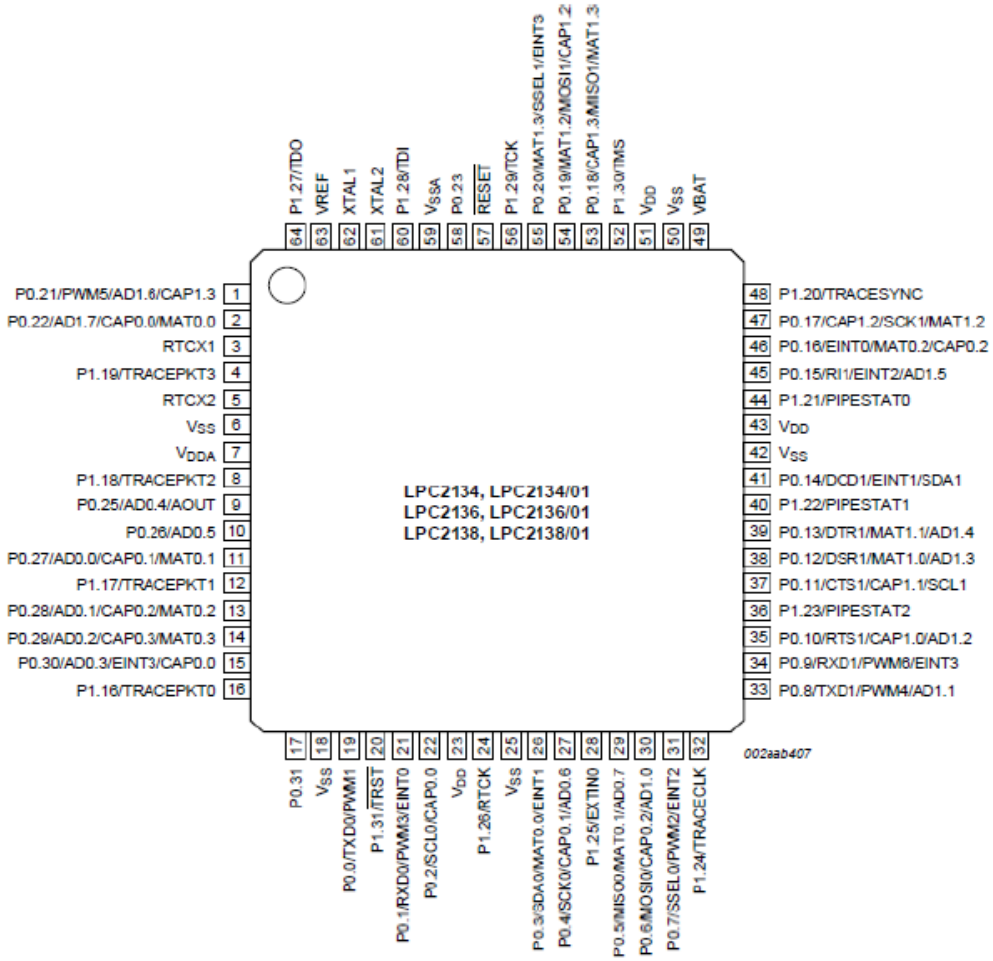
همانطور که در جدول زیر مشاهده می کنید، در این سری ۵ میکروکنترلر LPC2131, LPC2132, LPC2134, LPC2136و LPC2138 وجود دارد و ۵ میکروکنترلر دیگر به صورت LPC213X/01 نیز در کنار آن مشاهده می شود.

میکروکنترلرهایی که به آن پسوند ۰۱ اضافه شده است، نسخه جدیدتر و با قابلیت‌های بهبود یافته ی نسخه بدون پسوند می‌باشد.

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

پایه‌های میکروکنترلر LPC2138

از میان میکروکنترلرهای سری LPC213x ، میکرو دارای بیشترین قابلیت یعنی LPC2138 انتخاب می شود. این میکروکنترلر در بسته بندی TQFP و ٦٤ پایه عرضه می‌شود. پایه های این میکروکنترلر را در شکل زیر مشاهده می‌کنید.



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

چند نکته مهم :

- همانطور که مشاهده می‌کنید هر یک از پایه های حداقل یک و حداکثر چهار عملکرد مختلف دارند که به طور همزمان فقط از یک عملکرد آن استفاده می‌گردد.
- 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 آی‌سی : پایه هایی که با نام VDD وجود دارند تغذیه مثبت آی‌سی و پایه‌های VSS تغذیه منفی(زمین) آی‌سی می باشند.

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

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

کریستال واحدRTC : این آی‌سی دارای واحد RTC مجزا است که برای فعالسازی آن نیاز به اتصال کریستال ساعت به پایه‌های RTXC1 و RTXC2 می‌باشد. همچنین پایه VBAT ، تغذیه مثبت واحد RTC است که به منظور فعال بودند ائمی این واحد به باتری پشتیبان (Backup Battery) متصل می گردد.

واحدGPIO : 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.

واحدADC : 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 می‌باشد.

پایه‌های تغذیه آنالوگ آی‌سی تغذیه واحدهای (ADCوDAC) : شامل پایه‌های VREF ولتاژ مرجع ، VDDA تغذیه مثبت آنالوگ و VSSA تغذیه منفی آنالوگ می‌باشند.

واحد UART0 : پایه های RXD0 وTXD0

واحد UART1 : پایه‌هایRXD1 ، TXD1 ، CTS1 ، DCD1 ، DSR1 ، DTR1 ، RI1 ، RTS1مربوط به واحد ارتباط سریال می‌باشد. پایه هایی که اضافه بر RXD1 و TXD1 برای این واحد وجود دارد مربوط به قابلیت ارتباط دو طرفه با اطمینان بالا (Handshaking) می‌باشد.

واحد SPI0 : پایه‌هایSCK0 ، SSEL0 ، MISO0وMOSI0

واحد SPI1/SSP : پایه‌هایSCK1 ، SSEL1 ، MISO1وMOSI1 که قابلیت برقراری هر دو پروتکل ارتباط سریال SPIو SSP را دارد.

واحدI2C0 : پایه‌های SDA0 وSCL0

واحدI2C1 : پایه‌های SDA1 وSCL1

واحدTimer : تمام پایه های به نامهای CAP0.X و CAP1.X به عنوان ورودی واحد تایمر/ کانتر به کار می رود که در آن‌ها X عددی بین ۰ تا ۳ می باشد. همچنین تمام پایه‌های به نامهای MAT0.X و MAT1.X به عنوان خروجی واحد تایمر به کار می رود که در آن X عددی بین ۰ تا ۳ می‌باشد.

واحدPWM : تمام پایه‌های به نامهای PWMX که در آن X بین ۱ تا ٦ می باشد ، مربوط به واحد مجزای PWM است.

واحدJTAG : پایه‌هایی که به نامهای TRST ، TMS، TCK، TDI، TDO، TCKوجود دارند، مربوط به واحد برنامه ریزی و عیب یابی میکرو می‌باشند.

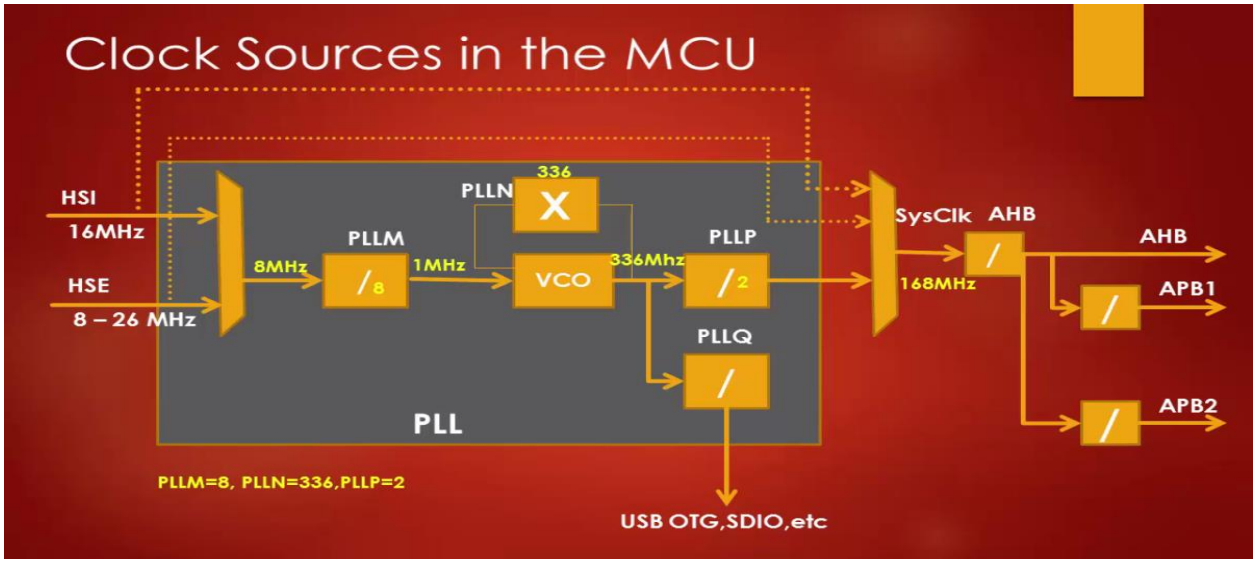
منابع Reset

با RESET شدن میکروکنترلر، تمام رجیسترهای I/O به مقدار اولیه‌شان تغییر می‌کنند و CPU شروع به اجرای دستورالعمل‌ها از بردار RESET خواهد کرد.

چهار منبع RESET وجود دارد که عبارتند از:

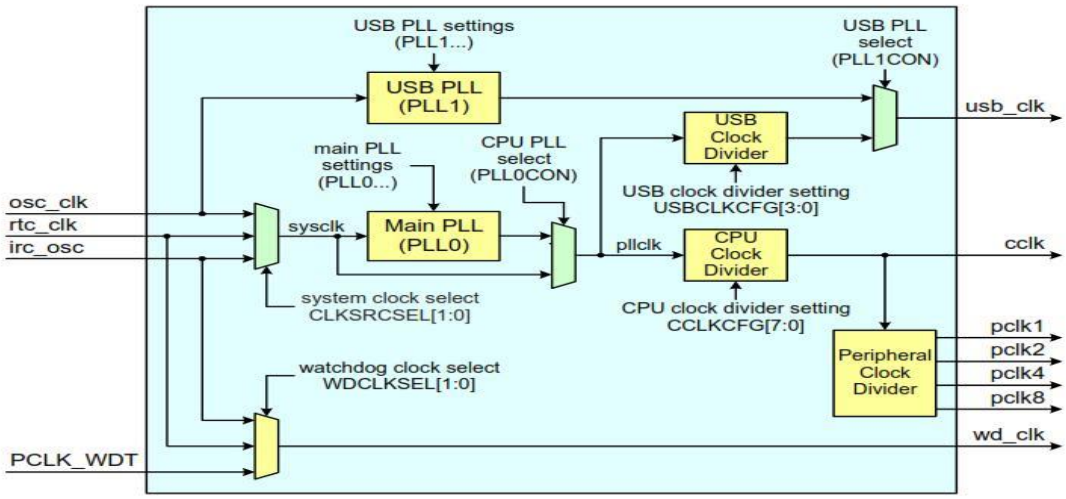
- Power-on Reset (ریست شدن میکرو هنگام روشن شدن)
- External Reset (ریست خارجی میکرو یعنی وصل کردن پایه ریست به زمین)
- Brown-out Reset (ریست شدن میکرو هنگامی‌که ولتاژ از حد آستانه بیشتر شود)
- Watchdog Reset (استفاده از تایمر سگ نگهبان و ریست شدن میکرو برای جلوگیری از هنگ کردن میکرو)

کلاک در میکروهای ARM



تمام پردازنده‌ها برای هماهنگ کردن قسمت‌های ترتیبی و ترکیبی خود نیاز به یک پالس هماهنگ کننده که اصطلاحا به آن کلاک می‌گویند دارند.

میکروکنترلرهای ARM سری lpc176x/5x دارای سه منبع کلاک مستقل شامل: اسیلاتور اصلی، اسیلاتور RC داخلی، اسیلاتور RC هستند. که هر کدام از این منابع بسته به نیاز فعال می‌شوند و می‌توان بدون استفاده از کریستال خارجی و به کمک نرم افزار از هر کدام از این منابع کلاک استفاده کرد. که در زیر منابع کلاک معرفی شده‌اند.



منابع تولید کلاک

اسیلاتور RC داخلی

همان‌طور که در بلوک دیاگرام منابع کلاک میکرو مشاهده می‌کنید، این اسیلاتور می‌تواند به عنوان منبع کلاک تایمر نگهبان یا به عنوان منبع کلاکی که PLL و سپس CPU برای عملیات راه اندازی میکرو راه می‌اندازد به کار رود. اما از این اسیلاتور در راه اندازی USB به علت نیاز به دقت زمانی بالا استفاده نمی‌شود. همچنین از واحد CAN به کمک این اسیلاتور در صورتی که نرخ ارسال این واحد بیشتر از Kbit/s ۱۰۰ باشد، استفاده می‌شود.

نکته : این منبع کلاک ٤ مگاهرتز هست و زیاد به درد کار ما نمی‌خورد چون دقت کافی رو ندارد.

اسیلاتور اصلی

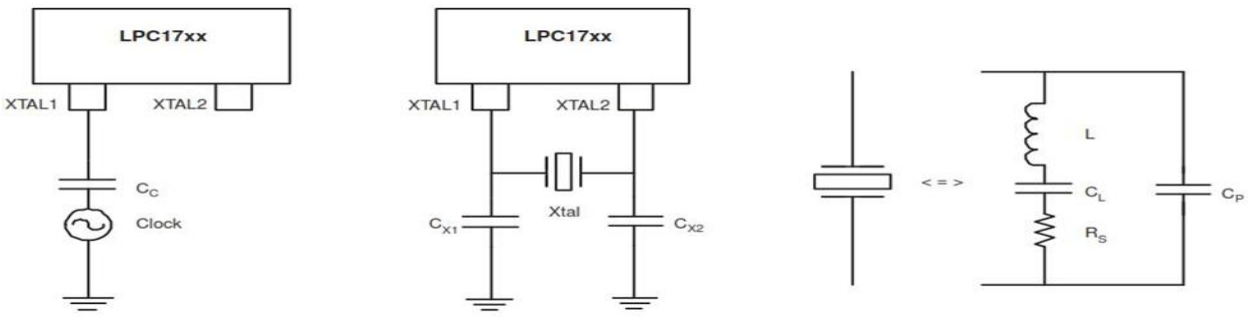
این اسیلاتور اصلی برای راه اندازی CPU میکرو می‌باشد که می‌توان از PLL در این مد استفاده کرد یا نکرد. اسیلاتور اصلی دارای فرکانسی بین ۱ تا ۲۵ مگاهرتز می‌باشد که به کمک PLL این فرکانس قابل افزایش می باشد. این اسیلاتور می‌تواند در دو مد به کار گرفته شود : مد slave و مد اسیلاتوری در مد slave مانند شکل زیر منبع کلاک توسط خازنی با مقدار مشخص باید سری شود و در مد اسیلاتوری کریستال یا منبع کلاک بین دو پین XTALx قرار گرفته و خازن‌هایی به این پایه‌ها مانند شکل زیر متصل می‌شوند. که برای فرکانس‌های مختلف مقادیر خازن ها متفاوت است.

نکته : این منبع کلاک یک کریستال خارج از میکرو هست که معمولا ۱۲ مگاهرتز دارد(البته اگه هدر بورد دارید)

اسیلاتور RTC

این اسیلاتور دارای فرکانسی از ۱ کیلوهرتز تا مقدار نامی فرکانس RTC است که معمولا از این اسیلاتور برای استفاده PLL و CPU و همچنین تایمر نگهبان watchdog استفاده می شود. اما باید به این نکته توجه شود که زمانی که خروجی PLL برای کنترل USB استفاده می‌کنیم از این اسیلاتور برای منبع کلاک استفاده نشود.

نکته : این منبع کلاک مخصوص بلوک RTC است و فرکانسش khz ۳۲ است که خیلی کمه و به کار ما نمی‌آید.



حال برای استفاده از این رجیستر ما فقط به دو بیت اول یعنی بیت های ۰ و ۱ احتیاج داریم! برای دسترسی به این رجیستر می‌توانید از کد زیر استفاده کنید.

```
LPC_SC->CLKSRCSEL=0X00000000;
```

شما باید به جای 0X00 عدد خودتان رو قرار بدهید که آنرا در زیر آورده‌ایم!

- برای استفاده از منبع کلاک داخلی بیت 00 رو قرار می‌دهیم! (باینری)
- برای استفاده از منبع کلاک اصلی (کریستال) بیت 01 رو قرار می‌دهیم.
- برای استفاده از RTC بیت 10 رو قرار می‌دهیم.

مثلا شما می‌خواهید از منبع کلاک خارجی استفاده کنید تکه کد مربوطه به شکل زیر است!

```
LPC_SC->CLKSRCSEL=0X1;
```

راه اندازی منابع کلاک با استفاده از رجیستر

برای انتخاب هر کدام از این اسیلاتورها با استفاده از رجیستر CLKSRCSEL می‌توان آن‌ها را انتخاب کرد. رجیستر به صورت زیر می‌باشد.

Bit number	Symbol	value	Description
۱:۰	CLKSRC	۰۰	RC oscillator as PLL source
		۰۱	Main oscillator
		۱۰	RTC oscillator as PLL
		۱۱	Reserved
۳۱:۲	–	۰	Reserved

منابع توزیع کلاک

تمامی امکانات جانبی میکرو از کلاک استفاده می‌کنند، مانند کلاک CPU، کلاک I/O ورودی خروجی، کلاک ADC، کلاک TIMER/COUNTER، کلاک سریال، که کلاک CPU همیشه فعال است.

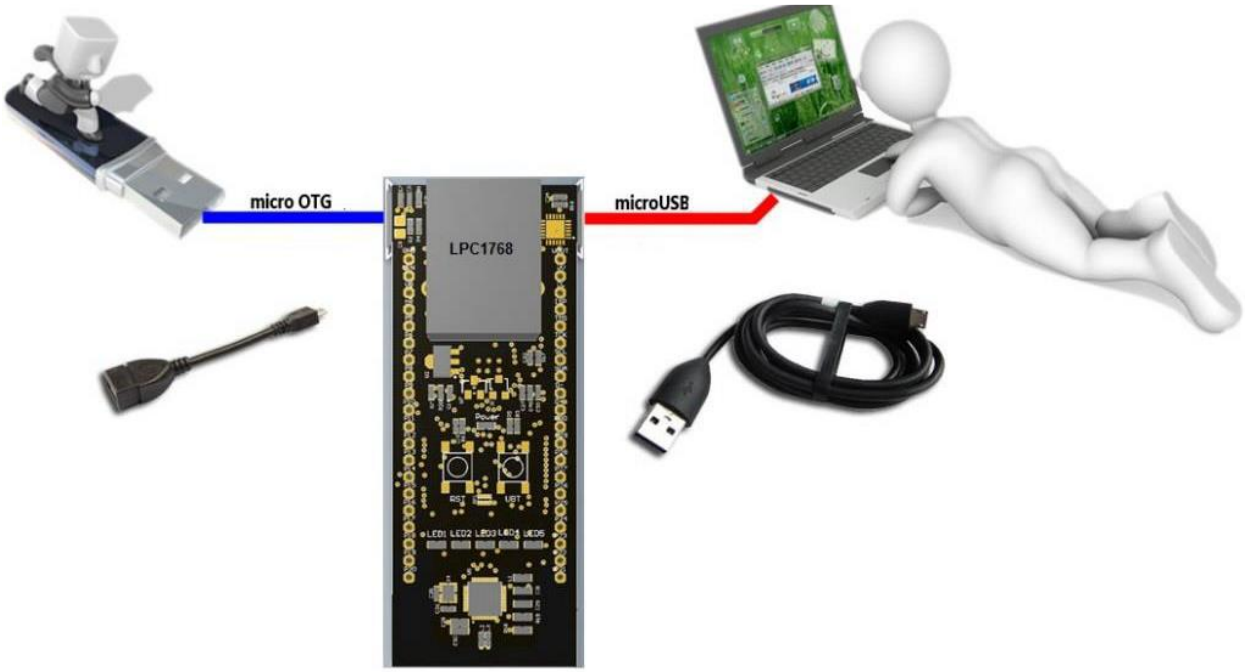
فصل سوم

انواع روش‌های پروگرام کردن میکروکنترلرهای ARM

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.

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 می‌کنید که قابلیت عملکرد در ولتاژ ۳/۳ ولت را دارد.

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 می‌باشند مانند (LPC1768) را پروگرام نمود. به این صورت که پس از اتصال پایه های +D و -D به پایه‌های مربوطه به میکرو ، به صورت اتوماتیک میکرو شبیه یک فلش توسط کامپیوتر شناسایی می‌شود و می‌توان فایل Hex برنامه دلخواه را مستقیم داخل آن Copy نمود.



فصل چهارم

کامپایلرها و مفسرهای ARM

برای میکروکنترلرهای ARM کامپایلرها و مفسرهای مختلفی ارائه شده است. زبان برنامه‌نویسی 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 می‌کنید.

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

شروع به کار با نرم افزار KEIL و Proteus

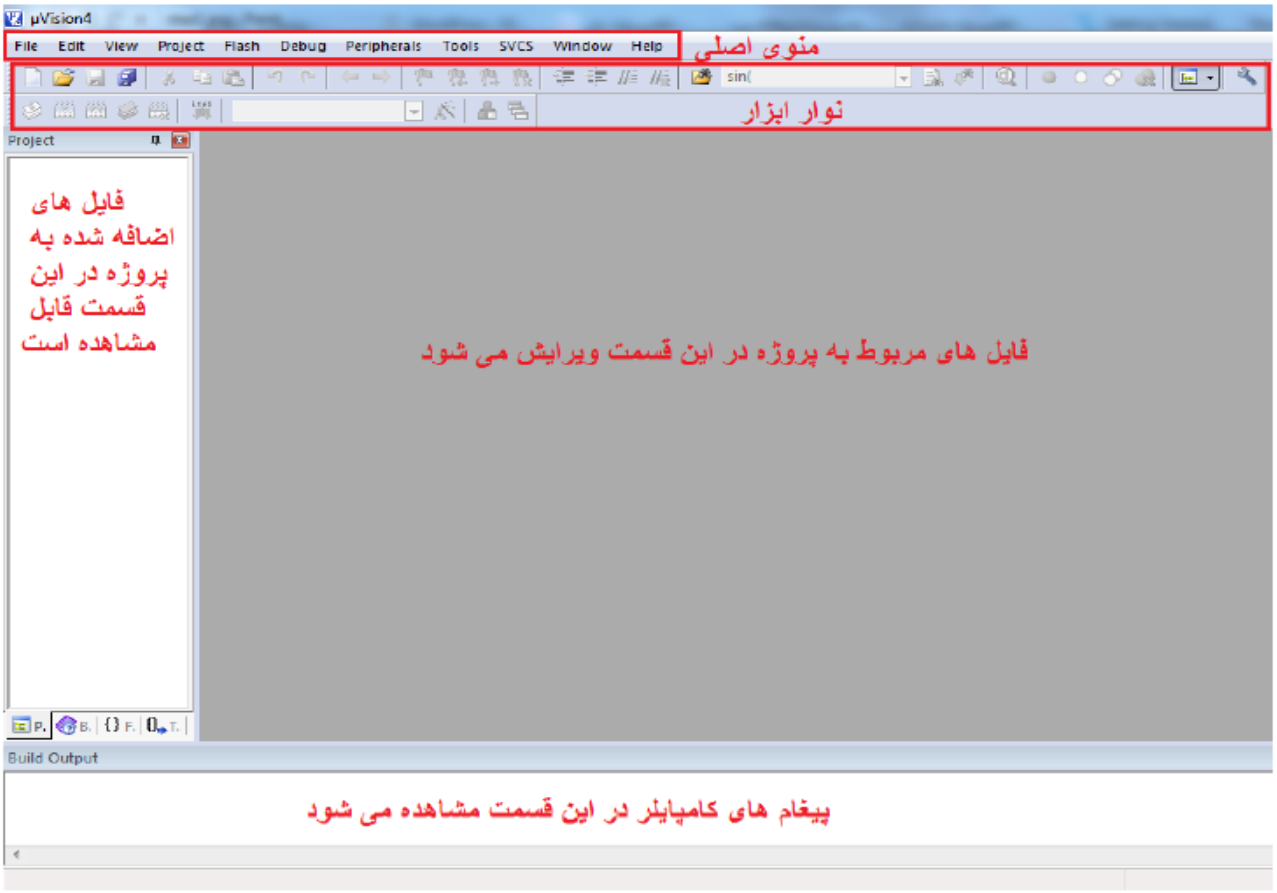
میکروکنترلرهای ARM سری LPC21xx یکی از راحت‌ترین میکروکنترلرها از نظر برنامه نویسی و سخت افزار مورد نیاز می‌باشند که طیف وسیعی از پروژه‌ها را نیز پوشش می‌دهند. ضمن اینکه جزو معدود میکروکنترلرهای ARM هستند که در نرم افزار پروتئوس وجود دارند و می‌توان آن‌ها را همانند میکروکنترلرهای AVR در پروتئوس شبیه سازی 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.

ابتدا نرم افزار KEIL uvision را دانلود نموده و نصب نمایید.

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 راحتی کار با نسخه MDK 4.7 کامپایلر Keil نسبت به نسخه ۵ آن ، از این نرم افزار برای برنامه نویسی کلیه میکرو کنترلرها استفاده خواهیم کرد.

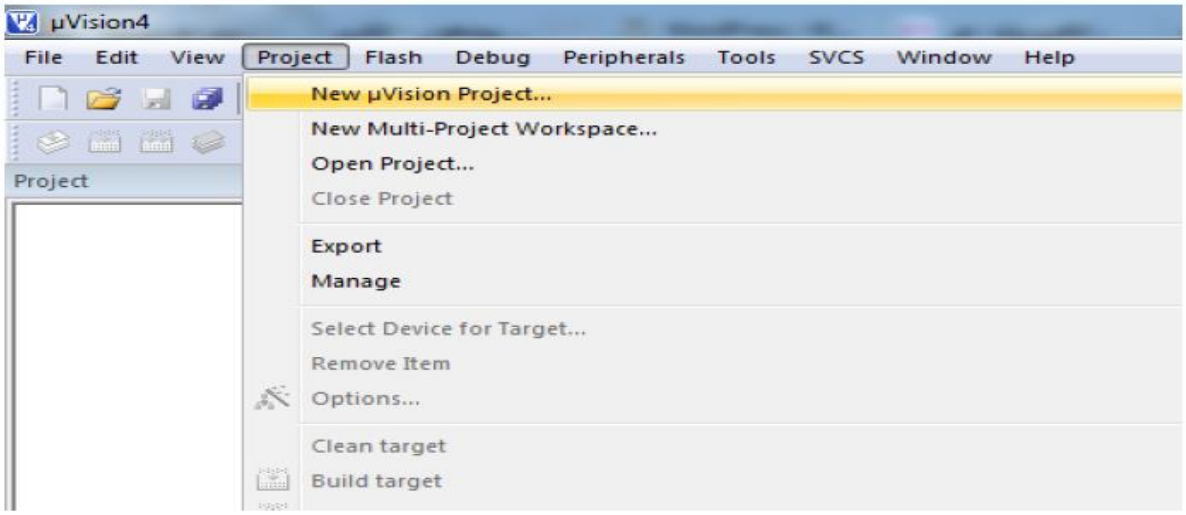
شروع به کار با نرم افزارKEIL

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

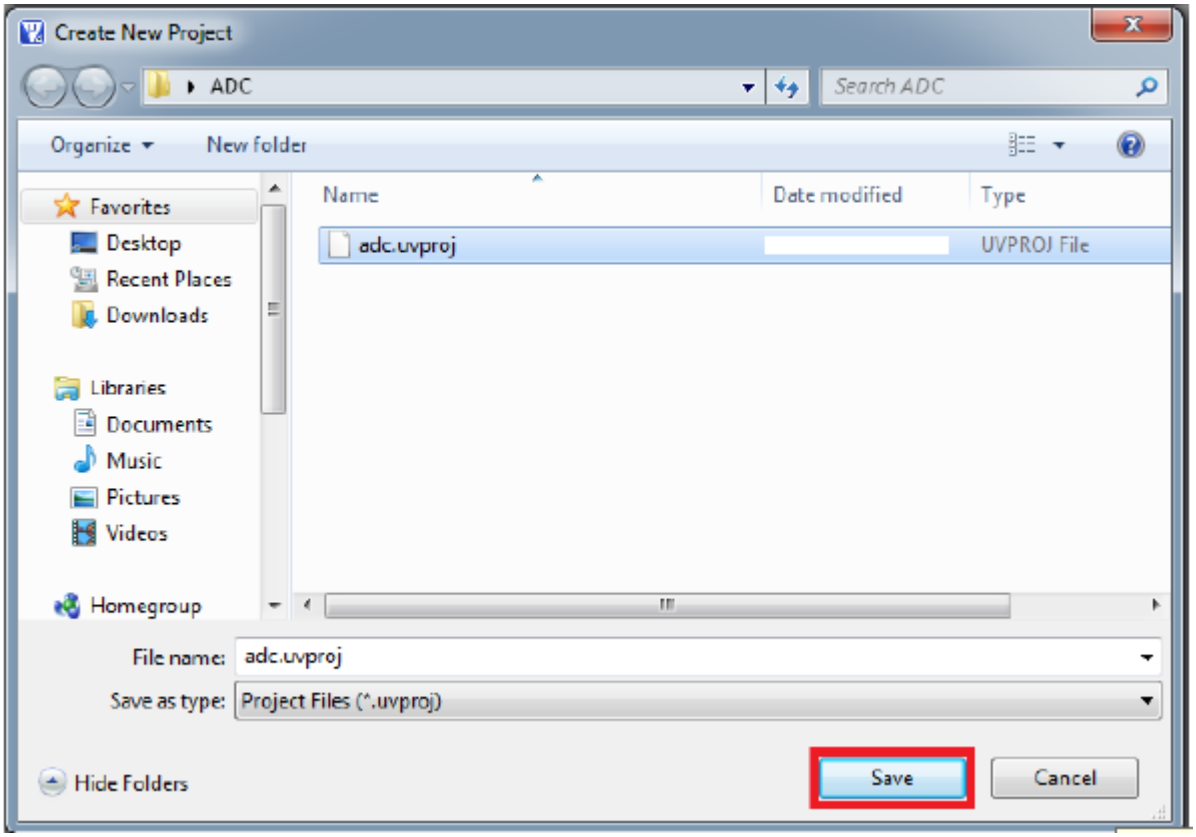


ایجاد یک پروژه جدید

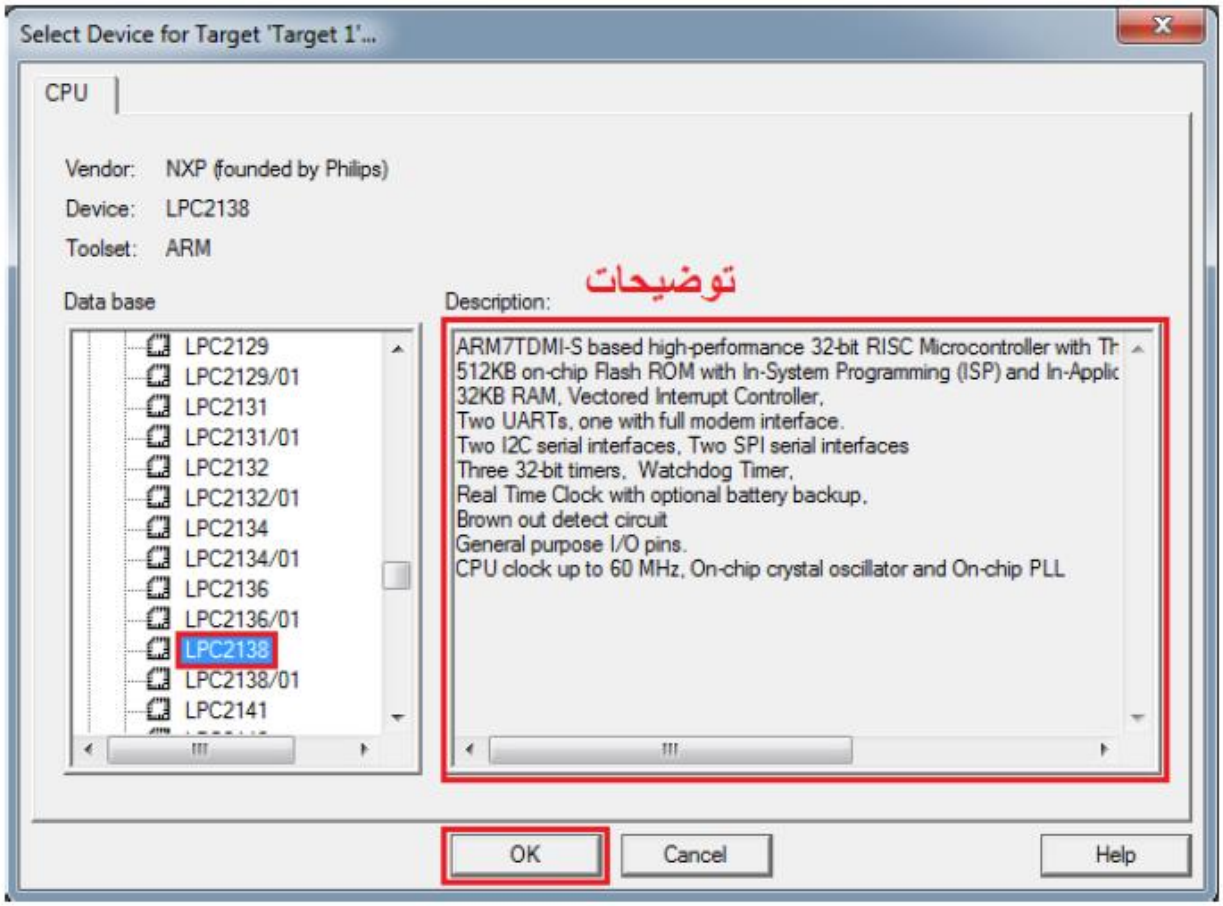
از منوی Project روی گزینه New Uvision Project کلیک کنید. با کلیک روی این گزینه از شما می‌خواهد تا یک مسیر برای ذخیره پروژه انتخاب نمایید.



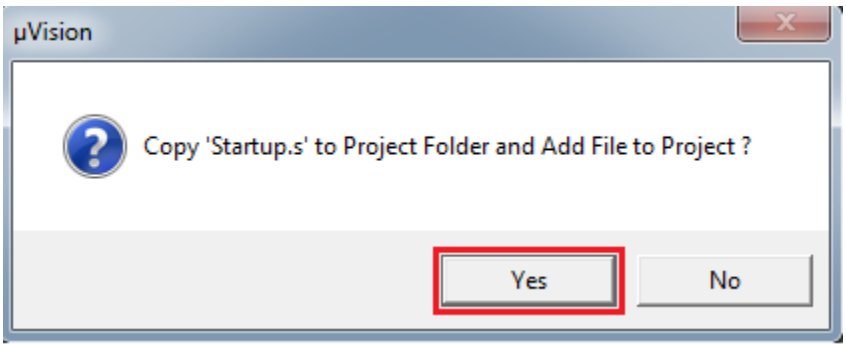
به علت جداسازی و در کنار هم بودن تمام پروژه آدرس پروژه جدید را در یک پوشه خالی انتخاب نمایید. سپس نام فایل اصلی پروژه را انتخاب نموده و روی گزینه Save کلیک کنید. با این کار فایل اصلی پروژه با پسوند uvproj ذخیره می‌شود.



در مرحله بعد پنجره ای باز می شود که نوع میکرو مورد نظر را باید در آن انتخاب نمود. در این پنجره روی شرکت NXP کلیک کنید و سپس LPC2138 را انتخاب نمایید. همانطور که مشاهده می‌کنید با انتخاب هر میکرو مشخصات آن در سمت راست پنجره نمایش داده می شود. بعد از انتخاب روی گزینه OK کلیک کنید.

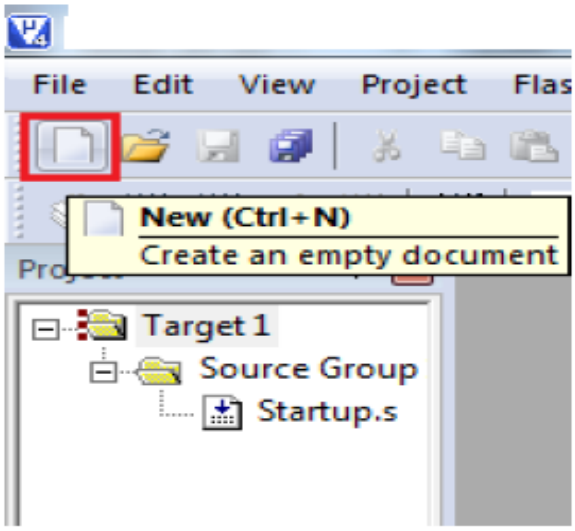


در نهایت از کاربر می‌خواهد که فایل مربوط به راه‌اندازی میکرو (Sturtup.s) که همیشه در پروژه وجود دارد و به همراه فایل های دیگر کامپایل می شود را به پروژه به صورت اتوماتیک اضافه نماید. با کلیک کردن بر روی گزینه Yes کار ایجاد پروژه جدید تمام می‌شود.

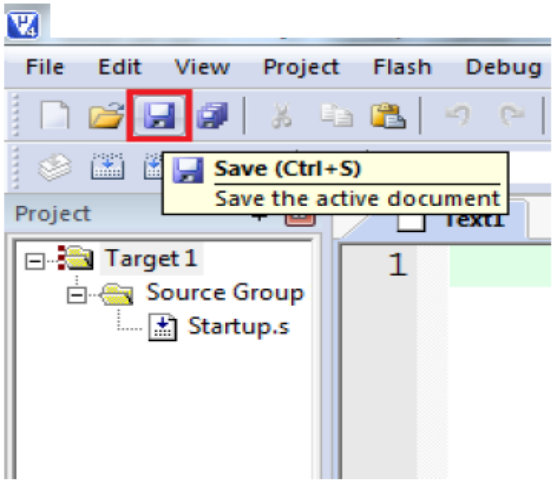


ایجاد یک فایل در ویرایشگر KEIL

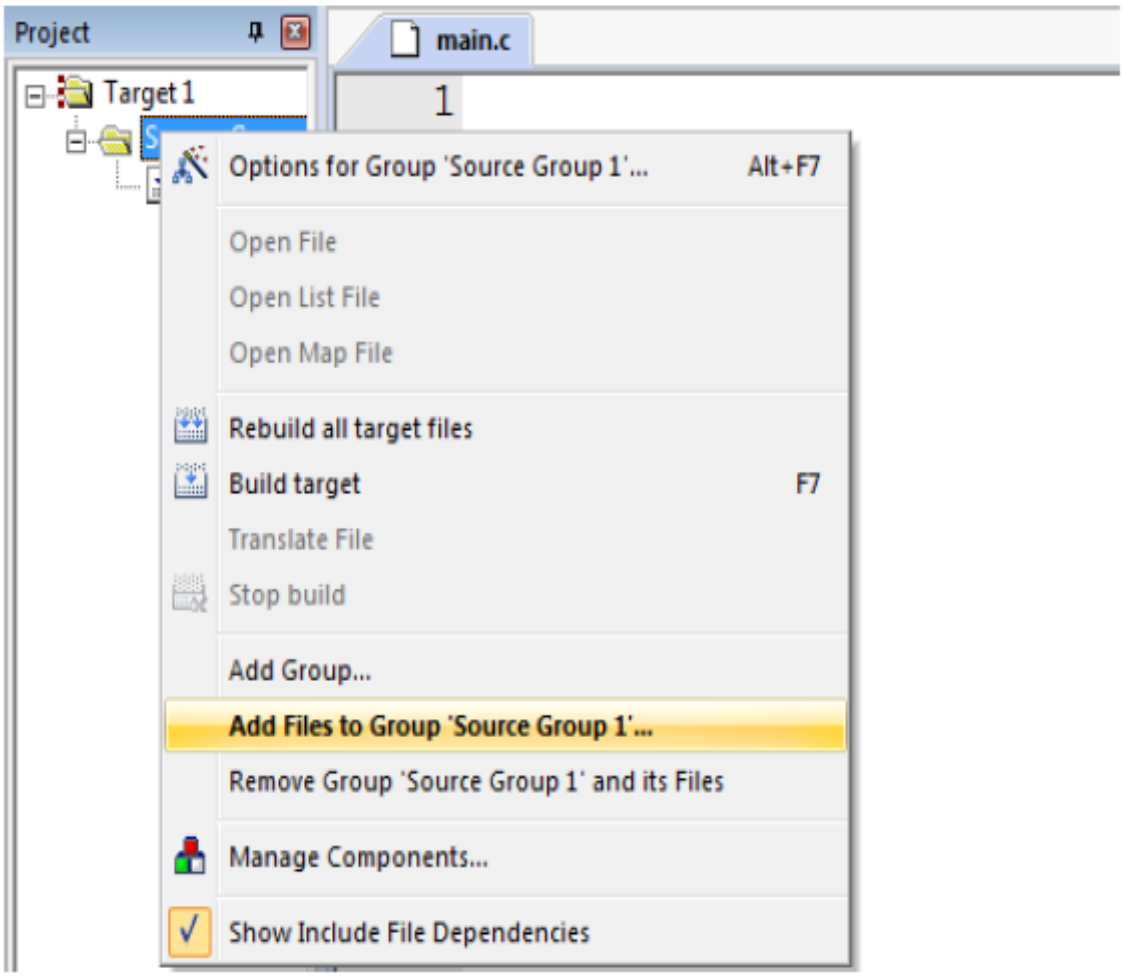
بعد از ساختن پروژه همانطور که در شکل زیر مشاهده می‌کنید، شاخه‌ای با نام Target 1 ایجاد می‌شود که درون آن فقط فایل Startup.s وجود دارد. برای نوشتن برنامه به زبان C ، باید یک فایل با پسوند c اضافه نماییم. برای این منظور از منوی File گزینه New را کلیک می‌کنیم و یا از نوار ابزار روی تصویر New کلیک می‌کنیم (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.

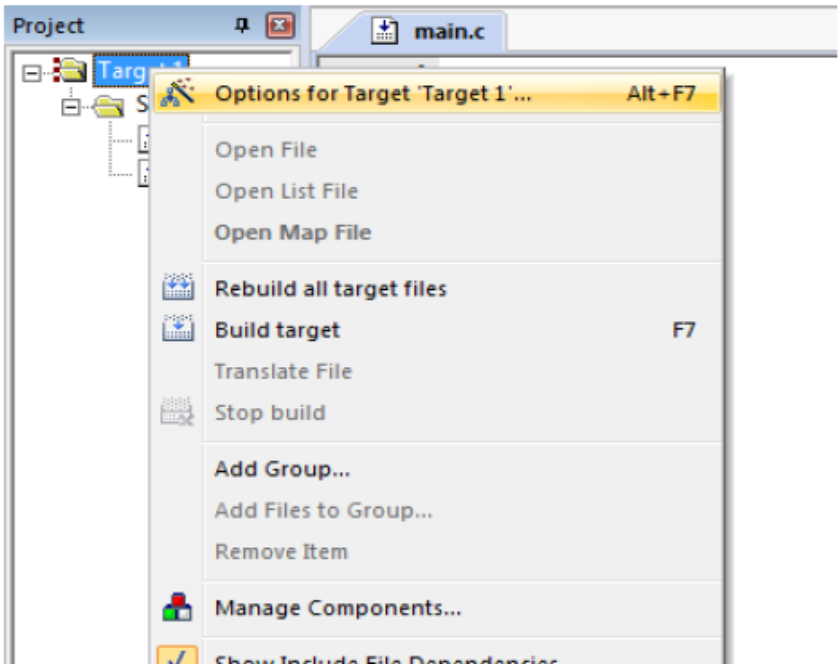


بعد از ذخیره کردن می‌بایست فایل مورد نظر را به پروژه اضافه نمود. برای اضافه کردن فایل جدید به پروژه ، از بخش Project روی Source Group کلیک راست می‌کنیم و از قسمت Add Files To Group، فایل main.c را به پروژه اضافه می‌کنیم.

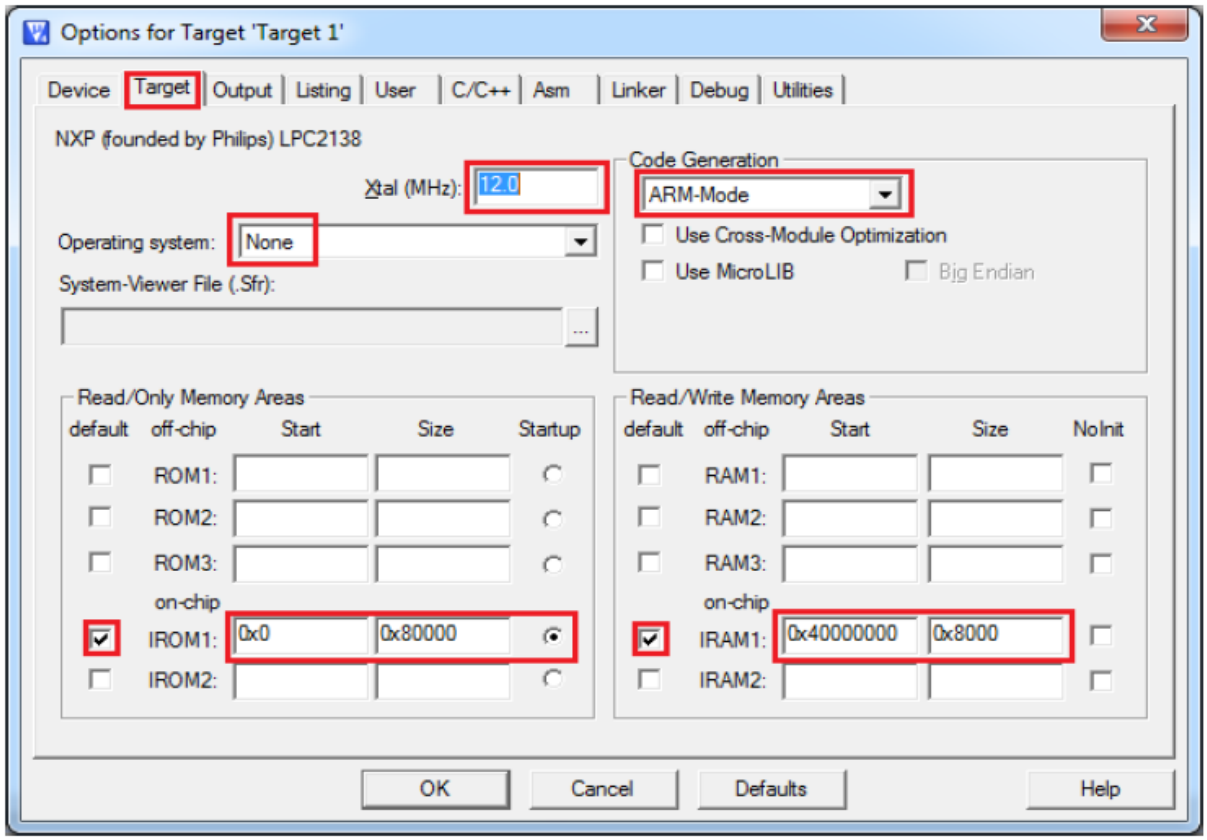


انجام تنظیمات پروژه

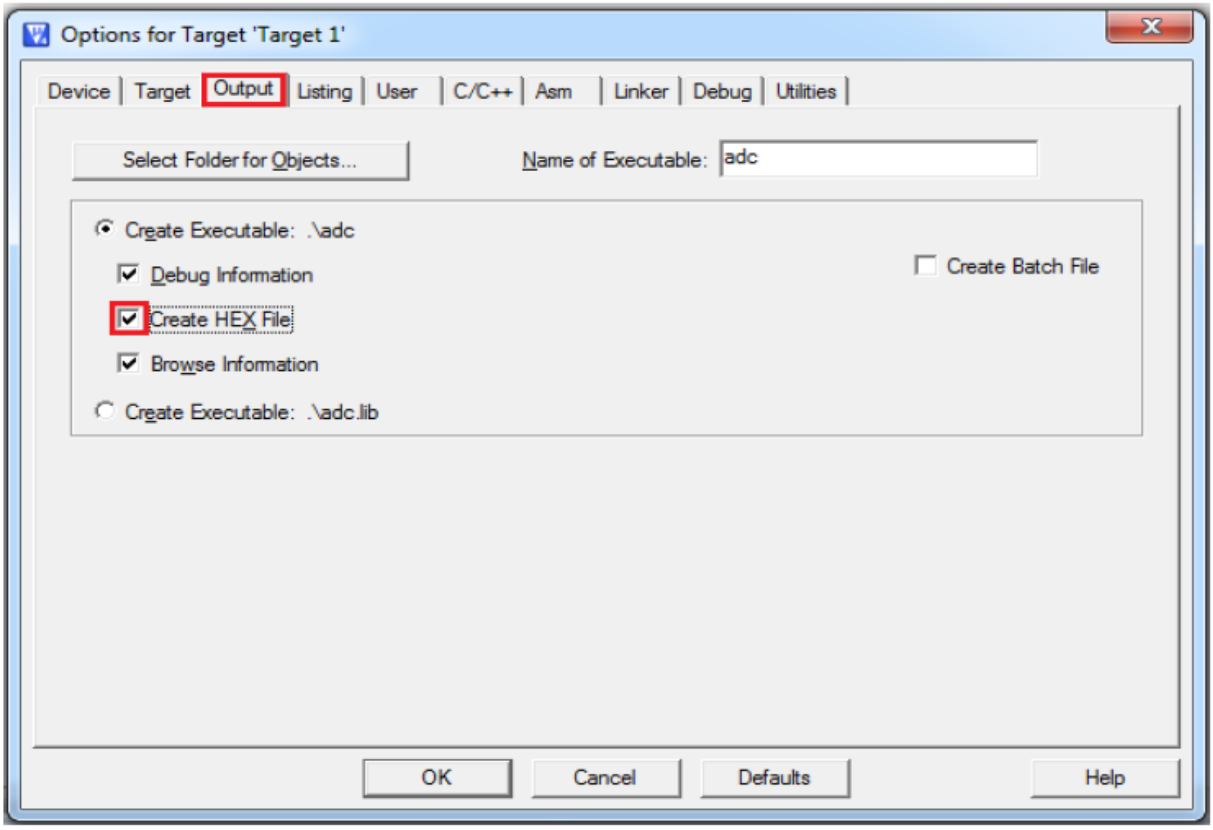
برای صحیح کامپایل شدن پروژه باید تنظیمات پروژه را انجام داد. برای این منظور از قسمت Project و با کلیک راست روی Target 1 و زدن Options For Target (یا از روی نوار ابزار) پنجره تنظیمات را باز می‌کنیم.



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 می‌زنیم.



ساختار برنامه نویسی ARM در نرم افزار KEIL

برنامه مورد نظر در نرم افزار Keil درون فایلی که اضافه کردیم به نام (main.c) نوشته می‌شود. 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 می‌کنیم، LPC213x.hنام دارد و تابع main در نرم افزار Keilباید خروجی int داشته باشد. بنابراین ساختار برنامه نویسی به صورت زیر خواهد بود.

```
#include <lpc213x.h>
int main (void){
while(1){
}
}
```

متغیرها و ثوابت

تعریف متغیر یعنی انتخاب نام مستعار برای مکانی از حافظه که برای جلوگیری از اضافه نویسی و اشتباه در بازخوانی کد استفاده می‌شود.

فرم تعریف متغیر: نوع متغیر نام متغیر ;

مثال : char a ;

انواع داده ها:

Type	Size (Bits)	Range
bit	1	0 , 1
char	8	-128 to 127
unsigned char	8	0 to 255
signed char	8	-128 to 127
int	16	-32768 to 32767
short int	16	-32768 to 32767
unsigned int	16	0 to 65535
signed int	16	-32768 to 32767
long int	32	-2147483648 to 2147483647
unsigned long int	32	0 to 4294967295
signed long int	32	-2147483648 to 2147483647
float	32	±1.175e-38 to ±3.402e38
double	32	±1.175e-38 to ±3.402e38

برای تعریف متغیر در حافظه EEPROM از کلمه کلیدی eeprom قبل از نام متغیر استفاده می‌کنیم.

مثال : **eeprom int code;**

می‌توان در زمان تعریف به متغیر مقدار اولیه داد.

مثال : **char s = 'a';**

تعریف ثابت با کلمه کلیدی const انجام می‌شود.

مثال: **const float pi =3.14**

با دستور #define می‌توان ماکرو تعریف نمود، در این حالت مقداری که برای ثابت تعریف می شود نوع داده را تعیین کرده و مقادیر تعریف شده، توسط پیش پردازنده با مقدار ثابت جایگزین می‌شود.

مثال: **#define code 100;**

دستور IODIRx :

با دستور بالا می‌توان پایه‌های پورت x که x می‌تواند پورت صفر یا یک باشد را بعنوان ورودی یا خروجی تعریف نمود. در فصل‌های بعدی این دستور مفصلا توضیح داده می‌شود.

IODIR0 = 0xFFFFFFFF;

دستور IOSETx , IOCLRx :

با دستور بالا نیز می‌توان پایه‌ایی از پورت x که x می‌تواند پورت صفر یا یک باشد را صفر منطقی یا یک منطقی نمود. در فصل‌های بعدی این دستور مفصلا توضیح داده می‌شود.

مثال :

```
IOSET0 =0X0FF01B5D;
IOCLR0 = 0XE1A0A589;
```

دستور IOPINx :

زمانیکه یک پایه بعنوان ورودی تعریف می‌شود برای اینکه از وضعیت این پایه متوجه شویم از دستور IOPIN0 در دستور شرطی استفاده می‌کنیم تا از یک بودن یا صفر بودن این پایه از پورت با خبر شویم.

مثال : در دستور زیر، چک خواهیم کرد که آیا پایه دوم و سوم از پورت یک، یک منطقی هستند یاخیر؟

```
if(IOPIN1==0X00000003)
```

تنوع عملگرها

- **عملگرهای حسابی و بیتی**

عملگرها در عملیات ریاضی و منطقی مورد استفاده قرار می‌گیرند. دارای ترتیب اولویت هستند همانطور که در جدول آمده از بالا دارای بیشترین اولویت و به ترتیب به سمت پایین اولویت کم می‌شود.

عملگر	عملکرد	مثال	نتیجه
*	ضرب	۳*۲	٦
/	تقسیم	۵/۲	۲,۵
+	جمع	۳+٦	۹
-	تفریق	۸-۳	۵
%	باقیمانده	۱۰٪۳	۱
&	AND	0xF0 & 0x0F	0x00
|	OR	0x00 | 0x03	0x03
^	XOR	0x0F ^ 0xFF	0xF0
~	مکمل یک	~(0xF0)	0x0F
>>	شیفت به راست	0xF0 >> 4	0x0F
<<	شیفت به چپ	0x0F << 4	0xF0

- **عملگرهای یکانی**

عملگر	عملکرد	مثال	نتیجه
-	قرینه	-a	a = a×-1
++	افزایش یک واحد	a++	a = a + 1
--	کاهش یک واحد	a--	a = a - 1

- **عملگرهای مقایسه ای**

نتیجه	مثال	عملکرد	عملگر
False	2>3	بزرگتر	>
True	'm'>'e'	کوچکتر	<
True	5>=5	بزرگتر یا مساوی	>=
True	2.5<=4	کوچکتر یا مساوی	<=
False	'A' =='B'	تساوی	==
True	2!=3	نامساوی	!=

- **عملگرهای منطقی**

نتیجه	مثال	عملکرد	عملگر
False	(2>3)&&(1!=3)	AND منطقی	&&
True	('a'<3)||(10)	OR منطقی	||
False	!(7>5)	نقیض	!

- **عملگرهای انتساب**

عملگر	عملکرد	مثال	نتیجه
=	انتساب	a=b	a ⇦ b
*=	ضرب و انتساب	a*=b	a=a*b
/=	تقسیم و انتساب	a/=b	a=a/b
+=	جمع و انتساب	a+= b	a=a+b
-=	تفریق و انتساب	a-=b	a=a-b
a=(value)?x:y	انتساب شرطی	اگر value مقدار صحیح باشد a برابر x و در غیراینصورت برابر y می شود.	

آرایه‌ها

آرایه‌ها مجموعه‌ایی از متغیرهای هم‌نوع هستند.

فرم تعریف: اسم متغیر نوع متغیرهای آرایه[تعداد عناصر] ;

مثال: int a[0 5];

با مقدار اولیه: int a[5] = { ۱،۲،۳،٤،٥ };

فرم تعریف آرایه های دو بعدی: نوع متغیرها اسم متغیر[تعداد عناصر سطر][عداد عناصر ستون];

مثال:

```
int a[2][3]={{1,2,3},{ ٤،٥،٦ }};
```

در زبان C اندیس آرایه‌ها از صفر شروع می‌شود.

رشته‌ها

رشته‌ها آرایه‌های کاراکتری هستند.

مثال :

```
char name[ ] = "Test";
```

یا:

```
char name[5]={'T','e','s','t','\ ٠ '};
```

رشته ها همواره به یک کاراکتر Null ختم می‌شوند.

دستور پرش goto :

پرش بدون شرط به یک برچسب انجام می‌شود.

```
IOSET0=0x00000001;
Int a;
Res:
a=0;
While(1){
IOCLR1=0x000000ff;
Goto Res;
}
```

دستورات شرطی :

۱. ساختار : if - else

```
if(شرط)
{
```

```
دستورات ۱ ;

}

else

{

دستورات ۲ ;

}
```

- در صورتی که دستورات یک خطی باشند می‌توان brace ({}) را حذف نمود.
- استفاده از بلاک else اختیاری است.

```
if(key==0)
{
   if(key!=p_state)
    {
     if(i==15)
          {
             i=0;
               n=digits[i];
                 }
        Elseif{
                  i++;
                  n = digits[i];
                p_state=0;
          };
          }
Else{
p_state=1;
}
```

مثال :

```
int a,b;

if(a= =0)
{
b=a*10;
```

```
}
else if(a>0)
{
b=a;
}
Else
{
b=++a;
}
```

۲. ساختار Switch – Case :

```
switch(مقدار)

{

case ۱ مقدار:

دستورات ۱;

break;

case ۲ مقدار:

دستورات ۲;

break;

.

.

default:

n دستورات;

}
```

مثال :

```
int a;
printf("Enter Month of your birth: ");
scanf("%d",&a);
switch(a){
case 1:
```

```
case 2:
case 3: printf("You've born is spring\n");
break;
case 4:
case 5:
case 6: printf("You've born is summer\n");
break;
case 7:
case 8:
case 9: printf("You've born is autumn\n");
break;
case 10:
case 11:
case 12: printf("You've born is winter\n");
break;
default: printf("Error! Enter a number between 1-12\n");
}
}
```

حلقه‌هاي تكرار

۱. ساختار While :

```
while(شرط)
{
دستورات ;
}
```

اگر بخواهیم این حلقه تکرار بی‌نهایت بار تکرار شود از دستور زیر استفاده می‌کنیم.

```
While(1)
{
دستورات ;
}
```

مثال :

```
while(temp>10){
IOSET0 = 0x0000FF00;
}
```

۲. ساختار Do/While :

```
do{
دستورات ;
}while(شرط)
```

اگر بخواهیم این حلقه تکرار بی‌نهایت بار تکرار شود از دستور زیر استفاده می‌کنیم.

```
do{
دستورات ;
}while (1)
```

نکته : در ساختار Do/While بر خلاف While شرط در انتهای حلقه آزمایش می‌شود، بنابراین دستورات داخل حلقه، حداقل یکبار اجرا می‌شوند.

مثال :

```
do{
IOCLR1 = 0x0000FF00;
} while (IOPIN1=0x00000005)
```

۳. حلقه‌هایFor :

```
for( مقدار ابتدای حلقه ; شرط پایان ; گام)
{
```

```
دستورات ;
{
```

و اگر بخواهیم این حلقه تکرار بی‌نهایت بار تکرار شود از دستور زیر استفاده می‌کنیم.

```
for(  ;  ;  )
{
دستورات ;
{
```

مثال :

```
for(i = 64; i > 1; i = i/2)
{
a = i;
delay_ms(100);
}
```

مثال :

```
int a=10,i=20,b;
long int fact=1;
while(1){
--a;
if(a<0){
a=a*2;
}
else if(a==0)
{
```

```
b=a;
}
else{
for(i=1;i<=a;i++)
fact*=i;
}
}
}
```

- دستور break باعث خروج بدون شرط از هر حلقه‌ایی می‌شود.
- دستور continue باعث می شود اجرای ادامه دستورات متوقف شده و حلقه از ابتدا آغاز شود.

توابع

تعریف توابع به صورت زیر می‌باشد:

```
(آرگومان‌های تابع) نام تابع    نوع داده خروجی
{
متغیرهای محلی
دستورات تابع
}
```

نکته : توابع داخل یکدیگر قابل تعریف نمی‌باشند و جدا از هم باید تعریف گردند.

مثال :

```
long int kelid(int x){
```

```
if(IOPIN1==0X00000001)
return x*x*x;
}
int main(void)
{
}
```

مثال :

```
int  max(int a,int b);
int main( void){
int a,b;
}
int max(int a,int b){
if(a>b)
return a;
else
return b;
}
```

خصوصیات ADC

- وجود ۸ کانال ورودی ADC که عمل تبدیل آن ۱۲ بیتی است که تبدیل از روش تقریبات متوالی انجام می‌دهد
- وجود مد Power-down (برا بحث کاهش جریان مصرفی میکروکنترلر)
- ولتاژ مرجع ۳ ولت
- دقت اندازه گیری = بیت ۱۲
- بالاترین سرعت نمونه برداری = ۲۰۰ Khz
- وجود مد تبدیل Burst برای یه تک سیگنال ورودی یا چندین سیگنال ورودی
- قابلیت شروع تبدیل با سیگنال های مقایسه تایمر و یا بعضی پایه های ورودی
- تبدیل صحیح و کامل نیاز به ٦٥ کلاک ADC دارد

پایه‌های AD0.[0:7] : این ۸ تا پایه ورودی آنالوگ، پایه‌های ADC ما هستند که وظیفه اندازه‌گیری ولتاژ را بر عهده دارند.

هشدار : اگر از این ۸ تا پایه به عنوان ADC استفاده کنید، نباید ولتاژهایی بالاتر از VDDA ، به این پایه‌ها اعمال کنید؛ در غیر این صورت، دیتای خونده شده معتبر نمی‌باشد؛ اگه مبدل A/D در پروژه استفاده نشه اون وقت پین های مرتبط با ورودی A/D می توانند بعنوان پایه‌های دیجیتال وروردی خروجی پنج ولت استفاده شوند.

در جدول زیر مکان پایه‌های ADC میکروکنترلر LPC1768 مشخص شده است.

Table 80. Pin function select register 0 (PINSEL0 - address 0x4002 C000) bit description

PINSEL0	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
5:4	P0.2	GPIO Port 0.2	TXD0	AD0.7	Reserved	00
7:6	P0.3	GPIO Port 0.3	RXD0	AD0.6	Reserved	00

Table 81. Pin function select register 1 (PINSEL1 - address 0x4002 C004) bit description

PINSEL1	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
15:14	P0.23[1]	GPIO Port 0.23	AD0.0	I2SRX_CLK	CAP3.0	00
17:16	P0.24[1]	GPIO Port 0.24	AD0.1	I2SRX_WS	CAP3.1	00
19:18	P0.25	GPIO Port 0.25	AD0.2	I2SRX_SDA	TXD3	00
21:20	P0.26	GPIO Port 0.26	AD0.3	AOUT	RXD3	00

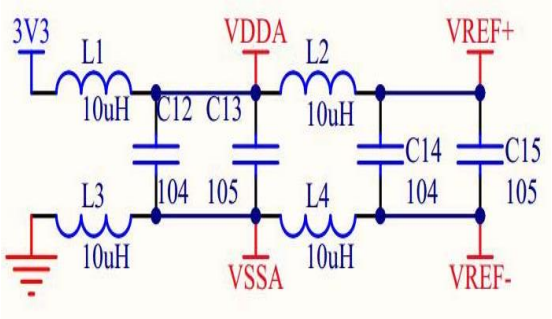
Table 83. Pin function select register 3 (PINSEL3 - address 0x4002 C00C) bit description ...continued

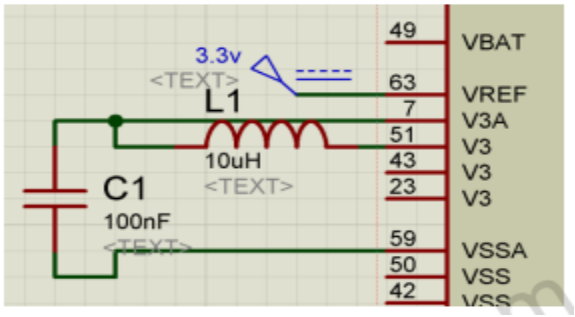
PINSEL3	Pin name	Function when 00	Function when 01	Function when 10	Function when 11	Reset value
29:28	P1.30	GPIO Port 1.30	Reserved	V_{BUS}	AD0.4	00
31:30	P1.31	GPIO Port 1.31	Reserved	SCK1	AD0.5	00

پایه‌های VREFP و VREFN : ولتاژ مرجع؛ این دو پایه ولتاژ مرجع برای ADC و DAC فراهم می‌کنند. ولتاژ مرجع، ولتاژی است که برای تبدیل، مبدل با این ولتاژ مقایسه می‌کند.

توجه : اگر از ADC یا DAC استفاده نمی‌کنید، باید پایه VREFP رو به ولتاژ ۳/۳ ولت پایه (VDD) و پایه VREFNرو به پایه VSS وصل کنید.

پایه‌های VDDA و VSSA : تغذیه + و – آنالوگ؛ این پایه‌ها عموما باید ولتاژی برابر V DD و V SS داشته باشند، اما برای به حداقل رساندن نویز و خطا باید ایزوله شوند. شکل زیر نمونه‌ایی از این مدار کاهش نویز است.





توجه : اگر از ADC یا DAC استفاده نمی‌کنید، باید پایه VDDA را به ولتاژ ۳/۳, ولت (پایه VDD) و پایه VSSA رو به پایه VSS وصل **کنید**.

نکته : دقت یا رزولوشن مبدل در ARM با توجه به رابطه زیر بدست می‌آید. که ۳/۳ ولتاژ مرجع و ۱۲ تعداد بیت های مبدل می‌باشد.

ADC = 3.3/(2^12) = 3.3/4096 =0.000805 = 0.8mV

همچنین میکرو lpc2138 دو واحد مجزا از هم برای ADC دارد که هر یک ۸ کانال ورودی مختلف نیز دارند. در نتیجه ۱٦ ورودی آنالوگ وجود دارد. در یک زمان فقط یکی از کانالها می‌تواند به کد دیجیتال تبدیل شود.

این واحد برای LPC 2138 دارای مشخصات زیر می‌باشد :

- ولتاژمرجع بین 0 تا VDDA (حداکثر ۳/۳ ولت)
- دقت رزولوشن ۱۰ بیت
- حداکثر نرخ نمونه برداری ٤/٥ میلیون نمونه در ثانیه
- هشت کانال برای ADC0 (پایه‌های ٤، ٥، ٢٥ ،٢٦، ٢٧، ٢٨، ٢٩، ٣٠ پورت صفر)
- هشت کانال برای ADC1 (پایه‌های ٨، ١٠، ١٢، ١٣، ١٥، ٢١، ٢٢ پورت صفر)

دستورات توابع زیر برای راه اندازی مبدل استفاده می‌شود.

```
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;
  }
```

تابع زیر نیز مربوط داده برداری از پایه های ورودی می‌باشد. که در ورودی خود شماره کانال را دریافت می‌کند و بعد از اتمام عملیات تبدیل، عدد دیجیتال را به خروجی تابع می‌دهد.

```
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);
  }
```

حال بعد از نوشتن این تابع، در قسمت اصلی برنامه بفرض وقتی سنسور دما را کانال یک از adc0 وصل کردیم دستور زیر را می‌نویسیم. عدد دیجیتال شده در متغیر dama قرار می‌گیرد.

```
dama = read_adc0(1);
```

و یا طبق تابع زیر، با دستور زیر می‌توان دیتای adc1 را خواند.

```
temp = read_adc1(1);
```

```
unsigned short read_adc1(unsigned char char
  {
  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);
  }
```

به دو روش می توان داده ها را پردازش کرد (انواع وقفه از دیدگاه CPU):

۱. روش سرکشی (Polling)

در روش سرکشی، میکروکنترلر دائما در حال خواندن دستورات بصورت ترتیبی بوده و آن‌ها را با فاصله زمانی معین و به نوبت اجرا می‌کند و بدون رسیدن نوبت دستورات، اجرای آن‌ها ممکن نیست. دراین روشCPU باید منتظر رسیدن نوبت اتفاق موردنظر باشد نه اینکه منتظر خود اتفاق که در این حین که CPU منتظر اتفاق افتادن کاری هست، این کار اتفاق بیفتد و قبل از رسیدن نوبتش، این کار اتمام شده باشد. مثلا در برنامه‌ایی که شامل ۲۰۰۰خط دستور هست، دستور زده شدن کلیدی در خط اول گفته شده که بعد ازین دستور، کلید بلافاصله زده می‌شود و میکرو هم دارد خطوط بقیه را بررسی می‌کند کلید برداشته شود و میکرو به دستور کلید برسد در حالیکه کلید قطع شده در آن صورت برنامه بدون متوجه شدن این کار به کار خودش ادامه میدهد.

همچنین ایراد این روش اینست که CPU باید بطور مداوم سرکشی کنه و بیت های پرچم المان‌ها را بررسی کنه که این کار باعث ازدست رفتن زمان بررسی المان‌های دیگه می‌شود.

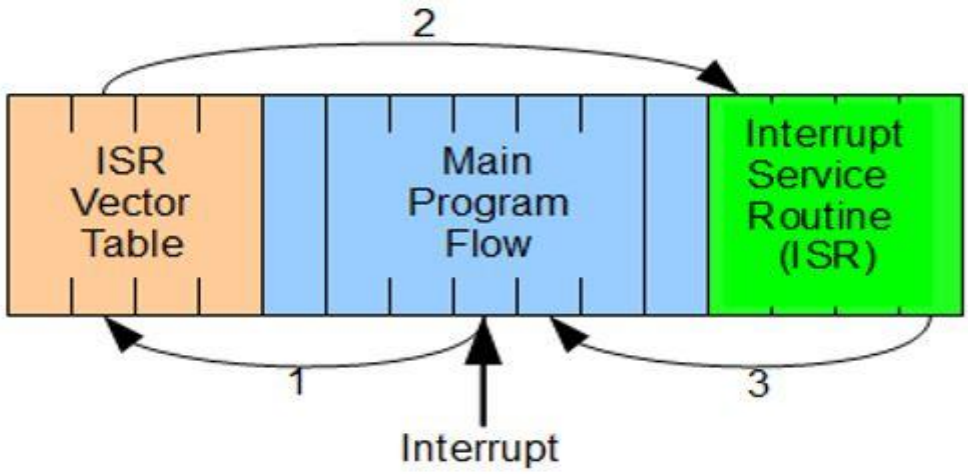
۲. روش وقفه Interrupt

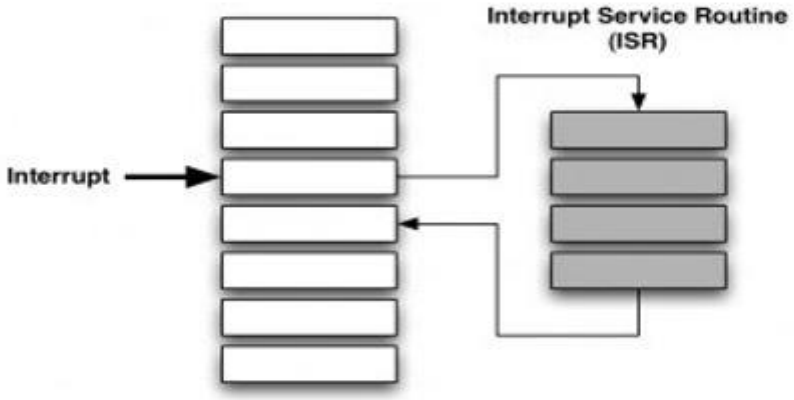
برای رفع مشکل سرکشی،CPU باید به انجام وظیفه ی اصلی خود بپردازد و در آن حین، اتفاق مورد نظر به هنگام وقوع، به صورت خودکار به CPU گزارش شود برای همین امکانی در میکروکنترلر تعبیه شده که اتفاقات مهم توسط برنامه به اطلاع CPU برسد.در این روش CPU دستورات برنامه رو بطور عادی انجام میدهد و المانی که تحت عنوان وقفه به CPU معرفی شده درصورت تحریک پایه وقفه، سیگنال وقفه، روال عادی CPU را متوقف کرده و آدرس دستور بعدی رو ذخیره کرده و دستورات زیرتابع وقفه رو اجرا می‌کند و پس از اتمام دستورات زیرتابع وقفه، به ادامه کار خودش(به آدرسی که ذخیره کرده) برمی‌گردد.اگه بطور خلاصه بگوییم اینطور

می‌شود که وقفه پیامی است که از طرف یک وسیله جانبی تولید می‌شود تا از CPU درخواست سرویس کند CPU با دریافت یک سیگنال وقفه، کارهای جاری را متوقف کرده و به وسیله مورد نظر سرویس می دهد.

در کارهای روزمره، همچین کاری زیاد اتفاق میفته مثلا در خانه در حال انجام کارهای خودمان هستیم که زنگ در زده می‌شود که در آن صورت کارهایمان را متوقف کرده و در را باز می‌کنیم.

این روش باعث میشه تا زمان cpu برای بررسی رخ دادن یک وضعیت تلف نشود.





.

وقفه‌ها در میکرو کنترلر به دو دسته وقفه‌های داخلی و خارجی تقسیم می‌شود و با توجه به نوع میکرو تعداد آن ها متفاوت است.

وقفه داخلی

همانطور که از اسمش پیداست المان‌های داخل میکرو، سیگنال‌های وقفه رو ایجاد می‌کنند. تقریبا تمام امکانات داخلی میکرو دارای وقفه بوده و اکثر این سیگنال‌ها از وسایل جانبی خود میکرو مانند تایمر/کانتر ها، سنکرون، پروتکل ارتباطی، مقایسه کننده‌ها ، ADCو ... تولید می‌شوند. این سیگنال‌ها همان درخواست سرویس از CPU می‌باشند.

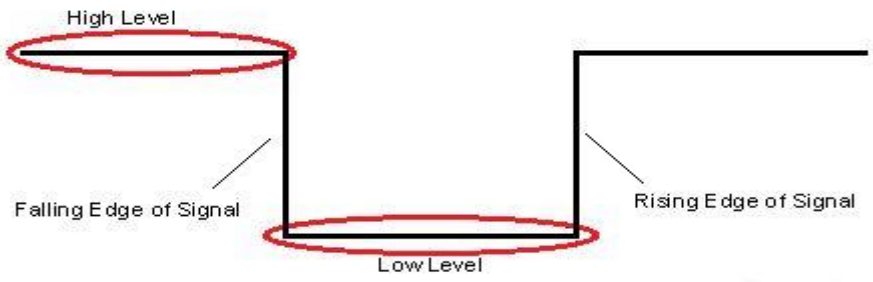
وقفه خارجی

وقفه‌های Interrupt، وقفه‌هایی هستند که منبع آن‌ها خارج از میکرو قرار دارد و با ارسال سیگنالی به پایه‌های مخصوصی(int0,int1,int2,int3) در میکروکنترلر، وقفه ایجاد می‌کنند. یعنی زمانی که پایه‌های INTx تحریک شود، درخواست وقفه Interrupt می‌شود و میکرو به تابع وقفه رفته و زیربرنامه‌های آنرا اجرا می‌کند. میکروی lpc21xx چهار پایه وقفه خارجی دارد که شماره پایه‌ها در جدول زیر بیان شده است.

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 می‌افتد :

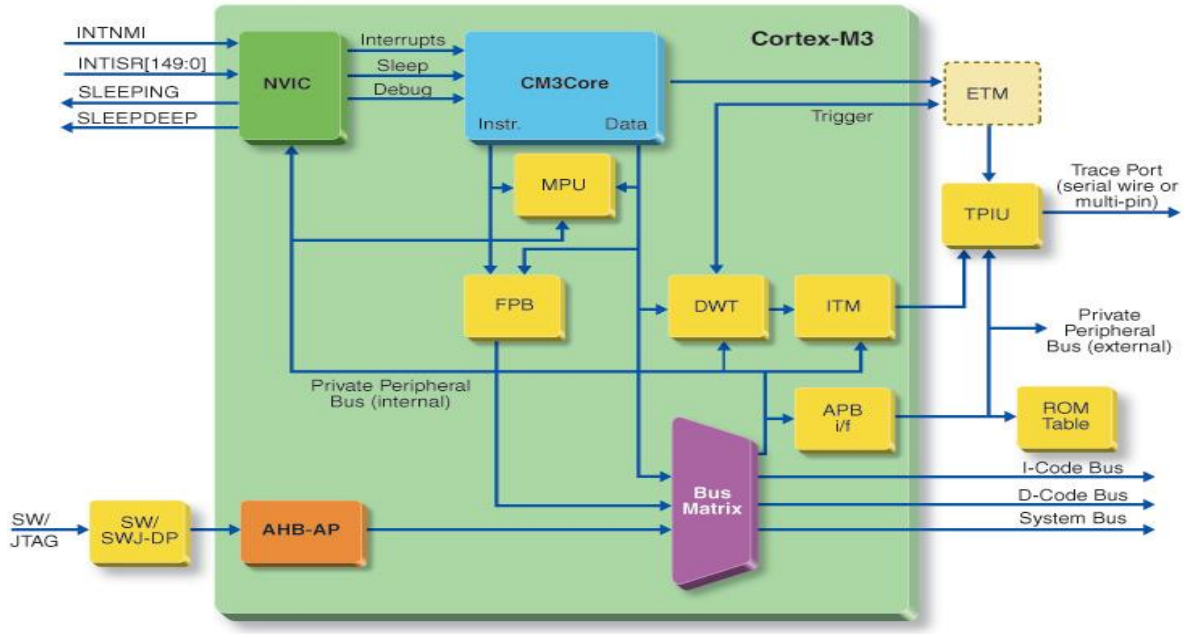
سطح صفر و سطح یک، لبه بالا رونده و لبه پایین رونده



نکته : هنگامی که یک پایه به عنوان وقفه استفاده شود دیگر نمی‌توانیم از آن پایه به عنوان ورودی یا خروجی استفاده کنیم.

برای استفاده از وقفه‌های خارجی باید بصورت زیر عمل کنیم:

- ابتدا پایه مربوط به وقفه Interrupt را ورودی کرده و pull-up یا pull-down می‌کنیم.
- وقفه مورد نظر را فعال می‌کنیم.
- مشخص می‌کنیم نسبت به چه حساس باشد.
- وقفه سراسری را فعال می‌کنیم.



اگر حساس به سطح منطقی صفر و لبه پایین رونده باشه pull-up و اگه حساس به سطح منطقی یک و لبه بالارونده باشه pull-down می‌کنیم.

نکته: کنترل کننده وقفه‌ها در LPC1768 از نوع NVIC یا کنترل کننده وقفه تودرتوی برداری می‌باشد که باعث افزایش کارایی میکروکنترلر می‌شود.

منابع وقفه میکروکنترلر LPC1768

میکروکنترلر LPC1768 از ۳۵ وقفه برای امکانات جانبی پشتیبانی می‌کند که بعضی از وقفه‌ها دارای چند خط

وقفه هستند، در عکس زیر منابع وقفه میکروکنترلر lpc1768 بهمراه شماره وقفه و اطلاعات بیشتر موجود است.

Table 50. Connection of interrupt sources to the Vectored Interrupt Controller

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s)
0	16	0x40	WDT	Watchdog Interrupt (WDINT)
1	17	0x44	Timer 0	Match 0 - 1 (MR0, MR1) Capture 0 - 1 (CR0, CR1)
2	18	0x48	Timer 1	Match 0 - 2 (MR0, MR1, MR2) Capture 0 - 1 (CR0, CR1)
3	19	0x4C	Timer 2	Match 0-3 Capture 0-1
4	20	0x50	Timer 3	Match 0-3 Capture 0-1
5	21	0x54	UART0	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
6	22	0x58	UART1	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) Modem Control Change End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
7	23	0x5C	UART 2	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
8	24	0x60	UART 3	Rx Line Status (RLS) Transmit Holding Register Empty (THRE) Rx Data Available (RDA) Character Time-out Indicator (CTI) End of Auto-Baud (ABEO) Auto-Baud Time-Out (ABTO)
9	25	0x64	PWM1	Match 0 - 6 of PWM1 Capture 0-1 of PWM1
10	26	0x68	I^2C0	SI (state change) Melec.ir
11	27	0x6C	I^2C1	SI (state change)
12	28	0x70	I^2C2	SI (state change)
13	29	0x74	SPI	SPI Interrupt Flag (SPIF) Mode Fault (MODF)

Table 50. Connection of interrupt sources to the Vectored Interrupt Controller

Interrupt ID	Exception Number	Vector Offset	Function	Flag(s) Melec.ir
14	30	0x78	SSP0	Tx FIFO half empty of SSP0 Rx FIFO half full of SSP0 Rx Timeout of SSP0 Rx Overrun of SSP0
15	31	0x7C	SSP 1	Tx FIFO half empty Rx FIFO half full Rx Timeout Rx Overrun
16	32	0x80	PLL0 (Main PLL)	PLL0 Lock (PLOCK0)
17	33	0x84	RTC	Counter Increment (RTCCIF) Alarm (RTCALF)
18	34	0x88	External Interrupt	External Interrupt 0 (EINT0)
19	35	0x8C	External Interrupt	External Interrupt 1 (EINT1)
20	36	0x90	External Interrupt	External Interrupt 2 (EINT2)
21	37	0x94	External Interrupt	External Interrupt 3 (EINT3). **Note:** EINT3 channel is shared with GPIO interrupts
22	38	0x98	ADC	A/D Converter end of conversion
23	39	0x9C	BOD	Brown Out detect
24	40	0xA0	USB	USB_INT_REQ_LP, USB_INT_REQ_HP, USB_INT_REQ_DMA
25	41	0xA4	CAN	CAN Common, CAN 0 Tx, CAN 0 Rx, CAN 1 Tx, CAN 1 Rx
26	42	0xA8	GPDMA	IntStatus of DMA channel 0, IntStatus of DMA channel 1
27	43	0xAC	I^2S	irq, dmareq1, dmareq2
28	44	0xB0	Ethernet	WakeupInt, SoftInt, TxDoneInt, TxFinishedInt, TxErrorInt, TxUnderrunInt, RxDoneInt, RxFinishedInt, RxErrorInt, RxOverrunInt.
29	45	0xB4	Repetitive Interrupt Timer	RITINT
30	46	0xB8	Motor Control PWM	IPER[2:0], IPW[2:0], ICAP[2:0], FES
31	47	0xBC	Quadrature Encoder	INX_Int, TIM_Int, VELC_Int, DIR_Int, ERR_Int, ENCLK_Int, POS0_Int, POS1_Int, POS2_Int, REV_Int, POS0REV_Int, POS1REV_Int, POS2REV_Int
32	48	0xC0	PLL1 (USB PLL)	PLL1 Lock (PLOCK1)
33	49	0xC4	USB Activity Interrupt	USB_NEED_CLK
34	50	0xC8	CAN Activity Interrupt	CAN1WAKE, CAN2WAKE

نکته : ممکن است در یک برنامه چند وقفه داشته باشیم که هرچه عدد بردارهای وقفه کمتر باشد اولویت وقفه بالاتر است، همانطور که از جدول بالا مشخص است وقفه‌های خارجی(وقفه Interrupt) عدد بردار وقفه ایی کمتری دارند پس اولویت بالاتری نسبت به وقفه های داخلی دارند.

اگر در هنگام اجرای روال سرویس وقفه‌ایی، وقفه‌ی دیگری رخ دهد عکس العمل میکرو بدین گونه است،

که ابتدا زیربرنامه وقفه جاری را به اتمام می‌رساند و سپس زیربرنامه وقفه دوم را شروع به اجرا می‌کند

و سپس به برنامه اصلی برمی‌گردد.

حال عکس العمل میکرو در مواجهه با دو وقفه همزمان بدین صورت است که جدول بردار وقفه مراجعه می‌کند، با توجه به اولویت بندی، آن وقفه‌ایی که از اولویت بالاتری برخوردار است را اجرا و بعد اتمام آن به وقفه دوم رسیدگی می‌کند.

روال سرویس دهی وقفه هاISR

روش سرویس وقفه‌ها بصورت زیر می‌باشد البته شکل‌های دیگری هم دارد ولی این روش بدلیل شبیه بودن به تابع حفظ کردنش راحتر است.

```
void NAME_IRQHandler(void)
{
//user instructions
}
```

مثال :

```
#include "LPC17xx.h"
 void EINT3_IRQHandler ( )  {

                //
        }
int main(void){
  NVIC_SetPriority(EINT3_IRQn, 0);
  NVIC_EnableIRQ(EINT3_IRQn);
        while(1);
}
```

مراحل اجرای یک وقفه :

۱. ابتدا برنامه‌ایی که میکرو در حال اجرای آن است را به پایان می رساند.
۲. آدرس خط بعدی از برنامه اصلی را در حافظه پشته ذخیره می‌کند.
۳. با توجه به بردار وقفه به زیربرنامه وقفه مورد نظر پرش می‌کند وآنرا اجرا می‌کند.
٤. سپس به حافظه پشته برمی‌گردد و آدرس خط اصلی برنامه را می‌گیرد و به آن خط پرش می‌کند و ادامه برنامه اصلی را خط به خط اجرا می‌کند.

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 ۲۳۲-۱ تغییر کرده و سپس سر ریز می‌شود. یعنی بعد از پر شدن به حالت اولیه 0 برمی‌گردد و مجددا به صورت افزایشی ادامه می‌یابد.

نکته : در میکروکنترلر ARM به علت استفاده زیاد از موج PWM ، یک واحد مجزا برای تولید PWM در نظر گرفته شده است. و از واحد تایمر / کانتر فقط برای تولید زمان خاص (تایمر) یا برای شمارش (کانتر) استفاده می‌شود.

پایه‌های مربوط به واحد تایمر / کانتر

در میکروهای ARM چندین خروجی به 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 ورودی با نام CAPX.n وجود دارد که هر یک از آن‌ها در زمان رخداد پالس ورودی یک واحد به رجیستر شمارنده اضافه می‌کند.

	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

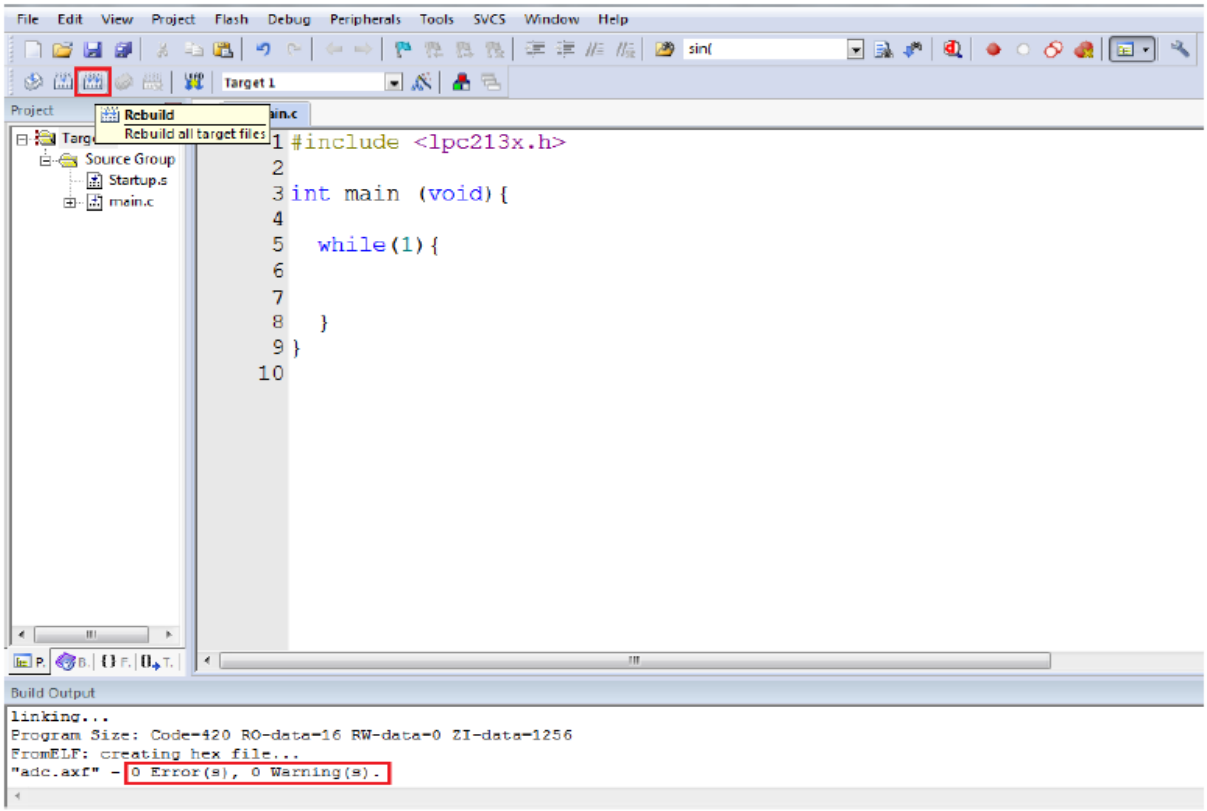
تابع راه اندازی تایمر در ARM LPC2138

```
#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;
}
```

تابع راه اندازی کانتر توسط TIMER0 روی CAP0.2

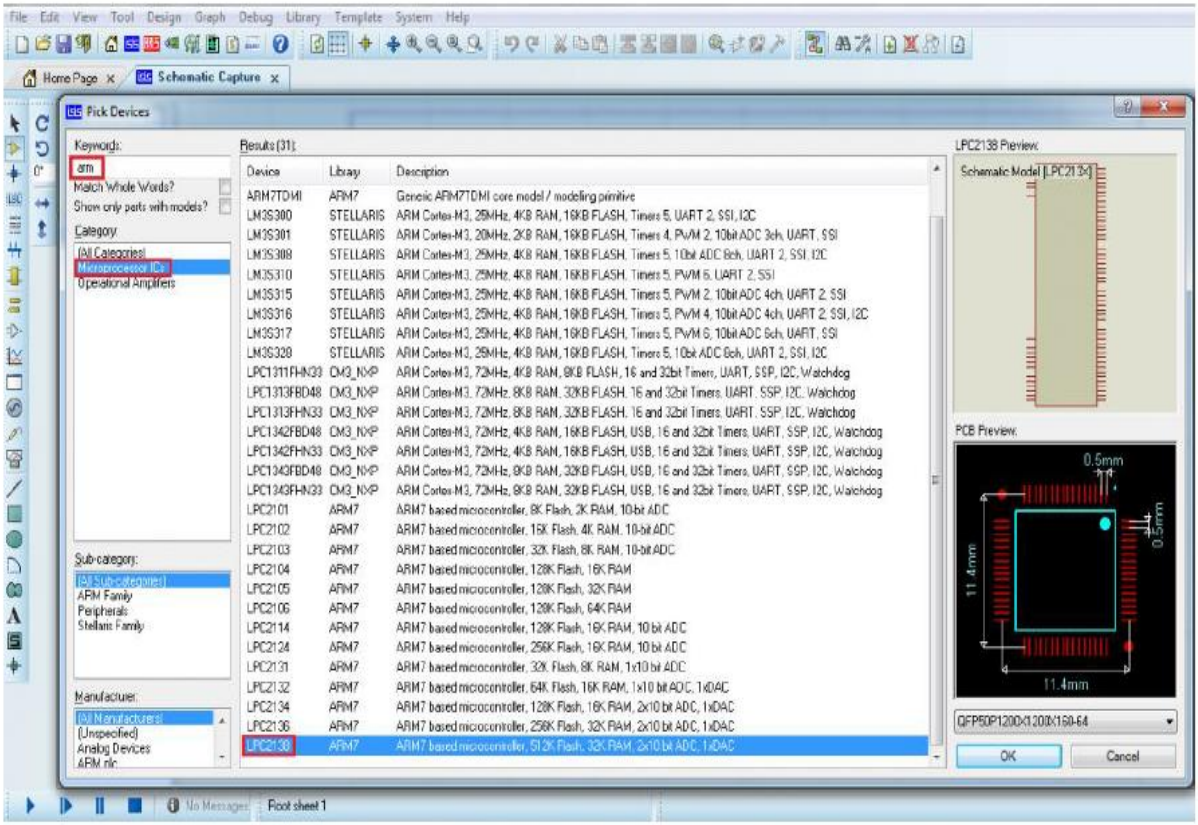
```
void counter0_init( void ) {
PINSEL1=(1<<25);
T0CTCR=9;
T0TC=0;
T0TCR=1;
}
```

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



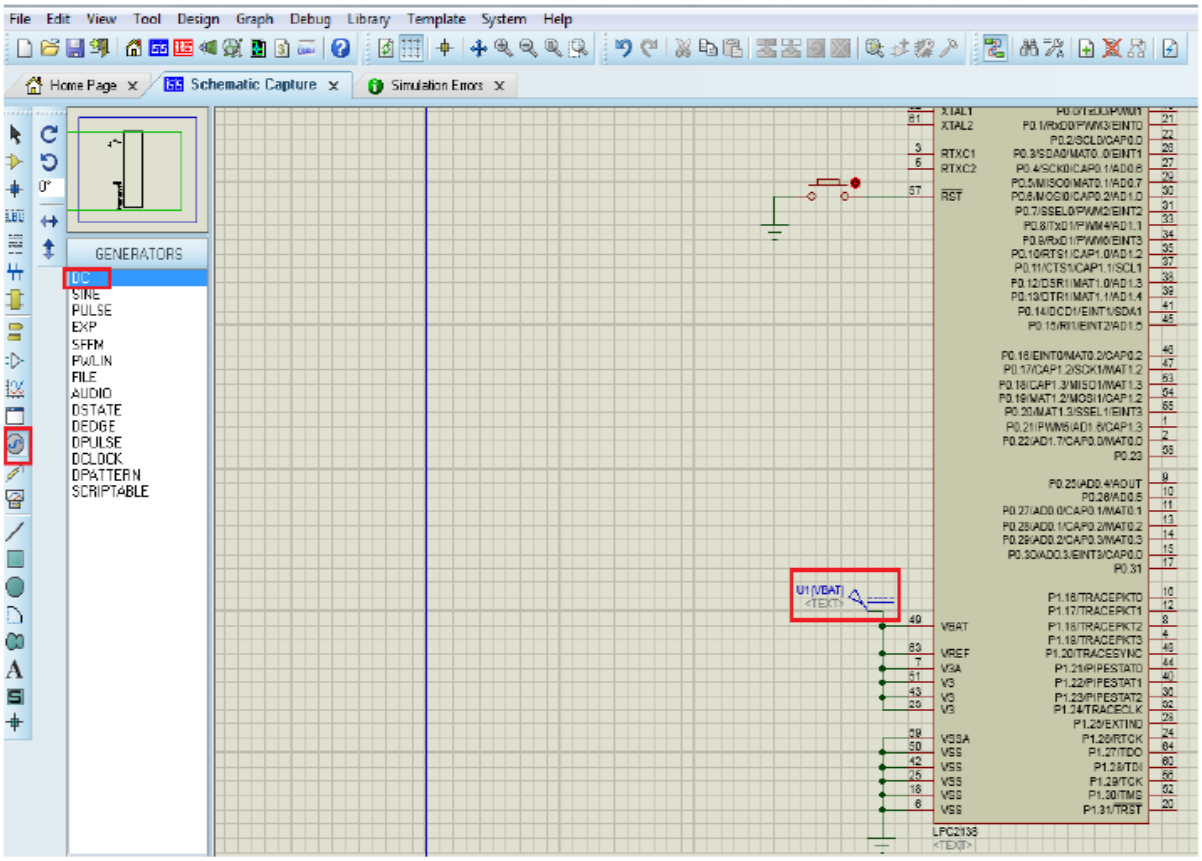
اضافه کردن فایل خروجی به شبیه‌سازی پروتئوس

برای شبیه سازی میکروکنترلرهای ARM می‌توان از خود نرم افزار Keil استفاده کرد. اما این شبیه سازی به صورت بسیار محدود صورت می‌گیرد. به طوری که تنها می‌توان تغییرات پایه‌ها و رجیسترها 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 بسیار نزدیک به واقعیت انجام دهد. برای اینکه بدانید نرم افزار پروتئوس از چه میکروکنترلرهای ARM پشتیبانی می‌کند، درون این نرم افزار رفته و در قسمت انتخاب قطعه عبارت arm را تایپ کنید و از کتابخانه‌های بوجود آمده روی Microprocessor ICs کلیک 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 می‌کنیم. سپس در نرم افزار پروتئوس به صورت زیر اتصال قطعات را تکمیل می‌کنیم.



نکته ۱ : چون پروتئوس صرفا یک نرم افزار شبیه سازی است می‌توان از قرار دادن کریستال و خازن ها و مقاومت های پایه‌های 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 آی‌سی از Generator در نرم افزار پروتئوس استفاده کردیم. بعد از جایگذاری این قطعه روی آن دابل کلیک کرده و ولتاژ آن را روی ۳/۳ ولت قرار می دهیم. در پایان روی میکرو LPC2138 دابل کلیک می‌کنیم و فرکانس کاری میکرو را روی ۱۲Mhz قرار می‌دهیم. سپس از قسمت انتخاب فایل می‌توانیم فایل Hex مورد نظرمان را برای شبیه سازی اضافه کنیم .

فصل پنجم

آشنایی مقدماتی با برنامه نویسی به زبان C در KEIL

بلوگ دیاگرام یک زبان برنامه به زبان C تقریبا به شکل زیر است :

- فراخوانی و راه اندازی پردازنده
- پیکربندی امکانات (مانند lcd و ...)
- معرفی متغیر ها
- شروع حلقه
- برنامه ای که باید انجام شود
- پایان حلقه

اولین خط در برنامه مربوط به معرفی پردازنده می‌باشد، معرفی پردازنده با دستور include # شروع شده و به.h ختم می‌شود. مثلا معرفی میکرو کنترلر lpc ۲۱۳۱ فیلیپس :

```
#include <LPC21xx.H>
```

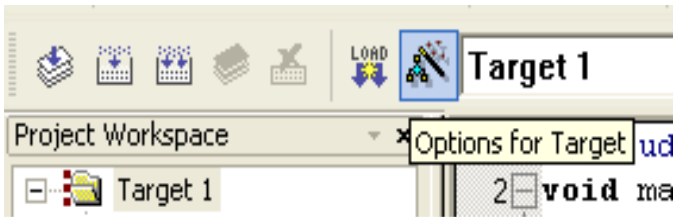
همان گونه که حدث زدید معرفی چیب براساس سری چیپ انجام می‌شود ، معرفی چیپ اصلی در قسمت ذخیره پروژه یا در options for target و در قسمت devise انجام می‌شود.

مثال چیپ STM32F101C8 ساخت شرکت STMicroelectronics را معرفی کنید :

در اولین خط برنامه عبارت زیر نوشته می‌شود :

```
#include <STM32F10x.h>
```

سپس در بالای پنجره project workspace و بر روی آیکون options for target کلیک کنید :



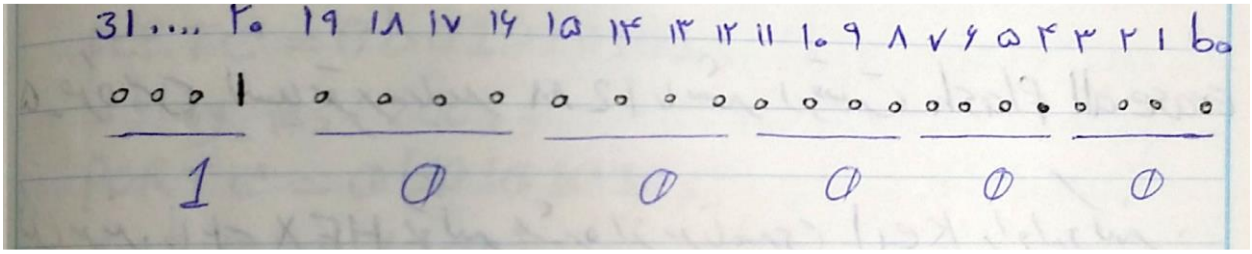
در پنجره باز شده ،به بخش Device بروید و STM32F101C8 را از کتابخانه STMicroelectronics انتخاب نمایید.

پیکربندی پورت ها:

در زبان C برای پیکربندی پورت ها باید اطلاعات را در رجیستر آن‌ها نوشت ، در keil برای رجیستر هر پورت یک نام انتخاب شده است که شما باید این رجیستر ها را به خاطر بسپارید.

رجیستر وضعیت IODIRx :

این ثبات (یا رجیستر) مشخص کننده وضعیت پورت است، پورت می‌تواند به صورت خروجی یا ورودی باشد، ورودی یا خروجی بودن پورت با نوشتن داده هگز در آن معین می‌شود. برای ورودی بودن باید صفر باشد هر پایه و برای خروجی بودن آن پایه رجیسترش باید یک باشد. از آنجایی که ARM دو پورت دارد، پورت صفر و پورت یک و هر پورت ۳۲ پایه دارد. برای تعیین وضعیت رجیستر هشت کاراکتر در نظر می‌گیریم هر کاراکتر بیانگر ٤ پایه است و پایه شماره ۱ از راست شروع می شود یعنی کاراکتر اول چهار پایه اول و کاراکتر دوم از راست پایه‌های ۵ و ٦ و ۷ و ۸ و به همین ترتیب آخرین کاراکتر از راست پایه های ۲۹و۳۰ و۳۱ و۳۲ می‌باشد.



هر دسته چهارتایی برای تبدیل باینری به هگز از ارزش ۱ ۲ ٤ ۸ برای تبدیل مبنا استفاده می‌کنیم (به تبدیل مبنا از ۲ به ۱٦ مراجعه شود) در ضمن لازم به یادآوری است که در این تبدیل مبنا وقتی جمع ارزش‌های هربیت از ۹ بیشتر شود A=۱۰ و B=11 و C=12 و D=13 و E=14 و D=15 و F=16 می‌گذاریم.

Decimal	Binary	Hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

مثلا تمام پایه‌های پورت صفر به عنوان خروجی تعریف شده است :

IODIR0 = 0xFFFFFFFF;

یا پورت یک به عنوان ورودی تعریف شده است:

IODIR1 = 0x00000000;

نکته : در زبان برنامه نویسی C تمامی دستورات به ; ختم می‌شوند.

به علت اینکه رجیسترها ۳۲ بیتی هستند ، در اکثر اوقات به صورت) hexadecimal مبنای ۱۶ (آن ها را

مقدار دهی می کنیم

ثبات داده:

وضعیت پورت (صفر یا یک بودن) توسط ثبات IOSET و IOCLR مشخص می‌شود.

IOSET برای یک کردن هر پایه و IOCLR برای صفر منطقی کردن هر پایه استفاده می‌شود.

اولی پایه‌های زوج پورت ۸ پینی یک دروضعیت یک و پایه‌های فرد در وضعیت صفر و دومی نیز پایه‌های متناظر یک میکروی ۳۲ بیتی یک منطقی قرار گرفته‌اند :

```
IOSET1 =0XAA;
IOSET0 =0X0FF01B5D;
```

کد نوشته در زیر نیز پایه‌های متناظر را صفر منطقی می‌کند. مثلا در دستور دوم تمام پایه‌های پورت یک، صفر منطقی یا خاموش می‌شود.

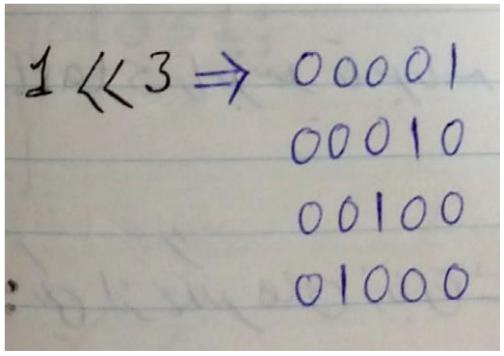
```
IOCLR0 = 0XE1A0A589;
IOCLR1 = 0FFFFFFFF;
```

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 (ShiftوOr) بنویسیم:

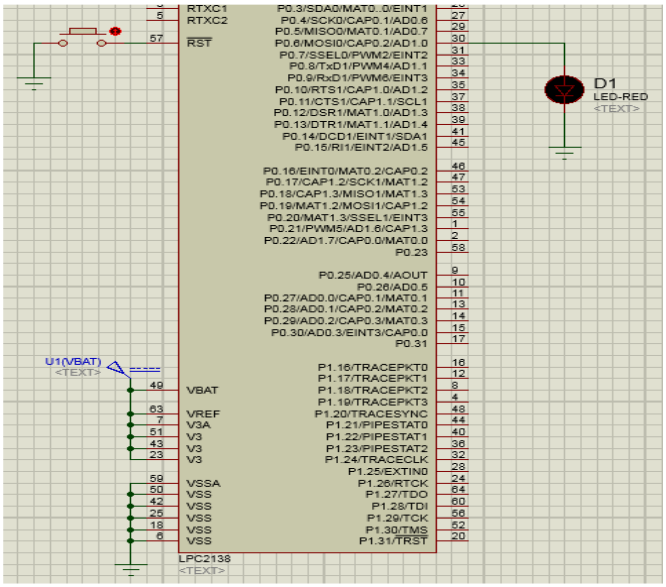
```
IO1DIR |= (1<<3);
IO1SET |= (1<<3);
```



مثال : برنامه‌ای بنویسید که تمامی پایه‌های چیپ LPC2131 را یک منطقی کند؟

```
#include <LPC21xx.h>
int main (void){
IODIR0 = 0xffffffff;
IODIR1 = 0xffff0000;
IOSET0=0xffffffff;
IOSET1 =0xffff0000;
}
```

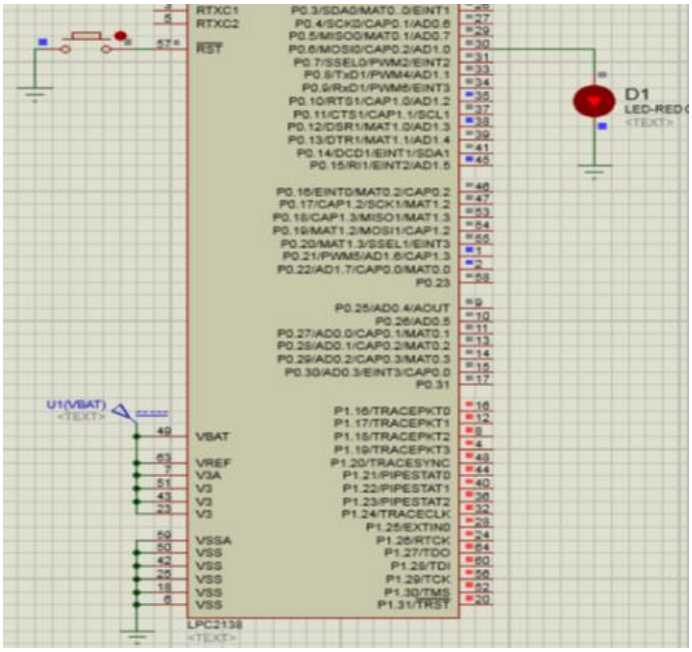
مثال : برنامه‌ایی بنویسید که LED موجود روی P0.6 را با سرعت قابل دیدن به صورت چشمک زن روشن و خاموش کند؟ سپس آنرا در نرم افزار Proteus شبیه سازی کرده و پس از اطمینان از عملکرد صحیح برنامه روی میکروکنترلر LPC2138 پیاده سازی نمایید؟




```
#include <lpc213x.h>
void delay(void){
unsigned int i=1000000;
while(i--);
}
int main (void){
IODIR0 |= (1<<6);
while(1){
IOSET0 |= (1<<6);
delay();
IOCLR0 |= (1<<6);
delay();
}
}
```

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.

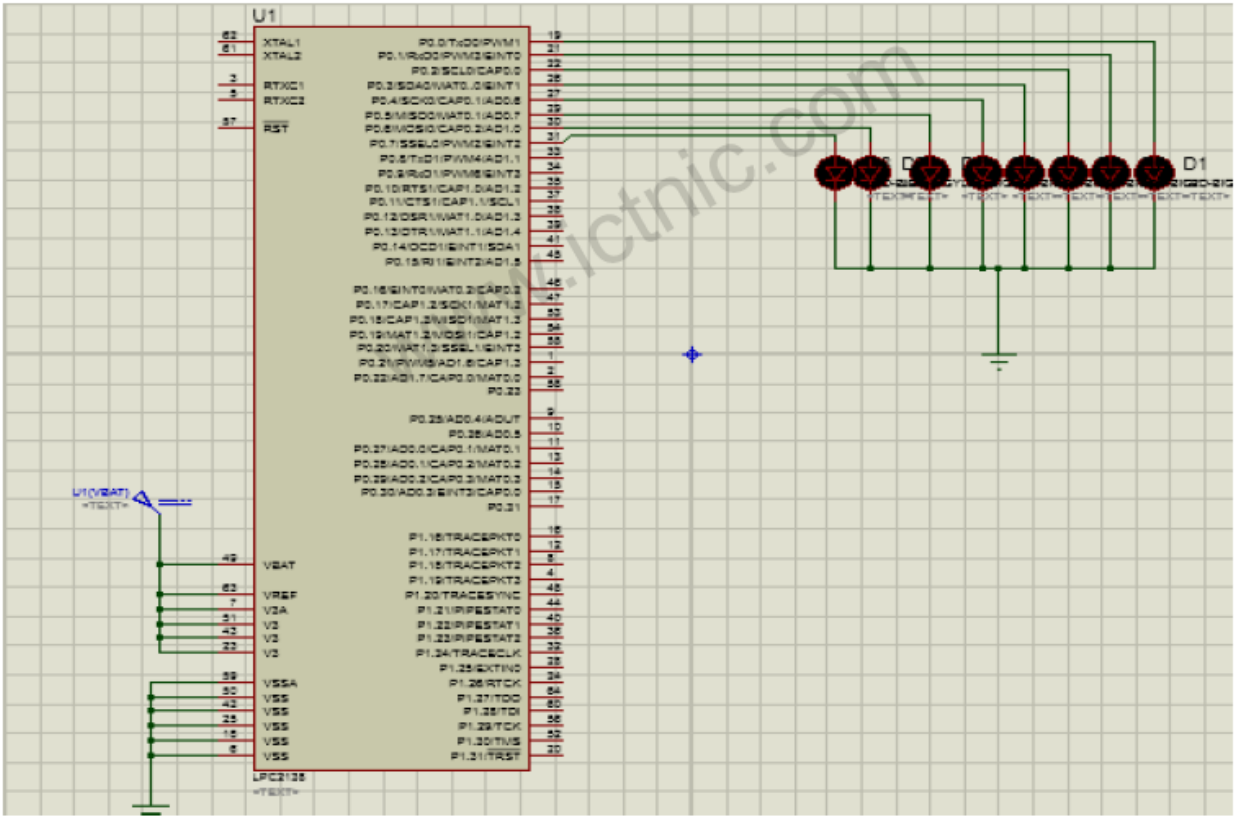


نکته : در برنامه نویسی ARM و کامپایلر KEIL هدر فایلی شبیه delay.h که در AVR و کامپایلر Codevision بود وجود ندارد تا آن را به برنامه اضافه کرده و تابع delay به اندازه میکروثانیه یا میلی ثانیه داشته باشیم .بنابراین مشابه چنین تابعی خودمان در برنامه تعریف می کنیم.

مثال : برنامه‌ای بنوسید که تمام پایه‌های زوج LPC2131 را صفر وتمام پایه‌های فرد آنرا یک کند؟

```
#include <LPC21xx.h>
int main (void){
IODIR0 = 0xffffffff;
IODIR1 = 0xffff0000;
IOSET0=0xaaaaaaaa;
IOSET1 =0xaaaa0000;
}
```

مثال : ۸ عدد LED را به یکی از پورت های ARM متصل کرده آن‌ها را یکی در میان روشن کنید؟نیم ثانیه در این حالت نگه دارید سپس LEDهای خاموش و روشن را عوض کرده نیم ثانیه نیز در این حالت نگه دارید و این عمل مداوم تکرار شود.




```
#include <LPC21xx.h>
Void delay(void){
Int i;
For (i=0;<0x6ffff;i++);
}
int main (void){
IODIR0 = 0xffffffff;
While(1){
IOCLR1 = 0xffffffff;
IOSET0=0xaaaaaaaa;
```

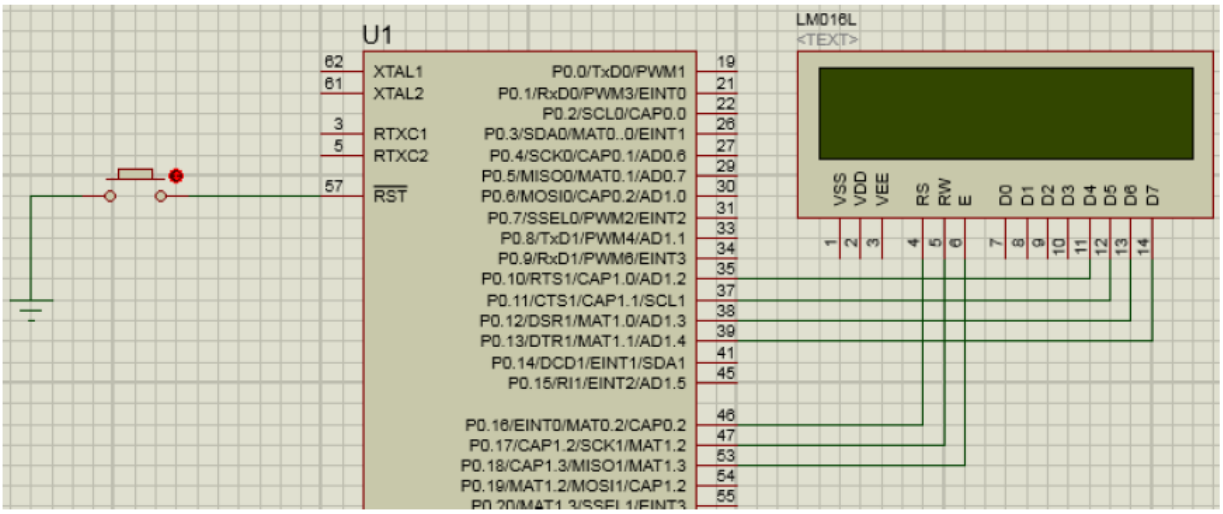
```
Delay();
}
}
```

راه اندازی LCD کاراکتری

ال‌سی‌دی‌های کارکتری اغلب دارای ۱٤ پایه هستند که ۸ پایه برای انتقال اطلاعات ۳ پایه برای کنترل نرم افزاری یک پایه برای کنترل سخت افزاری و دو پایه تغذیه می‌باشد در جدول زیر اطلاعات پایه‌ها را مشاهده می‌کنید :

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

lcdها را می‌توان با ۸ خط دیتا یا ٤ خط دیتا راه اندازی کرد اما معمولا آن را با ٤ خط دیتا راه اندازی می‌کنند 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 ٤ خط دیتا در این است که داده های ۸ بیتی LCD به جای یکبار، در دو مرحله ٤ بیتی ارسال می شوند. مزیت راه اندازی با ٤ خط دیتا در این است که اتصال LCD به میکروکنترلر تنها ۷ پایه از میکرو را اشغال می‌کند. این ۷ خط دیتا را می‌توان به هر پایه دلخواهی از میکرو متصل نمود اما بهتر است در این اتصال از پورت‌های کنار هم استفاده نمود. در اینجا ما LCD را 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 می‌کنیم 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 می‌کنیم 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.

توابع کار با LCD کاراکتری درARM :

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.

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.

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);

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

lcd_putchar(‘ کاراکتر ‘);

مثال:

lcd_putchar(‘A’);

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

lcd_print(“ رشته “);

lcd_print(ای رشته متغیر نام);

مثال:

lcd_print(“ MICRO 255”);

lcd_print(str);