مرجع کامل میکرو کنترلر های سری

AT91SAM شرکت ATMEL

تالیف و گرد آوری :

1nafar:1nafar@1nafar.com

Codev:codev_armin@hotmail.co.uk

Farzadsw: http://farzadsw.persianblog.ir/

مقدمه :

در اوایل سال 1388بود که بحثهای بر سر میکروکنترلرهای جدید و پرقدرتی که با نام ARM شناخته می شد، در انجمن های برق و الکترونیک ایرانی ایجاد شد .در این بین در انجمن ایران میکرو با همت تعدادی از کاربران و البته پشتیبانی گروه کویر الکترونیک (تامین بردهای آموزشی ،میکروکنترلر و سایر ادوات مورد نیاز) ، آموزش میکروکنترلر های ARM به زبان فارسی آغاز شد . کاربران مذکور که نام "گروه ARM " را برای خود انتخاب کرده بودند ، مطالب آموزشی مربوط به این میکروکنترلرها را در مجلاتی با نام PMM (مجله میکروکنترلر فارسی – Persian Microcontroller Magazine) که به صورت رایگان و در فرمت PDF و در اواخر هر ماه منتشر میشد ، در اختیار علاقه مندان قرار میدادند. این گروه میکروکنترلر های مبتنی بر هسته ی ARM شرکت اتمل را برای آموزش انتخاب کرده بود (تنها میکروکنترلر مبتی بر هسته ی ARM که در آن زمان در بازار وجود داشت) .

بعد از گذشت تقریبا یک سال از آغاز به کار گروه ، با انتشار نهمین شماره از مجلات PMM ، آموزش میکروکنترلر های مبتنی بر هسته ی ARM اتمل به پایان رسید و... .

در مورد میکروکنترلرهای مبتنی بر هسته ی ARM اتمل بیش از 500 صفحه آموزش فارسی توسط گروه منتشر شد و ما به امروز رسیدیم . امروز در بازار الکترونیک کشور محصولات شرکت های دیگری همچون ST ، فیلیپس ، TI و... یافت میشود و کتاب های آموزشی مختلفی برای این میکروکنترلر به صورت رایگان در سطح وب ، یا به صورت تجاری در کتاب فروشی ها وجود دارد .

کتابی که در حال مطالعه ی آن هستید ، مرجع کامل میکروکنترلرهای سری AT91SAM شرکت ATMEL نام دارد . که در شش بخش / حالت / فرمت زیر منتشر خواهد شد:

1- مباحث مقدماتی : در این بخش شما با میکروکنترلر های مبتی بر هسته ی ARM آشنا شده و نحوه ی استفاده از آنها در کامپایلر KEIL را فرا گرفته و بعد از آشنایی با برخی از دستورات زبان C ، می آموزید که چطور از ورودی/خروجیهای این میکروکنترلرها استفاده کرده و چطور کتابخانه های مورد نیاز خود را ایجاد کنید .
2- راه اندازی منابع تامین کلاک : در این بخش شما با نحوه ی پیکربندی منابع کلاک میکروکنترلر آشنا می شوید .
3- راه اندازی واحد تایمر/ کانتر و راه اندازی LCD گرافیکی رنگی : در فصل اول این بخش با نحوه ی راه اندازی راه اندازی واحد تایمر/ کانتر و در فصل بعدی با اصول راه اندازی LCD گرافیکی رنگی آشنا خواهید شد .
4- پروتکل های ارتباطی : در این بخش با نحوه ی راه اندازی و استفاده از پروتکل های ارتباطی موجود در این میکروکنترلر ها ، شامل پروتکل USART ، پروتکل SPI ، پروتکل I2C و پروتکل I2S آشنا خواهید شد .

5- راه اندازی امکانات جانبی دیگر: در این بخش در مورد راه اندازی بخش های مانند مبدل آنالوگ به دیجیتال ، واحد دیباگ یونیت و... بحث شده است .

6- نمایشگر های گرافیکی : در این بخش با اصول رراه اندازی LCD گرافیکی آشنا میشوید .

این کتاب به دلایل زیر در چند بخش منتشر شده است :

1- بخش های 1 و 4 و 5 به صورت آزاد و در فرمت Docx منتشر شده است ، در این رابطه اطلاعات بیشتری در آدرس زیر وجود دارد :

http://www.iranmicro.ir/forum/showthread.php?t=12189

2- بخش های 2 و 3 و 6 به صورت متن باز و در فرمت pdf منتشر شده است ، اطلاعات بیشتری در این مورد در پست شماره 11 تاپیک بالا وجود دارد .

3- در تدوین مطالب این کتاب (جمع آوری و درج آنها در مجلات PMM) 3 نفر به صورت مستقیم حضور داشته اند که نام هر نفر در بخشی که تهیه کرده ، آورده شده است .

4- مطالب بخش های 4 و5 تا حدودی ناقص بوده و نیاز به ویرایش دارند . برای کمسب اطلاعات بیشتر به آدرس اینترنتی درج شده در بند یک مراجعه کنید .

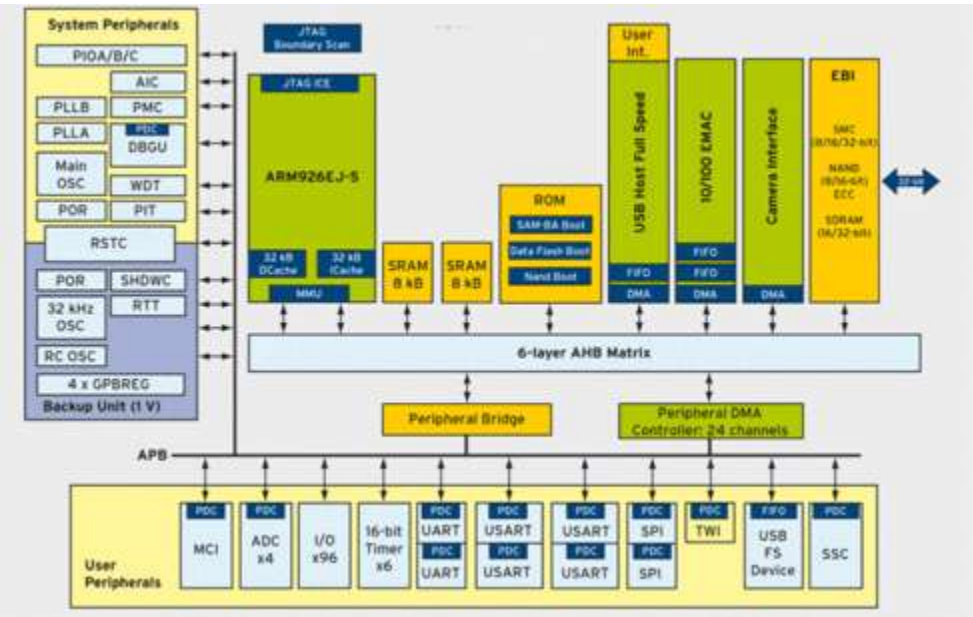
1nafar

www.1nafar.com

فصل صفر

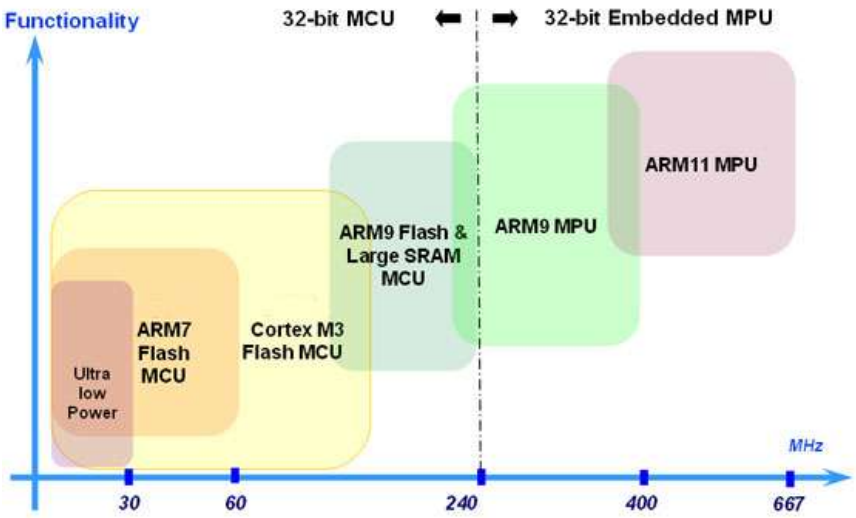
در این فصل به معرفی کلی میکرو کنترلر های سری AT91SAM پرداخته ایم ، ما امکانات کلی موجود در این خانواده و کاربرد ان را شرح داده ایم ، برای کسب اطلاعات بیشتر میتوانید به بخش ضمائم و دیتا شیتهای فارسی مراجعه نمایید . در صورتی که احساس میکنید با میکروکنترلر های ARM اشنایی دارید ، نیازی به خواندن این فصل نخواهید داشت .

در اوایل دهه ی 80 اولین پردازنده ی ARM در شرکت ACRON طراحی و به بازار ارائه گردید . تقریبا 80 سال بعد و در سال 1990 با پیوستن شرکت های APPLE و VLSI به کمپانی ACRON شرکت ARM تاسیس شد . این شرکت با طراحی معماری های مختلف از پردازنده های RISC و ارائه ی آن به بازار توانست در مدت کوتاهی 75 درصد بازار پردازنده های 32 بیتی را در دست بگیرد .



به بیان ساده تر ، ARM فقط یک معماری (یا یک نقشه) میباشد که به شرکت های تولید کننده ی میکرو کنترلر فروخته میشود و آنها باید تحت لیسانس شرکت ARM آن را تولید نمایند . وجود معماری متناسب و قابلیت توسعه ی سخت افزار به صورت نامحدود و.... از ویژگی های این معماری میباشد .

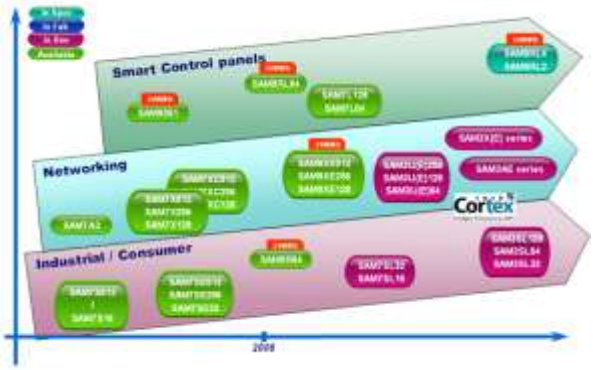
هسته های که تاکنون ارائه شده اند ARM11 , ARM9 , ARM7 و... میباشند . همچنین این هسته ها خود ، دارای زیر مجموعه های همچون Cortex-M3 و Cortex-M3 هستند . شما میتوانید به مراجعه به منابع موجود در انتهای کتاب اطلاعاتی بیشتری را در مورد تاریخچه ی ARM بدست آورید .



امروزه بیش از 120 شرکت میکرو کنترلر های مبتنی بر هسته ی ARM را تولید میکنند که حاصل این تولیدات بیش از 9 هزار میکرو کنترلر است . به دلیل یکسان بودن هسته در تمامی میکرو کنترلرها (محصولات شرکت های مختلف) کار با تمامی آنها تقریبا یکسان است و کاربرانی که با محصولات یک شرکت آشنایی کامل داشته باشند به سادگی میتوانند با محصولات شرکت های دیگر کار کنند .

در این بین شرکت های نرم افزاری نیز فعالیت گسترده ای دارند ، در دنیای نرم افزار بیش از 50 کامپایلر برایARM وجود دارد که میتوانیم کامپیالر های معروف KEIL و IAR را در این زمینه معرفی نماییم .

بررسی سری AT91SAM :



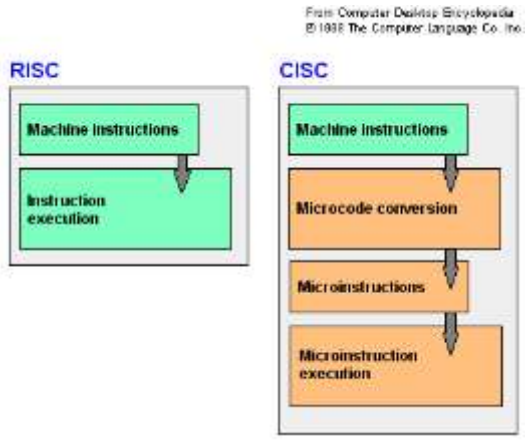
در زمان نگارش این کتاب ، شرکت اتمل دارای 32 محصول مبتنی بر هسته ی ARM میباشد ، این محصولات دارای امکانات تقریبا مشابهی میباشند و تفاوت اصلی آنها در کلاک پردازنده و مقدار حافظه ی جانبی وتعداد I/O و امکانات جانبی است . در ادامه به بررسی ویژگی های اصلی این گروه پرداخته ایم :

(توجه داشته باشید که این ویژگی ها در تمامی میکروکنترلر های ARM (حتی میکروکنترلر های تولید شده توسط شرکت های دیگری همچون ST یا NXP یا ...) وجود دارد و ما جهت جلوگیری از پراکندگی مطلب و تخصصی بودن بحث ، این ویژگی ها را برای میکروکنترلر های AT91SAM اتمل شرح میدهیم)

ساختار داخلی:

استفاده از معماری RISC : .

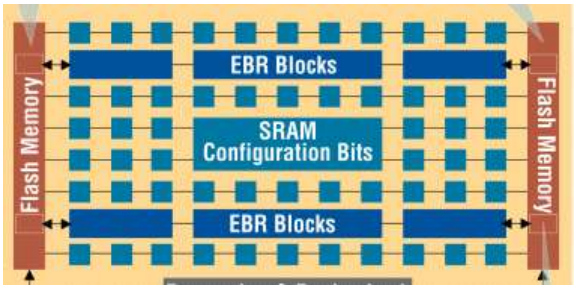
معماری (RISC (Reduced Instruction Set Coding/Computer نوعی طراحی برای سخت افزار های میکرو الکترونیک میباشد ، در این معماری با تغییرات سخت افزاری مجموعه دستورات برنامه نویسی کم میشود،در مقابل این معماری ، معماری(CISC (Complex InstructionSet Coding قرار دارد ، در معماری CISC حجم دستورات زیاد تر میشود ، اما سخت افزار میکرو کنترلر ساده تر میگردد .



وجود High-speed Flash و SRAM .

Static random access memoryیا SRAM نوعی حافظه موقت میباشد ، که اطلاعات قبل از آنکه به CPU برسد در آن ذخیره میشود . با این کار دیگر نیاز نیست وقت CPU برای دریافت اطلاعات از حافظه فلش تلف شود ، در هنگام پردازش اطلاعات توسط CPU اطلاعات از فلش به این حافظه منتقل میشود و همیشه اطلاعات در دسترس CPU خواهد بود..

برای دستیابی به سرعت بالا و تازگی اطلاعات موجود در SRAM ، حافظه فلش میکرو کنترلر های arm طوری طراحی میشوند که بتوانند با سرت بالا 30 مگاهرتز اقدام به تبادل داده با sram و cpu نمایند .



میکرو کنترلر های ها سری AT91SAM مجهز به واحد های MC و RSTC و PMC و AIC و ...میباشند ، وجود این واحد باعث مصونیت میکرو در برابر نویز میشود و پایداری کامل آن در کلیه شرایط را تضمین میکند .

واحد MC یا Memory Controller وظیفه کنترل کردن حافظه فلش و رفع خطا های احتمالی ان را به عهده دارد .

واحد RSTC یا Reset Controller وظیفه باز نشانی میکرو در شرایط اضطراری همچون کم شدن ولتاژ تغذیه و ...را به عهده دارد .این واحد امکان باز نشانی میکرو از طریق پایه nrst (ریست دستی) را نیز فراهم اورده است.

واحد PMC یا Power Management Controller با فعال بودن این واحد ، میکرو میتواند عملکرد خود را با توجه به ولتاژ تغذیه تنظیم کند . میکرو میتواند در شرایط کمبود ولتاژ و جریان تغذیه ، فرکانس کاری خود را تا زیر 500 هرتز کاهش دهد.

یکی از بخش های مهم در هر میکرو کنترلر بخش مدیریت وقفه ها میباشد ، در این خانواه واحد AICیا Advanced Interrupt Controller عملیات کنترلر کردن وقفه ها را به عهده دارد .

راهکارهای شبکه :

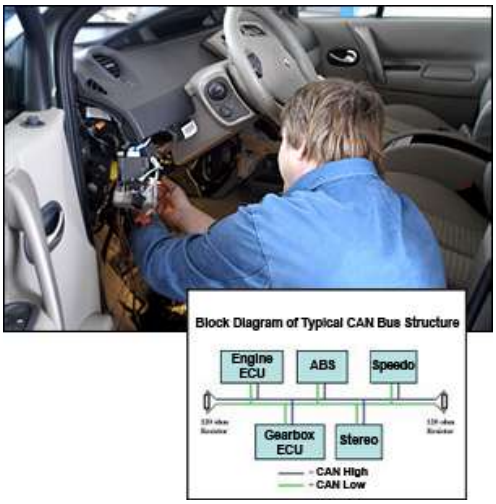
Ethernet :

تراشه های سری AT91 دارای یک ماژول اترنت که به صورت EMAC module طراحی شده میباشد.این ماژول با استفادندارد IEEE802.3 سازگار بوده و میتواند با سرعت های 10 و 100 مگابیت بر ثانیه مورد استفاده قرار گیرد .قابلیت mac یا media access control به شما کمک میکند تا با اتصال میکرو کنترلر به تراشه های مختلف ، پروتکل های مختلفی همچون TCP/IP ، FTP و... را بر روی تراشه پیدا سازی کنید

.

CAN :

Controller–area network ((CAN or CAN-bus یک پروتکل ارتباطی برای متصل کردن میکرو کنترلرها و سایر وسایل الکترونیک به یکدیگر بدون نیاز به وسیله کنترل کننده میباشد.



نمونه قابل مشاهده کاربرد can در ECU اتومبیل میباشد . سیستم ECU تمامی امکانات اتومبیل را کنترل میکند ، امکانی از قبیل ، چراغ ها ، سیستم صوتی ، درب ها و موتور ، سیستم سوخت رسانی و ....قطعا برای اتصال تمامی این امکانات نیاز به تعداد زیادی سیم خواهیم داشت ، اما با پروتکل can و از طریق دو سیم تمامی موارد را به پردازنه اصلی مرتبط میکنیم .

Serial Synchronous Controller (SSC) :

SSC یکی از امکانات موجود در arm میباشد که به شما امکان کار با قطعات جانبی که از پروتکل I2S استفاده میکنند (دارای پورت I2S هستند) را میدهد ، I2S (Inter-IC Sound) یک پروتکل سه سیمه برای منتقل کردن داده است ، در این پروتکل داده موجود معمولا صدا های دیجتال میباشد .در این سیستم یک خط انتقال سریال برای تبادل داده بین دو دستگاه یا قطعه صوتی ایجاد میشود ، شما میتوانید پورت i2s را در پشت اغلب دستگا های صوتی و تصویری بیابید . .

پروتکل SSC یک رابط همزمان برای ارتباط با کلیه قطعات و لوازمی میباشد که به صورت همزمان اقدام به ارسال و دریافت داده میکنند ، برای نمونه میتوان به I2S, Short Frame Sync, Long Frame Sync, اشاره کرد ، کلیه این پروتکل ها داده را به صورت همزمان به خروجی ارسال میکنند .

USART :

این پروتکل تقریبا در تمامی وسایل الکترونیکی استاندارد استفاده میشود ، ما در کامپیوتر آن را به نام پروتکل RS232 و خروجی آن را نام پورت com میشناسیم . میکروکنترلر های سریat91sam عموما دارای دو پورت usart میباشند در این

خانواده امکان استفاده از این پورت در حالت های مختلفی همچون ISO7816 T0/T1 Smart Card ، IRDA ، ارتباط سریال همزمان و غیر همزمان ، RS485 ، RS232 و... وجود دارد .

SPI:

Serial Peripheral Interfaceیا پروتکل spi یک رابط سریال برای انتقال داده به صورت همزمان بین دو وسیله میباشد ،اکثر میکرو کنترلر های arm دارای دو عدد خط ارتباطی SPI مجزا میباشد ، سرعت بالا در انتقال داده ، قابلیت کنترل کردن چندین اسلیو به صورت مجزا ، امکان اتصال به تراشه های حافظه و... از ویژگی های بارز این بخش است .

Two-wire :

پروتکل Two-wire یا ارتباط دو سیمه ، یک پروتکل ارتباطی سریال میباشد . در این پروتکل از دو سیم برای تبادل داده و کلاک ، و سیم های گراند و vcc استفاده میشود . در این پروتکل میتوان توسط دو سیم تعداد 128 وسیله را به هم متصل نمود ، هر وسیله دارای یک ادرس منحصر بفرد میباشد و توسط همان ادرس شناسایی میشود .

مدیریت حافظه و برنامه ریزی:

برای برنامه ریزی این خانواده روش های مختلفی وجود دارد :

توسط SAM-BA ® Boot میتوانید میکرو کنترلر را از طریق پورت usb و بدون نیاز به سخت افزار پرگرامر به راحتی پرگرام کنید .

تمامی میکرو کنترلر های خانواده ARM از واسط JTAG پشتیبانی میکنند. در پروتکل JTAG دسترسی کامل به CPU و حافظه ها فراهم میباشد ، و شما میتوانید داده های پردازش شده یا در حال پردازش توسط آنها را مشاهده کنید.

وجود واحدDebug Unit کار عیب یابی و تست سخت افزار را بسیار ساده کرده است ، در این خانواده پورتی مجزا برای انتقال داده از میکرو به کامپوتر و بالعکس تعبیه شده است که توسط آن میتوانید وضعیت کلیه رجیستر ها و متغیر های دلخواه را در کامپیوتر خود مشاهده کنید .

هیچ کمبودی وجود ندارد ...

میکروکنترلر های مبتنی بر هسته ARM ...

دارای تعداد زیادی خطوط I/O میباشند ، این خطوط در تمامی میکرو کنترلر ها قابل برنامه ریزی اند . برای مثال ، میکرو AT91SAM7X256 دارای دو پورت میباشد ، این دو پورت تعداد 62 خط ورودی و خروجی قابل برنامه ریزی را در اختیار شما میگزارد.

انجام محاسبات در قالب های 16 و 32 بیتی ، یکی دیگر از مزیت های اصلی این خانواده است . با این قابلیت ، امکان اتصال تمامی وسایل جانبی نظیر هارد دیسک ، فلش های سخت افزاری و... به سادگی میسر خواهد بود و میکرو میتواند در یک کلاک خروجی تمامی آنان را بخواند .

دارای تایمر کانتر ها 16 تا 24 بیتی میباشند . همچنین در این میکرو ها خروجی های 16 PWM بیتی برای کار کنترلی و کار با صوت ایجاد شده است .

بسیار کم مصرف میباشند . همانطور که قبلا گفتیم واحد PMC یا Power Management Controller وظیفه تنظیم کردن مصرف توان میکرو را به عهده دارد . مد های کم مصرف یا sleep میکرو کنترلر های arm گاهی به 7 مورد میرسد . در بعضی از میکرو کنترلر ها جریان مصرفی میتواند تا زیر 1 میکرو آمپر نیز کاهش یابد .

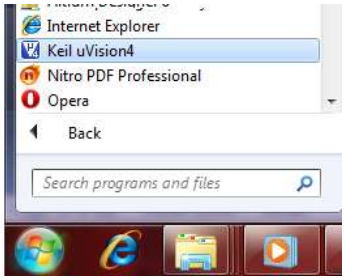
وجود مبدل های انالوگ به دیجتال 10 تا 16 بیتی ، RTC داخلی ، ورودی های وقفه و.... تماس با دنیای واقعی را ساده کرده است .

فصل یک : گذری کوتاه در کامپایلر keil و اشنایی با ....

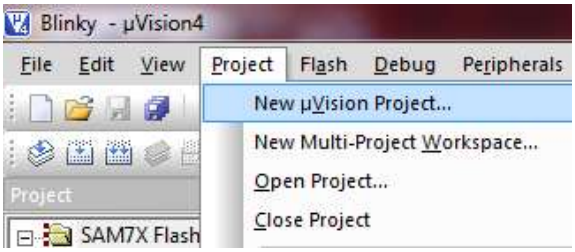
در این فصل قصد نداریم تا با معرفی منو ها و موارد اضافه شما را خسته و دل زده کنیم ، به همین جهت به سراغ اصل مطلب رفته ایم . در این فصل شما یاد خواهید گرفت که چگونه با نرم افزار KEIL کار کنید . شما با نحوه ی ایجاد پروژه ، نوشتن برنامه ، کامپایلر کردن برنامه ، شبیه سازی برنامه و برنامه ریزی میکروکنترلر آشنا خواهید شد . تمامی موراد مورد نیاز برای برنامه ریزی یک میکرو کنترلر در این فصل وجود دارد ، در فصل های بعدی ما مطالب این بخش را تکمیل تر خواهیم کرد .

ایجاد پروژه و نوشتن برنامه :

بعد از اینکه نرم افزار را نصب کردید آن را باز کنید(مراحل نصب در بخش ضمائم موجود است) :

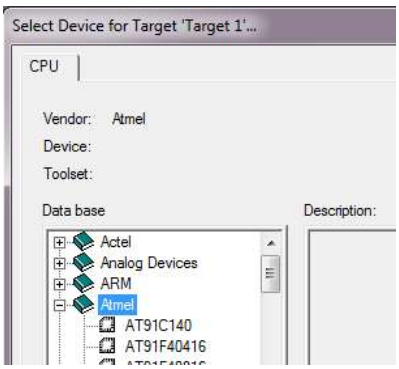


برای نوشتن برنامه به زبان c ابتدا باید یک پروژه ایجاد کنید ، به منوی project برید و در انجا گزینه ی new uvision project را انتخاب کنید :

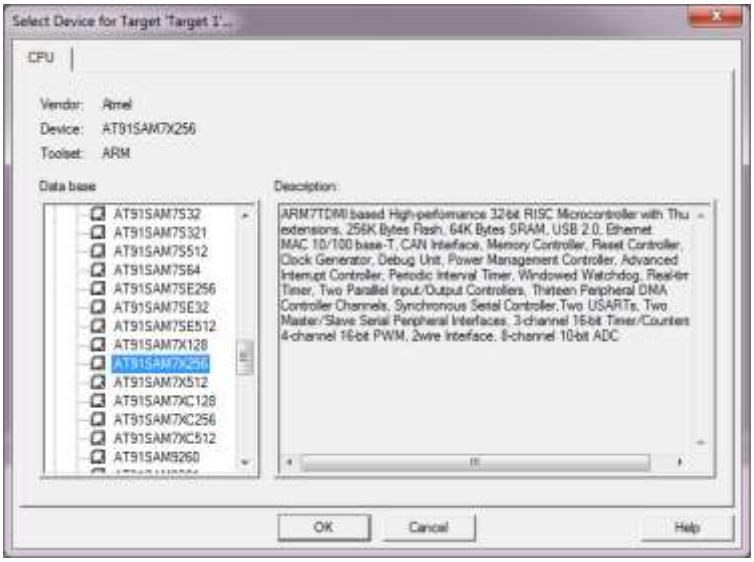


در پنجره ای که باز میشود یک نام مناسب برای پروژه وارد کنید و آن را در مسیر دلخواه ذخیره کنید .

بعد از انجام عملیات ذخیره سازی پنجره ای باز میشود ، در این پنجره شما باید پردازنده مورد نظر خود را انتخاب کنید (پردازنده ای که میخواهید برایش برنامه بنویسید) جهت هماهنگی با مطالب کتاب از شاخه ی Atmel ،پردازنده ی AT91SAM7X256 را انتخاب کنید و سپس بر روی ok کلیک کنید ، پیغامی که ظاهر میشود را نیز تایید کنید تا فایل های Startup به پوشه ی پروژه اضافه شود .

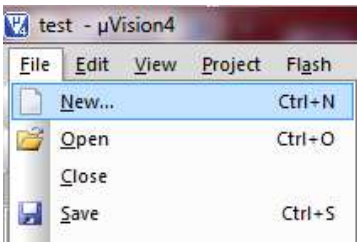


مطالب کتاب در مورد تمامی اعضای خانواده ی at91sam صادق است و میکروکنترلر AT91SAM7X256 صرفا جهت ایجاد هماهنگی در مطالب کتاب انتخاب شده است . شما در آینده میتوانید پردازنده دلخواه خود را در این پنجره انتخاب کنید .

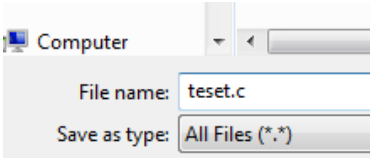


در این پنجره ی خلاصه ای از دیتا شیت میکرو کنترلر وجود دارد ، در این خلاصه امکانات میکرو کنترلر اورده شده است .

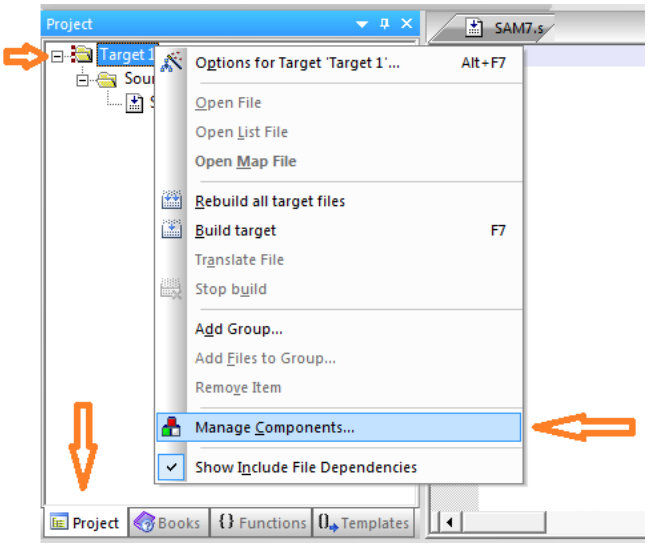
از منوی file گزینه ی new را انتخاب کنید :



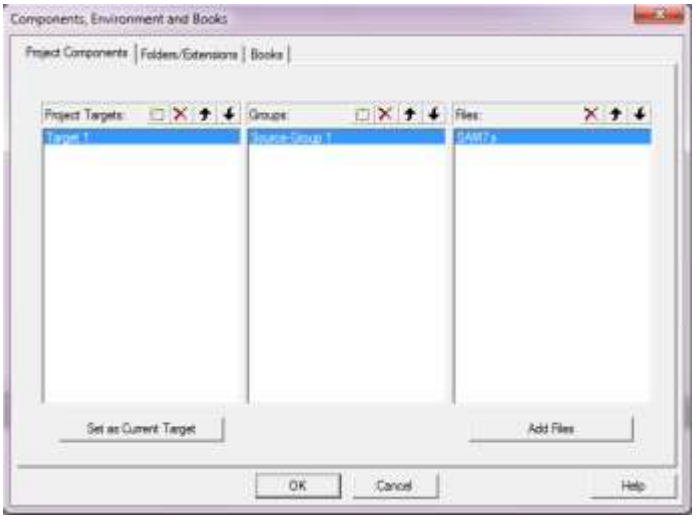
مشاهده میکنید که یک ویرایش گر متن در صفحه باز میشود ، از منوی فایل گزینه ی save را انتخاب کنید و فایل را در کنار پروژه با نام دلخواه و با پسوند .c (برای درج پسوند در اخر نام عبارت .c را بنویسید)ذخیره کنید .



اکنون باید فایل متنی را به پروژه معرفی کنید برای اینکار در پالت project workspace روی گزینه ی target 1 کلیک راست کلیک کنید و در آنجا گزینه ی manege components را انتخاب کنید . در صورتی که پالت project workspace در برنامه شما وجود ندارد از منوی view گزینه ی project window را انتخاب نمایید ، همچنین دقت کنید که گزینه ی file یا project انتخاب شده باشد (گزینه ای که در پایین پالت با فلش مشخص شده).



بعد از انتخاب manege components پنجره ی زیر باز میشود :



بر روی add file کلیک کنید و در پنجره ای که باز میشود ، فایل متنی که با پسوند .c ذخیره کردید باز کنید (بر روی add یکبار کلیک کنید و سپس پنجره را ببندید) . مشاهده میکنید که با کلیک روی ok فایل متنی به project workspace افزوده میشود .

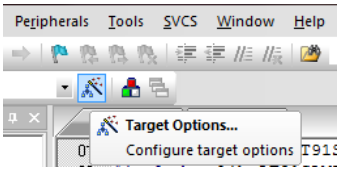
مراحل ایجاد پروژه به پایان رسید ، اکنون باید برنامه خود را با زبان C بنویسیم و آن را کامپایل کنیم . برای ادامه کار ، برنامه زیر را در فایل متنی که ایجاد کردید ، در نرم افزار کپی کنید(فایلی که با پسوند.C ذخیره شده) :

در این برنامه بعد از معرفی میکروکنترلر پایه 19 از پورت B در حالت خروجی پیکربندی شده و وضعیت آن بعد از گذشت زمان تقریبی یک ثانیه بین حالت صفر و یک تغییر میکند . در ادامه با دستورات زیر بیشتر آشنا خواهیم شد .

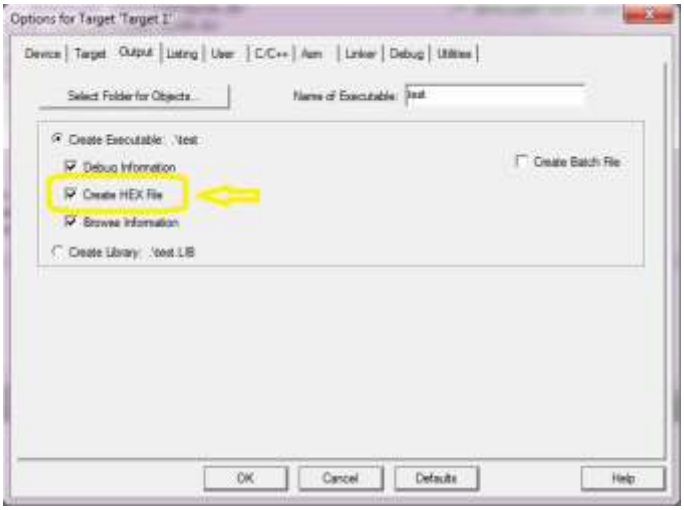
```
#include <AT91SAM7X256.H>                    /* AT91SAMT7X256 definitions */
void delay_s(void);
int main (void) {
*AT91C_PIOB_PER = 0x80000; // Set in PIO mode
*AT91C_PIOB_OER = 0x80000; // Configure in Output
while(1){
     *AT91C_PIOB_SODR = 0x80000  ; // PB.19 to be set
     delay_s();
   *AT91C_PIOB_CODR = 0x80000;    // PB.19 to be cleared
     delay_s();
}
}
void delay_s (void) {
  unsigned int n;
  for (n = 0; n < 7372800; n++);
}
```

ما از این به بعد به فایل متنی ویرایشگر خواهیم گفت . (از سخت بودن این کد ها نترسید ، وقتی که آنها را یادبگیرید به سادگیشان پی خواهید برد)

در بالای پنجره ی project workspace و بر روی ایکون target options کلیک کنید یا از منوی flash گزینه ی Configure Flash tools را انتخاب نمایید :

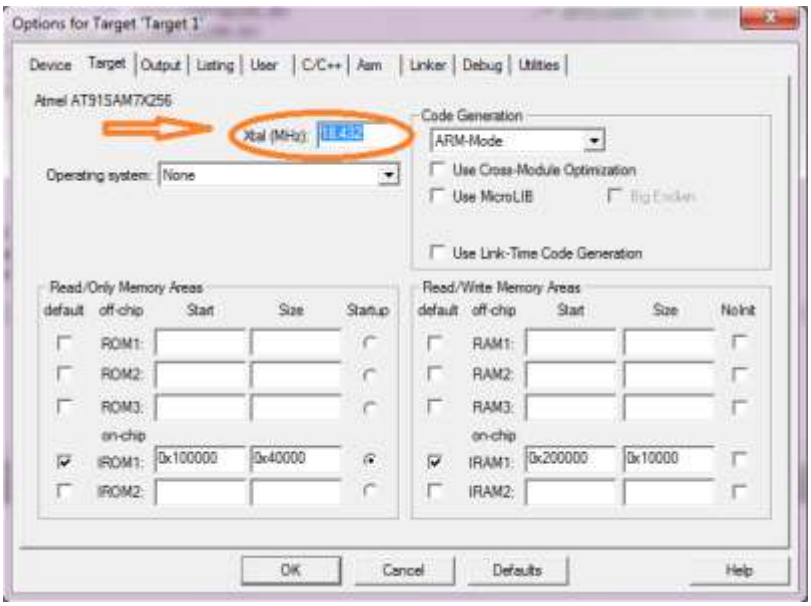


در پنجره باز شده ، تب output را انتخاب کنید و گزینه ی greate hex file را تیک بزنید :



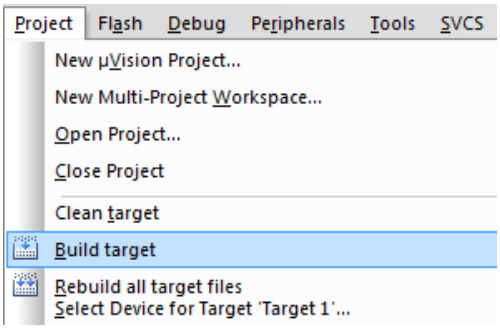
با انجام دادن عمل بالا فایل هگز به خروجی افزوده میشود .

در همین پنجره (پنجره بالا) بر روی target کلیک کنید و در بخش xtal{mhz} مقدار فرکانس کاری میکرو را مشخص کنید (مقدار کریستالی که به میکرو متصل است را در این بخش بنویسید .) و در نهایت بر روی ok کلیک نمایید .

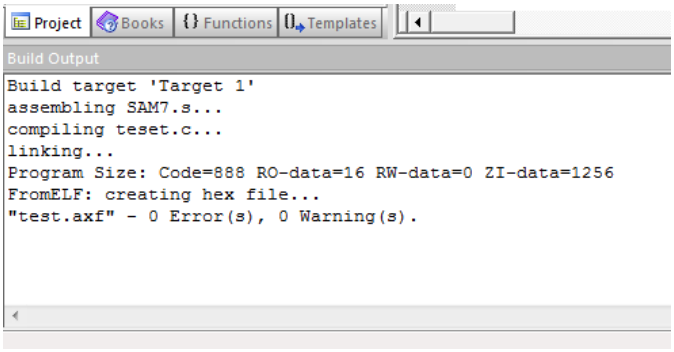


فعلا مقدار کریستال را 18.432 مگاهرتز وارد کنید . مقدار کریستال ، با زمان تولید شده رابطه مستقیم دارد ، در صورتی که مقدار کریستال نوشته شده در این مکان ، با کریستال متصل شده به میکرو یکی نباشد ، برنامه به درستی اجرا نمیشود .

در این مرحله قصد کمپایل کردن برنامه را داریم ، برای اینکار به منوی Project بر وید و در انجا گزینه ی build target را انتخاب کنید :



با این کار برنامه کامپایل میشود و کد هگز مربوطه در محل ذخیره فایل اصلی ذخیره میشود . در صورتی که خطا یا اشکالی در برنامه وجود داشته باشد ،در قسمت output window پیغام خطا به نمایش در میاید .(در صورتی که پالت output window در برنامه شما وجود ندارد از منوی view گزینه ی output window را انتخاب نمایید)



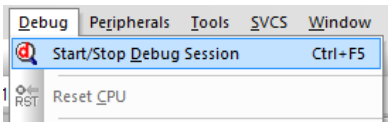
مراحل بالا برای تمامی پروژه ها ثابت است و شما برای هر برنامه ی جدیدی که می نویسید ، این مراحل انجام دهید . در صورتی که قصد دارید برنامه خود را شبیه سازی کنید ، مراحل زیر را انجام دهید ، در غیر این صورت به سراغ بخش برنامه ریزی میکروکنترلر بروید :

شبیه سازی برنامه :

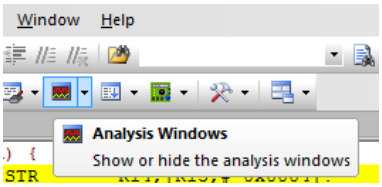
در نرم افزار KEIL امکان اشکال یابی برنامه به دو روش نرم افزاری و سخت افزاری وجود دارد ، در این بخش ما نحوه ی اشکال یابی نرم افزاری را شرح داده و اشکال یابی سخت افزاری را به فصل های بعدی موکول میکنیم .

برای استفاده از اشکال یابی نرم افزاری از منوی flash گزینه ی Configure Flash tools را انتخاب نمایید و در پنجره ی باز شده به تب Debug رفته و تیک گزینه ی Use simulator را بگذارید:

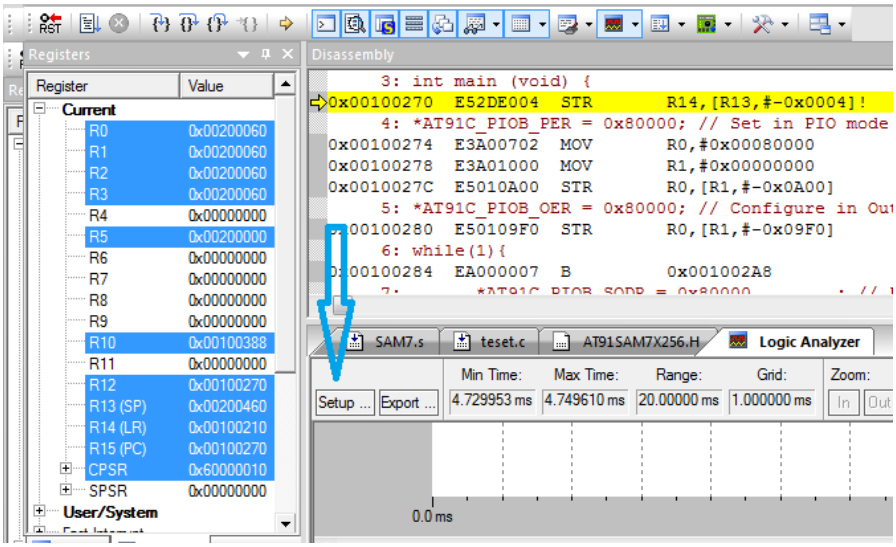
برای ورود به محیط شبیه سازی ، بعد از کامپایل کردن برنامه از منوی debug گزینه ی start / stop debog session را انتخاب کنید:



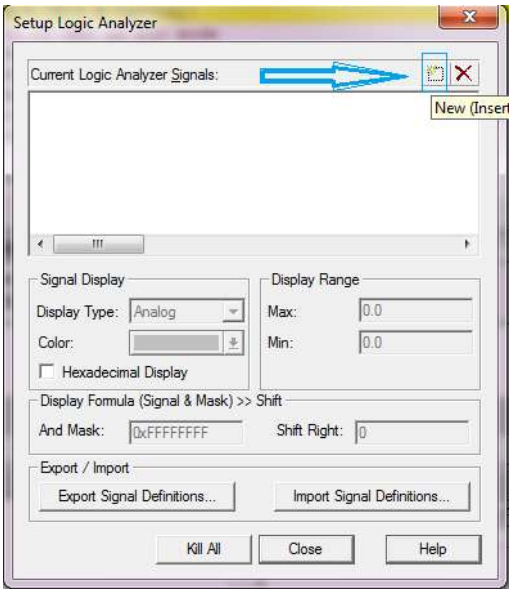
پنجره شبیه سازی بازمیشود ، در تولبار بر روی گزینه ی logic analyzer کلیک کنید :



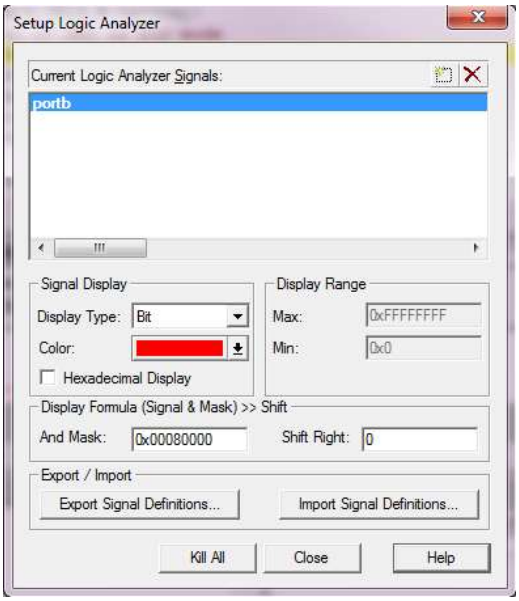
در پنجره باز شده بر روی setup کلیک کنید :



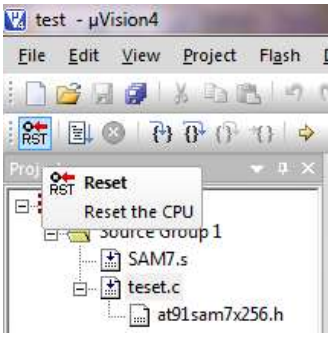
در پنجره باز شده ، بر روی new کلیک نمایید و در بخش ایجاد شده ، پورت که قصد شبیه سازی ان را داریم وارد کنید (portb)



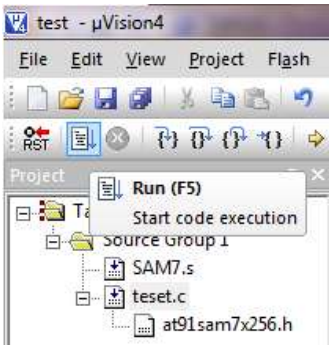
در همین پنجره ، بعد از وارد کردن نام پورت ، در بخش display type خاصیت را به bit تغییر دهید ، همچنین در بخش and mask ادرس پایه مورد نظر را وارد کنید (پایه شماره 19 به ادرس 0 x00080000)



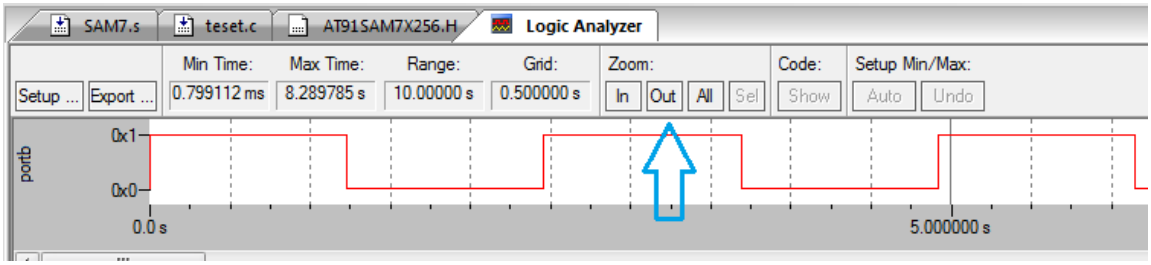
پنجره بالا را ببندید و در تولبار بر روی reset cpu کلیک کنید



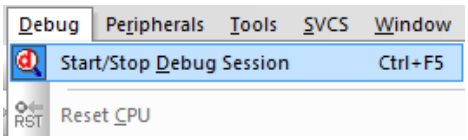
در تولبار بر روی run کلیک کنید و در صفحه اصلی عمل کرد برنامه را ببنید



مشاهده میکنید که ما یک موج مربعی با زمان تناوب 1.23 ثانیه اجاد کرده ایم ، در صورتی که مقدار بزرگنمایی (zoom) را تغییر دهید ، میتوانید بهتر شکل موج را ببنید :



شما موفق به کامپایل و اجرا یک برنامه به زبان c شدید ،برای شبیه سازی پورت ها روش های ساده تری نیز وجود دارد که در ادامه به بررسی آنها خواهیم پرداخت . برای خروج از محیط شبیه سازی از منوی debug گزینه ی start / stop debog session را انتخاب کنید.



برنامه ریزی میکروکنترلر:

اکنون باید برنامه خود را به میکرو منتقل کنیم ، برای انتقال برنامه به میکرو یا پروگرام کردن آن چند روش وجود دارد که در زیر به بررسی آنها پرداخته ایم :

روش های پروگرام کردن میکرو :

Serial Fast Flash Programming(IEEE® 1149.1 JTAG)

SAM-BA ® Boot

Parallel Fast Flash Programming

برای روش اول به سخت افزار(پروگرامر) JTAG و برای مورد سوم به سخت افزار (پروگرامر) ppi (Parallel Programming Interface) نیاز دارید . مورد دوم تقریبا به سخت افزار جانبی نیاز ندارد .

SAM-BA ® Boot

برای راه اندازی این قابلیت به نرم افزار AT91-ISP نیاز دارید ، این نرم افزار به صورت رایگان و برای سیستم عامل های مختلف ، از طرف شرکت اتمل ارائه شده است ، به آدرس زیر مراجعه کنید و نرم افزار مناسب را دانلود نمایید (با توجه به نوع سیستم عامل)

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3883

Software:

	SAM-BA_CDC.zip (Windows Vista - v1.13 current release) (6 MB, revision 1.13, updated 6/09) Install files for the SAM-BA package only. Uses standard CDC driver.
	SAM-BA_CDC.zip (Linux - v1.13 current release) (7 MB, revision 1.13, updated 6/09) Install files for the SAM-BA package only. Uses standard CDC driver.
	AT91-ISP.zip (Windows XP - v1.13 current release - patch) (8 MB, revision 1.13, updated 11/09) This patch shall be applied on AT91ISP v1.13. It fixes several SAM-BA application issues, in particular the SAM-ICE connection issue.
	AT91-ISP.exe (Windows XP - v1.13 current release) (3 MB, revision 1.13, updated 6/09) Install files for the AT91 ISP. Includes SAM-BA package.

فایل Install AT91-ISP.exe را نصب کنید (کافی است در تمامی پنجره ها بر روی next کلیک کنید) . بعد از نصب نرم افزار ، میکروکنترلر خود را را به پورت USB متصل کنید :

کدام میکروکنترلر را به پورت USB متصل کنیم ؟

قبل از خواندن ادامه مطلب به ضمیمه شماره 3 مراجعه کنید ، در این ضمیمه نحوه ی ساختن یک برد آموزشی برای کار با میکروکنترلر AT91SAM7X256 شرح داده شده است . بعد از خرید یا ساخت برد آموزشی ادامه مطلب را بخوانید .

در اولین اتصال برد پیغام روبرو نمایش داده میشود :

اندکی صبر کنید تا شناسی سخت افزار به پایان برسد بعد از گذشت چند ثانیه پنجره found new hardware wizard باز میشود . در صورتی که این پنجره باز نشد به کنترل پانل کامپیوتر خود بروید و گزینه ی add hardware را انتخاب نمایید (در صورتی که پیغام های بالا به نمایش در نیامد ، به قست های بعدی مراجعه کنید).در پنجره found new hardware wizard گزینه ی yes, this time only را انتخاب کنید و سپس بر روی next کلیک کنید :

در پنجره بعدی نیز بر روی next کلیک نمایید ، مقدار صبر کنید تا سیستم درایو مناسب را برای سخت افزار متصل شده پیدا کند :

بعد از گذشت چند ثانیه پنجره ای باز میشود و از شما در مورد کپی کردن درایو سوال میکند ، در ان پنجره گزینه ی continue anyway را انتخاب نمایید تا فایل های درایو به سیستم عامل شما منتقل شود . بعد از نصب سخت ، بر روی finish کلیک کنید تا پروسه تمام شود .

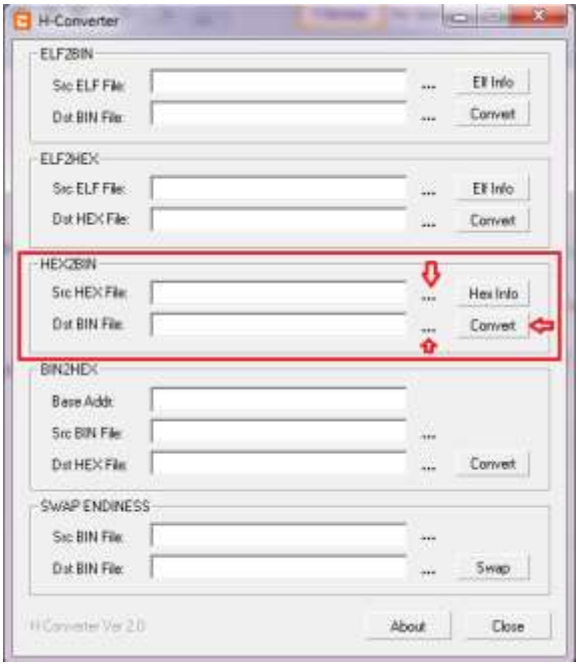
با استفاده از نرم افزار sam-prog قادر خواهید بود کد های باینری (name.bin) را به میکروکنترلر منتقل کنید ، اما همان طور که میدانید خروجی نرم افزار keil به فرم هگز (name.hex) میباشد به همین جهت شما باید قبل از استفاده از نرم افزار ، کد های هگز خروجی را به کد باینری تبدیل کنید .

برای تبدیل کردن کد هگز به کد باینری نرم افزار های مختلفی وجود دارد که یکی از آنها بسته ی H-JTAG است. H-JTAG از سه نرم افزار H-Converter و H-Flasher و H-JTAG تشکیل شده است . ما از نرم افزار H-Converter برای تبدیل کد هگز به باینری و از سایر نرم افزار ها ، در ادامه برای برنامه ریزی میکرو از طریق واسط jtag استفاده خواهیم نمود .نرم افزار hjtag را از ادرس زیر دانلود و سپس نصب کنید :

http://www.hjtag.com/download.html

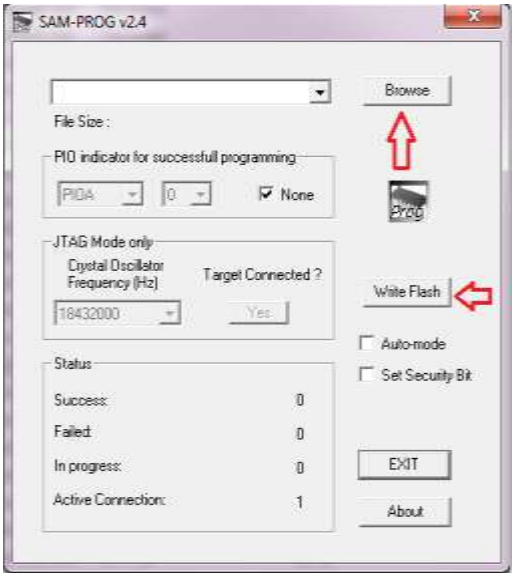
سپس گزینه ی H-Converter را مطابق شکل از منوی استارت انتخاب نمایید :

بعد از باز شدن نرم افزار در بخش hex2bin و در قسمت src hex file بر روی .... کلیک کنید و کد هگز را باز کنید . بر روی convert کلیک کنید تا عملیات تبدیل اغاز شود ، بعد از اتمام تبدیل با پیغام نرم افزار مبنی بر موفقیت تبدیل روبرو خواهید شد . نرم افزار به صورت پیش فرض کد باینری را در کنار کد هگز ذخیره میکند .



در این نرم افزار امکان تبدیل سایر کد ها به یکدیگر نیز وجود دارد .

بعد از تبدیل کردن کد هگز به کد باینری از مسیر Start Menu\Programs\ATMEL Corporation\AT91-ISP v1.13 گزینه ی SAM-PROG Vx.X را انتخاب کنید .پنجره SAM-PROG باز میشود ، بر روی browse کلیک کنید و فایل bin. را در مسیر ذخیره برنامه ، انتخاب نمایید و سپس گزینه ی write flash را بزنید .



میکرو را ریست کنید ، مشاهده میکنید که LED موجود بر روی برد شروع به چشمک زدن میکند. (برنامه ی مذکور پایه ی 19 از پورت B را هر 1.2 ثانیه روشن و خاموش میکند)

نکته ها :

- برای استفاده از sam-ba ، باید کریستال 18.432 مگا هرتز به میکرو متصل کنید (sam-ba فقط با این کریستال کار میکند.)
- برای استفاده از sam-ba نیاز به انجام هیچ گونه تنظیماتی در ابتدای کار نیست (در صورتی که سخت افزار درست باشد ، با اولین اتصال دستگاه شناخته میشود .)
- توجه داشته باشید که باید برد را مانند انوع سخت افزار های usb ، همچون فلش مموری و... متوقف کنید و سپس آن را از پورت بیرون بکشید

در صورتی که در هنگام اتصال دستگاه با پیغام مقابل زیر روبرو شدید موارد زیر را انجام دهید :

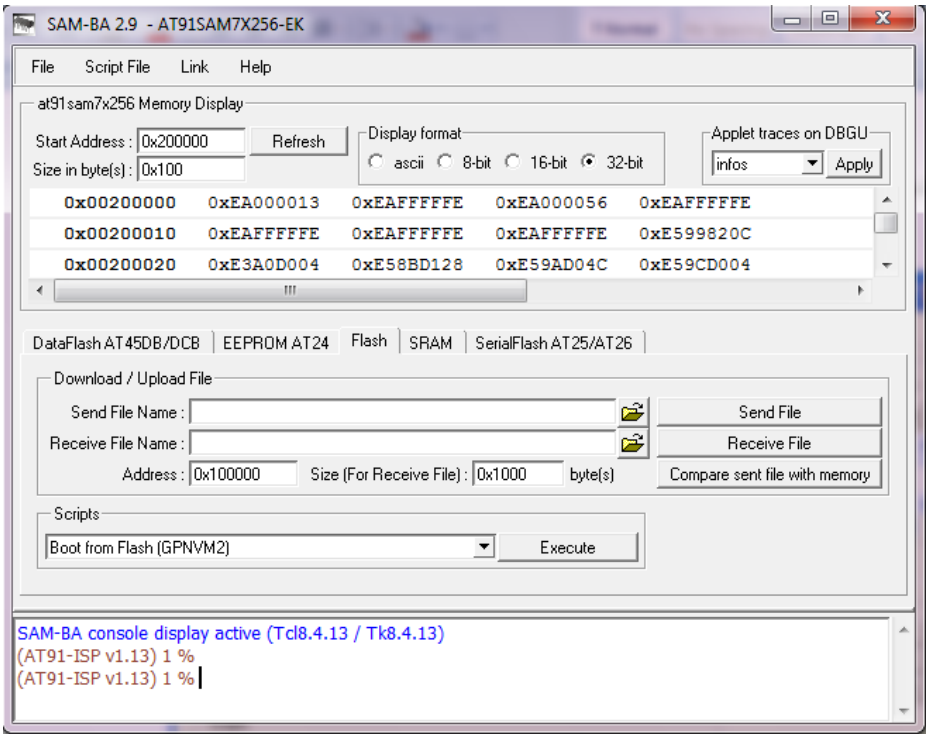
- تغذیه برد را متصل کنید .جامپر پایه erase را متصل نمایید .
- تغذیه برد را قطع کنید .جامپر erase را بردارید .
- اکنون برد را به usb متصل کنید ، با این کار مشکل حل میشود .
- برای هربار استفاده از نرم افزار sam-ba شما باید با اتصال جامپر erase برنامه قبلی را پاک کنید .

برای کسب اطلاعات بیشتر در مورد جامپر erase به ضمیمه شماره 3 مراجعه کنید .

از مسیر Start Menu\Programs\ATMEL Corporation\AT91-ISP v1.13 گزینه ی SAM-BA v2.9 را انتخاب کنید .

در پنجره باز شده و در بخش select your board میکرو متصل شده به پورت usb را انتخاب نمایید (گزینه ی at91sam7x256-ek را انتخاب کنید) و سپس بر روی connect کلیک کنید :

مشاهده میکنید که پنجره sam-ba باز میشود :



با نرم افزار SAM-PROG شما میتوانید باینری را از کامپیوتر خود به حافظه فلش میکرو منتقل کنید . همچنین نرم افزار SAM-ba امکان خواندن و نوشتن حافظه flash ، eeprom و... را به شما می دهد ، این نرم افزار بیشتر برای کار با برد های که قابلیت نصب سیستم عامل دارند مناسب است ، در بخش های بعدی به بررسی کامل این نرم افزار خواهیم پرداخت .

کار با دیباگر و پروگرامر JTAG

JTAG یا IEEE Standard 1149.1-1990 Test Access Portand Boundary-Scan Architecture یک پروتکل ارتباطی میباشد که توسط تعدادی از شرکت های وابسته به سازمان IEEE و تحت استاندارد آن به ثبت رسیده است .در پروتکل JTAG دسترسی کامل به CPU و حافظه ها فراهم میباشد ، و شما میتوانید داده های پردازش شده یا در حال پردازش توسط آنها را مشاهده کنید .رابط JTAG از 4 پایه اصلی برای ارتباط با سخت افزار استفاده میکند ،هر وسیله ای که با استاندارد JTAG سازگار باشد لازم است تا پین های زیر را داشته باشد :

72- TCK (Test Clock Input) : Clock ورودی ، این پالس برای همزمانی میان دستگاه مورد تست و پروگرامر jtag میباشد .

2- TDI (Test Data In) : از طریق این پایه ، داده از پروگرامر به دستگاه در حال تست وارد میشود .

3- TDO (Test Data Out): از طریق این پایه ، داده از دستگاه مورد تست به پروگرامر میرود .

4- TMS (Test Mode Select) : از طریق این پورت حالتهای مختلف تست انتخاب می شود.

همچنین در این میان دو پایه دیگر نیز وجود دارد :

5- TRST (Test Reset Input) این پایه از پروگرامر به ریست دستگاه متصل میشود و قبل از شروع کار آن را باز نشانی میکند .

6- JTAGSEL (JTAG SELECT) : برای راه اندازی پروتکل JTAG ، این پایه باید یک شود .

برای کار با JTAG به موارد زیر نیاز دارید :

1- کابل ارتباطی

2- سخت افزار JTAG

3- نرم افزار JTAG

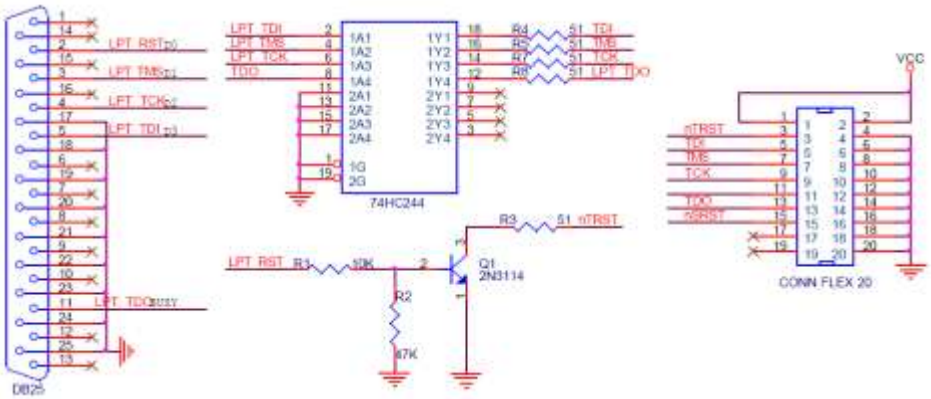
کابل ارتباطی وظیفه اتصال دستگاه مورد تست را به پروگرامر را به عهده دارد ، معمولا پروگرامر از طریق یک کابل دیگر به یکی از پورت های کامپیوتر متصل میشود . سخت افزار JTAG وظیفه کنترل داده های ارسالی و دریافتی و همچنین تغییر دادن انها به کد قابل فهم کامپیوتر را به عهده دارد ، همچنین وظیفه نرم افزار JTAG ،ایجاد یک واسط ارتباطی گرافیکی ، میان کاربر و سخت افزار است .

برای میکروکنترلر های مبتنی بر هسته ی ARM ، شرکت های مختلف اقدام به تولید و عرضه پروگرامر و دیباگر مبتنی بر پروتکل JTAG نموده اند که در این میتوانیم به پروگرامر و دیباگر رایگان H-JTAG و پروگرامر و دیباگر J-LINK اشاره کنیم ؛ H-JTAG از طریق پورت LPT و J-LINK از طریق پورت USB به اطلاعات را از کامپیوتر به میکروکنترلر منتقل

میکند . در ادامه ما نحوه ی برنامه ریزی میکروکنترلر با هر دو پروگرامر را شرح داده ایم ، دیباگ و اشکال یابی برنامه در بخش های بعدی شرح داده خواهد شد :

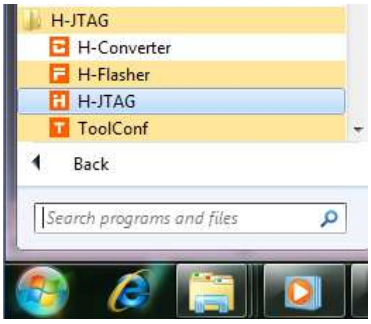
برنامه ریزی میکروکنترلر با استفاده از H-JTAG :

برای راه اندازی واسط JTAG از نرم افزار های رایگان H-Flasher و H-JTAG که در بسته ی نرم افزاریH-JTAG وجود دارند استفاده میشود، در صفحات قبل لینک دانلود نرم افزار موجود میباشد. برای اتصال میکرو به کامپیوتر به یک سخت افزار نیاز دارید ، این سخت افزار مطابق شکل زیر است :

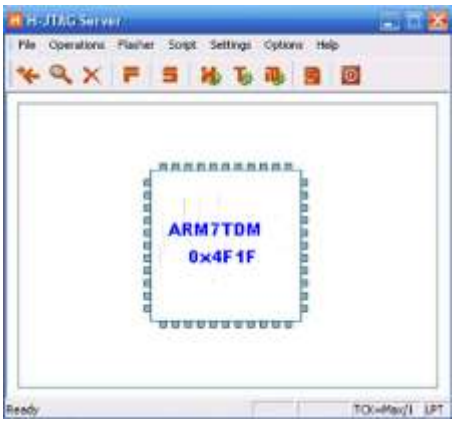


کانکتور تعبیه شده برای اتصال کابل jtag باید مشابه کانکتور تعبیه شده بر روی برد باشد، تا در اتصال سیم ها مشکلی بوجود نیاید . در صورتی که پایه های پورت LPT را روبروی خود بگیرید ، شماره پایه ها در کنار آنها نوشته شده است . به دلیل اختلاف منطقی در سطح پالس ریست ، استفاده از ترانزیستور و مقاومت ها الزامی است .

بعد از نصب نرم افزار و ساختن پروگرامر ، ان را به برد متصل نمایید . شما باید جامپر jselect را بر روی برد متصل کنید. پروگرامر را به پروت lpt متصل کنید و سپس نرم افزار را باز نمایید .

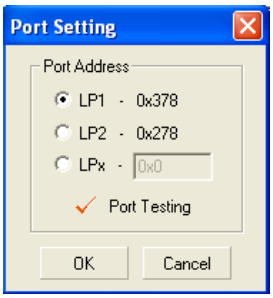


در صورتی که سخت افزار پروگرامر درست باشد ، در پنجره hjtag نام هسته استفاده شده در میکرو به نمایش در میاید :



در صورتی که با پیغام زیر روبرو شدید :

به منوی settings بروید و بعد از انتخاب گزینه ی port setting در پنجره باز شده اولین مورد را انتخاب کنید (در صورتی که کامپیوتر شما بیشتر از یک پورت دارد ، گزینه های بعدی را انتخاب نمایید)



بعد از انجام تنظیمات از منوی felasher گزینه ی start h-flasher را انتخاب کنید .

ابتدا بر روی check کلیک کنید تا میکرو شناسایی شود ، سپس در محل دو بر روی .... کلیک کنید و فایل با پسوند hex. یا bin. را انتخاب کنید ؛ با زدن گزینه ی program برنامه از کامپیوتر به میکرو منتقل میشود .

برنامه ریزی میکروکنترلر با استفاده از J-LINK :

پروگرامر و دیباگر J-LINK که از طرف گروه SEGGER برای برنامه ریزی و دیباگ کردن میکرو کنترلر های مبتنی بر هسته ی ARM طراحی و ارائه شده است ، میتواند از طریق کانکتور JTAG میکرو و پورت USB کامپیوتر کد هگز یا باینری ساخته شده توسط کامپایلر را به میکرو منتقل کند ، این دستگاه در کامپایلر KEIL به صورت کامل پشتیبانی میشود و شما میتوانید برنامه خود را مستقیما از داخل آن به میکروکنترلر منتقل کنید .شما میتوانید این پروگرامر را با قیمت تقریبی 60 هزار تومان (در زمان نگارش این کتاب) از فروشگاه های معتبر برق و الکترونیک تهیه کنید .

برای راه اندازی پروگرامر به نرم افزار J-Link ARM نیاز دارید ، معمولا این نرم افزار در CD همراه پروگرامر وجود دارد ، شما همچنین میتوانید آن را از سایت سازنده ی پروگرامر و به صورت رایگان دانلود کنید :

www.segger.com

نصب این نرم افزار مانند سایر نرم افزار های ویندوز است و نیاز به توضیح اضافی ندارد . بعد از نصب نرم افزار ، پروگرامر را به USB خود متصل کنید ، مشاهده میکنید که ویندوز به صورت خودکار آن را شناسایی کرده و درایوش را نصب میکند .

برای برنامه ریزی میکرو دو روش وجود دارد :

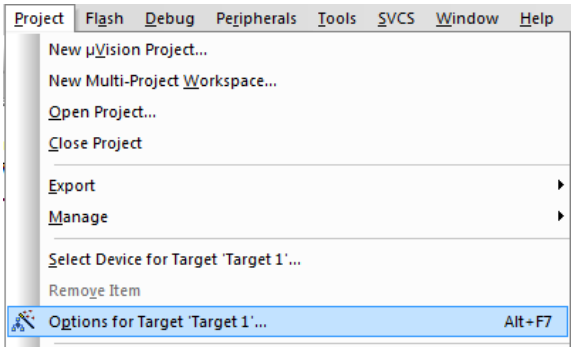
1- استفاده از نرم افزار J-Flash ARM

2- استفاده از نرم افزار KEIL

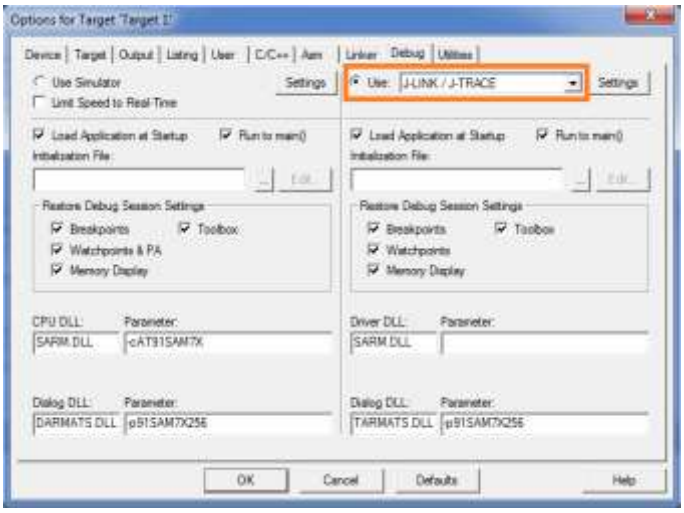
ما ابتدا نحوه ی راه اندازی با استفاده از نرم افزار KEIL را بیان میکنیم و سپس به بررسی J-Flash ARM میپردازیم .

راه اندازی J-Link با استفاده از نرم افزار KEIL :

در نرم افزار keil و بعد از کامپایل کردن پروژه ی خود از منوی project گزینه ی ... 'target 1' options for target را انتخاب کنید :



در پالت باز شده بر روی گزینه ی debug کلیک کنید و در بخش مشخص شده ، تیک گزینه ی use در حالتی که از پالت روبرو گزینه ی J-LINK/J-TRACE انتخاب شده را بگذارید :

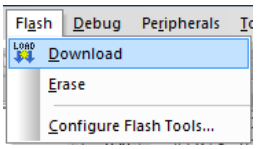


در پنجره ی بالا بر روی پالت Utilities کلیک کنید و در بخش مشخص شده در تصور زیر ، گزینه ی J-LINK/J-TRACE را انتخاب کنید :

در پنجره ی بالا بر روی گزینه ی setting کلیک کنید :

در پنجره ی باز شده ، بر روی add کلیک کرده و در پنجره ی جدید ، هسته ی میکرو کنترلر خود را انتخاب نمایید ، با گذاشتن تیک گزینه ی reset and run ، میکرو بعد از برنامه ریزی شروع به اجرای برنامه میکند . در زیر تنظیمات مربوط به میکرو کنترلر At91sam7x256 آورده شده است :

بعد از انجام دادن تنظیمات بالا ، در کلیه پنجره ی باز شده بر روی ok کلیک کنید ، سپس از منوی flash گزینه ی download را انتخاب کنید ، تا کد هگز ایجاد شده به میکرو منتقل شود .



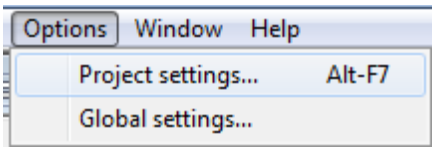
توجه داشته باشید که ، بعد از انجام دادن مراحل بالا ، در صورتی که قصد شبیه سازی برنامه را داشته باشید (از منوی debug گزینه ی start / stop debug session را انتخاب کنید) ، شبیه سازی به صورت سخت افزاری انجام شده و برنامه بر روی میکروکنترلر شبیه سازی میشود ، جهت انجام شبیه سازی به صورت نرم افزاری از منوی flash گزینه ی Configure Flash tools را انتخاب نمایید و در پنجره ی باز شده به تب Debug رفته و تیک گزینه ی Use simulator را بگذارید:

شبیه سازی سخت افزاری یا دیباگ کردن (اشکال یابی) برنامه در فصل های بعدی توضیح داده خواهد شد .

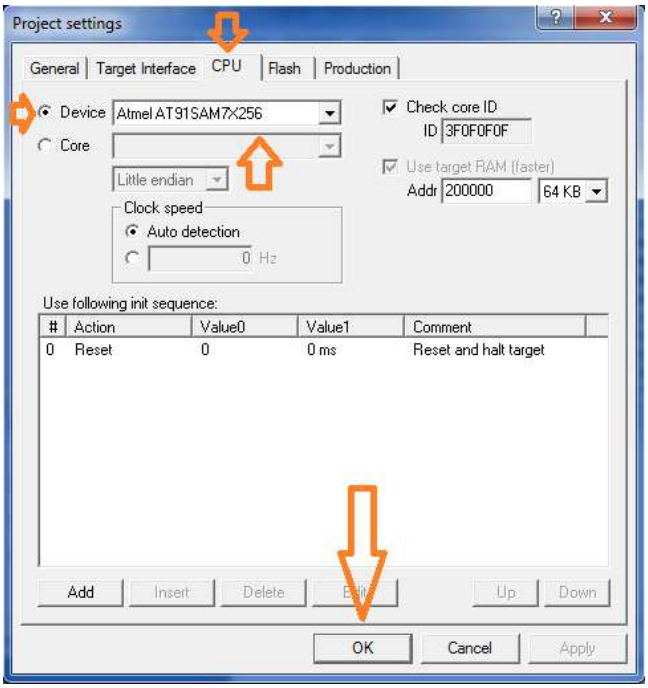
راه اندازی J-Link با استفاده از نرم افزار J-Flash :

برای ورود به نرم افزار J-Flash به منوی start بروید و از زیر منوی SEGGER آن را انتخاب کنید :

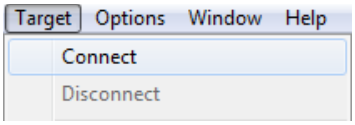
بعد از ورود به نرم افزار از منوی options گزینه ی project settings را انتخاب کنید :



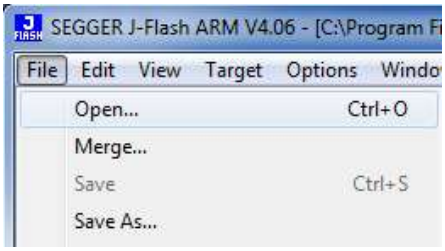
در پنجره ی باز شده بر روی پالت CPU کلیک کنید و بعد از قرار دادن تیک گزینه ی device میکرو کنترلر خود را از پالت روبرو انتخاب کنید و در نهایت بر روی OK کلیک نمایید :



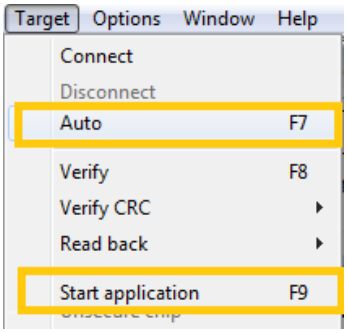
بعد از انجام دادن تنظیمات بالا از منوی target گزینه ی connect را نتخاب نمایید :



با انتخاب گزینه ی open از منوی فایل ، به مسیر ذخیره ی پروژه ی خود رفته و کد هگز یا باینری مربوط به آن را باز کنید :



از منوی Target گزینه ی Auto را نتخاب کنید تا برنامه به حافظه ی فلش میکرو منتقل شود ، با انتخاب گزینه ی start application میکرو ریست شده و برنامه ی ریخته شده بر روی آن اجرا میشود .



چند نکته :

- برای برنامه ریزی میکرو با j-link مقدار فرکانس کاری آن (کریستال متصل شده) مهم نمیباشد و میکرو میتواند با هر فرکانسی کار کند .
- پروگرامر j-link قادر به تهیه تغذیه ی برد نمیباشد ، شما باید برد را از طریق تغذیه ی مخصوص روشن کنید.
- در صورتیکه در هنگام برنامه ریزی با خطایی روبرو شدید ، قبل از دستکاری تنظیمات، میکرو را ریست کنید.
- در صورتی که از نرم افزار keil برای دیباگ کردن برنامه استفاده میکنید ، با قطع تغذیه یا ریست شدن میکرو ، ارتباط قطع میشود .

فصل سوم : شروع برنامه نویسی به زبان C

در این بخش دستورات زبان C و نحوه ایجاد فایل های هدر شرح داده شده است .. بهتر است تمامی مثال ها را شبیه سازی کنید و سپس به میکرو منتقل کنید . جهت تسریع در امر یادگیری و توانایی اجرای مثال ها در عمل ، دستورات مورد نیاز برای LCD کاراکتری و پورت ها توضیح داده شده است .

مقدمه ای بر زبان c

زبان برنامه نویسی C چیست؟

در سال 1967 زبان برنامه نویسی BCPL توسط martin 37ubrouti طراحی و به جامعه جهانی معرفی شد. با توسعه و تکمیل این زبان طی سال های 1969 تا 1970 توسط ken 37ubrouti زبان برنامه نویسی دیگری به نام b متولد شد ، بعد از گذشت 2 سال از ایجاد زبان c تحولات بسیاری در ان توسط برنامه نویسان و... ایجاد شد و کلیه خطا ها و باگ های ان برطرف گردید و در نهایت در 1971 این زبان با نام جدید C به دنیا معرفی شد .در علم الکترونیک و کار با میکرو ها ، کامپایلر ها و مفسر های زیادی برای این زبان وجود دارد ، این کامپایلر ها کد های نوشته شده به زبان C را به اسمبلی و سپس به HEX تبدیل میکنند . تبدیل کد C به اسمبلی توسط برنامه کامپایلر (مانند CODE VSION یا IAR یا KIEL یا ...) انجام میشود و تبدیل برنامه اسمبلی به کد هگز بر عهده اسمبلر ارائه شده توسط سازنده میکرو کنترلر است .

معرفی دستورات اولیه(ساختاری)

بلوک دیاگرام یک برنامه به زبان c تقریبا به شکل زیر است :

راه اندازی پردازنده و فراخوانی کتابخانه ها و فایل های هدر و....

پیکربندی امکانات (مانند lcd و...)

معرفی متغیر ها

شروع حلقه صلی

برنامه ای که باید انجام شود

پایان حلقه

پایان برنامه

زیر برنامه ها (در صورت وجود)

ساختار کلی برنامه نوشته شده به زبان C در کامپایلر KEIL :

```
#include <نام تراشه.h>
#define name “ARM”
.
.
.
unsigned char x,y;
unsigned int n,count;
.
.
.
void function() {
unsigned char k,j;
دستورالعمل ها;
}
.
.
.
void main () {
دستورالعمل ها;
function1();
while(1) {
};
}
```

مثال :

```
#include <AT91SAM7X256.H>  /* AT91SAMT7X256 definitions */
#define led_on *AT91C_PIOB_SODR = 0x80000 /* led_on = *AT91C_PIOB_SODR = 0x80000*/
void delay_s(void);
int main (void) {
*AT91C_PIOB_PER = 0x80000; // Set in PIO mode
```

```
*AT91C_PIOB_OER = 0x80000; // Configure in Output
while(1){
    led_on          ; // PB.19 to be set
    delay_s();
   *AT91C_PIOB_CODR = 0x80000;    // PB.19 to be cleared
    delay_s();
}
}
void delay_s (void) {
 unsigned int n;
 for (n = 0; n < 7372800; n++);
}
```

استفاده از توضیحات (comments)

افزودن توضیحات به یک برنامه اگر هیچ فایده ای نداشته باشد حداقل باعث خوانایی آن می شود.توضیحات هیچ نقشی در اجرای برنامه ندارند و همان طور که از اسمشان برمی آید فقط توضیح اند.اگر توضیح شما چندخطی باشد برای شروع از /* و برای پایان از */ استفاده کنید. مثال:

```
/*this is a comment */

/* this is a

multiline comment */
```

برای توضیح یک خطی بهتر است از // استفاده کنید :

```
// this is  a comment
```

دستور #include

این دستور برای رجوع به فایل های دیگر بکار می رود. هرگاه شما توابع یا توضیحاتی را که در فایل های دیگری تعریف شده اند بکار ببرید باید نام فایل آن را توسط دستور #incude به برنامه اصلی اضافه کنید.

دستور #include به دوشکل زیر بکار می رود :

```
#include<filename.h>

#include"filename.h"
```

در حالت اول کامپایلر filename را تنها درپرونده های پیشفرض خود (که معمولا در پوشه ای به نام INC یا lib در مسیر نصب کامپایلر موجود هستند) جستجو می کند. اما در حالت دوم ابتدا مسیر پروژه جاری را جستجو کرده، در صوتی که فایلی پیدا نشود پوشه پرونده را بررسی می کند.در واقع حالت اول برای کتابخانه های استاندارد خود برنامه و حالت دوم برای کتابخانه های ساخته شده توسط کاربر استفاده می شود.

به فایل ها که توسط دستور بالا به برنامه وارد میشود ، فایل های هدر گفته میشود .

این فایل ها تعاریف و توابعی را شامل می شوند که در زبان C از آنها استفاده می شود.کامپایلر C برای استفاده از هر تابع، باید نام فایل هدر آن را بداند در غیر اینصورت آن تابع برایش ناشناخته خواهد بود.

به این نکته توجه کنید که زبان C نسبت به بزرگی و کوچکی حروف حساس است. تمام دستوراتC با حروف کوچک نوشته می شوند در غیر این صورت خطا رخ خواهد داد. در تعریف متغیرها نیز دقت کنید زیرا متغیر a با متغیرA فرق می کند.

به هر برنامه ممکن است فایل های هدر مختلفی افزوده شود ، یکی از هدر های که در تمام پروژه ها مورد نیاز است فایل مربوط به تراشه مورد استفاده می باشد. برای افزودن فایل هدر میکروکنترلر AT91SAM7X256 به برنامه بایددستور زیر را در اولین خط برنامه استفاده کنیم :

```
#include <at91sam7x256.h>
```

فایل at91sam7x256.h در مسیر C:\Keil\ARM\INC\Atmel\SAM7X در محل نصب نرم افزار KEIL قرار دارد ، آن را با نرم افزار NOTEPAD باز کنید ، همانطور که مشاهده میکنید در این فایل مجموعه ای از دستورات برنامه نویسی مخصوص میکروکنترلر AT91SAM7X256 وجود دارد که در صورت نیاز ما میتوانیم از آنها در برنامه استفاده کنیم .

در صورتی که هر یک از این دستورات قبل از معرفی فایل هدر آورده شود ، کامپایلر قادر به شناسایی آنها نخواهد بود . توجه کنید که هدرهای استاندارد در C به h. ختم می شوند.

دستور #define

این دستور برای تعریف ماکرو(Macro) استفاده می شود.ماکرو در واقع نوعی جایگزینی کد است که به صورت کلی به صورت زیر به کار می رود :

```
#define name description
```

با دستور بالا ، کامپایلر هر جا که با عبارت name برخورد کند آن را با description جایگزین می کند.مثلا فرض کنید مبی خواهید در برنامه چند بار از عدد π با اعشار طولانی استفاده کنید برای این کار می توانید دستور زیر را به کار ببرید :

```
#define pi 3.141592
```

از این به بعد pi همان عدد 3.141592 است .

در خط 2213 فایل at91sam7x256.h با دستور زیر جمله AT91C_PIOB_PER به جای آدرس حافظه ی 0xFFFFF600 معرفی شده است :

```
#define AT91C_PIOB_PER  (AT91_CAST(AT91_REG *)  0xFFFFF600) // (PIOB) PIO Enable Register
```

از این به بعد میتوانیم به جای دستور دشوار (AT91_CAST(AT91_REG *) 0xFFFFF600) از دستور کوتاه و مختصر AT91C_PIOB_PER استفاده کنیم .

متغیرها(identifires)

متغیر نامی است که به یک عدد، تابع، برچسب یا... نسبت داده میشود .یک متغیر می تواند از حروف a..z ، A..Z ، ارقام 0..9 وکاراکتر آندرلاین (_) تشکیل گردد(اولین کاراکتر متغیر نمی تواند رقم باشد).

برای تعریف یک متغیر ابتدا نوع و سپس نام آن را ذکر می کنیم:

```
char x;
int I,j;
bit a,b,c;
```

انواع متغیرهای پشتیبانی شده در کامپایلر KEIL عبارت است از :

Type	Size in bits	Natural alignment in bytes
char	8	1 (byte-aligned)
short	16	2 (halfword-aligned)
int	32	4 (word-aligned)
long	32	4 (word-aligned)
long long	64	8 (doubleword-aligned)
float	32	4 (word-aligned)
double	64	8 (doubleword-aligned)
long double	64	8 (doubleword-aligned)
All pointers	32	4 (word-aligned)
_Bool (C only[1])	8	1 (byte-aligned)

مقداردهی به متغیرهای صحیح char و int در مبناهای دیگر به صورت زیر است :

مبنا	فرم	مثال
hexadecimal	0xNNN	0xB6
octal	0NNN	010
binary	0bNNN	0b10111001

متغیر نوع char می تواند کاراکترهای ASCII را نیز در خود نگاه دارد. مثلا :

```
char c;
c='A';
```

آرایه ها

آرایه مجموعه ای از متغیرهای هم نوع و مرتبط با هم است که با یک نام واحد شناخته می شوند.تعریف یک آرایه به صورت زیر است :

نوع داده نام آرایه [size];

مثال :

```
char a[5];
```

برای دسترسی به عنصر یک ارایه می بایست از اندیس آرایه استفاده کرد که از صفر شروع می شود.در مثال فوق آرایه a شامل عناصر a[0] تا a[4] خواهد بود.

```
Int var[];
```

با دستور بالا ، تعداد بینهایت متغیر با نام var در برنامه معرفی میشود ، هر متغیر var با عدد قرار گرفته در داخل گروشه شناخته میشود :

```
var[1]=1;
var[2]=455;
var[100]=300;
```

هر متغیری را می توان در هنگام تعریف مقدار دهی کرد مثلا:

```
unsigned char a=5,b=4,c;
```

با دستور زیر آرایه var دارای 3 متغیر var[2] و var[1] و var[0] است(آرایه شماره 3 غیر قابل استفاده است) :

```
short  var[3];
```

مقداردهی اولیه آرایه ها بطور نمونه به صورت زیر است :

```
int a[4]={11,5,0,125};
char c[5]={‘a’,’b’,’c’,’d’,’e’};
```

اما برای مقداردهی اولیه متغیرهای رشته ای روش زیر مناسب تر است :

```
char c[]=”abcde”;
```

مثال بالا روشی برای تعریف یک متغیر رشته ای است زیرا در زبان C بر خلاف زبانهایی مثل pascal یا Basic نوع خاصی برای متغیر رشته ای تعبیه نشده است و با این وجود برای کار با متغیرهای رشته ای بهتر است از اشاره گرها استفاده کنیم ، اشاره گرها در ادامه توضیح داده شده اند .

آرایه ها به صورت چند بعدی نیز معرفی میشوند ، شکل کلی تعریف آرایه های چند بعدی به صورت زیر است :

نوع داده نام آرایه [size1,size2,…];

نوع داده نام آرایه [size1] [size2] […];

مثلا:

```
unsigned int x[5,5];
char c [2] [3]={ ‘a’,’b’,’c’,’d’,’e’,’f’};
```

با دستور زیر متغیر var به صورت یک آرایه ماتریسی 3 در 3 معرفی شده است :

```
char  var[3][3];
```

همچنین امکان مقدار دهی آرایه ها به صورت خطی یا ماتریسی نیز وجود دارد :

```
char  var[3][3] = {{‘1’,’2’,’3’},
                   {‘4’,’5’,’6’},
                    {‘7’,’8’,’9’}};
var[1][1]=1;
short  var[16]={4,6,8,12,22,30,37,45,55,75,90,110,132,160,200,250};
```

توجه داشته باشید :

- نام متغیر ها نمیتواند شامل کلمات رزرو شده (دستورات که در برنامه نویسی به کار میروند مانند void و int و...) باشد ، همچنین نام متغیر نباید از 31 کاراکتر (حرف) بیشتر باشد و زبان c بین حروف کوچک و بزرگ تفاوت قائل میشود (aبا A فرق دارد.)
- معرفی متغیر ها باعث مصرف شدن فضای حافظه ی RAM میگردد . در صورتی که به متغیری نیاز ندارید ، از معرفی و مقدار دهی آن پرهیز کنید .
- در حالت پیشفرض تمامی متغیر های معرفی شده در برنامه از نوع unsigned (بدون علامت) بوده و تنها قادر به دریافت مقادیر مثبت میباشند ، در صورتی که در ابتدای عنوان متغیر عبارت signed (علامت دار) آورده شود . متغیر قادر به دریافت مقادیر منفی نیز خواهد بود :

```
signed short AN0=-10;
```

اشاره گرها (pointers)

اشاره گر، متغیری است که آدرس حافظه یک شئ را در خود نگاه می دارد. هر متغیر در فضای برنامه، حافظه ای اشغال می کند و این حافظه یک آدرس دارد که اشاره گر مقدار عددی آن را در خود ذخیره می کند شکل کلی تعریف اشاره گر ه صورت زیر است :

نام متغیر* نوع داده;

مثلا:

```
unsigned char *a;
```

عملگر * محتوای آدرس و عملگر & آدرس متغیر را بر می گرداند.

مثال:

```
viod main() {
char b,*a,c;
b=100;
a=&b;
c=*a;
}
```

در این مثال ابتدا b=100 وسپس a برابر آدرس b (ونه خود آن) قرار داده شده است.مثلا اگر b در خانه 200H حافظ باشد a=200H خواهد بود. در پایلن c برابر محتوای آدرسی که a به ان اشاره می کند قرار گرفته و برابر همان b یا 100 می شود.

مثال :

```
*AT91C_PIOB_PER = 0x80000;
```

با دستورات بالا رقم 80000 در مبنای هگز در آدرسی که متغیر AT91C_PIOB_PER به ان اشاره میکند ، کپی میشود .

با جستوجو کردن AT91C_PIOB_PER در هدر AT91SAM7X256.H میبینید که این واژه در خط زیر به کار رفته است :

```
#define AT91C_PIOB_PER  (AT91_CAST(AT91_REG *)   0xFFFFF600) // (PIOB) PIO Enable Register
```

در واقع رقم 80000 در منبای هگز در ریجستر فعال کننده ی واحد PIOB کپی میشود . در ادامه در این مورد بیشتر توضیح داده ایم .

مثال :

```
char *str;
*str=”ARM”
*str=”KEIL”;
```

عملگرها در زبان C

عملگرهای یگانی

عملگر	مفهوم	مثال	نتیجه
&	آدرس	b=&a	b برابر آدرس a می شود
*	محتوای آدرس	B=*addr	b برابر مقدار موجود در حافظه ادرس addr می شود.

عملگرهای حسابی

عملگر	مفهوم	مثال	نتیجه
*	ضرب	2*3	6
/	تقسیم	6/2	3
٪	باقیمانده تقسیم	17%3	2
+	جمع	3+2	5

++	افزایش به اندازه 1 واحد	a++	a=a+1
--	کاهش یه اندازه 1 واحد	b--	b=b-1
–	تفریق	3-2	1
<<	شیفت چپ	0x01<<2	0x04
>>	شیف راست	0x04>>2	0x01

عملگرهای مقایسه ای

عملگر	مفهوم
<	کمتر از
<=	کوچکتریا مساوی
>	بیشتر از
>=	بزرگتر یا مساوی
==	برابر است با
!=	مخالف است با

عملگرهای بیتی

&	AND بیتی	0xF0 & 0x22	0x20
|	OR بیتی	0xFF | 0x01	0xFF
^	XOR بیتی	0xFF ^ 0xFF	0
~	مکمل 1	~0x55	0xaa

عملگرهای منطقی (logical)

&&	AND منطقی
||	OR منطقی
!	Notمنطقی

عملگرهای انتسابی یا ترکیبی

عملگر	مفهوم	مثال	معادل
=	مساوی	X=10	
*=	انتساب ضرب	X*=y	X=x*y

/=	انتساب تقسیم	x/=10	X=x/10
%=	انتساب باقیمانده تقسیم	X%=10	X=x%10
+=	انتساب جمع	X+=y	X=x+y
-=	انتساب تفریق	x-=y	X=x-y

دستورات کنترلی و شرطی

دستور شرطی if-else

شکل کلی استفاده از این دستور به صورت زیر است :

```
if (شرط)
دستورالعملها;
else
دستورالعملها;
```

مثال :

```
if (num<0)
  num++;
else
num+=10;
```

اگر پس از عبارات if() یا else تنها یک دستور نیاز باشد مستقیما آن دستور را می نویسیم و اگر تعداد دستورات بیشتر بود باید آنها را بین علامت های { و } قرار دهیم. مثلا :

```
if(num<0) {
abs= -num;pos=1;
}
else {
abs=num;
pos=-1;
}
```

در عبارت فوق چنانچه مقدار عبارت داخل if() درست باشد دستورات مقابل آن اجرا خواهد شد در غیراین صورت دستورات مقابل else اجرا می شوند.اگر else به کار نرفته باشد در صورت صحت شرط عبارت داخل بلوک if() اجرا می شود و در غیر

این صورت سطر بعد از بلوک اجرا خواهد شد. عبارات if-else را می توان به صورت لایه لایه و تو در تو نیز به کار برد که شکل کلی آن به صورت زیر است:

```
if(1شرط) {
}
else if (2شرط) {
}
else if (3شرط) {
}
else {
}
```

حلقه شرطی while()

شکل کلی این دستور به صورت زیر است :

```
While (شرط) {
دستورالعملها;
}
```

در این دستور مادامی که شرط درست باشد عبارات داخل بلوک while اجرا می شود. مثلا:

```
While(i>0) {
i=i-1;
a[i]=I;
}
```

در این مثال هر بار مقدار i با صفر مقایسه می شود و اگر بزرگتر از آن بود دستورات داخل بلوک اجرا می شود.در مورد عبارات شرطی در زبان C رخلاف زبان هایی مثل پاسکال متغیر نوع بولی(Boolean) وجود ندارد و برای هر عبارت درست یا نادرست مقداری عددی در نظر گرفته می شود.عدد صفر معادل یک عبارت نادرست و هدر عدد غیر صفر نشانه یک عبارت درست است.

تذکر: عبارت while(1) یک حلقه دائمی و بدون پایان ایجاد می کند.

حلقه شرطی do-while()

شکل کلی آن به صورت زیر است :

```
do{
دستورالعملها;
}while(شرط);
```

تفاوت این حلقه با حلقه قبلی در این است که در این حلقه ابتدا دستور داخل حلقه اجرا می شود و در پایان شرط حلقه بررسی می شود یعنی عبارات حلقه یک بار اجرا خواهد شد.

مثال:

```
do{
i++;
a=I;
}while(i<11);
```

حلقه for()

شکل کلی آن به صورت زیر است :

```
for(مقداراولیه;شرط حلقه;شمارنده حلقه){
}
```

مثال :

```
for(i=0;i<10;i++){
a=I;
b=2*I;
}
```

i در حلقه بالا اندیس حلقه است. این اندیس مقدار دهی اولیه می شود و تا وقتی که شرط حلقه درست باشد دستورات حلقه اجرا می شود. در صورت نادرستی شمارنده حلقه برنامه از حلقه خارج می شود. پس از هر بار اجرای حلقه شمارنده حلقه اجرا می شود که معمولا وظیفه اش افزایش اندیس است.

دستور goto

از این دستور برای پرش به یک برچسب استفاده می شود.برچسب نام منحصر به فردی است که به کاراکتر ":" ختم می شود(استفاده از دستور goto تا جای امکان توصیه نمی شود.)

```
loop:
دستورها;
goto loop;
```

دستور break

برای خروج بدون شرط از حلقه استفاده می شود و هر گاه برنامه به این دستور برسد از حلقه خارج شده و ادامه برنامه از انتهای حلقه ادامه می یابد.

مثال :

```
while(1) {
i=i+1;
if(i==20) break;
}
```

در مثال فوق هر گاه مقدار i برابر 20 شود برنامه از حلقه خارج خواهد شد.

دستور switch()

شکل کلی این دستور به صورت زیر است :

```
switch (عبارت) {
case 1:
دستورها;
break;
Case 2:
دستورها;
break;
.
.
```

```
.
case n :
دستورها;
break;
default :
دستورها;
}
```

این برنامه مقدار درون ()switch را با عبارات مقابل case ها مقایسه می کند . در صورت پیدا کردن تساوی دستورات مقابل آن را اجرا می کند.اگر هیچ تطبیقی یافت نشود دستورات مقابل default اجرا می شود و اگر default وجود نداشته باشد(گذاشتن default اختیاری است) هیچ کاری انجام نمی شود.

مثال:

```
switch(x) {
case 1:
i=10;
break;
case 2 : case3:
i=20;
break;
default :
i=30;
}
```

دستورات مربوط به پورت ها

با توجه به تعداد زیاد میکروکنترلرهای مبتنی بر هسته ی ARM و تنوع بسیار زیاد در سخت افزار آنها امکان ایجاد دستورات بهینه و مشابه برای تمامی میکروکنترلرها وجود ندارد . به بیان ساده تر در کامپایلر KEIL دستور استانداردی برای راه اندازی ، روشن یا خاموش کردن و... امکانات جانبی مثل پورت های ورودی/ خروجی و... وجود ندارد و کاربر الزاما باید از ثبات ها (رجیستر ها) یا کتابخانه های ارائه شده از طرف سایر کاربران یا شرکت سازنده میکروکنترلر برای نوشتن برنامه ی خود استفاده کند .

ثبات چیست ؟

شرکت های تولید کننده ی میکرو کنترلر ، خانه های را در حافظه ی فلش میکروکنترلر برای فعال یا غیر فعال کردن بخش های مختلف میکروکنترلر در نظر گرفته اند . با قرار دادن 0 یا 1 در هر یک از این خانه ها میتوان بخش های جانبی میکروکنترلر مانند پورت ها ، مبدل آنالوگ به دیجیتال ، واسط SPI و... را فعال یا غیر فعال کرد ، این بخش ها میتوانند داده های مورد نیاز خود را از این خانه ها بخوانند یا داده های پردازش شده را در این خانه ها ذخیره کنند .

هر یک از این خانه ها دارای آدرس اختصاصی هستند ، مثلا در آدرس 0xFFFFF600 حافظه ، یک بخش 32 بیتی برای فعال سازی پایه های پورت B وجود دارد ، با قرار دادن یک در هر خانه ، پایه ی متناظر فعال شده و میتواند در حالت ورودی یا خروجی قرار گیرد .

جهت سهولت در یادگیری آدرس ها ، تمام آدرس ها در فایل هدر میکرو کنترلر جمع آوری شده و با دستورات ساده تر که به آنها رجیستر یا ثبات گفته میشود ، جایگزین گردیده است .

```
#define AT91C_PIOB_PER  (AT91_CAST(AT91_REG *)  0xFFFFF600) // (PIOB) PIO Enable Register
```

از این به بعد با مقدار دهی رجیستر AT91C_PIOB_PER آدرس 0xFFFFF600 در حافظه میکروکنترلر مقدار دهی شده و در نهایت پایه های دلخواه از پورت B در حالت ورودی / خروجی فعال میشود .

با این تفاسیر ، اولین قدم برای نوشتن برنامه برای میکروکنترلر های ARM ، شناخت ثبات های مربوط به واحد های مختلف است در کامپایلر KEIL ، ثبات های مخصوص میکروکنترلر های سری AT91SAM شرکت اتمل ، با عبارت AT91C شروع میشوند . بعد از آشنایی با ثبات ها ، کاربر میتواند کتابخانه ها و هدر های مورد نیاز خود را ایجاد کرده و از آنها استفاده کند .

هر چند به خاطر سپردن نام رجیستر ها نیز به دلیل زیاد بودن تعدادشان ، مقداری دشوار به نظر میرسد ، اما در برنامه های حرفه ای چاره ای جز استفاده از آنها نداریم . ما در ابتدا به معرفی کلیه رجیستر ها مربوط به واحد pio پرداخته ایم و سپس

دو هدر pio.h و lib_AT91SAMxxxxx.h را بررسی نموده ایم ، با استفاده از این هدر دیگر نیازی به رجیسترها ندارید و دستورات ساده تر جای آنها را خواهد گرفت .

برای کار با واحد pio (Parallel Input/Output) سری at91sam 29 رجیستر و مطابق جدول زیر در نظر گرفته شده است :

توضیح	نوع استفاده	نام کامل	نام ریسجتر
فعال کننده ی واحد pio	Write-only	PIO Enable Register	PIOx_PER
غیر فعال کننده ی واحد pio	Write-only	PIO Disable Register	PIOx_PDR
دریافت وضعیت واحد pio		PIO Status Register	PIOx_PSR
پیکربندی واحد pio در حالت خروجی	Write-only	Output Enable Register	PIOx_OER
ساقط کننده ی واحد pio از حالت خروجی	Write-only	Output Disable Register	PIOx_ODR
وضعیت ورودی یا خروجی پایه ها	Read-only	Output Status Register	PIOx_OSR
فعال کننده ی فیلتر اشکار خطای ورودی	Write-only	Glitch Input Filter Enable Register	PIOx_IFER
غیر فعال کننده ی فیلتر اشکار خطای ورودی	Write-only	Glitch Input Filter Disable Register	PIOx_IFDR
وضعیت فیلتر بر روی پایه ها	Read-only	Glitch Input Filter Status Register	PIOx_IFSR
یک کردن پورتی که در قبلا فعال شده است	Write-only	Set Output Data Register	PIOx_SODR
صفر کردن پورتی که در قبلا فعال شده است	Write-only	Clear Output Data Register	PIOx_CODR
وضعیت پایه های خروجی (تحریک توسط واحد pio)	Read-write	Output Data Status Register	PIOx_ODSR
وضعیت پایه ها ورودی (تحریک توسط عامل بیرونی)	Read-only	Pin Data Status Register	PIOx_PDSR
پیکربندی پورت در حالت ورودی وقفه	Write-only	Interrupt Enable Register	PIOx_IER
ساقط کردن پورت از حالت ورودی وقفه	Write-only	Interrupt Disable Register	PIOx_IDR
دریافت پوشش وقفه ی موجود روی پورت	Read-only	Interrupt Mask Register	PIOx_IMR
دریافت وضعیت ورودی وقفه یا ورودی عادی بودن پایه ها	Read-only	Interrupt Status Register	PIOx_ISR
توضیحات 1	Write-only	Multi-driver Enable Register	PIOx_MDER
توضیحات 1	Write-only	Multi-driver Disable Register	PIOx_MDDR
توضیحات 1	Read-only	Multi-driver Status Register	PIOx_MDSR
فعال سازی مقاومت pull up بر روی پایه های ورودی	Write-only	Pull-up Disable Register	PIOx_PUDR

PIOx_PUER	Pull-up Enable Register	Write-only	غیر فعال سازی مقاومت pull up بر روی پایه های ورودی
PIOx_PUSR	Pad Pull-up Status Register	Read-only	وضعیت مقاومت pull up بر روی پایه های ورودی
PIO_ASR	Peripheral A Select Register	Write-only	توضیحات 2
PIO_BSR	Peripheral B Select Register	Write-only	توضیحات 2
PIO_ABSR	AB Status Register	Read-only	توضیحات 2
PIOx_OWER	Output Write Enable	Write-only	توضیحات 3
PIOx_OWDR	Output Write Disable	Write-only	توضیحات 3
PIOx_OWSR	Output Write Status Register	Read-only	توضیحات 3

1- فعال / غیر فعال سازی پورت

به دلیل وجود مد های کم مصرفی مختلف ، واحد pio در سری at91sam به صورت پیش فرض غیر فعال می باشد . با استفاده از رجیستر PIOx_PER میتوانید پورت x (A یا B) را فعال نمایید . در این حالت پورت میتواند با مقدار دهی رجیستر های که در ادامه بررسی میشوند در حالت ورودی ، خروجی ، درین باز و یا ورودی/خروجی وسیله ی جانبی (spi ، adc و..) پیکربندی شود .

با استفاده از رجیستر PIOx_PDR میتوانید پورت x را غیر فعال نمایید . در این حالت پایه ها از داخل درین باز شده و تغذیه ی و کلاک کنترل کننده ی آنها قطع میشود .

با استفاده از PIOx_PSR می توانید وضعیت فعال یا غیره فعال بودن پایه های یک پورت را چک کنید . پایه های که فعال باشند مقدار 1 و پایه های که غیر فعال باشند مقدار 0 را در رجیستر قرار می دهند .

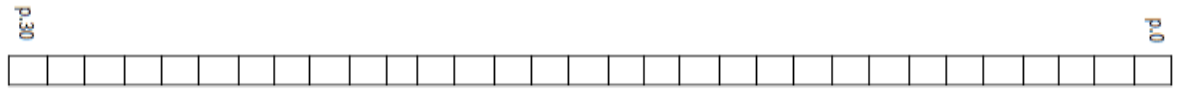
```
Int c;

C=*AT91C_PIOB_ PSR
```

با دستور بالا وضعیت پایه های پورت b در متغیر c که از نوع اینتیجر معرفی شده ، ریخته می شود .

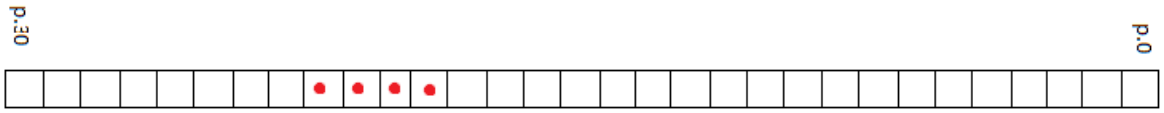
مثال : پایه های 19 و 20 و 21 و 22 از پورت b را فعال کنید :

همان طور که مشاهده کردید ، برای مقدار دهی رجیستر ها باید از کد هگز استفاده کنیم ، برای پیدا کردن کد هگز مورد نظر که از این به بعد به آن آدرس می گوییم باید به روش زیر عمل کنید :

تصویر زیر را رسم کنید:



در تصویر بالا هر مربع نشانگر یک پایه از میکرو می باشد ، در صورتی که اولین مربع از سمت راست (p.0) پایه B.0 باشد . سیزدهمین مربع نشانگر پورت B.12 است و بیستمین مربع نشانگر پایه 19 از پورت b است .اکنون پایه های (مربع های) که قصد فعال کردنشان را داریم علامت بزنید :



به مربع های خالی رقم صفر و به مربع های رنگی عدد یک بدهید و مجموعه ی اعداد را در کنار هم قرار دهید ، عدد بدست آمده آدرس پایه های مورد نظر در مبنای باینری است ، با بردن این عدد در مبنای هگز ، آدرس مورد نیاز بدست می اید (برای تبدیل عدد باینری به هگز می توانید از ماشین حساب ویندوز استفاده کنید) .

00000000111100000000000000000000= 11110000000000000000000 bin = 780000 hex

*AT91C_PIOB_PER = 0x780000;

2- پیکربندی پورت در حالت خروجی و استفاده از آنها:

برای فعال سازی پورت در حالت خروجی نیز مانند فعال کردن آن ، سه رجیستر وجود دارد :

PIOx_OER : با مقدار دهی این رجیستر می توانید پایه های دلخواه را در حالت خروجی پیکربندی کنید ، حداقل و حداکثر مقدار مجاز برای این رجیستر به ترتیب برابر 0x0000 و 0XFFFF می باشد .

PIOx_ODR : با مقدار دهی این رجیستر می توانید می توانید پایه های که قبلا به عنوان خروجی تعیین شده بودند را غیر فعال کنید . این رجیستر بیشتر در حالتی استفاده می شود که قصد داشته باشید پایه ی خروجی را به عنوان ورودی پیکربندی نمایید .

PIOx_OSR : وضعیت پایه های که آزاد هستند یا در مد خروجی قرار گرفته اند در این رجیستر قرار داده می شود . مقدار دهی این رجیستر ها مشابه مثل قبلی است .

PIOx_SODR : با استفاده از این رجیستر می توانید پایه های دلخوه را یک کنید .

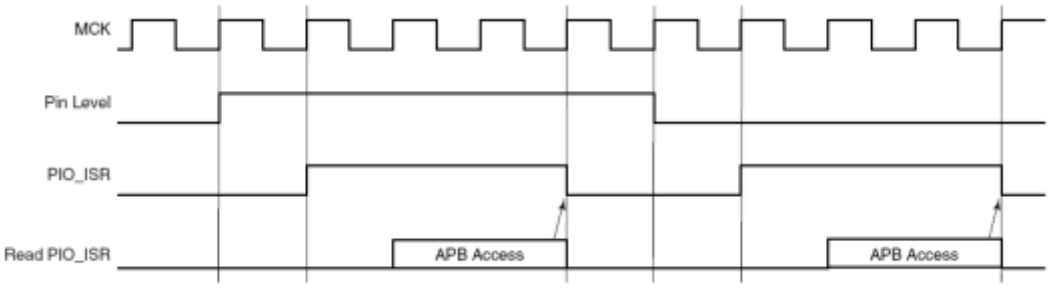
PIOBx_CODR : با این رجیستر می توانید پایه های دلخواه از میکرو کنترلر را صفر کنید .

توجه داشته باشید که دو رجیستر بالا کاملا از یکدیگر مجزا بوده و هر کدام کاربرد خود را دارد ، در صورتی که با مقدار دهی PIOx_SODR پایه ای را یک کنید ، برای صفر کردن آن حتما باید از رجیستر PIOBx_CODR استفاده کنید و مقدار دهی مجدد PIOx_SODR عملا نقشی در تغییر وضعیت پایه مذکور نخواهد داشت .

PIOx_ODSR : وضعیت صفر یا یک بودن پایه ها در این رجیستر قرار داده می شود .

3- پیکربندی پورت ها در حالت ورودی و استفاده از آنها .

هنگامی که پایه ای در حالت ورودی قرار داده می شود ، می تواند با ورودی سطح صفر یا یک تحریک شود . در این حالت CPU با توجه به کلاک سیستم و کلاک PIO ، پایه مذکور را در دوره های زمانی منظم چک می کند و در صورتی که پایه تغییر وضعیت دهد مقدار رجیستر PIOx_PDSR تغییر می کند .



در حالت پیش فرض کلاک pio غیر فعال است و اگر شما پایه ای را به صورت ورودی پیکربندی کنید ، cpu توانایی خواندن آن را ندارد . پس اولین قدم برای راه اندازی پورت در حالت ورودی راه اندازی کلاک آن می باشد .

PMCx_PCER : با مقدار دهی این رجیستر ، کلاک مورد نیاز برای همزمانی واحد PIO با CPU از کلاک اصلی میکرو کنترلر (MCK/X) تامین می شود . اکنون می توانید با استفاده از دستورات شرطی وضعیت رجیستر PIOx_PDSR را چک کنید و وضعیت پایه ی مورد نظر را مشاهده نمایید ، رجیستر PMCx_PCER در بخش مدیریت توان ، تشریح خواهد شد .

توجه داشته باشید که اگر قبلا پایه های مذکور (که قرار است در حالت ورودی پیکربندی شوند) را در حالت خروجی استفاده کرده باشید باید رجیستر PIOx_ODR (Output Disable Register) را مقدار دهی کنید تا پایه از حالت خروجی ساقط شود ، همچنین فعال سازی واحد PIO با رجیستر PIOx_PER (PIO Enable Register) الزامی است .

4- فعال سازی مقاومت pullup :

در سری AT91SAM ، تمامی پایه ها دارای یک مقاومت pull up داخلی می باشند ، مقاومت pull up دارای سه رجیستر زیر است :

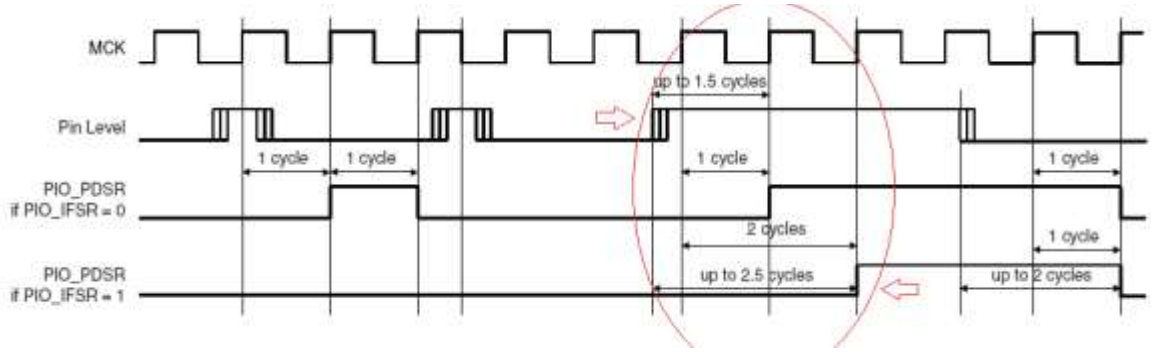
PIOx_PUER : با مقدار دهی این رجیستر مقاومت pull up داخلی فعال می شود و پایه ی ورودی را در سطح 1 منطقی نگه می دارد . در این حالت پایه فقط به ورودی صفر پاسخ می دهد.

PIOx_PUDR : با مقدار دهی این رجیستر مقاومت pull up داخلی غیر فعال شده و پایه می تواند به سطوح منطقی صفر یا یک که به ورودی اعمال می شوند پاسخ دهد . (در صورتی که قصد دارید یک شدن پایه را چک کنید ، حتما از مقاومت pull down استفاده کنید) .

PIOx_PUSR : وضعیت فعال یا غیر فعال بودن مقاومت ها در این رجیستر ذخیره می شود .

5- Input Glitch Filtering

گاهی اوقات ممکن است پالسی به میکرو وارد شود که فرکانس آن بیشتر از کلاک PIO باشد و نیاز باشد که CPU به آن پالس پاسخ ندهد (ان را نادیده بگیرد). در سری AT91SAM واحدی به نام Input Glitch Filtering تدارک دیده شده است که می تواند نقاط همزمانی پالس موجود بر روی پایه و کلاک را آشکار کرده و بقیه نقاط را حذف نماید ، در این حالت رجیستر PIOx_PDSR فقط در مواقعی که پالس ورودی دارای عرض مناسب باشد یا با کلاک PIO همزمان شود دارای مقدار می گردد . در شکل زیر وضعیت رجیستر PIOx_PDSR هنگامی که Input Glitch Filtering فعال یا غیر فعال است آورده شده است :



وجود لرزش در هنگام وصل شدن کلید ها ، وجود اتصالات ناقص در ترمینال ها یا وجود نویز بر روی مسیر های متصل شده بر روی پایه های ورودی ، عواملی هستند که ممکن است توسط میکرو به عنوان فشرده شدن کلیدیا تحریک شدن پایه ورودی ، لحاظ شوند . با فعال کردن Input Glitch Filtering میتوان از بروز این حالت ها جلوگیری کرد . Input Glitch Filtering مانند یک فیلتر RC پایین گذر ، بر روی پایه های ورودی عمل میکند .

PIO_IFER : با مقدار دهی این رجیستر می توانید Input Glitch Filtering را بر روی پایه ی دلخواه فعال کنید .

PIO_IFDR : این رجیستر برای غیر فعال کردن فیلتر به کار می رود .

PIO_IFSR : وضعیت فعال یا غیر فعال بودن Input Glitch Filtering بر روی پایه های مختلف با این رجیستر مشخص می شود .

توضیحات :

1- پایه های موجود در میکرو کنترلر های سری AT91SAM می تواند در حالت ورودی یا خروجی راه اندازی شوند . هنگامی که رجیستر PIOx_MDER (Multi-driver Enable Register) را مقدار دهی می کنید ، پایه های مورد نظر چه ورودی باشند چه خروجی به حالت Open Drain می روند . در این حالت مقاومت های pull up ، یک یا صفر بودن پایه و... غیر فعال شده و برای راه اندازی مجدد آن باید تمامی رجیستر ها را مجددا مقدار دهی کنید . با مقدار دهی رجیستر PIOx_MDDR می توانید حالت Multi-driver را غیر فعال کنید ، همچنین رجیستر PIOx_MDSR وضعیت پایه ها را مشخص می کند .

2- اغلب پایه های میکروکنترلر های سری AT91SAM دارای چند نقش کاری هستند ، مثلا پایه PB27 علاوه بر داشتن نقش ورودی/خروجی ، میتواند به عنوان ورودی مبدل آنالوگ به دیجیتال شماره 0 یا پایه خروجی تایمر

شماره 2 پیکربندی شود .با استفاده از رجیستر PIO_ASR و PIO_BSR می توان امکانات جانبی مورد نیاز را فعال کرد . مثال :

```
*AT91C_PIOA_BSR|=AT91C_PA25_SPI1_NPCS1|AT91C_PA21_SPI1_NPCS0|AT91C_PA22_SPI1_SPCK|AT91C_PA23_SPI1_MOSI|AT91C_PA24_SPI1_MISO;
```

پایه های 21 تا 25 پورت A از این بعد به عنوان ورودی خروجی های باس SPI ایفای نقش میکنند و دیگر نمیتوان از آنها به عنوان ورودی / خروجی معمولی استفاده کرد . در بخش های بعدی توضیحات بیشتری در مورد این رجیستر آورده شده است .

3- قبلا گفتیم که PIOx_ODSR رجیستری است که وضعیت صفر یا یک بودن پایه های خروجی در آن قرار داده می شود . این رجیستر فقط خواندنی است و نمی توان عددی را در آن قرار داد .

اما در صورتی که قبلا رجیستر PIOx_OWER را مقدار دهی کرده باشید ، این رجیستر (PIOx_ODSR) از حالت Read-only (فقط خواندنی) بیرون می آید و می توانید مستقیما وضعیت صفر یا یک بودن پایه ها را توسط آن تعیین کنید .

از رجیستر PIOx_OWDR برای غیر فعال سازی این قابلیت و از رجیستر PIOx_OWSR برای دریافت وضعیت پایه ها ی که به صورت بالا صفر یا یک شده اند ، استفاده می شود

مثال :

```
#include <AT91SAM7X256.H>                    /* AT91SAMT7X256 definitions */
void delay_s(void);
int main (void) {
*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in  Read-write mode
while(1){
    *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
   delay_s();
   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared
   delay_s();
}
}
void delay_s (void) {
  unsigned int n;
```

```
  for (n = 0; n < 7372800; n++);
}
```

4- در سری AT91SAM تمامی پایه های قابلیت پیکربندی در حالت ورودی وقفه را دارند ، در این حالت با اعمال شدن پالس مشابه با شرایط برنامه ، CPU ، برنامه ی در دست اجرای خود را رها کرده و به زیر برنامه ی وقفه می رود . برای کار با وقفه ها نیاز مند دانستن نحوه ی کار واحد AIC خواهید بود ، به همین جهت بحث را ادامه نداده و ادامه ی آن را به بخش راه اندازی AIC موکول می کنیم .

5- در تمامی رجیستر ها منظور از x حرف A یا B یا C یا .... به نشانه ی پورت های A و B و c و ... است .

6- هدف از معرفی ثبات ها در این بخش آشنا کردن شما با نحوه ی کار میکرو کنترلر و یاری شما در درک بهتر هدر های که در ادامه آورده می شوند است ، با استفاده از هدر ها دیگری نیازی به کار با رجیستر ها نیست و به می توان به سادگی دستورات بالا را به دستورات ساده و قابل فهم تبدیل کرد .

ایجاد تاخیر در برنامه

گاهی موقع لازم است ، برنامه برای مدتی اجرا نشود ، برای این کار از دستورات تاخیر استفاده میشود ، در KEIL تابعی برای تاخیر (در برنامه) وجود ندارد ، و شما باید برای ایجاد تاخیر CPU را به کار دیگری مشغول کنید ، یکی از این کار ها شمردن اعداد میباشد :

برنامه زیر را را شبیه سازی کنید:

```
#include <AT91SAM7X256.H>                    /* AT91SAMT7X256 definitions */
void delay_s(void);
int main (void) {
*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
while(1){
    *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
   delay_s();

   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared
   delay_s();
}
}
void delay_s (void) {
 unsigned int n;
 for (n = 0; n < 7372800; n++);
}
```

برای ایجاد تاخیر زمانی شما باید یک تابع ایجاد کنید تا در هنگام تاخیر فراخوانی شود ، برای ایجاد تابع باید ابتدا آن را معرفی کنید . برای معرفی کلیه توابع از دستور زیر استفاده میشود : (خود برنامه اصلی نیز یک تابع میباشد)

void نام تابع (نوع فضا) ;

در قسمت نام تابع ، باید یک نام مناسب نوشته شود ؛در برنامه ، تابع با همین نام فراخوانی میشود. در قسمت نوع فضا ، نام و جنس متغیر های مورد نیاز جهت ارسال داده به تابع مشخص میشود ، در صورتی که نیازی به ارسال داده به تابع نباشد از دستور void (فضای خالی) در این مکان استفاده میشود .

با رسیدن CPU به نام تابع که در برنامه اصلی ، CPU به زیر برنامه ی تابع که شکلی مانند زیر دارد پرش میکند :

```
void نام تابع ( نوع فضا) ;
{
 دستورات
}
```

و بعد از انجام دادن دستورات موجود در داخل تابع با رسیدن به اکولاد دوم(}) ، به برنامه ی اصلی برمیگردد .

در برنامه ی بالا ، با رسیدن CPU به دستور delay_s(); ، به تابع void delay_s (void) پرش میشود ، در این تابع یک حلقه ی for وجود دارد که در آن مقدار اولیه ی 0 برای n در نظر گرفته شده است ، با دستور n++ مدام به متغیر n یک واحد اضافه میشود ، هنگامی که n به 1843200 رسید CPU به خط بعد از شرط پرش میکند (به دلیل درست نبودن شرط حلقه)، در آن جا یک اکولاد وجود دارد که CPU را به حلقه ی اصلی برمیگرداند و CPU برنامه را از خط بعد از delay_s(); در حلقه ی اصلی ادامه میدهد .

مقدار تقریبی زمانی که توسط روش بالا ایجاد میشود از رابطه ی زیر بدست میاید :

$$\text{زمان} = \frac{x*5}{f}$$

X مقدار رقمی است که در حلقه ی for شمرده میشود و f مقدار فرکانس کریستال برحسب هرتز میباشد .(البته همانطور که میدانید میزان تاخیر ایجاد شده در این روش به ضرائب PLL و حجم برنامه نیز وابسته است ، فرمول بالا فقط برای مقادیر پیشفرض PLL صادق است و با تغییر کردن مقدار PLL دیگر مورد قبول نخواهد بود . در بخش مدیریت توان ، این فرمول را تکمیلتر خواهیم کرد .)

مثال برنامه ای بنویسید که پایه b.19 میکرو کنترلر AT91SAM7X256 را به مدت 2 ثانیه 0 کند ، بعد به مدت 1.5 ثانیه 1 کند و سپس به مدت 2.5 ثانیه 0 کند و این حلقه مدام تکرار شود .

```
#include <AT91SAM7X256.H>
#include <lib_AT91SAM7X256.h>
int n;
void wait(void ;(
int main (void} (
```

```
  *AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
wait;()
wait;()
wait;()
wait   ;()
    *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
wait;()
wait    ;()
wait;()
   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared
wait;()
wait    ;()
wait;()
wait    ;()
wait;()
}
void wait (void  } (
  for (n = 0; n < 1843200; n;(++
}
```

ما میتوانستیم برنامه ی بالا را به شکل زیر نیز بنویسیم :

```
#include <AT91SAM7X256.H>                    /* AT91SAMT7X256 definitions */
int main (void) {
int n;
*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
while(1){
```

```
    *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
   for (n = 0; n < 7372800; n++);
   *AT91C_PIOB_ODSR = 0x00000000;          // PB.19 to be cleared
for (n = 0; n < 7372800; n++);}
}
```

در برنامه ی بالا حلقه ی for را مستقیما در حلقه ی اصلی برنامه آورده ایم ، اما این برنامه ظاهر مناسبی ندارد و ممکن است خواننده ی بعدی در درک برنامه دچار مشکل شود . به هر حال در ادامه بیشتر در مورد توابع و نحوه ی کار با آنها بحث کرده ایم و این مطالب صرفا برای درک برنامه های که در ادامه آورده شده میباشد .

معرفی هدر lib_AT91SAMxxxxx.h :

یکی از هدر های استانداردی که برای کار با واحد pio و سایر بخش های میکرو کنترلر های سری AT91SAM ارائه شده است هدر های سری lib_AT91SAMxxxxx.hاست . در این هدر ها رجیستر های مربوط به هر بخش به دستورات ساده تر تبدیل شده اند . به دلیل ساده و نزدیک به گفتار بودن این دستورات ، استفاده از آنها به مراتب ساده از کار با رجیستر ها است .

البته یکی از مشکلات عمده ی هدر مذکور طولانی بودن دستورات آن میباشد ، مثلا :

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOB, AT91C_PIO_PB12);
```

دستور بالا که بیش از 50 حرف دارد ، فقط پورت b.12 را به عنوان خروجی پیکربندی میکند ، با این حال این هدر توسط شرکت اتمل ارائه شده و برنامه نویسان مختلف از آن در پروژه های مختلفی استفاده نموده اند و ما چاره ای جز یادگیری دستورات آن نداریم (در صورتی که قصد داشته باشیم سورسی را مطالعه کنیم نیازمند این دستورات خواهیم بود .)

برای استفاده از این کتابخانه باید آن را در مسیر زیر یا پوشه ی پروژه ی خود ، کپی کنید : (کتابخانه پیوست کتاب میباشد)

```
Program Files\Keil\ARM\INC\Atmel.
```

(در پوشه های SAM7Xو SAM7A3 و AT91SAM9G20 و سایر پوشه های موجود).

برای استفاده از این کتابخانه ، ابتدا باید آن را در برنامه فراخوانی کنید ، برای اینکار از دستور زیر استفاده می شود :

```
#include < lib_AT91SAMxxxxx.h >
```

مثلا برای میکرو کنترلر AT91SAM7x256 :

```
#include <lib_AT91SAM7X256.h>
```

در صورتی که هدر را در پوشه ی پروژه کپی کرده باشید باید از دستور زیر استفاده کنید :

```
#include “lib_AT91SAM7X256.h”
```

با فرا خوانی این کتابخانه ، دستوراتی به نرم افزار KEIL اضافه می شود که شرح آنها در ادامه آورده شده است :

تعریف پایه به عنوان خروجی :

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOx, AT91C_PIO_Pxy);
```

دستور بالا پایه y از پورت x را به عنوان خروجی تعریف می کند ، در واقع این دستور تابع زیر که در هدر موجود می باشد را اجرا می کند :

```
__inline void AT91F_PIO_CfgOutput(
        AT91PS_PIO pPio,            // \arg pointer to a PIO controller
        unsigned int pioEnable)     // \arg PIO to be enabled
{
        pPio->PIO_PER = pioEnable; // Set in PIO mode
        pPio->PIO_OER = pioEnable; // Configure in Output
}
```

شما می توانید به جای AT91C_PIO_Pxy از آدرس پایه یا پایه های مورد نظر استفاده کنید .

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOx, y);
```

در دستور بالا پایه های y از پورت x به عنوان خروجی تعریف می شوند(برای درک بهتر موضوع مثال را مشاهده کنید)

مثال : پایه 12 از پورت b و پایه های 9 و 10 و 11 و 16 از پورت A میکرو کنترلر AT91SAM7x256 را به عنوان خروجی تعریف کنید .

روش اول:

پورت b :

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOB, AT91C_PIO_PB12);   // portb.12  is output.
```

پورت a :

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA, AT91C_PIO_PA9);   // porta.9  is output.
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA, AT91C_PIO_PA10);   // porta.10  is output.
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA, AT91C_PIO_PA11);   // porta.11  is output.
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA, AT91C_PIO_PA16);   // porta.16  is output.
```

روش دوم :

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA,AT91C_PIO_PA9|AT91C_PIO_PA10|AT91C_PIO_PA11|AT91C_PIO_PA16);
```

در روش بالا بیت های متناظر با دستورات AT91C_PIO_PA16 و... با یکدیگر OR منطقی شده و نتیجه در دستور AT91F_PIO_CfgOutput قرار گرفته است .

در فایل AT91SAM7X256.h دستور AT91C_PIO_PA16 را جستوجو کنید (این فایل را با Notepad ویندوز باز کنید) ، در خط 2667 این فایل این دستور با دستور زیر ایجاد شده است :

```
#define AT91C_PIO_PA16       (1 << 16) // Pin Controlled by PA16
```

روش چهارم :

ما میتوانیم به جای استفاده از دستورات AT91C_PIO_PA9 و AT91C_PIO_PA10 و AT91C_PIO_PA11 و AT91C_PIO_PA16 از دستورات معادل آنها که در فایل AT91SAM7X256.h اورده شده است نیز استفاده کنیم :

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA,AT91C_PIO_PA9|AT91C_PIO_PA10|AT91C_PIO_PA11|AT9 1C_PIO_PA16);
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA,(1 << 9)|(1 << 10)|(1 << 11)|(1 << 16));
```

در دستور بالا ابتدا دستورات داخل پرانتز اجرا میشوند ، به این ترتیب ، در اولین مرحله عدد 1 نه بار به سمت چپ شیفت داده میشود و نتیجه که عدد 1000000000 است در حافظه ی RAM به صورت موقت ذخیره میشود ، در مرحله ی بعدی عدد 1 ده بار و بعد عدد 1 یازده بار و در نهایت عدد 1 شانزده بار به سمت چپ شیفت داده میشود . در این حالت 4 عدد 1000000000 و 10000000000 و 100000000000 و 1000000000000000که از دستورات موجود در داخل پرانتز ها ایجاد شده اند ، با دستور | با هم OR میشوند ، در نهایت نتیجه برابر با 10000111000000000 در مبنای باینری است که در دستور AT91F_PIO_CfgOutput قرار میگیرد .

روش پنجم :

پورت B :برای پیدا کردن آدرس ، به پایه های که قرار است خروجی شوند ، رقم 1 و به سایر پایه ها رقم صفر تعلق می گیرد :

در تصویر بالا هر مربع نشانگر یک پایه از میکرو می باشد ، در صورتی که اولین led از سمت راست پایه B.0 باشد . سیزدهمین LED نشانگر پورت B.12 است . با قرار دادن رقم یک به جای مربع های علامت دار و رقم صفر به جای دیگر مربع ها ، داده ی پورت در مبنای باینری بدست می اید . با بردن این عدد در مبنای هگز ، آدرس پورت بدست می اید (در نرم افزار KEIL ، آدرس ها باید در مبنای هگز باشند)

72 >>> تبدیل به هگز >>> 1000

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOB, 0x1000);  // portb.12  is output.
```

0X نمایانگر مبنای هگز می باشد ، 0x1000 یعنی 1000 در مبنای هگز .

پورت A :برای پورت A نیز مراحل بالا را تکرار می کنیم ، با تبدیل عدد 10000111000000000 (در مبنای باینری) به مبنای هگز ، آدرس مورد نیاز KEIL برای پیکربندی پایه های مورد نظر به عنوان خروجی به دست می آید :

10000111000000000 >>> تبدیل به هگز >>> 10E00

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA, 0x10E00);  // porta.9  , porta.10  , porta.11  , porta.16  are  output.
```

پیکر بندی پایه به عنوان ورودی :

```
AT91F_PIO_CfgInput(AT91C_BASE_PIOx, AT91C_PIO_Pbxy);
```

دستور بالا پایه y از پورت x را به عنوان ورودی تعریف می کند ، این دستور تابع زیر را در هدر lib_AT91SAM7X256.h اجرا می کند :

```
__inline void AT91F_PIO_CfgInput(
        AT91PS_PIO pPio,          // \arg pointer to a PIO controller
        unsigned int inputEnable)     // \arg PIO to be enabled
{
        // Disable output
        pPio->PIO_ODR  = inputEnable;
        pPio->PIO_PER  = inputEnable;
}
```

همانطور که مشاهده می کنید در این تابع ، ابتدا وضعیت خروجی غیر فعال شده و سپس واحد PIO راه اندازی می شود .

شما می توانید به جای AT91C_PIO_Pxy از آدرس پایه یا پایه های مورد نظر استفاده کنید .آدر س مـی توانـد بـه فـرم هگـز یـا عددی باشد :

```
AT91F_PIO_CfgInput(AT91C_BASE_PIOx, y);
```

در دستور بالا پایه های y از پورت x به عنوان ورودی تعریف می شوند(برای درک بهتر موضوع مثال را مشاهده کنید)

مثال : پایه های B.4 تا B.7 و A.5 و A.0 را به عنوان ورودی پیکربندی کنید :

روش اول :

```
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, AT91C_PIO_PB4);
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, AT91C_PIO_PB5);
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, AT91C_PIO_PB6);
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, AT91C_PIO_PB7);
AT91F_PIO_CfgInput(AT91C_BASE_PIOA, AT91C_PIO_PA5);
AT91F_PIO_CfgInput(AT91C_BASE_PIOA, AT91C_PIO_PA0);
```

روش دوم :

```
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, 0xF0);
AT91F_PIO_CfgInput(AT91C_BASE_PIOA, 0x21);
```

روش سوم :

```
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, (1<<4));
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, (1<<5));
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, (1<<6));
AT91F_PIO_CfgInput(AT91C_BASE_PIOB, (1<<7));
AT91F_PIO_CfgInput(AT91C_BASE_PIOA, (1<<0));
AT91F_PIO_CfgInput(AT91C_BASE_PIOA, (1<<5));
```

در روش دوم باید همانند پیکربندی به عنوان خروجی ، آدرس پایه های مورد نظر را بدست اورید . (مانند مرحله قبل ، بـه پایـه های ورودی رقم 1 و به سایر پایه ها رقم صفر تعلق می گیرد.) ، از روش عددی می توان در پیکربندی بـه عنـوان خروجـی نیـز استفاده کرد .

بعد از پیکربندی پورت به عنوان ورودی یا خروجی ، مـی تـوانیم بـا دسـتورات زیـر داده را بـر روی پـورت بریـزیم یـا از آن بخوانیم .

خاموش کردن پایه (reset pin):

```
AT91F_PIO_ClearOutput(AT91C_BASE_PIOx, AT91C_PIO_Pxy)
```

دستور بالا پایه y از پورت x را صفر می کند .

شما می توانید به جای AT91C_PIO_Pxy از آدرس پایه یا پایه های مورد نظر استفاده کنید .

```
AT91F_PIO_ ClearOutput (AT91C_BASE_PIOx, y);
```

در دستور بالا پایه های y از پورت x صفر می شوند(برای درک بهتر موضوع مثال را مشاهده کنید) .

روشن کردن پایه (set pin) :

```
AT91F_PIO_SetOutput (AT91C_BASE_PIOx, AT91C_PIO_Pxy)
```

دستور بالا پایه y از پورت x را یک می کند .

شما می توانید به جای AT91C_PIO_Pxy از آدرس پایه یا پایه های مورد نظر استفاده کنید .

```
AT91F_PIO_ SetOutput (AT91C_BASE_PIOx, y);
```

در دستور بالا پایه های y از پورت x یک می شوند(برای درک بهتر موضوع مثال را مشاهده کنید) .

مثال : برنامه ای بنویسید که پایه های a.11 , a.16, b.18 , b.17 , b.15 را یک و پایه های a.6 , a.5 , b.9 , b.2 , b.1 را صفر کند (میکرو کنترلر AT91SAM7x256).

طبق آموزش های قبلی اقدام به نوشتن برنامه می کنیم ، اولین خطوط مربوط به فراخوانی هدر ها (معرفی میکرو) می باشد :

```
#include <AT91SAM7X256.H>
#include <lib_AT91SAM7X256.h>
```

در ادامه باید یک حلقه ایجاد کنیم (تمامی برنامه در زبان C به یک حلقه اصلی نیاز دارند):

```
int main (){
```

اکنون باید پایه های مورد نظر را به عنوان خروجی پیکر بندی کنیم :

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOB, 0X68206);
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA, 0X10860);
```

روشن کردن پایه های مورد نظر :

```
AT91F_PIO_SetOutput (AT91C_BASE_PIOA, 0X10800);
AT91F_PIO_SetOutput (AT91C_BASE_PIOB, 0X68000);
```

خاموش کردن پایه های مورد نظر :

```
AT91F_PIO_ClearOutput(AT91C_BASE_PIOA,0X60);
AT91F_PIO_ClearOutput(AT91C_BASE_PIOB,0X206);
```

پایان حلقه :

```
}
```

شما می توانید مقدار یک متغیر را در پورت بریزید :

```
#include “AT91SAM7X256.h"
#include “lib_AT91SAM7X256.h"
int main(){
int a;
a=1023;
AT91F_PIO_CfgOutput(AT91C_BASE_PIOA, a);
AT91F_PIO_ClearOutput(AT91C_BASE_PIOA,a);
}
```

در برنامه بالا مقدار 1024 (در مبنای دسیمال) در متغیر a قرار گرفته است ، با دستور AT91F_PIO_SetOutput(AT91C_BASE_PIOA, a); عدد 1024 به هگز تبدیل شده و در پورت a ریخته می شود . با این عمل 10 پایه اول پورت a روشن خواهد شد .

خواندن پایه یا پورت با دستور زیر انجام می شود :

```
AT91F_PIO_GetInput(AT91C_BASE_PIOx) & AT91C_PIO_Paxy
```

دستور بالا مقدار موجود بر روی پورت را برمی گرداند ، شما می توانید با مقایسه کردن این دستور با مقدار دلخواه (صفر یا یک) وضعیت ورودی را چک کنید .

```
AT91F_PIO_GetInput(AT91C_BASE_PIOy) & y
```

در این دستور امکان چک کردن گروهی پایه ها نیز وجود دارد . y آدرس پایه های است که قصد خواندشان را داریم . همانطور که قبلا توضیح داده شد ، برای خواندن مقدار موجود بر روی یک پورت ابتدا باید کلاک PIO را فعال کنید ، برای فعال کردن کلاک PIO از دستور زیر استفاده می شود :

```
AT91F_PMC_EnablePeriphClock(AT91C_BASE_PMC, 1 << AT91C_ID_PIOx);
```

X می تواند A برای راه اندازی کلاک پورت A و B برای راه اندازی کلاک پورت B باشد .

راه اندازی مقاومت های Pullup داخلی :

```
AT91F_PIO_CfgPullup(AT91C_BASE_PIOx, AT91C_PIO_Pxy);
```

دستور بالا مقاومت Pullup (مقاومت بالا کشنده) پایه y از پورت x را فعال می کند . شما می توانید به جای AT91C_PIO_Pxy از آدرس پایه یا پایه های مورد نظر استفاده کنید .

```
AT91F_PIO_CfgPullup(AT91C_BASE_PIOx, y);
```

در دستور بالا مقاومت پایه های y از پورت x فعال می شوند .

مثال : به پایه A.30 یک کلید و به پایه B.19 یک LED متصل است ، برنامه ای بنویسید که با تحریک کلید led روشن شود:

```
#include <AT91SAM7X256.H>    /* AT91SAMT7X256 definitions */
#include “lib_AT91SAM7X256.h”  //include lib_AT91SAM7X256.h in to project
int main (void) {
AT91F_PMC_EnablePeriphClock(AT91C_BASE_PMC, 1 << AT91C_ID_PIOA);  //enable pio clack
 AT91F_PIO_CfgInput(AT91C_BASE_PIOA, AT91C_PIO_PA30);    // enable porta.30 as output
 AT91F_PIO_CfgPullup(AT91C_BASE_PIOA,AT91C_PIO_PA30);   //enable pull up resistor in   porta.30
 AT91F_PIO_CfgOutput(AT91C_BASE_PIOB,AT91C_PIO_PB19);  // enable portb.19 as output
 AT91F_PIO_ClearOutput(AT91C_BASE_PIOB,AT91C_PIO_PB19); // reset portb.19
 while(1) {
 if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) & AT91C_PIO_PA30) == 0) {   //chake porta.30
  AT91F_PIO_SetOutput(AT91C_BASE_PIOB, AT91C_PIO_PB19); //if porta.30 is 0 set portb.19
 }}}
```

در جدول زیر کلیه ی دستوراتی که برای کار با پورت ها در هدر lib_AT91SAMxxxxx.h وجود دارد اورده شده است:

عمل کرد	نام دستور	عمل کرد	نام دستور
فعال سازی پورت در حالت	AT91F_PIO_OutputEnable	پیکربندی پورت در حالت کنترل	AT91F_PIO_CfgPeriph

خروجی		وسلیه ی جانبی	
ساقط کردن پورت از حالت خروجی	AT91F_PIO_OutputDisable	پیکربندی پورت در حالت خروجی	AT91F_PIO_CfgOutput
دریافت وضعیت واحد PIO (خروجی یا معلق)	AT91F_PIO_GetOutputStatus	پیکربندی پورت در حالت ورودی	AT91F_PIO_CfgInput
تست کردن خروجی بودن پورت	AT91F_PIO_IsOutputSet	پیکربندی پورت در حلت معلق (Open drain)	AT91F_PIO_CfgOpendrain
فعال سازی فیلتر ورودی (Glitch Input Filter)	AT91F_PIO_InputFilterEnable	پیکربندی مقاومت های PULL UP	AT91F_PIO_CfgPullup
غیر فعال سازی فیلتر ورودی (Glitch Input Filter)	AT91F_PIO_InputFilterDisable	پیکربندی پورت ها برای تحریک مستقیم	AT91F_PIO_CfgDirectDrive
دریافت وضعیت فیلتر ورودی (فعال یا غیر فعال)	AT91F_PIO_GetInputFilterStatus	پیکربندی فیلتر ورودی (Glitch Input Filter)	AT91F_PIO_CfgInputFilter
تست کردن فعال بودن فیلتر ورودی	AT91F_PIO_IsInputFilterSet	دریافت وضعیت پورت دلخواه ورودی (صفر یا یک)	AT91F_PIO_GetInput
دریافت وضعیت پروت های خروجی (صفر یا یک)	AT91F_PIO_GetOutputDataStatus	تست کردن PIO برای حالت ورودی	AT91F_PIO_IsInputSet
فعال سازی پورت در حالت ورودی وقفه	AT91F_PIO_InterruptEnable	یک کردن پورت دلخواه	AT91F_PIO_SetOutput
لغو کردن پورت از حالت ورودی وقفه	AT91F_PIO_InterruptDisable	صفر کردن پورت دلخواه	AT91F_PIO_ClearOutput
دریافت پوشش وقفه	AT91F_PIO_GetInterruptMaskStatus	مقدار دهی PIO_ODSR (به صفحه ی 64 مراجعه کنید)	AT91F_PIO_ForceOutput
در یافت نوع وقفه	AT91F_PIO_GetInterruptStatus	فعال کردن واحد PIO	AT91F_PIO_Enable
تست وجود پوشش برای وقفه	AT91F_PIO_IsInterruptMasked	غیر فعال کردن واحد PIO	AT91F_PIO_Disable
تست وجود وقفه	AT91F_PIO_IsInterruptSet	دریافت وضعیت واحد PIO (فعال یا غیر فعال)	AT91F_PIO_GetStatus
به صفحه ی 64 مراجعه کنید	AT91F_PIO_MultiDriverEnable	تست کردن یک بودن پورت	AT91F_PIO_IsSet
به صفحه ی 64 مراجعه کنید	AT91F_PIO_OutputWriteDisable	به صفحه ی 64 مراجعه کنید	AT91F_PIO_MultiDriverDisable

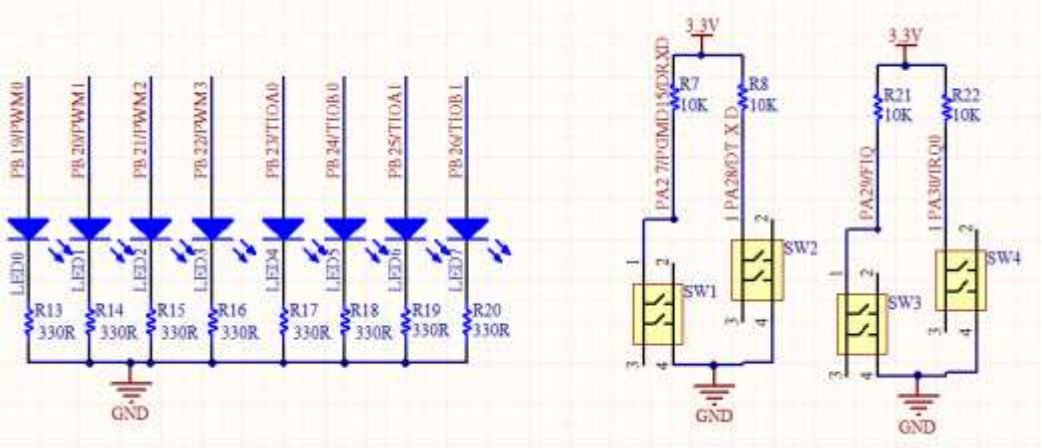
به صفحه ی 64 مراجعه کنید	AT91F_PIO_GetOutputWriteStatus	به صفحه ی 64 مراجعه کنید	AT91F_PIO_GetMultiDriverStatus
به صفحه ی 64 مراجعه کنید	AT91F_PIO_IsOutputWriteSet	به صفحه ی 64 مراجعه کنید	AT91F_PIO_IsMultiDriverSet
پیکربندی مقاومت های PULL UP	AT91F_PIO_GetCfgPullup	به صفحه ی 64 مراجعه کنید	AT91F_PIO_A_RegisterSelection
به صفحه ی 64 مراجعه کنید	AT91F_PIO_IsOutputDataStatusSet	به صفحه ی 64 مراجعه کنید	AT91F_PIO_B_RegisterSelection
تست وجود مقاومت های PULL UP	AT91F_PIO_IsCfgPullupStatusSet	به صفحه ی 64 مراجعه کنید	AT91F_PIO_Get_AB_RegisterStatus
به صفحه ی 64 مراجعه کنید	AT91F_PIO_OutputWriteEnable	به صفحه ی 64 مراجعه کنید	AT91F_PIO_IsAB_RegisterSet

در بخش های بعدی با سایر دستورات جدول آشنا خواهیم شد.

- در جدول بالا برخی از دستورات مکمل هم هستند ، مثلا هنگامی که دستور AT91F_PIO_CfgOutput اجرا می شود ، مانند این هست که دستورات AT91F_PIO_Enable و AT91F_PIO_OutputEnable با هم اجرا شوند.
- برای کسب اطلاعات بیشتر نام هر دستور را در فایل lib_AT91SAM7X256.h یا سایر فایل های هدر مشابه جستو جو کنید .
- در این جدول پورت یک واژه ی کلی و به معنای یک یا چند پایه می باشد .
- با استفاده از کتابخانه ی pio.h که ادامه آورده شده است ، شکل ساده تری از دستورات در اختیار شما میگیرد .

مثال :

در این مثال که شماتیک آن را در ادامه مشاهده میکنید ، تعداد 4 عدد کلید و 8 عدد LED به میکروکنترلر متصل شده است .در حالت عادی هشت LED موجود با تاخیر تقریبی 500 میلی ثانیه روشن و خاموش میشوند .با فشردن sw4 ، LED ها به ترتیب و از راست به چپ روشن و خاموش میشوند ، کلید sw3 جهت چرخش را تغییر میدهد .شما میتوانید با فشردن و نگه داشتن کلید های sw1 و sw2 سرعت خاموش و روشن شدن led ها را به ترتیب کم و زیاد کنید.



همانطور که در تصویر بالا مشاهده میکنید ، led ها به پایه های 19 تا 26 پورت B و کلید ها به پایه های 27 تا 30 پورت A میکرو کنترلر AT91SAM7X256 متصل شده اند ، در صورتی که مقاومت های PULLUP داخلی میکرو را فعال نمایید ، نیازی به اتصال مقاومت های R7 و R8 و R21 و R22 نمی باشد . برنامه ی میکرو :

```
#include <AT91SAM7X256.H>
#include “lib_AT91SAM7X256.h”
 int I, n,LEDSpeed= 1000000;
74ubroutin led_mask[] = { AT91C_PIO_PB19, AT91C_PIO_PB20, AT91C_PIO_PB21, AT91C_PIO_PB22,AT91C_PIO_PB23,
AT91C_PIO_PB24, AT91C_PIO_PB25, AT91C_PIO_PB26};
void left_to_right(void) ;
void right_to_left(void);
 void change_speed_roll (void) ;
 void wait (void) ;
int main (void) {
  AT91F_PIO_CfgOutput(AT91C_BASE_PIOB, 0x07F80000);
  AT91F_PMC_EnablePeriphClock(AT91C_BASE_PMC, 1 << AT91C_ID_PIOA);
  AT91F_PIO_CfgInput(AT91C_BASE_PIOA, 0x78000000);
 // AT91F_PIO_CfgPullup(AT91C_BASE_PIOA,0x78000000);
            while(1){
  AT91F_PIO_SetOutput(AT91C_BASE_PIOB, 0x07F80000);
wait();
              AT91F_PIO_ClearOutput(AT91C_BASE_PIOB, 0x07F80000);
wait();
}}
```

```
void wait (void) {
  for (n = 0; n <LEDSpeed; n++);
  change_speed_roll();
}
   void change_speed_roll (void) {
  if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) & AT91C_PIO_PA27) == 0)
   LEDSpeed =LEDSpeed-50000;
  if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) &AT91C_PIO_PA28) == 0)
      LEDSpeed=LEDSpeed+50000;
if (LEDSpeed>=3000000)
      LEDSpeed=1000000;
  if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) & AT91C_PIO_PA29) == 0)    //chake porta.30
right_to_left();
if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) & AT91C_PIO_PA30) == 0)   //chake porta.30
left_to_right();
}
void right_to_left(void) {
 AT91F_PIO_SetOutput(AT91C_BASE_PIOB, 0x07F80000);
  while(1){
    for (I = 0; I <8 ; i++) {
      AT91F_PIO_ClearOutput(AT91C_BASE_PIOB, led_mask[i]);
      wait();
      AT91F_PIO_SetOutput  (AT91C_BASE_PIOB, led_mask[i]);
      wait();
              }}}
void left_to_right(void) {
 AT91F_PIO_SetOutput(AT91C_BASE_PIOB, 0x07F80000);
 while(1){
    for (I =7; I >= 0; i--) {
      AT91F_PIO_ClearOutput(AT91C_BASE_PIOB, led_mask[i]);
      wait();
      AT91F_PIO_SetOutput  (AT91C_BASE_PIOB, led_mask[i]);
      wait();
              }}}
```

توضیح برنامه :

```
#include <AT91SAM7X256.H>
#include “lib_AT91SAM7X256.h”
 int I, n,LEDSpeed= 1000000;
76ubroutin led_mask[] = { AT91C_PIO_PB19, AT91C_PIO_PB20, AT91C_PIO_PB21, AT91C_PIO_PB22,AT91C_PIO_PB23,
AT91C_PIO_PB24, AT91C_PIO_PB25, AT91C_PIO_PB26};
void left_to_right(void) ;
void right_to_left(void);
 void change_speed_roll (void) ;
 void wait (void) ;
```

در اولین بخش برنامه ، دو کتابخانه ی AT91SAM7X256.H و lib_AT91SAM7X256.h معرفی شده است .

در بخش بعدی سه متغیر معمولی i و n و LEDSpeed با مقدار اولیه 1000000 معرفی شده است . آرایه ی int led_mask[] که دارای بینهایت خانه است ، برای ذخیره کردن آدرس پایه ها معرفی و استفاده شده است .در خطوط آخر این بخش ، نام 4 تابعی که در انتهای برنامه مورد استفاده قرار گرفته است ، آورده شده است . در صورتی که این توابع قبل از تابع اصلی آورده می شد نیازی به این دستورات نبود .

```
int main (void) {
  AT91F_PIO_CfgOutput(AT91C_BASE_PIOB, 0x07F80000);
  AT91F_PMC_EnablePeriphClock(AT91C_BASE_PMC, 1 << AT91C_ID_PIOA);
  AT91F_PIO_CfgInput(AT91C_BASE_PIOA, 0x78000000);
 // AT91F_PIO_CfgPullup(AT91C_BASE_PIOA,0x78000000);
            while(1){
  AT91F_PIO_SetOutput(AT91C_BASE_PIOB, 0x07F80000);
wait();
              AT91F_PIO_ClearOutput(AT91C_BASE_PIOB, 0x07F80000);
wait();
}}
```

این بخش از برنامه تابع اصلی یا main برنامه میباشد . با دستور AT91F_PIO_CfgOutput مقدار 7F80000 هگز (11111111000000000000000000 در مبنای باینری) در ریجستر فعال ساز پورت B قرار داده شده و باعث فعال شدن آن در حالت خروجی میشود .دستوراتAT91F_PMC_EnablePeriphClock و AT91F_PIO_CfgInput به ترتیب باعث فعال سازی

کلاک پورت A و فعال سازی آن در حالت خروجی میشود .(در صورتی که کلاک پورتی ، فعال نشود ، نمی توان از آن در حالت ورودی استفاده کرد) .

دستور اختیاری AT91F_PIO_CfgPullup باعث فعال شدن مقاومت بالاکش (PULLUP) بر روی پایه های مذکور میشود ، در صورتی که کلید های ورودی را به مقاومت بالاکش متصل نکرده اید میتوانید از این دستور استفاده کنید . (مقدار 78000000 هگز (برابر با 11110000000000000000000000000000 در مبنای باینری) باعث راه اندازی پایه های 27 تا 30 پورت A میشود) .

در ادامه ی این بخش یک حلقه ی بینهایت توسط دستور while(1) ایجاد شده که در آن تمامی 8 پایه ای که قبلا به عنوان خروجی تعریف شده بود با تاخیر WAIT چشمک میزنند . هنگامی که CPU به دستور wait(); میرسد ، به تابع void wait (void) پرش کرده و بعد از اجرا کردن دستورات موجود در آن به ادامه ی برنامه باز می گردد .

```
void wait (void) {
  for (n = 0; n <LEDSpeed; n++);
  change_speed_roll();
}
```

در تابع wait با استفاده از حلقه ای که توسط دستور شرطی for بوجود آمده ، cpu از عدد 0 تا LEDSpeed میشمارد . ما در ابتدا ی برنامه مقدار اولیه 1000000 را برای این متغیر در نظر گرفتیم ، در این حالت تاخیر بوجود امده تقریبا برابر با 500 میلی ثانیه است . (در ادامه در مورد نحوه ی ایجاد تاخیر بیشتر بحث خواهیم نمود) .

در خط بعدی تابع void change_speed_roll (void) با دستور change_speed_roll(); فراخوانی میشود ، مثل همیشه ، هنگامی که CPU به چنین دستور میرسد ، به تابع معرفی شده پرش کرده و بعد از اجرا کردن دستورات موجود در آن به ادامه ی برنامه باز میگردد (ادامه ی برنامه در اینجا پایان تابع wait است که cpu را به حلقه ی main باز میگرداند .).

```
   void change_speed_roll (void) {
  if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) & AT91C_PIO_PA27) == 0)
   LEDSpeed =LEDSpeed-50000;
  if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) &AT91C_PIO_PA28) == 0)
      LEDSpeed=LEDSpeed+50000;
if (LEDSpeed>=3000000)
      LEDSpeed=1000000;
```

```
 if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) & AT91C_PIO_PA29) == 0)    //chake porta.30
right_to_left();
if ((AT91F_PIO_GetInput(AT91C_BASE_PIOA) & AT91C_PIO_PA30) == 0)   //chake porta.30
left_to_right();
}
```

همانور که قبلا اشاره شد ، cpu با افزودن مقدار متغیر n از صفر تا LEDSpeed (1000000) تاخیری را بوجود می آورد . در این بخش با استفاده از دو دستور شرطی اول ، میتوانیم مقدار این متغیر را کم یا زیاد کنیم . برای جلوگیری از تجاوز مقدار متغیر از مقدار قابل قبول ، دستور if (LEDSpeed>=3000000) تدارک دیده شده است ، در صورتی که مقدار متغییر از 3000000 بیشتر شود ، مقدار آن به مقدار اولیه تغییر می یابد .

دو شرط بعدی cpu را به توابع void right_to_left(void) و void left_to_right(void) که الگوی چرخش led ها هستند هدایت میکنند .

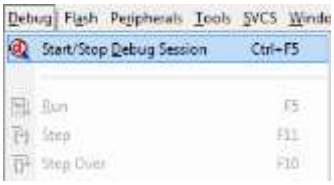
```
void right_to_left(void) {
 AT91F_PIO_SetOutput(AT91C_BASE_PIOB, 0x07F80000);
  while(1){
    for (I = 0; I <8 ; i++) {
      AT91F_PIO_ClearOutput(AT91C_BASE_PIOB, led_mask[i]);
      wait();
      AT91F_PIO_SetOutput  (AT91C_BASE_PIOB, led_mask[i]);
      wait();
              }}}
void left_to_right(void) {
 AT91F_PIO_SetOutput(AT91C_BASE_PIOB, 0x07F80000);
 while(1){
    for (I =7; I >= 0; i--) {
      AT91F_PIO_ClearOutput(AT91C_BASE_PIOB, led_mask[i]);
      wait();
      AT91F_PIO_SetOutput  (AT91C_BASE_PIOB, led_mask[i]);
      wait();
              }}}
```

عمل کرد این دو تابع کاملا مشابه میباشد و تنها تفاوت موجود در آنها جهت چرخش led روشن است . در اولین بخش این تابع تمامی led ها روشن میشوند . با دستور for موجود ، که مقدار 0 تا 7 را به متغیر i میدهد ، پایه ی I ام از پورت b روشن و بعد از تاخیر wait خاموش میشود .

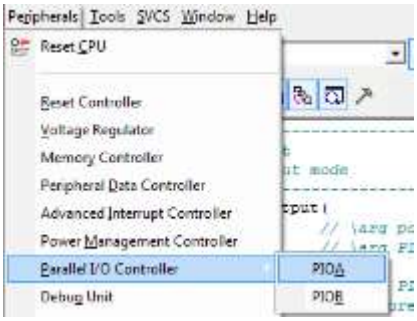
در این بخش با فرخوانی تابع wait میتوانید جهت و سرعت روشن و خاموش شدن led ها رو تغییر دهید .

شبیه سازی برنامه بالا در نرم افزار KEIL :

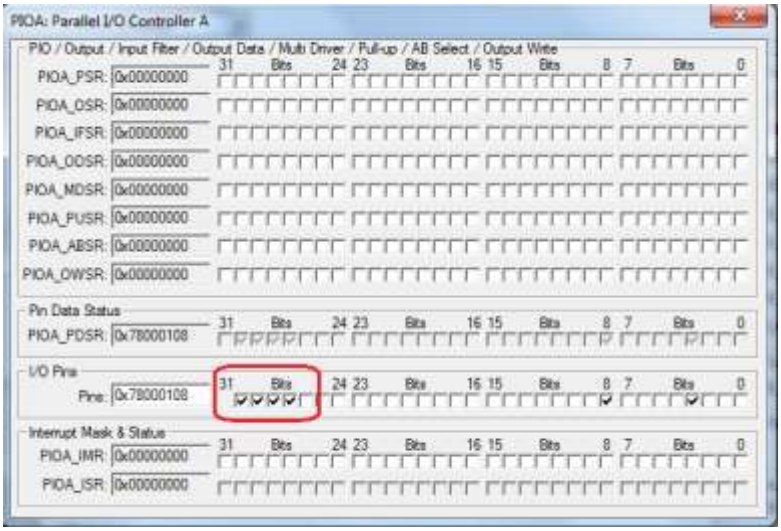
بعد از کامپایل کردن برنامه ، وارد محیط شبیه سازی شوید :



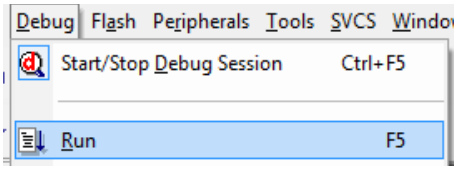
از منوی peripherals و بخش parallel I/O controller گزینه های PIOA و PIOB را انتخاب کنید :



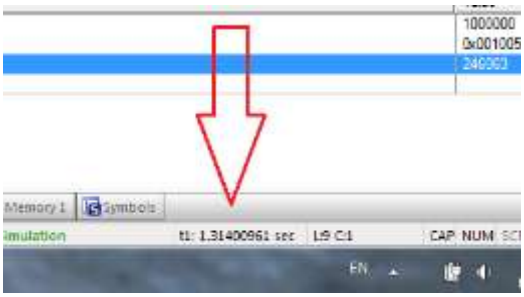
در پنجره parallel I/O controller وجود علامت به نشانه یک شدن و عدم وجود آن نماینگر صفر بودن پایه است ، با توجه به استفاده از سطح صفر در دستورات شرطی مربوط به پایه های ورودی ، در پنجره PIOA: parallel I/O controller و در بخش I/O pins پایه A.27 تا A.30 را تیک بزنید :



از منوی debug گزینه ی run را انتخاب کنید :



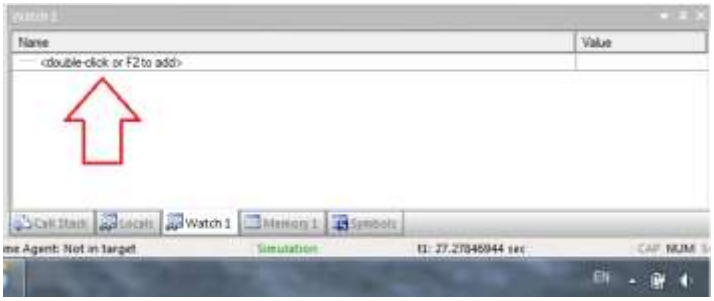
زمان سپری شده از اجرای برنامه در پایین نرم افزار (گوشه سمت راست مانیتور) نمایش داده میشود ، وضعیت پایه های b.19 تا b.22 در پنجره PIOB: parallel I/O controller تقریبا هر 600 میلی ثانیه تغییر میکند.



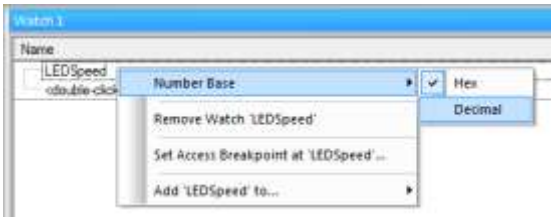
در پنجره PIOA : parallel I/O controller تیک پایه PA.30 را بردارید ، مشاهده میکنید که در پنجره PIOB: parallel I/O controller وضعیت پایه ها به صورت چرخشی تغییر میکند (به زمان سپری شده از آغاز شبیه سازی دقت کنید ، شبیه سازی با سرعت کمی انجام میشود) . پایه PA.30 را تیک بزنید .

از منوی View و زیر منوی Watch windows گزینه ی watch1 را انتخاب کنید :

در پالت باز شده و محل مشخص شده دو بار کلیک کنید و نام یکی از متغیر های موجود در برنامه ، مانند LEDSpeed را وارد کنید و کلید enter را بفشارید :



مطابق تصویر زیر بر روی نام متغیر کلیک راست کنید و منوی Number base گزینه ی decimal را انتخاب کنید تا مقدار متغیر در فرمت دسیمال نمایش داده شود :



متغیر n را نیز به پنجره ی watch1 اضافه کنید و فرمت نمایش آن را دسیمال تغییر دهید .

در پنجره PIOA : parallel I/O controller تیک پایه PA.27 را بردارید و متغیر LEDSpeed را در پنجره ی watch1 چک کنید ، همانطور که مشاهده میکنید مقدار این متغیر بعد از سپری شده هر واحد تاخیر ، 50000 هزار واحد کم میشود . وقتی که LEDSpeed به 50000 پایه PA.27 را تیک بزنید .

تیک پایه های 28 و 29 پورت A را بردارید و نتیجه را مشاهده کنید .

معرفی هدر pio.h :

همانطور که قبلا نیز اشاره کردیم ، یکی از مشکلات عمده ی کار با هدر lib_AT91SAMxxxxx.h طولانی بودن دستورات و دشواری به خاطر سپردن آنها بود . برای رفع این مشکل میتوانید از هدر PIO.H استفاده کنید ، در این هدر کلیه دستورات مربوط به پورت ها تقریبا مشابه دستورات نرم افزار های کدویژن و ccs میباشند ، و با وجود اندازه ی کوتاه ، یادگیری و به خاطر سپردن آنها به مراتب ساده تر خواهد بود ، برای مثال می توانیم با وارد کردن این کتابخانه به نرم افزار از دستور PORTB_OUTPUT=0x01; به جای دستور زیر استفاده کنیم :

```
AT91F_PIO_CfgOutput(AT91C_BASE_PIOB, AT91C_PIO_PB0);
```

برای استفاده از این کتاب خانه باید آن را در مسیر زیر یا پوشه ی پروژه ی خود ، کپی کنید : (کتابخانه در پوشه ی پیوست کتاب موجود میباشد)

```
Program Files\Keil\ARM\INC\Atmel.
```

(در پوشه های SAM7X و SAM7A3 و AT91SAM9G20 و سایر پوشه های موجود).

برای استفاده از این کتابخانه ، ابتدا باید آن را در برنامه فراخوانی کنید ، برای اینکار از دستور زیر استفاده می شود :

```
#include <pio.h>
```

در صورتی که هدر را در پوشه ی پروژه کپی کرده باشید باید از دستور زیر استفاده کنید :

```
#include “pio.h”
```

با فرا خوانی این کتابخانه ، دستوراتی به نرم افزار KEIL اضافه می شود که شرح آنها در ادامه آورده شده است :

تعریف پایه به عنوان خروجی :

```
PORTx_OUTPUT=var
```

دستور بالا پایه های پورت x که آدرس آنها در متغیر var وجود دارد را به عنوان خروجی تعریف می کند . شما همچنین میتوانید به جای VAR از PA0 تا PA31 و PB0 تا PB31 (و PC0 تا PC31 و PD0 تا PD31 برای میکرو کنترلر های که بیشتر از دو پورت دارند) استفاده نمایید .

مثال : پایه 12 از پورت b و پایه های 9 و 10 و 11 و 16 از پورت a میکرو کنترلر AT91SAM7x256 را به عنوان خروجی تعریف کنید .

روش اول:

پورت b :

```
PORTB_OUTPUT=PB12;    // portb.12  is output.
```

پورت a :

```
PORTA_OUTPUT=PA9);    // porta.9  is output.
PORTA_OUTPUT=PA10;    // porta.10  is output.
PORTA_OUTPUT=PA11;    // porta.11  is output.
PORTA_OUTPUT=PA16;    // porta.16  is output.
```

روش دوم :

پورت b :

برای پیدا کردن آدرس ، مشابه کتابخانه ی lib_AT91SAMxxxxx.h ، به پایه های که قرار است خروجی شوند ، رقم 1 و به سایر پایه ها رقم صفر تعلق می گیرد :

در تصویر بالا هر مربع نشانگر یک پایه از میکرو می باشد ، در صورتی که اولین led از سمت راست پایه B.0 باشد . سیزدهمین LED نشانگر پورت B.12 است . با قرار دادن رقم یک به جای مربع های علامت دار و رقم صفر به جای دیگر مربع ها ، داده ی پورت در مبنای باینری بدست می اید . با بردن این عدد در مبنای هگز ، آدرس پورت بدست می اید (در نرم افزار KEIL ، آدرس ها باید در مبنای هگز باشند)

72 >>> تبدیل به هگز >>> 1000

```
PORTB_OUTPUT=0x1000;    // portb.12  is output.
```

0X نمابانگر مبنای هگز می باشد ، 0x1000 یعنی 1000 در مبنای هگز .

پورت A :

برای پورت A نیز مراحل بالا را تکرار می کنیم ، با تبدیل عدد 10000111000000000 (در مبنای باینری) به مبنای هگز ، آدرس مورد نیاز kiel برای پیکربندی پایه های مورد نظر به عنوان خروجی به دست می آید :

10000111000000000 >>> تبدیل به هگز >>> 10E00

```
PORTA_OUTPUT=0x10E00;    // porta.9  , porta.10  , porta.11  , porta.16  are  output.
```

پیکر بندی پایه به عنوان ورودی :

```
PORTx_INPUT=VAR
```

دستور بالا پایه های پورت x که آدرس آنها در متغیر var وجود دارد را به عنوان ورودی تعریف می کند . شما همچنین میتوانید به جای VAR از PA0 تا PA31 و PB0 تا PB31 (و PC0 تا PC31 و PD0 تا PD31 برای میکرو کنترلر های که بیشتر از دو پورت دارند) استفاده نمایید .

مثال :پایه های b.4 تا b.7 و a.5 و a.0 را به عنوان ورودی پیکربندی کنید :

روش اول :

```
PORTB_INPUT= PB4;
PORTB_INPUT= PB5;
PORTB_INPUT= PB6;
PORTB_INPUT= PB7;
PORTA_INPUT= PA5;
PORTA_INPUT= PA0;
```

یا :

```
PORTB_INPUT=( PB4|PB5|PB6|PB7);
PORTA_INPUT= (PA5| PA0);
```

روش دوم :

```
PORTB_INPUT= 0xF0;
PORTA_INPUT= 0x21;
```

روش سوم :

```
PORTB_INPUT= (1<<4);
PORTB_INPUT= (1<<5);
PORTB_INPUT= (1<<6);
PORTB_INPUT= (1<<7);
PORTA_INPUT= (1<<0);
PORTA_INPUT= (1<<5);
```

در روش دوم باید همانند پیکربندی به عنوان خروجی ، آدرس پایه های مورد نظر را بدست اورید . (مانند مرحله قبل ، به پایه های ورودی رقم 1 و به سایر پایه ها رقم صفر تعلق می گیرد.) ، از روش عددی می توان در پیکربندی به عنوان خروجی نیز استفاده کرد . بعد از پیکربندی پورت به عنوان ورودی یا خروجی ، می توانیم با دستورات زیر داده را بر روی پورت بریزیم یا از آن بخوانیم .

خاموش کردن پایه (reset pin):

```
RST_PORTx=VAR
```

دستور بالا پایه های پورت x که آدرس آنها در متغیر var وجود دارد را خاموش می کند . شما همچنین میتوانید به جای VAR از PA0 تا PA31 و PB0 تا PB31 (و PC0 تا PC31 و PD0 تا PD31 برای میکرو کنترلر های که بیشتر از دو پورت دارند) استفاده نمایید .

روشن کردن پایه (set pin) :

```
SET_PORTx= VAR
```

دستور بالا پایه های پورت x که آدرس آنها در متغیر var وجود دارد را روشن می کند . شما همچنین میتوانید به جای VAR از PA0 تا PA31 و PB0 تا PB31 (و PC0 تا PC31 و PD0 تا PD31 برای میکرو کنترلر های که بیشتر از دو پورت دارند) استفاده نمایید .

تغییر وضعیت مستقیم :

```
PORTx= VAR
```

در این روش میتوانید وضعیت تمام پایه های پورت x را به جای var قرار دهید . برای این دستور نیز کافی است پایه های روشن را یک و پایه های خاموش را صفر در نظر بگیرید و سپس رقم حاصله را به هگز تبدیل کنید .

مثال : برنامه ای بنویسید که پایه های a.11 , a.16, b.18 , b.17 , b.15 را یک و پایه های a.6 , a.5 , b.9 , b.2 , b.1 را صفر کند (میکرو کنترلر AT91SAM7x256).

طبق آموزش های قبلی اقدام به نوشتن برنامه می کنیم ، اولین خطوط مربوط به فراخوانی هدر ها (معرفی میکرو) می باشد :

```
#include <AT91SAM7X256.H>
#include “pio.h”
```

در ادامه باید یک حلقه ایجاد کنیم (تمامی برنامه در زبان C به یک حلقه اصلی نیاز دارند):

```
int main (){
```

اکنون باید پایه های مورد نظر را به عنوان خروجی پیکر بندی کنیم :

```
PORTB_OUTPUT =0X68206;
PORTA_OUTPUT =0X10860;
```

روشن کردن پایه های مورد نظر :

```
SET_PORTA= 0X10800;
SET_PORTB= 0X68000;
/* OR
PORTA=0X10800;
PORTB=0X68000;
*/
```

خاموش کردن پایه های مورد نظر :

```
RST_PORTA=0X60;
RST_PORTB=0X206;
```

```
/* OR
PORTA=0X00;
PORTB=0X00;
*/
```

پایان حلقه :

```
}
```

شما می توانید مقدار یک متغیر را در پورت بریزید :

```
#include “AT91SAM7X256.h"
#include “pio.h”


 int main(){
int a;
a=128;
PORTA_OUTPUT=a;
SET_PORTA= a;
}
```

در برنامه بالا مقدار 128 (در مبنای دسیمال) در متغیر a قرار گرفته است ، با دستور SET_PORTA= a; عدد 128 به هگز تبدیل شده و در پورت a ریخته می شود . با این عمل 8 پایه اول پورت a روشن خواهد شد .

خواندن وضعیت پایه ی ورودی :

هنگامی که پایه ای را به عنوان ورودی معرفی میکنید ، وضعیت ان پایه در متغیر PINx(y) که از کتابخانه ی pio.h برگردانده میشود ، ذخیره میگردد . در این حالت وضعیت پایه همیشه در دسترس شما قرار دارد و شما میتوانید در هر زمان آن را با دستورات شرطی چک کنید .

با استفاده از این دستور نیازی به فعال سازی کلاک pio و سایر تنظیمات اضافی نخواهید داشت ، برای کسب اطلاعات بیشتر مثال صفحه ی بعد را مشاهده کنید .

راه اندازی مقاومت های Pullup داخلی :

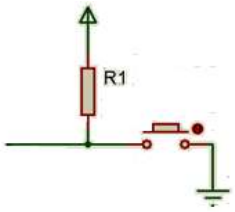
```
PULL_UP_x=VAR
```

دستور بالا مقاومت بالا کش پایه های پورت x که آدرس آنها در متغیر var وجود دارد را فعال می کند . شما همچنین میتوانید به جای VAR از PA0 تا PA31 و PB0 تا PB31 (و PC0 تا PC31 و PD0 تا PD31 برای میکرو کنترلر های که بیشتر از دو پورت دارند) استفاده نمایید .مثال :

به پایه a.30 یک کلید و به پایه b.19 یک led متصل است ، برنامه ای بنویسید که با تحریک کلید led روشن شود :

```
#include <AT91SAM7X256.H>    /* AT91SAMT7X256 definitions */
#include “pio.h”


int main (void) {
  PORTA_INPUT= PA30;        // enable porta.30 as output
  PULL_UP_A= PA30;          //enable pull up resistor in   porta.30
  PORTB_OUTPUT= PB19;   // enable portb.19 as output
RST_PORTB= PB19; // reset portb.19
  while(1) {
  if (PINA(30) == 0){   //chake porta.30
   SET_PORTB= PB19;         //if porta.30 is 0 set portb.19
  }
}
}
```

با فعال کردن مقاومت Pullup ، نیازی به قرار دادن مقاومت خارجی نیست ، در این حالت مقاومتی مانند R1 پایه را از داخل پایه ی ورودی را به VCC متصل می کند .



برای غیر فعال کردن مقاومت PULL-UP کافی است پایه را یک بار به عنوان ورودی تعریف کنید . (هر چند رجیستر مخصوصی برای این کار وجود دارد) .

- در ابتدا ی هدر PIO.H کلیه ی دستوراتی که برای کار با پورت ها وجود دارد آورده شده است در بخش های بعدی یا این دستورات بیشتر آشنا می شویم
- در هدر PIO.H از دستوراتی که کمتر استفاده میشوند ، صرف نظر شده است . برای دستیابی به امکانات این دستورات از رجیستر ها استفاده کنید .
- در این بخش پورت یک واژه ی کلی و به معنای یک یا چند پایه می باشد .
- مهم نیست از کدام دستور یا هدر برای نوشتن برنامه ی خود استفاده میکنید ، مهم این است که برنامه ی شما قابل فهم بوده و فردی که بعدا از آن استفاده میکند گمراه نشود .

مثال :

پروژه ی موجود در صفحه ی 72 که با کتابخانه ی PIO.H نوشته شده است را در زیر مشاهده میکنید :

```
#include <AT91SAM7X256.H>
#include “pio.h”
 int I, n,LEDSpeed= 1000000;
88ubroutin led_mask[] = { PB19, PB20, PB21, PB22,PB23, PB24, PB25, PB26};
void left_to_right(void) ;
void right_to_left(void);
 void change_speed_roll (void) ;
 void wait (void) ;
int main (void) {
  PORTB_OUTPUT= 0x07F80000;
 PORTA_INPUT= 0x78000000;
 // PULL_UP_A=0x78000000;
            while(1){
SET_PORTB= 0x07F80000; //OR  PORTB= 0x07F80000
wait();
RST_PORTB= 0x07F80000; //OR PORTB= 0x00
wait();}}
void wait (void) {
  for (n = 0; n <LEDSpeed; n++);
  change_speed_roll();}
   void change_speed_roll (void) {
  if (PINA(27) == 0)
```

```
  LEDSpeed =LEDSpeed-50000;
 if (PINA(28) == 0)
     LEDSpeed=LEDSpeed+50000;
if (LEDSpeed>=3000000)
     LEDSpeed=1000000;
 if (PINA(29) == 0)    //chake porta.30
right_to_left();
if (PINA(30) == 0)   //chake porta.30
left_to_right();}
void right_to_left(void) {
SET_PORTB= 0x07F80000;
 while(1){
   for (I = 0; I <8 ; i++) {
RST_PORTB=led_mask[i];
    wait();
SET_PORTB=led_mask[i];
    wait();
            }}}
void left_to_right(void) {
SET_PORTB= 0x07F80000;
 while(1){
   for (I =7; I >= 0; i--) {
RST_PORTB=led_mask[i];
    wait();
SET_PORTB=led_mask[i];
    wait();
            }}}
```

آشنایی با توابع و نحوه ی نوشتن کتابخانه

قبل از خواندن مطالب این فصل شما باید با نحوه ی پیکربندی منابع کلاک میکروکنترلرهای سری AT911SAM آشنایی داشته باشد ، بدین منظور حتما فایل " راه اندازی واحد های PMC و CLKG.pdf " نوشته آقای آرمین غنی ، که در پوشه پیوست این کتاب آورده شده است را مطالعه کنید .

معرفی دستورات کار با تابع :

توابع در زبان C یعنی همه چیز! توابع ارکان یک برنامه C را تشکیل می دهند. هر برنامه C از مجموعه توابعی تشکیل شده که تابع اصلی آن main() است.پس از شروع برنامه اولین تابعی که اجرا می شود تابع main() می باشد و سایر توابع در درون این تابع فراخوانی می شوند.(به جز توابع وقفه).تابع در واقع زیر روالی (subroutine90) است که عمل خاصی را انجام می دهد و به دفعات قابل فراخوانی است. شکل کلی یک تابع به صورت زیر است :

```
return_type function_name(parameter_list)
(آرگومانهای تابع)نام تابع_نوع برگشتن تابع
{
دستورها;
return عدد یا متغیر
}
```

مثال :

```
int max(int a,int b) {
hf (a<b) return b;
else return a;
}
```

در قالب فوق return_type نوع خروجی تابع است. اگر تابع خروجی نداشته باشد نوع برگشتی آن باید viod تعریف شود. برای برگرداندن مقدار تابع از دستور return استفاده می شود(اگر نوع برگشتی void باشد احتیاجی به این دستور نیست).

Parameter_list لیست آرگومانهای تابع می باشد و همانند متغیر می باشد و با "," از هم جدا می شوند.اگر تابع پارامتر ورودی نداشته باشد در لیست پارامترها نیز از void استفاده می شودو یا هیچ چیز نوشته نمی شود.(تنها پرانتزها نوشته می شوند مانند (function()

در ابتدای بدنه تابع، می توان متغیر های محلی تعریف کرد که تنها در داخل تابع معتبر باشند و با خارج شدن از تابع از بین بروند. برای تعریف یک متغیر همگانی باید آن را در خارج از بدنه تابع تعریف کرد.توابع متغیرهای محلی بیتی نمی پذیرند. هر تابع درون خود می تواند توابعی دیگر را فراخوانی کند. اما در این کار باید ترتیب نوشتن توابع رعایت شود. بدین صورت که هر تابع می تواند توابعی را که قبل از خودش تعریف شده اند را فراخوانی کند اما به توابع بعد از خودش دسترسی ندارد. لذا main() آخرین تابعی است که نوشته می شود.(در صورتی که تابع قبل از تابع mine() معرفی شود ، میتوانید دستورات آن را بعد از mine() بیاورید)

توابع در زبان C به دوصورت هستند. یکی توابعی که توسط کاربر تعریف می شوند مانند تابع main() و دیگری توابعی که توسط کامپایلر فراهم شده اند و به توابع کتابخانه ای معروفند. برای استفاده از توابع کتابخانه ای باید نام فایل هدر آن را از طریق دستور #include به برنامه اضافه کرد.به عنوان نمونه تابع sin در فایل math تعریف شده است و مثالی از کاربرد آن به صورت زیر است :

```
#include<math.h>
viod main() {
float a=180,b;
b=sin(a);
}
```

در KIEL این امکان وجود دارد که توابع را در یک فایل دیگر بنویسیم و سپس فایل را در برنامه فراخوانی کنیم ، به فایل جدید که در برنامه فراخوانی می شود ، هدر می گویند . هدر ها بخشی از برنامه هستند که در یک مکان دیگر آورده می شوند ، مکان دیگر یک فایل متنی با پسوند .c یا .h است ، در صورتی که هدر نوشته شده همگانی بوده و بتوان از آن در برنامه های مختلف استفاده کرد ، میتوان گفت که این هدر یک کتابخانه است .

نوشتن کتابخانه تاخیر:

برای درک بهتر کاربرد توابع ، با هم یک کتابخانه می نویسیم ...

برای نوشتن هر کتابخانه باید مراحل زیر طی شود :

1- در مورد نحوه ی عمل کرد بخش مورد نظر مطالعه شود (شما باید به نحوه ی عملکرد واحدی که قصد نوشتن کتابخانه برای آن را دارید مسلط باشید) .
2- بخش جانبی با استفاده از دستورات مورد نیاز پیکربندی شود .
3- برنامه در بخش های مختلف تست شده و کلیه بخش ها در آن گنجانده شود .
4- کلیه دستورات تکراری در یک تابع قرار داده شود .
5- برای کتاب خانه یک راهنمای جامع تهیه شود تا دیگران نیز بتوانند از ان استفاده کنند .

برای نوشتن کتابخانه ابتدا لازم است بخش جانبی مورد نظر را راه اندازی کنیم (بدون اینکه به نوشتن کتابخانه فکر کنیم) کتابخانه ای که در این بخش قصد نوشتن آن را داریم ، کتابخانه ای برای ایجاد تاخیر میباشد، در صورتی که قبلا با سایر میکروکنترلر ها و کامپایلر های مربوطه کار کرده اید ، میدانید که در آنها سه دستور برای ایجاد تاخیر وجود دارد :

اولین دستور تاخیر در حد میکرو ثانیه را ایجاد کند :

```
delay_us()
```

دستور بعدی باید تاخیر در حد میلی ثانیه و دستور سوم برای ایجاد تاخیر های ثانیه ای به کار می رود :

```
delay_ms();
delay_s();
```

پس ما به سه تابع برای ایجاد تاخیر نیاز داریم :

```
void delay_us();
void delay_ms();
void delay_s();
```

البته به جای دستورات بالا می توانیم از دستورات دیگری نیز استفاده کنیم ، مثلا به جای delay_us() از wait-us استفاده کنیم ، در این صورت باید نام تابع را نیز به نام دستور تغییر دهیم :

```
void wait-us ();
```

به هر حال بهتر است نامی که برای دستورات انتخاب می کنید با کاربرد هدر همخوانی داشته باشد تا در استفاده های بعدی دچار مشکل نشوید .

همانگونه که قبلا اشاره کردیم ، برای ایجاد تاخیر باید cpu را به کار دیگری مشغول کنیم (امکان توقف CPU برای ما وجود ندارد) ، که ساده ترین کار اجرا کردن دستورات دیگری همچون شمردن عدد از صفر تا عدد x می باشد ، در این

حالت عدد مذکور در هر پالس کلاک یک واحد افزایش یافته تا به x برسد ، بعد از برابری عدد با x ، cpu شمارش را متوقف کرده و به حلقه ی اصلی برمی گردد . CPU میکروکنترلر میتواند هر دستور را در یک سیکل اجرا کند .

برای شروع کار باید ببینیم شمارش هر واحد عدد چقدر زمان میبرد ،قبل از خواندن ادامه ، مطالب بخش " راه اندازی واحد های PMC و CLKG " را یکبار دیگر مرور کنید .

مثال برنامه ای بنویسید که پایه b.19 میکرو کنترلر AT91SAM7X256 را به مدت 0.5 ثانیه 0 کند ، بعد به مدت 1 ثانیه 1 کند و این حلقه مدام تکرار شود :

برای نوشتن این برنامه نیاز است تا cpu را در دو مکان مشغول نگه داریم :

```
#include <AT91SAM7X256.H>
int main (void} (
while(1){
  *AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
    *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
دستوراتی که n بار اجرا میشوند
   *AT91C_PIOB_ODSR = 0x00000000;          // PB.19 to be cleared
دستوراتی که m بار اجرا میشوند
}
}
```

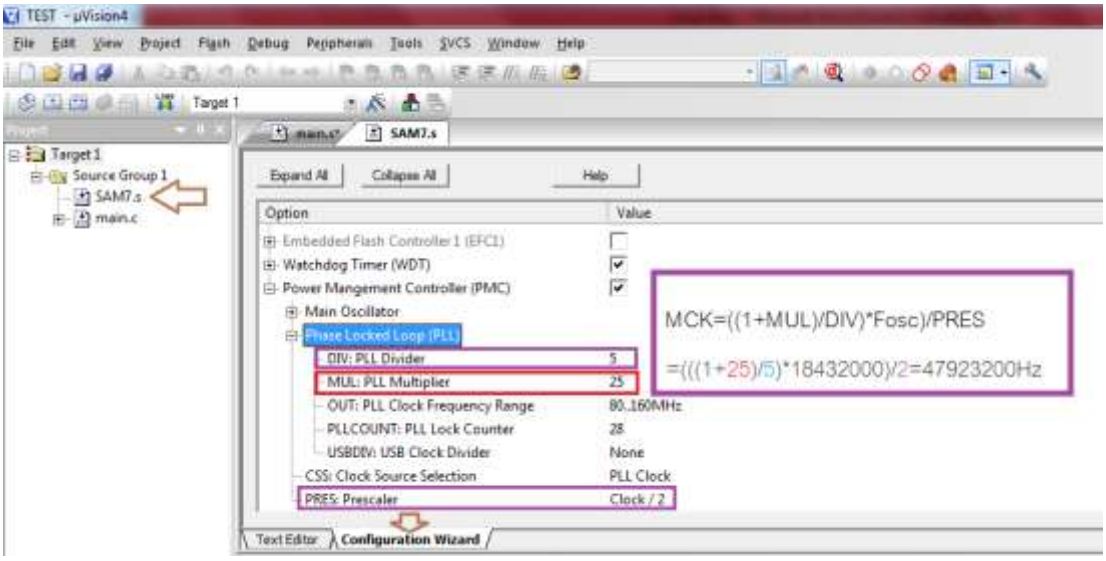
بعد از ایجاد کردن پروژه و کپی کردن کد های بالا در فایل MAIN.C به یکی از روش های زیر مقدار فرکانس کاری CPU را بدست آورید : (فعلا برنامه را کامپایلر نکنید)

روش اول :

فایل SAM7.S را باز کنید و از طریق فرمول زیر مقدار فرکانس کاری CPU را محاسبه کنید :

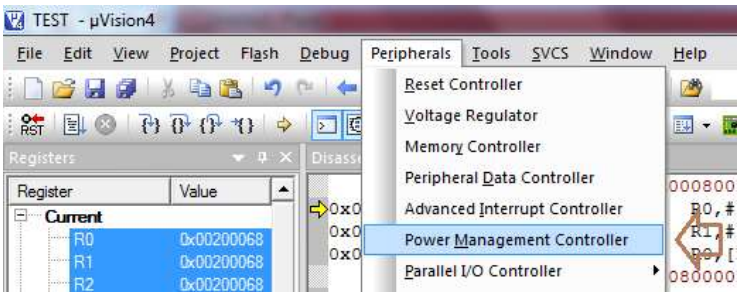
MCK=((1+MUL)/DIV)*Fosc)/PRES

در این رابطه Fosc فرکانس کریستال خارجی متصل شده به میکروکنترلر میباشد (من از کریستال 18.432 مگاهرتز استفاده میکنم) .

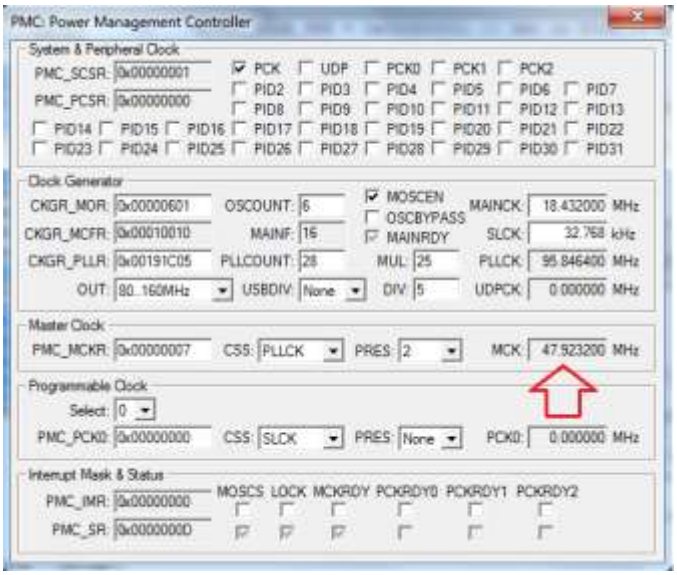


روش دوم :

در محیط کامپایلر از منوی debug گزینه ی start / stop debog session را انتخاب کنید و سپس در محیط شبیه سازی از منوی Peripherals گزینه ی Power Management Controller را انتخاب کنید .



در این پنجره فرکانس کلاک اصلی سیستم (MCK) نمایش داده شده است :



در صورتی که MCK را به 48 مگاهرتز روند کنیم ، اجرا کردن هر دستور تقریبا 20.8 نانو ثانیه زمان میبرد ، در این حالت برای ایجاد تاخیر 500 میلی ثانیه باید 24 میلیون دستور را اجرا کنیم .

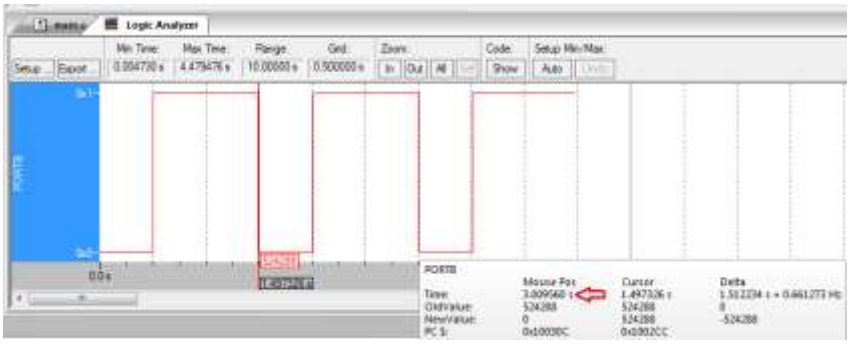
برای اجرا کردن 24 میلیون دستور ، روش های مختلفی وجود دارد که یکی از آنها استفاده از حلقه های شرطی است ، برای شروع من حلقه ی for را انتخاب میکنم ؛ در صورتی که حلقه FOR برای شمارش یک متغیر 32 بیتی شامل 20 دستور باشد و مجموع این دستورات در 25 سیکل اجرا شوند (برای فهمیدن تعداد سیکل اجرای هر یک از دستورات زبان c باید کد اسمبلی متناظر با آنها را مشاهده کنید و تعداد دستورات مربوط به آن را شمارش کنید ، برخی از دستورات اسمبلی در دو سیکل اجرا میشوند) افزایش هر واحد به عدد شمرده شده 25 سیکل زمان میبرد ، در برنامه بالا به جای جمله " دستوراتی که n بار اجرا میشوند " دستور for (n = 0; n < x; n++); را قرار می دهم ، در این حلقه x برابر با 24000000/25 است .

```
for (n = 0; n < 960000; n++)
```

برای زمان یک ثانیه نیز cpu باید 48 میلیون دستور را اجرا کند ، در این حالت عددی که در حلقه for قرار میگیرد برابر با 1920000 خواهد بود . در زیر برنامه نهایی را مشاهده میکنید :

```
#include <AT91SAM7X256.H>
int n;
int main (void) {
  *AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
while(1){
   for (n = 0; n < 960000; n++);
     *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
   for (n = 0; n < 1920000; n++);
    *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared
}
```

برنامه بالا را شبیه سازی کنید و شکل موج ایجاد شده بر روی پایه B.19 را توسط بخش logic analyzer مشاهده کنید :



{در تابع بالا مقدار اولیه حلقه برابر صفر می باشد ، در حلقه ی for یک واحد به متغیر d افزوده می شود و هنگامی که مقدار آن به 4 رسید cpu حلقه را ترک می کند .

اکنون برنامه بالا برای زمان های 1 میکروثانیه و یک میلی ثانیه توسعه میدهیم ، در این حالت برنامه رو طوری مینویسیم که پایه B.19 علاوه بر وضعیت های 500 میلی ثانیه خاموش ، 1 ثانیه روشن ، مجددا 1 میلی ثانیه خاموش و یک 1 میکرو ثانیه روشن شود :

```
#include <AT91SAM7X256.H>

int n;

int main (void) {

*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode

*AT91C_PIOB_OER = 0x00080000; // Configure in Output

*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode

while(1){

  for (n = 0; n < 960000; n++);          //DELAY FOR 500mS

    *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set

  for (n = 0; n < 1920000; n++);          //DELAY FOR 1S

   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared

              for (n = 0; n < 1920; n++);             //DELAY FOR 1mS

    *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set

  for (n = 0; n < 2; n++);                                  //DELAY FOR 1uS

   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared

}

}
```

توابع مورد نیاز را در برنامه ایجاد میکنیم :

```
#include <AT91SAM7X256.H>
int n;
void delay_us(){
  for (n = 0; n < 2; n++);}                  //DELAY FOR 1sS
void delay_ms(){
  for (n = 0; n < 1920; n++);}                        //DELAY FOR 1mS
void delay_s(){
  for (n = 0; n < 1920000; n++);}         //DELAY FOR 1S}
void delay(){
  for (n = 0; n < 960000; n++);}    //DELAY FOR 500mS
int main (void) {
*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
while(1){
            delay();
   *AT91C_PIOB_ODSR = 0x00080000           ; // PB.19 to be set
            delay_s();
   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared
            delay_ms();
   *AT91C_PIOB_ODSR = 0x00080000           ; // PB.19 to be set
            delay_us();
   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared
}
}
```

در برنامه بالا توابع مورد نیاز جهت ایجاد تاخیری فاقد مقدار برگشتی هستند ، به همین دلیل از نوع void معرفی شده اند.

در صورتی که بخواهیم مقدار زمان را مقدار دهی کنیم ، کافی است برای هر یک از توابع یک آرگومان یا متغیر معرفی کنیم و رقم تاخیر مورد نیاز را به تابع ارسال کنیم :

```
void delay_s(int time;){
```

```
  for (n = 0; n < 1920000; n++);}        //DELAY FOR 1S}
```

اکنون تابع ایجاد تاخیر ثانیه ای میتواند عددی را در بازه متغیر int دریافت کند ، برای تاخیر n ثانیه حلقه for (n = 0; n < 1920000; n++); باید n بار اجرا شود ، پس :

```
void delay_s(int time)
{ //DELAY FOR 1S
        int temp;
        for (temp= 0; temp < time; temp ++)
            for (n = 0; n < 1920000; n++);
}
```

متغیر time آرگومان تابع delay_s می باشد و یک متغیر است که مقدارش را در برنامه اصلی دریافت می کند . در تابع delay_s مقدار این متغیر با متغیر temp مقایسه میشود و به تعداد آن حلقه for (n = 0; n < 1920000; n++); اجرا میشود . توجه داشته باشبد که وجود دستور ";" در انتهای حلقه FOR باعث اجرا شدن خود حلقه (به تنهایی) میشود . در صورتی که این دستور در انتهای حلقه وجود نداشته باشد در هر بار افزایش متغیر tempدستور بعدی حلقه نیز اجرا میشود و اگر دستورات بعد از حلقه بیشتر از یک دستور باشد باید آنها را در بین "{ }" قرار دهیم .

```
int n;
void delay_us(int time)
{ //DELAY FOR 1sS
  int temp;
  for (temp= 0; temp < time; temp ++)
                for (n = 0; n < 2; n++);
}
void delay_ms(int time)
{ //DELAY FOR 1mS
  int temp;
```

```
  for (temp= 0; temp < time; temp ++)
                        for (n = 0; n < 1920; n++);
}
void delay_s(int time)
{ //DELAY FOR 1S
        int temp;
        for (temp= 0; temp < time; temp ++)
            for (n = 0; n < 1920000; n++);
}
void delay()
{  //DELAY FOR 500mS
  for (n = 0; n < 960000; n++);
}
```

توابع در زبان C میتوانند از متغیر های محلی یا متغیر های سراسری استفاده کنند ، در برنامه بالا هر یک از توابع دارای متغیر های محلی time و temp مخصوص به خود هستند و از متغیر عمومی n استفاده میکنند . بهتر است متغیر های مورد نیاز تمامی توابع را به صورت محلی معرفی کنید تا در آینده مشکلاتی نظیر تداخل متغیر ها ایجاد نشود . در زیر ورژن نهایی برنامه را مشاهده میکنید : در این برنامه چهار توابع برای ایجاد تاخیر از یک تا 2^32 ثانیه / میلی ثانیه / میکرو ثانیه و 500 میلی ثانیه وجود دارد :

```
#include <AT91SAM7X256.H>
void delay_us(int time)
{ //DELAY FOR 1sS
  int temp,n;
  for (temp= 0; temp < time; temp ++)
                        for (n = 0; n < 2; n++);
}
void delay_ms(int time)
{ //DELAY FOR 1mS
  int temp,n;
  for (temp= 0; temp < time; temp ++)
```

```
                    for (n = 0; n < 1920; n++);
}
void delay_s(int time)
{ //DELAY FOR 1S
        int temp,n;
        for (temp= 0; temp < time; temp ++)
            for (n = 0; n < 1920000; n++);
}
void delay()
{  //DELAY FOR 500mS
int n;
  for (n = 0; n < 960000; n++);
}
int main (void) {
*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
while(1){
        delay();
   *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
        delay_s(1);
   *AT91C_PIOB_ODSR = 0x00000000;          // PB.19 to be cleared
        delay_ms(1000);
   *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
        delay_us(1000000);
   *AT91C_PIOB_ODSR = 0x00000000;          // PB.19 to be cleared
}
}
```

در صورتی که برنامه بالا را شبیه سازی کنید ، مشاهده میکنید که کلیه تاخیر ها در این ورژن از تابع دارای خطای غیر قابل قبولی هستند!

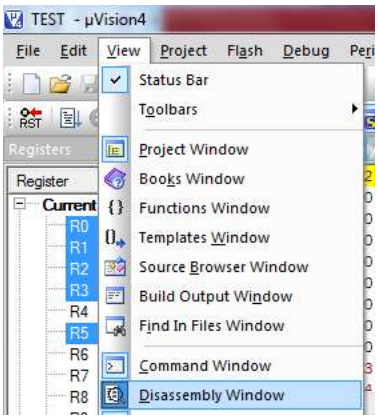
برنامه را حالت زیر تغییر دهید و مجدا نتایج شبیه سازی را ببینید :

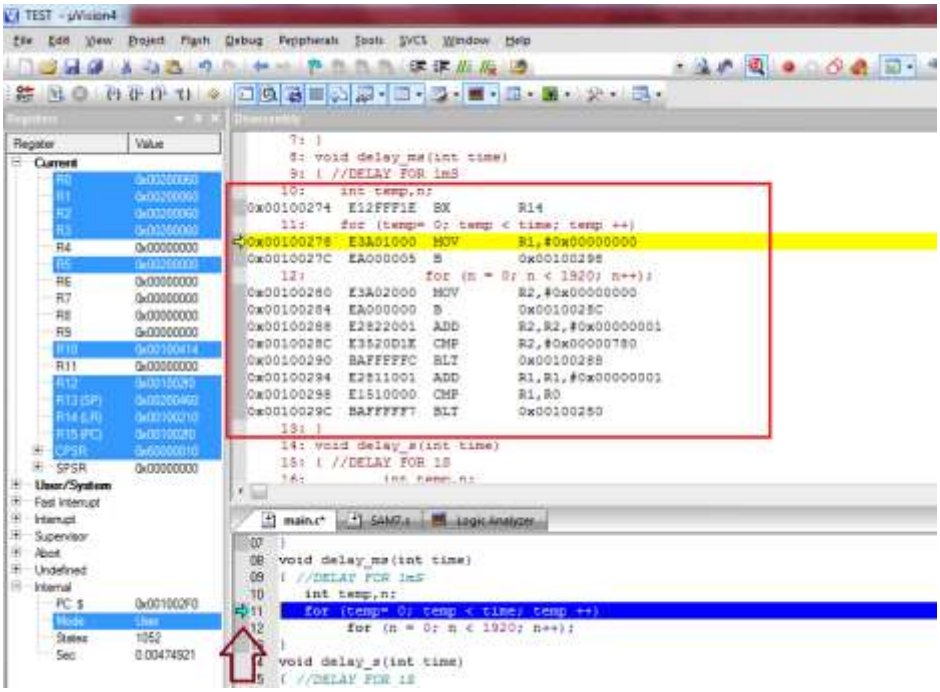
```
#include <AT91SAM7X256.H>
int temp,n;
int main (void) {
*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
while(1){
            for (n = 0; n < 960000; n++); //DELAY FOR 500mS
   *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
            for (temp= 0; temp < 1; temp ++)//DELAY FOR 1S
                  for (n = 0; n < 1920000; n++);
   *AT91C_PIOB_ODSR = 0x00000000;          // PB.19 to be cleared
            for (temp= 0; temp < 1000; temp ++)//DELAY FOR 1000mS
                        for (n = 0; n < 1920; n++);
   *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
            for (temp= 0; temp < 1000000; temp ++)      //DELAY FOR 1000000uS
                        for (n = 0; n < 2; n++);
   *AT91C_PIOB_ODSR = 0x00000000;          // PB.19 to be cleared
}
}
```

در برنامه بالا فقط تاخیر های میلی و میکرو ثانیه دارای خطا هستند ، دلیل وجود این خطا نیز بسیار ساده است ، به بخش ایجاد delay_us توجه کنید ، در این تابع برای ایجاد تاخیر یک میکروثانیه 40 دستور در حلقه for (n = 0; n < 2; n++); اجرا میشود ، بعد از سپری شدن هر میکرو ثانیه 20 دستور در حلقه for (temp= 0; temp < time; temp ++) جهت افزایش متغیر تاخیر اجرا میشود که این 20 دستور تقریبا 1 میکرو ثانیه زمان میبرد ، با این تفاسیر تاخیر 1000000 میکروثانیه در 1.56 ثانیه میشود . از طرفی عدد 2 که در این حلقه وجود دارد روند شده عدد 1.92 که در بخش های قبلی محاسبه شد میباشد در صورتی خطای 0.08 را نیز به خطای 0.5 ثانیه اضافه کنیم ، تاخیر ایجاد شده به جای 1000000 تقریبا میکروثانیه 1.8 ثانیه خواهد شد . پس بهتر است از هر تابع برای ایجاد تاخیر در محدوده خاص خودش استفاده شود تا میزان خطا قابل قبول باشد .

دلیل تفاوت تاخیر 1 ثانیه و 500 میلی ثانیه در دو برنامه بالا چیست ؟

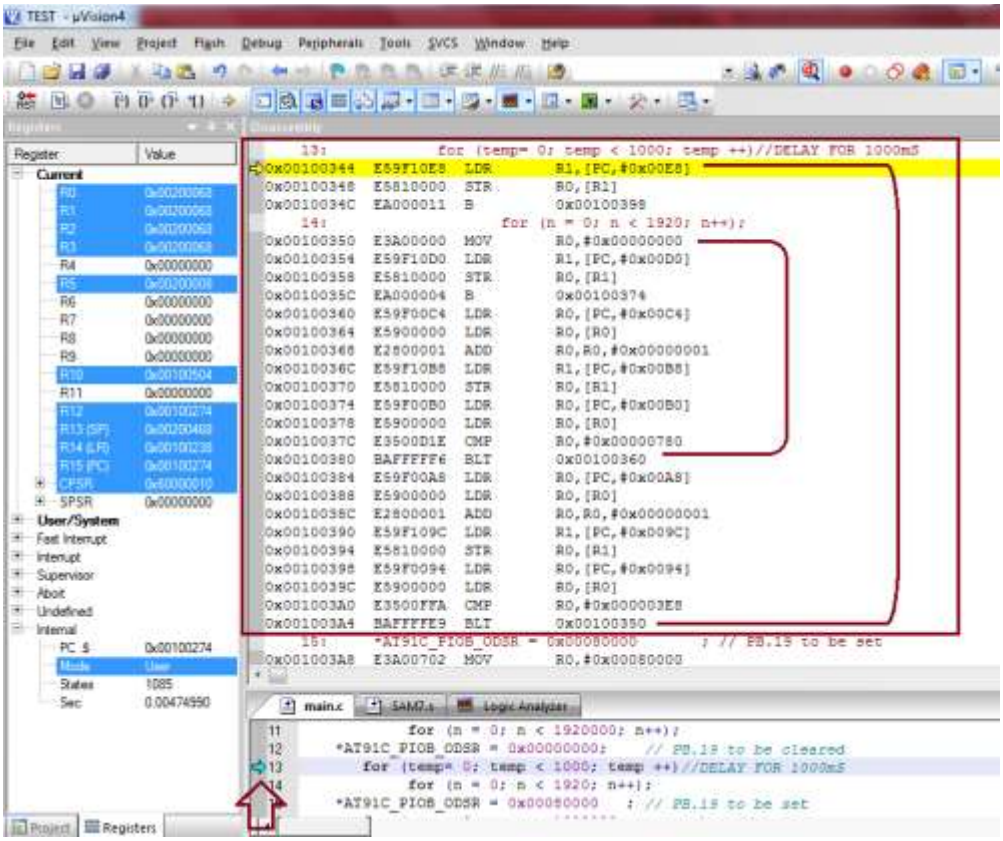
برنامه اول را در کامپایل کرده و سپس وارد محیط شبیه سازی شوید و از منوی view گزینه Disassembly window را انتخاب کنید تا پنجره Disassembly باز شود :





جهت مشاهده سورس اسمبلی تابع delay_ms بر روی قسمت خاکستری رنگ که با فلش قهوای مشخص شده است کلیک کنید (خط شماره 11 برنامه main.c) ، همانطور که در سورس اسمبلی میبینید ،حلقه for (n = 0; n < 1920; n++) با 6 دستور اسمبلی (در 8 سیکل) و حلقه for (temp= 0; temp < time; temp ++) با 6 دستور اسمبلی اجرا میشود . (1میلی ثانیه = 1920*6 دستور)

اکنون برنامه دوم را کامپایل کنید و سورس اسمبلی آن را مشاهده کنید :



همانطور که مشاهده میکنید در این برنامه حلقه for (n = 0; n < 1920; n++) با 18 دستور اسمبلی اجرا میشود !

از بررسی این دو مثال میتوان نتیجه گرفت که کد های قرار گرفته در توابع در کامپایلر keil به صورت بهینه تر اجرا میشوند (البته ممکن است در کامپایلر های دیگر این موضوع برعکس باشد) .

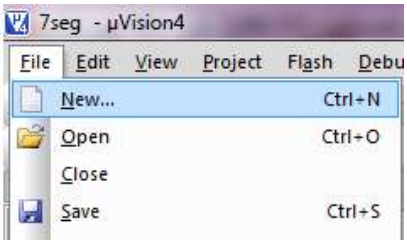
به این بحث همین جا خاتمه داده و متغیر ها را برای توابع بهینه سازی میکنم :

```
#include <AT91SAM7X256.H>

void delay_us(int time)

{ //DELAY FOR 1uS

  int temp,n;

  for (temp= 0; temp < time; temp ++)

                  for (n = 0; n <= 6; n++);

}

void delay_ms(int time)

{ //DELAY FOR 1mS
```

```
  int temp,n;
  for (temp= 0; temp < time; temp ++)
                        for (n = 0; n <= 6000; n++);
}
void delay_s(int time)
{ //DELAY FOR 1S
            int temp,n;
            for (temp= 0; temp < time; temp ++)
                for (n = 0; n <= 6000000; n++);
}
void delay()
{  //DELAY FOR 500mS
int n;
  for (n = 0; n < 3000000; n++);
}
int main (void) {
*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode
*AT91C_PIOB_OER = 0x00080000; // Configure in Output
*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode
while(1){
            delay();
   *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
            delay_s(1);          //-1.068377
   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared
            delay_ms(1);
   *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set
            delay_us(1);
   *AT91C_PIOB_ODSR = 0x00000000;           // PB.19 to be cleared
}
}
```

در حال حاضر توابع بالا دارای بهینه ترین حالت برای ایجاد تاخیر +1 میلی ثانیه / 1 میکروثانیه ، یک ثانیه و 500 میلی ثانیه هستند . برای تاخیر 1 میکرو ثانیه تابع delay_us دارای خطای 380 نانو ثانیه است (1 میکروثانیه ، 1.38 میکروثانیه طول میکشد) ، با افزایش زمان این خطا در تعداد اعداد ضرب میشود ، بگونه ای که تاخیر 1000 میکروثانیه دارای خطای 40 میکرو ثانیه است(1000 میکرو ثانیه ، 960 میکرو ثانیه طول می کشد) . (دلیل مثبت و منفی بودن خطا چیست ؟)

اکنون میتوانیم توابع ایجاد شده را در فایل دیگری ذخیره کرده و در تمامی برنامه های که در آینده خواهیم نوشت استفاده کنیم ، برای اولین مرحله ایجاد فایل هدر ، از منوی file گزینه ی new را انتخاب کنید :



پنجره جدیدی در محیط keil باز می شود . دوباره از همان منو گزینه ی save as را انتخاب کنید و در پنجره باز شده فایل موجود را با نام Delay.h ذخیره کنید : (فایل را در مکان دلخواه ذخیره کنید .)

در اولین خط ، کد های زیر را کپی کنید :

```
#ifndef _DELAY_INCLUDED_
#define _DELAY_INCLUDED_
```

دستور ات بالا باعث می شود که کامپایلر هدر را فقط یک بار لود کند . با این کار در صورتی که هدر در فایل های دیگری نیز فراخوانی شده باشد ، دیگر کامپایل نخواهد شد و حجم متغیر ها و کد نهایی برنامه بی دلیل زیاد نمی شود .

تابع تاخیر را بعد از دستور بالا قرار داده و در نهایت با دستور زیر برنامه هدر را به پایان برسانید :

```
#endif
```

محتوای فایل Delay.h:

```
#ifndef __DELAY_H__
```

```
#define __DELAY_H__
void delay_us(int time)
{ //DELAY FOR 1uS
  int temp,n;
  for (temp= 0; temp < time; temp ++)
                    for (n = 0; n <= 6; n++);
}
void delay_ms(int time)
{ //DELAY FOR 1mS
  int temp,n;
  for (temp= 0; temp < time; temp ++)
                    for (n = 0; n <= 6000; n++);
}
void delay_s(int time)
{ //DELAY FOR 1S
        int temp,n;
        for (temp= 0; temp < time; temp ++)
            for (n = 0; n <= 6000000; n++);
}
void delay()
{  //DELAY FOR 500mS
int n;
  for (n = 0; n < 3000000; n++);
}
 #endif
```

با فرخوانی هدر بالا سه دستور برای ایجاد تاخیر های زمانی میکروثانیه ، میلی ثانیه و ثانیه و 500 میلی ثانیه به نرم افزار keil اضافه میشود :

```
#include <AT91SAM7X256.H>
#include "pio.h"
#include "delay.h"
```

```
int main (void) {
PORTB_OUTPUT= PB19;
 while(1){
SET_PORTB= PB19;
    delay_s(1);
RST_PORTB= PB19;
    delay_s(1);
          }}
```

کتابخانه ی DELAY.h

برای استفاده از این کتاب خانه باید فایل DELAY.h را در مسیر زیر کپی کنید (بهتر است از کتابخانه ی کامل که در پوشه پیوست کتاب موجود می باشد، استفاده کنید) :

```
Program Files\Keil\ARM\INC\Atmel.
```

(در پوشه های SAM7Xو SAM7A3 و AT91SAM9G20 و سایر پوشه های موجود).

برای استفاده از این کتابخانه ، ابتدا باید آن را در برنامه فراخوانی کنید ، برای اینکار از دستور زیر استفاده می شود :

```
#include <delay.h>
```

با فرا خوانی این کتابخانه ، چهار دستور به نرم افزار KEIL اضافه می شود که شرح آنها در زیر آورده شده است :

```
delay_us(x);
```

دستور بالا تاخیر X میکرو ثانیه را بوجود می آورد ، X می تواند از 1 تا 4294967295 باشد .

```
delay_ms(x);
```

دستور بالا تاخیر X میلی ثانیه را بوجود می آورد ، X می تواند از 1 تا 4294967295 باشد .

```
delay_s(x);
```

دستور بالا تاخیر X ثانیه را بوجود می آورد ، X می تواند از 1 تا 4294967295 باشد

```
Delay();
```

دستور بالا با هر بار اجرا شدن تاخیر 500 میلی ثانیه را در برنامه ایجاد میکند .

مثال : برنامه ای بنویسید که پایه B.19 میکرو کنترلر AT91SAM7X256 را به مدت 2 ثانیه صفر کند و به مدت 1.5 ثانیه 1 ، حلقه مجددا تکرار شود .

```
#include <AT91SAM7X256.H>
#include "pio.h"
# include <delay.h>
int main (void){
while (1)
{
PORTB_OUTPUT=0x80000 ;
SET_PORTB=0x80000 ;
delay_s(2);
RST_PORTB=PIO_PB19 ;
delay_ms(1500);
}}
```

توسعه کتابخانه تاخیر :

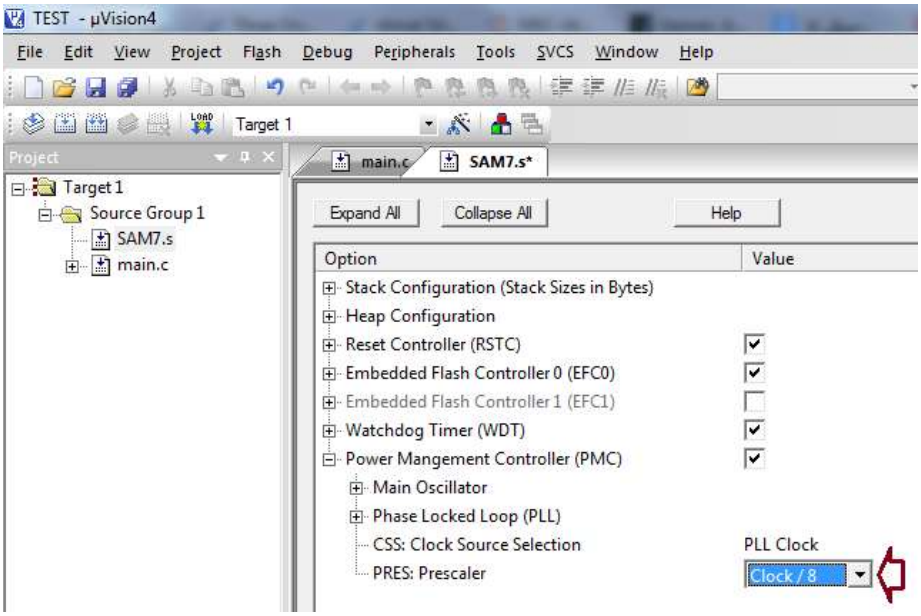
کتابخانه ای که نوشتیم دارای 2 مشکل به شرح زیر است :

1- با تغییر میزان کریستال یا ضرائب PLL ، کلیه زمان ها به هم میریزد .

2- تاخیر ایجاد شده در آن خطی نیست ، مثلا 1 میلیون میکرو ثانیه ، تاخیر 960 میلی ثانیه را به جای تاخیر یک ثانیه ایجاد میکند .

البته گزینه دوم زیاد مهم نیست ، چون کاربر از وجود خطا در این کتابخانه مطلع است و باید برای ایجاد تاخیر های دقیق از تایمرها استفاده کند .

برای مشاهده عملی مشکل اول فایل SAM7.S را باز کنید و در بخش PRES:Prescaler را به clock/8 تغییر دهید تا فرکانس MCK به 12 مگاهرتز کاهش یابد :



اکنون برنامه زیر شبیه سازی کنید :

```
#include <AT91SAM7X256.H>

#include "delay.h"

int main (void) {

*AT91C_PIOB_PER = 0x00080000; // Set in PIOB.19 mode

*AT91C_PIOB_OER = 0x00080000; // Configure in Output

*AT91C_PIOB_OWER = 0x00080000; // Configure in PIOX_ODSR  in Read-write mode

while(1){

   *AT91C_PIOB_ODSR = 0x00080000          ; // PB.19 to be set

            delay_s(1);

   *AT91C_PIOB_ODSR = 0x00000000;          // PB.19 to be cleared

            delay_s(1);

}}
```

همانطورکه مشاهده میکنید تاخیر 1 ثانیه در 4 ثانیه روی میدهد ، چون کلیه محاسبات مروبط به توابع تاخیر برای فرکانس 48 مگاهرتز انجام شده است .

برای رفع این مشکل دو روش زیر به ذهن من میرسد :

1- ایجاد یک متغیر و وادار کردن کاربر به وارد کردن فرکانس کاری میکروکنترلر (MCK) در برنامه :

در هدر delay.h بعد از دستورات مروبط به معرفی هدر ، دستورات زیر را وارد کنید و برنامه را کامپایل کنید :

```
#ifndef MCK
  #error MCK value not define, define it by '#define MCK x'(x is MCK value in hertz)statement  Before #include "delay.h"
#endif
```

توسط دستور پیش پردازشی بالا وجود متغیر mck در برنامه بررسی می شود و در صورت عدم وجود آن خطای به کاربر نمایش داده میشود ، در واقع کاربر باید قبل از معرفی کتابخانه در برنامه اصلی توسط دستور زیر مقدار فرکانس کلاک پردازنده را مشخص کند :

```
#define MCK 12000000
```

اکنون باید با استفاده از MCK مقدار عددی که باید در توابع شمرده شود را محاسبه کنیم :

```
#ifndef __DELAY_H__
#define __DELAY_H__
#ifndef MCK
  #error MCK value not define, define it by '#define MCK x'(x is MCK value in hertz)statement  Before #include "delay.h"
#endif
void delay_us(int time){ //DELAY FOR 1uS
  int temp,n;
  for (temp= 0; temp < time; temp ++)
                        for (n = 0; n <= (MCK/8000000); n++);}
void delay_ms(int time){ //DELAY FOR 1mS
  int temp,n;
  for (temp= 0; temp < time; temp ++)
                        for (n = 0; n <= (MCK/8000); n++);}
void delay_s(int time){ //DELAY FOR 1S
            int temp,n;
            for (temp= 0; temp < time; temp ++)
                for (n = 0; n <= (MCK/8); n++);}
void delay(){ //DELAY FOR 500mS
int n;
  for (n = 0; n < 3000000; n++);          }
#endif
```

در روش بالا ممکن است برای برخی از فرکانس ها ، تاخیر ها دارای خطا باشد ، مثلا برای فرکانس 12 مگاهرتز ، تابع delay_us باید در هر بار شمارش تا عدد 1.5 برای ایجاد تاخیر 1 میکرو ثانیه شمارش کند و امکان شمارش تا این عدد برای cpu وجود ندارد و اجبارا بخش اعشاری حذف میشود (تاخیر میکرو ثانیه تا 50 درصد دارای خطا خواهد بود) .

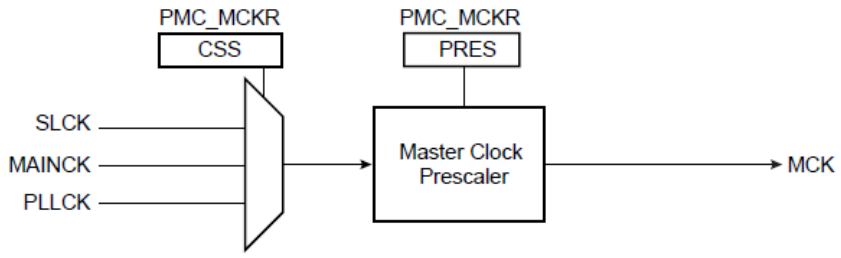
2- ما میتوانیم با خواندن مقادیر رجیستر های CKGR_MOR و CKGR_MCFR و CKGR_PLLR و PMC_MCKR فرکانس MCK را محاسبه کنیم ، استفاده از این روش کاربردی تر بوده و امکان ایجاد خطا در تاخیر های ایجاد شده ، در شرایطی همچون تغییر منبع تامین کلاک اصلی پردازنده ، تغییر ضرائب PLL و ... را از بین میبرد .

در اولین مرحله باید تابعی را جهت ایجاد عملیات های مورد نیاز در فایل delay.h ایجاد کنیم :

```
unsigned int get_MCK_value_4_delay_lib (void)
{


Return MCK
}
```

در این تابع فرکانس MCK محاسبه شده و به برنامه اصلی ارسال میشود . برای محاسبه MCK مطابق مطالب ذکر شده در بخش قبلی باید منبع تامین کننده کلاک اصلی پردازنده را مشخص کرد ، این کار با خواندن بیت های CSS رجیستر PMC_MCKR انجام میشود :

CSS		Clock Source Selection
0	0	Slow Clock is selected
0	1	Main Clock is selected
1	0	Reserved
1	1	PLL Clock is selected.



در صورتی که منبع تامین کننده کلاک slck باشد ، کافی است عدد 32768(فرکانس slck برحسب هرتز) موجود در فیلد PRES در رجیستر PMC_MCKR تقسیم شود تا MCK بدست آید .

در صورتی که منبع تامین کلاک MAINCK باشد کافی است فرکانس کریستال خارجی متصل شده به میکروکنترلر به موجود در فیلد PRES در رجیستر PMC_MCKR تقسیم شود تا MCK بدست آید .

در صورتی که منبع کلاک PLLCK باشد باید از طریق فرمول زیر MCK را در برنامه محاسبه کنیم :

MCK=((1+MUL)/DIV)*Fosc)/PRES

MUL و DIV در رجیستر CKGR_PLLR قرار دارند .

```
unsigned int get_MCK_value_4_delay_lib (void)
{
float mck2;
        switch ( *AT91C_PMC_MCKR & AT91C_PMC_CSS)
        {
                case 0: // Slow Clock is selected
                        mck2=32768/((*AT91C_PMC_MCKR & AT91C_PMC_PRES) >> 2);
                break;
                case 1: // Main Clock is selected
                        mck2=crystal / ((*AT91C_PMC_MCKR & AT91C_PMC_PRES) >> 2);
                break;
                case 3:{ // PLLB clock is selected
                        mck2=(((*AT91C_CKGR_PLLR  & AT91C_CKGR_MUL) >> 16)+1) ;
                        mck2=mck2/(AT91C_BASE_CKGR->CKGR_PLLR  & AT91C_CKGR_DIV);
                        mck2=((mck2*crystal) /(1<<((*AT91C_PMC_MCKR & AT91C_PMC_PRES) >> 2)));
                  cr1y= (1<<((*AT91C_PMC_MCKR & AT91C_PMC_PRES) >> 2));
                break;}    //prescaler
                default :
                        mck2=0;
                break;
        }
//      cr1y=mck2;
```

```
  return mck2;
}
```

در این تابع مقدار کلاک MCK محاسبه شده و با فراخوانی تابع ، توسط دستور return mck2; برگردانده میشود .

فایل delay.h در پوشه ی پیوست باز کنید و نحوه ی استفاده از تابع بالا در توابع تاخیر را مشاهده کنید .

ضمیمه شماره یک : دیتاشیت فارسی میکرو کنترلر های AT91SAM7X512 و AT91SAM7X256 و AT91SAM7X128

AT91SAM7X512 و AT91SAM7X256 و AT91SAM7X128 دقیقا مشابه هم بوده و تنها تفاوت بین آنها در مقادیر حافظه میباشد :

Device	Flash	Flash Organization	SRAM
AT91SAM7X512	512 Kbytes	dual plane	128 Kbytes
AT91SAM7X256	256 Kbytes	single plane	64 Kbytes
AT91SAM7X128	128 Kbytes	single plane	32 Kbytes

ویژگی ها :

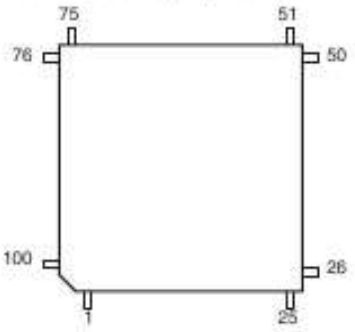
- کارایی بالا در معماری RISC 32 بیتی
- دستورات 32 بیتی با کارایی بالا
- ICE داخلی (ICE یا In-circuit Emulation به کاربر امکان دیباگ کردن برنامه ، بر روی میکرو را میدهد)
- 512 کیلو بایت حافظه سازمان یافته برای AT91SAM7X512
- 256 کیلو بایت حافظه سازمان یافته برای AT91SAM7X256
- 128 کیلو بایت حافظه سازمان یافته برایAT91SAM7X128
- حافظه فلش این پردازنده ها که مقدار آن در بالا بیان شد ، در دوبانک برای AT91SAM7X512 و یک بانک برای دو میکرو دیگر طراحی شده اند ، هر بانک دارای قطعات 256 بایتی میباشد ، این یعنی هر دستور فقط در یک خانه ذخیره میشود و این کار سرعت خواندن حافظه را به مقدار زیادی افزایش میدهد .
- پیشرو در سرعت و اجرای دستورات
- 128کیلو بایت حافظه sram برای AT91SAM7X512
- 64 کیلو بایت حافظه sram برای AT91SAM7X256
- 32 کیلو بایت حافظه sram برایAT91SAM7X128
- سرعت حافظه sram بسیار بالا میباشد ، به طوری که خواندن هر دستور فقط یک سیکل طول میکشد .
- قابلیت بازرسی حافظه (این واحد از نارسایی های که باعث اختلال در عملکرد میشود (معمولا تاخیر در نوشتن و...) ، چشم پوشی میکند)

- واحد کنترل کننده ریست (این واحد در شرایطی که تغذیه اعمال شده مناسب نباشد یا فرمان بازنشانی به پایه ریسیت اعمال شود ، میکرو را با سرعت بالا ریست میکند)
- درای نوسان ساز rc داخلی با فرکانس 0 تا 20 مگاهرتز به همراه یک عدد pll
- چندین مد توان (بهینه سازی توان ، کار در فرکانس پایین و مد Idle)
- 5 منبع سیگنال پالس خارجی قابل برنامه ریزی
- دو منبع وقفه خارجی (یکی از منابع ، وقفه سریع میباشد و دیگری وقفه عادی)
- دو کانال UART با قابلیت پشتیبانی از ارسال داده ی همزمان و غیر همزمان
- دارای کانتر 20 بیتی قابل برنامه ریزی
- تایمر Watchdog
- درای تایمر بلادرنگ ، این تایمرمیتواند به عنوان یک کانتر 32 بیتی به همراه آلارم (متصل به پین مجزا) راه اندازی شود.
- دو پورت ورودی و خروجی موازی
- هر پورت دارای 31 پایه مجزا میباشد .
- امکان استفاده ی هر کدام از پایه های i/o به عنوان ورودی وقفه
- تمامی پایه قابلیت برنامه ریزی به عنوان ورودی و با حالت درین باز و مقاومت بالا کشنده را دارند .
- شما میتوانید همزمان یک پایه را به عنوان ورودی و خروجی تعریف کنید .
- قابلیت کنترلر 13 وسیله جانبی به صورت مجزا (در مد spi).
- درای یک عدد پورت USB 2.0 کامل (با سرعت 12 Mbits بر ثانیه)
- فرستنده گیرنده داخلی قابل برنامه ریزی برای تبادل داده با USB
- دارای یک اترنت MAC 10/100 base-T
- امکان رابطه مستقل یا معمولی (MII یا RMII)
- امکان ورود داده و خروج ان به ترتیب ورود (FIFOs) ، دسترسی مستقیم به حافظه (DMA) و ارسال و دریافت داده با سرعت بالا
- درای رابط CAN
- 8پیغام قابل برنامه ریزی برای صندوق ایمیل با شمارنده 16 بیتی

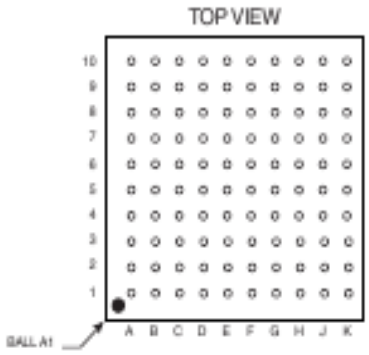
- درای یک کنترل کننده سریال هم زمان .
- درای کلاک مستقل و و هم زمان ساز برای هر سیگنال دریافتی و ارسالی
- پشتیبانی از رابط I^2S
- دارای تقسیم گر زمان چند گانه
- سرعت بالا در انتقال داده های 32 بیتی
- پشتیبانی USART از ISO7816 T0/T1 Smart Card و IrDA ® Infrared Modulation/Demodulation و Full Modem Line
- درای دو کانال مجزا برای رابط SPI
- یک تایمر / کانتر 16 بیتی 3 کاناله
- سه کلاک ورودی خارجی
- تولید PWM دوبل و دارای مد مقایسه ای و...
- یک واحد کنترل PWM 16 بیتی 5 کاناله
- پشتیبانی از پروتکل Two-wire
- پشتیبانی از EEPROMs سریال
- 8 کانال ADC 10 بیتی ، 5 کانال Multiplexed
- پشتیبانی از SAM-BA (SAM-BA به کاربر این امکان را میدهد تا بدون نیاز به سخت افزار پروگرامر میکرو را برنامه ریزی کند)
- پشتیبانی از واسط JTAG
- ولتاژ خروجی هر پورت برابر ولتاژ تغذیه میباشد و هر پین میتواند جریانی تا 16 میلی آمپر بدهد .
- عملکرد کاملا ثابت ، کارد کرد در فرکانس 0 تا 55 مگا هرتز ، و دمای –40 تا +85 درجه سانتی گراد .
- درای 100 پایه و تولید در دو بسته بندی LQFP و TFBGA

وضعیت پایه ها :

100-lead LQFP Package Outline (Top View)



100-ball TFBGA Package Outline (Top View)



شماره	نام پایه	عملکرد پایه	نوع پایه
1	ADVREF	ورودی ولتاژ مرجع ADC	Power
2	GND	گراند	GND
3	AD4	ورودی ADC(4)	Analog
4	AD5	ورودی ADC(5)	Analog
5	AD6	ورودی ADC(6)	Analog
6	AD7	ورودی ADC(7)	Analog
7	VDDOUT	ولتاژ خروجی رگولاتور (1.8 ولت)	Power
8	VDDIN	ولتاژ تغذیه	Power
9	PB27/AD0/TIOA2	پورت B.27 / ورودی ADC (0) /i.o برای تایمر	I/O
10	PB28/AD1/TIOB2	پورت B.28 / ورودی ADC (1) /i.o برای تایمر	I/O
11	PB29/AD2/PCK1	پورت B.29 / ورودی ADC (2)/خروجی پالس	I/O
12	PB30/AD3/PCK2	پورت B.30 / ورودی ADC (3) /خروجی پالس	I/O
13	PA8/PGMM0/RTS1	پورت A.8 / انتخاب مد برنامه ریزی/ درخواست ارسال	I/O
14	PA9/PGMM1/CTS1	پورت A.9/انتخاب مد برنامه ریزی/Clear To Send	I/O
15	VDDCORE	ولتاژ تغذیه CPU	Power
16	GND	گراند	GND
17	VDDIO	ولتاژ تغذه خطوط I/O	Power

18	PA10/PGMM2/TW D	پورت A.10/ انتخاب مد برنامه ریزی/ خط داده tow-wire	I/O
19	PA11/PGMM3/TW CK	پورت A.11/ انتخاب مد برنامه ریزی/خط کلاک tow-wire	I/O
20	PA12/PGMD0/SPI_NPC S0	پورت A.12/خط داده ی برنامه نویسی /خط انتخاب کننده دستگاه (پروتکل spi)	I/O
21	PA13/PGMD1/SPI0_NP CS1	پورت A.13/خط داده ی برنامه نویسی/ خط انتخاب کننده دستگاه (پروتکل spi)	I/O
22	PA14/PGMD2/SPI0_NP CS2	پورت A.14/خط داده ی برنامه نویسی/ خط انتخاب کننده دستگاه (پروتکل spi)	I/O
23	PA15/PGMD3/SPI0_NP CS3	پورت A.15/خط داده ی برنامه نویسی خط انتخاب کننده دستگاه (پروتکل spi)	I/O
24	PA16/PGMD4/SPI0_MIS O	پورت A.16/خط داده ی برنامه نویسی/ خط MISO برای رابط SPI شماره0	I/O
25	PA17/PGMD5/SPI0_MO SI	پورت A.17/خط داده ی برنامه نویسی/ خط MOSI برای رابط SPI شماره0	I/O
26	PA18/PGMD6/SPI0_SP CK	پورت A.18/خط داده ی برنامه نویسی/ خط کلاک برای رابط SPI شماره0	I/O
27	PB9/EMDIO	پورت B.9/مدیریت داده ورودی و خروجی (Ethernet)	I/O
28	PB8/EMDC	پورت B.9/مدیریت کلاک (Ethernet)	I/O
29	PB14/ERX3	پورت B.9/خط دریافت داده (Ethernet)	I/O
30	PB13/ERX2	پورت B.9/ خط دریافت داده (Ethernet)	I/O
31	PB6/ERX1	پورت B.9/ خط دریافت داده (Ethernet)	I/O
32	GND	گراند	GND
33	VDDIO	ولتاژ تغذه خطوط I/O	Power
34	PB5/ERX0	پورت B.5/ خط دریافت داده (Ethernet)	I/O
35	PB15/ERXDV/ECRSDV	پورت B.15/چک اعتبار داده ورودی (Ethernet)	I/O
36	PB17/ERXCK	پورت B.17/ خط دریافت کلاک(Ethernet)	I/O
37	VDDCORE	ولتاژ تغذیه CPU	Power
38	PB7/ERXER	پورت B.7/ خط دریافت خطا (Ethernet)	I/O
39	PB12/ETXER	پورت B.12/خط ارسال خطا (Ethernet)	I/O

40	PB0/ETXCK/EREFCK	پورت B.0/خط ارسال کلاک (Ethernet)	I/O
41	PB1/ETXEN	پورت B.1/ فعال سازی تبادل داده (Ethernet)	I/O
42	PB2/ETX0	پورت B.2/ خط ارسال داده (Ethernet)	I/O
43	PB3/ETX1	پورت B.3/ خط ارسال داده (Ethernet)	I/O
44	PB10/ETX2	پورت B.10/ خط ارسال داده (Ethernet)	I/O
45	PB11/ETX3	پورت B.11/ خط ارسال داده (Ethernet)	I/O
46	PA19/PGMD7/CANRX	پورت A.19/خط داده ی برنامه نویسی/ورودی CAN	I/O
47	PA20/PGMD8/CANTX	پورت A.20/خط داده ی برنامه نویسی/خروجی CAN	I/O
48	VDDIO	ولتاژ تغذه خطوط I/O	Power
49	PA21/PGMD9/TF	پورت A.21/خط داده ی برنامه نویسی/خط ارسال داده همزمان سازی (Synchronous)	I/O
50	PA22/PGMD10/TK	پورت A.22/خط داده ی برنامه نویسی/خط ارسال کلاک (Synchronous)	I/O
51	TDI	Test Data In(واسط JTAG)	Input
52	GND	گراند	GND
53	PB16/ECOL	پورت B.16/اشکارساز اتصال (Ethernet)	I/O
54	PB4/ECRS	پورت B.4/خط تشخیص حامل(Ethernet)	I/O
55	PA2 3/PGMD11/TD	پورت A.22/خط داده ی برنامه نویسی/خط ارسال داده (Synchronous)	I/O
56	PA24/PGMD12/RD	پورت A.22/خط داده ی برنامه نویسی/ خط دریافت داده (Synchronous)	I/O
57	NRST	خط بازنشانی قطعه (ریست)	Input
58	TST	انتخاب حالت تست	Input
59	PA25/PGMD13/RK	پورت A.22/خط داده ی برنامه نویسی/ خط دریافت کلاک (Synchronous)	I/O
60	PA26/PGMD14/RF	پورت A.22/خط داده ی برنامه نویسی/ خط دریافت داده همزمان سازی (Synchronous)	I/O
61	VDDIO	ولتاژ تغذه خطوط I/O	Power
62	VDDCORE	ولتاژ تغذیه CPU	Power
63	PB18/EF100	پورت B.18/ Force 100 Mbits.sec	I/O
64	PB19/PWM0	پورت B.19/PWM0	I/O
65	PB20/PWM1	پورت B.20/PWM1	I/O

66	PB21/PWM2	پورت PWM2/B.21	I/O
67	PB22/PWM3	پورت PWM3/B.22	I/O
68	GND	گراند	GND
69	PB23/TIOA0	پورت i.o/B.23 برای تایمر	I/O
70	PB24/TIOB0	پورت i.o/B.24 برای تایمر	I/O
71	PB25/TIOA1	پورت i.o/B.25 برای تایمر	I/O
72	PB26/TIOB1	پورت i.o/B.26 برای تایمر	I/O
73	PA2 7/PGMD15/DRXD	پورت A.22/خط داده ی برنامه نویسی/ خط ارسال داده Debug	I/O
74	PA28/DT X D	پورت A.28/ خط دریافت داده Debug	I/O
75	PA29/FIQ	پورت A.29/ ورودی وقفه فوری	I/O
76	TDO	Test Data Out(واسط JTAG)	Output
77	JTAGSEL	انتخاب گر JTAG	Input
78	TMS	Test Mode Select (واسط JTAG)	Input
79	TCK	Test Clock(واسط JTAG)	Input
80	PA30/IRQ0	پورت A.30/ورودی وقفه خارجی	I/O
81	PA0/PGMEN0/RXD0	پورت A.0 / فعال ساز برنامه ریزی/ ورودی داده سریال USART0	I/O
82	PA1/PGMEN1/TXD0	پورت A.1 / فعال ساز برنامه ریزی/ خروجی داده سریال USART0	I/O
83	GND	گراند	GND
84	VDDIO	ولتاژ تغذه خطوط I/O	Power
85	PA3/RTS0	پورت A.3/ درخواست ارسال	I/O
86	PA2/SCK0	پورت A.2/ خط کلاک سریال	I/O
87	VDDCORE	ولتاژ تغذیه CPU	Power
88	PA4/PGMNCMD/CTS0	پورت A.4/ خط فرمان برنامه ریزی/Clear To Send	I/O
89	PA5/PGMRDY/RX D 1	پورت A.5 /خط خواندن برنامه ریزی/ ورودی داده سریال USART1	I/O
90	PA6/PGMNOE/T X D1	پورت A.6/خط اماده کردن برنامه ریزی/ خروجی داده سریال USART1	I/O
91	PA7/PGMNVALID/SCK1	پورت A.7 / مسیر هدایت داده / خط کلاک سریال	I/O
92	ERASE	پاک کردن فلش (با اتصال این پایه به VCC حافظه ی فلش پاک میشود)	Input
93	DDM	USB+	Analog

Analog	USB-	DDP	94
Power	ولتاژ تغذیه حافظه ی فلش	VDDFLASH	95
GND	گراند	GND	96
Input	ورودی کلاک به میکرو از کریستال	XIN/PGMCK	97
Output	خروجی کلاک ازمیکرو به کریستال	XOUT	98
Input	اتصال به فیلتر RC مربوط به PLL	PLLRC	99
Power	ولتاژ تغذیه ی واحد PLL	VDDPLL	100

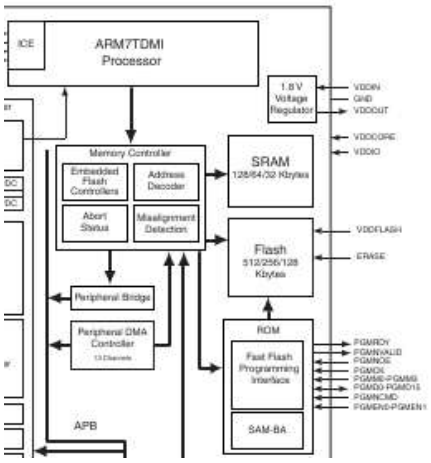
توضیحات بیشتر در مورد امکانات :

1– معماری risc چیست ؟

معماری risc (Reduced Instruction Set Coding/Computer) نوعی طراحی برای سخت افزار های میکرو الکترونیک میباشد ، دراین معماری با تغییرات سخت افزاری مجموعه دستورات برنامه نویسی کم میشود ، در مقابل این معماری ، معماری cisc (Set Coding Complex Instruction) قرار دارد ، در معماری cisc حجم دستورات زیاد تر میشود ، اما سخت افزار میکرو کنترلر ساده تر میگردد ، مثلا در معماری cisc برای روشن کردن یک led شما باید ، یکی از متغییر های حافظه را تغییر دهید و سپس آن را به cpu ارسال کنید ، cpu بعد از پردازش دستور فرمان set کرد پورت را صادر میکند ، اما در معماری risc شما برنامه را در حافظه فلش میکرو قرار میدهید ، بعد از روشن شدن میکرو ، برنامه به حافظه موقت منتقل میشود و سپس از آنجا به سمت cpu میرود ، در cpu دستور پردازش شده و سپس فرمان به کنترل کننده پورت ارسال میشود و.... در نهایت پایه مورد نظر set میگردد .

2– sram چیست ؟

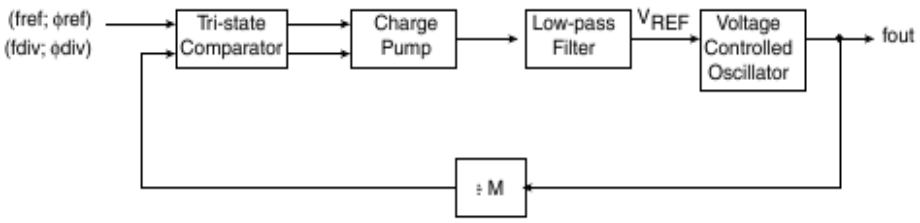
Static random access memory یا SRAM نوعی حافظه موقت میباشد ، که اطلاعات قبل از آنکه به CPU برسد در آن ذخیره میشود . با این کار دیگر نیاز نیست وقت CPU برای دریافت اطلاعات از حافظه فلش تلف شود ، در هنگام پردازش اطلاعات توسط CPU اطلاعات از فلش به این حافظه منتقل میشود و همیشه اطلاعات در دسترس CPU خواهد بود . در این نوع از حافظه



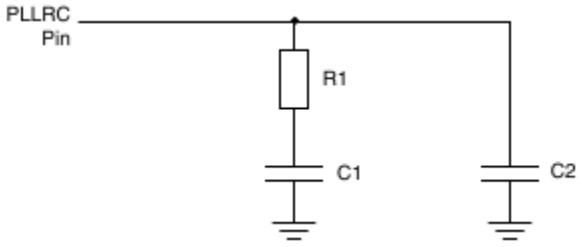
ها از فلیپ فلاپ برای ذخیره سازی هر بیت داده استفاده می گردد. یک فلیپ فلاپ برای یک سلول حافظه، از چهار تا شش ترانزیستور استفاده می کند . حافظه های SRAM نیازمند بازخوانی / بازنویسی اطلاعات نخواهند بود، بنابراین سرعت این نوع حافظه ، از دیگر حافظه های هم سطح (DRAM و..) بسیار بیشتر میباشد .

3–Pll چیست ؟

Phase-locked-loop (حلقه فازی قفل شده)یا واحد pll در میکرو کنترلر های arm وظیفه ایجاد یک پالس با فرکانس ثابت را بر عهده دارد . واحد pll برای ایجاد فرکانس نیاز به شبکه rc خارجی دارد .بخش pll در میکرو کنترلر های ساخته شده توسط شرکت اتمل از چهار قسمت زیر تشکیل شده است :



واحد pll برای نوسان سازی به شبکه rc روبرو نیاز دارد مقدار خازن ها و مقاومت از فرمول های خاص ، برای فرکانس های مختلف محاسبه میشوند ، در آموزش برنامه نویسی با نحوه محاسبه آنها اشنا خواهیم شد .



4– تایمر Watchdog چیست ؟

WATCHDOG یکی از تایمر های میکرو است که میتواند تا یک زمان خاص بشمارد وبعد میکرو را ریست کند ، این تایمر میتواند تا 8 زمان 16،32 ، 64 ، 128 ، 256 ، 512 ، 1024 و 2048 و در بعضی از میکروها 4096, 8192 میلی ثانیه بشمارد ، بعد از سپری شدن زمان میکرو ریست میشود و برنامه دوباره از ابتدا اجرا میشود. از این تایمر در زمان وجود خطا یا هنگ کردن میکرو استفاده میشود .

5–Ethernet MAC 10/100 base-T (اترنت MAC 10/100 base-T) چیست ؟

Ethernet :یکی از شبکه های استاندارد کامپیوتری میباشد ، توسط این شبکه شما میتوانید چندین کامپیوتر را به یکدیگر متصل نمایید و داده بین آنها رد و بدل کنید.برای اتصال کامپیوتر به دنیای مجازی نیاز مند یک کارت شبکه هستیم ، میکرو کنترلر های arm میتوانند توسط یک چیپ که به ان PHY گفته میشود ، به شبکه Ethernet متصل شوند و با سایر دستگاه های متصل به شبکه تبادل اطلاعات نمایند .

MAC (Media Access Controller) :منظور از این مورد دسترسی به پروتکل های مختلف شبکه از طریق چیپ های مختلف میباشد ، در این حالت برای میکرو کنترلر نوع پروتکل مهم نخواهد بود و این چیپ واسط است که خود را با پروتکل سازگار میکند .

10/100 : این دو عدد بیان گر سرعت شبکه میباشند ، شبکه اترنت در دو سرعت استاندار د 10 Mbit/s و 100 Mbit/s در دسترس شما میباشد (البته سرعت بالا تر نیز برای شبکه های کامپیوتری وجود دارد ، اما میکرو کنترلر های arm فقط از این دو سرعت پشتیبانی میکند .)

base-T (T="Twisted" Pair Copper) :شبکه Ethernet درای مبنا های مختلفی میباشد :

شبکه های سیمی

BASE-TX100 2.1

BASE-T4100 2.2

BASE-T2100 2.3

شبکه های فیبر نوری

BASE-FX100 3.1

BASE-SX100 3.2

BASE-BX100 3.3

البته میتوانیم به موارد بالا شبکه های بیسیم را نیز اضافه کنیم .... . اما انچه که باید از base-T بدانیم این است که این شبکه توسط کابل های زوج تابیده اطلاعات را منتقل میکند . یعنی برای تبادل داده به 8 سیم نیاز داریم (به غیر از رابط های

همزمانی و کلاک) و...در آموزش برنامه نویسی با Ethernet MAC 10/100 base-T بیشتر اشنا خواهیم شد ، شما میتوانید برای دریافت اطلاعات بیشتر به آدرس زیر مراجعه کنید :

http://en.wikipedia.org/wiki/Ethernet

http://en.wikipedia.org/wiki/Fast_Ethernet

6- رابط CAN چیست ؟

Controller–area network (CAN or CAN-bus) یک پروتکل ارتباطی برای متصل کردن میکرو کنترلرها و سایر وسایل الکترونیک به یکدیگر بدون نیاز به وسیله کنترل کننده میباشد .

نمونه قابل مشاهده کاربرد can در ECU اتومبیل میباشد . سیستم ECU تمامی امکانات اتومبیل را کنترل میکند ، امکانی از قبیل ، چراغ ها ، سیستم صوتی ، درب ها و موتور ، سیستم سوخت رسانی و.... قطعا برای اتصال تمامی این امکانات نیاز به تعداد زیادی سیم خواهیم داشت ، اما با پروتکل can از طریق دو سیم تمامی موارد را به پردازنده اصلی مرتبط میکنیم و.....

برای اطلاعت بیشتر به آدرس زیر مراجعه کنید :

http://en.wikipedia.org/wiki/CAN_bus

7- کنترل کننده سریال هم زمان (Synchronous Serial Controller (SSC)) چیست ؟

SSC یکی از امکانات موجود در ARM می باشد که به شما امکان کار با قطعات جانبی که از پروتکل I2S استفاده می کنند (دارای پورت I2S هستند) را می دهد .

✓ I2S Audio Bus چیست ؟

I2S (IC Sound–Inter) یک پروتکل سه سیمه برای منتقل کردن داده می باشد ، در این پروتکل داده موجود معمولا صدا های دیجیتال می باشد .در این سیستم یک خط انتقال سریال برای تبادل داده بین دو دستگاه یا قطعه صوتی ایجاد می شود ، شما می توانید پورت i2s را در پشت اغلب دستگا ههای صوتی و تصویری بیابید .

پروتکل SSC یک رابط همزمان برای ارتباط با کلیه قطعات و لوازمی می باشد که به صورت همزمان اقدام به ارسال و دریافت داده می کنند ، برای نمونه می توان به I2S, Short Frame Sync, Long Frame Sync, اشاره کرد ، کلیه این پروتکل ها داده را به صورت همزمان به خروجی ارسال می کنند .

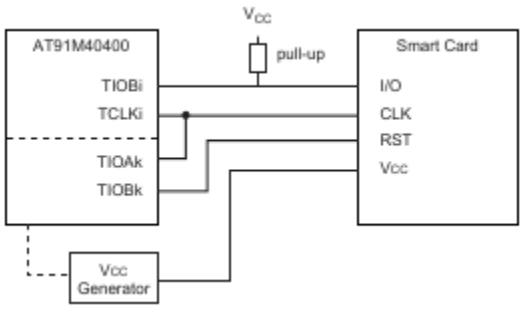
8–ISO7816 T0/T1 Smart Card چیست ؟

ISO7816 یک پروتکل استاندارد برای تبادل اطلاعات بین چیپ های الکترونیکی و پردازنده ها میباشد .چیپ های الکترونیکی ، بر روی کارت های از جنس پلاستیک نصب میشود و معمولا شما آنها را به نام اسمارت کارت میشناسید ، کارت های تلفن و کارت های اعتباری نمونه های مختلف از اسمارت کارت هستند .

پردازنده نیز وظیفه دریافت اطلاعات از کارت و تبدیل آن به داده قابل فهم برای انسال یا ارسال ان به پردازنده مرکزی را به عهده دارد .یک Smart Card درای پایه های زیر است :

GND	common reference
VCC	power supply for the Smart Card
CLK	system clock of the Smart Card
RST	system reset of the Smart Card
I/O	data signal

پایه های vcc و gnd ، مربوط به تغذیه کارت میباشند ، ولتاژ تغذیه کارت 5 ولت است و باید از تغذیه میکرو تامین شود .بر روی پایه کلاک ، یک پالس ایجاد میشود ، این پالس برای همزمانی کارت و میکرو به کار میرود و از طرف میکرو تولید میشود . پایه ریست (rst) نقش بازنشانی چیپ ، قبل از انتقال داده را به عهده دارد .، بازنشانی توسط میکرو انجام میشود .نقش پایه i/o نیز تبادل داده بین میکرو و چیپ میباشد ، این پایه باید توسط یک مقاومت pull-up به vcc متصل شود .



نقشه زیر ، یک مثال از اتصال چیپ به میکرو میباشد ، ممکن است در دیگر میکرو ها نام پایه ها متفاوت باشد .(در برنامه نویسی ، بیشتر با نحوه اتصال آشنا میشویم)انتقال داده بین میکرو و چیپ در 4 مرحله انجام میشود :

- one start bit at low level
- 8 data bits of information (from Least Significant Bit to Most Significant Bit)
- one parity bit
- two (if no error) or three (if error) stop bits at high level

ابتدا پایه i/o صفر میشود ، با این کار چیپ فعال شده و آماده تبادل داده میشود . سپس 8 بیت اطلاعات موجود بر روی کارت ، به میکرو ارسال میگردد .بعد از ارسال اطلاعات یک بیت خطا ارسال میشود ، (بیت توازن یا خطا میتواند صفر یا یک باشد ، این بیت تعداد صفر ها و یک های موجود در 8 بیت اطلاعات را زوج یا فرد میکند و در صورتی که اطلاعت رسیده متناقض باشد ، ارسال اطلاعات تکرار میگردد).در صورتی که خطایی وجود داشته باشد ، پردازنده سه بیت را برای چیپ ارسال میکند، با انجام این عمل پردازنده میفهمد که باید دوباره اطلاعات را ارسال کند .در صورتی که خطایی وجود نداشته باشد ، میکرو دو بیت را به چیپ ارسال میکند و سپس بیت شروع و ادامه اطلاعات ...

Figure 1. Communication on the I/O Line without Parity Error

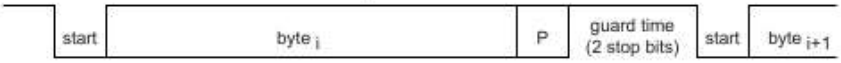
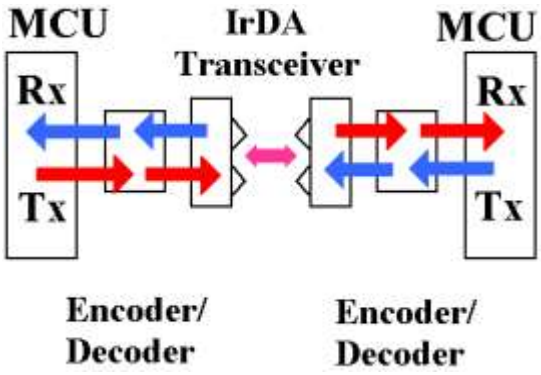


Figure 2. Communication on the I/O Line with Parity Error

9– IrDA ® Infrared Modulation/Demodulation چیست ؟

تاکنون led های مادون قرمز را مشاهده کرده اید ؟ شاید فکر کنید از این led ها فقط برای تشخیص مانع و ... استفاده میشود اما چنین نیست . در الکترونیک میتوانیم از این نوع led ها برای ارسال و دریافت داده استفاده نماییم ، میکرو کنتلر های arm به شما این امکان را میدهند تا داده خود را به فرم دلخواه مدوله کنید و سپس آن را با این نوع led ها در فضا انتشار دهید . در میکرو گیرنده شما بعد از دریافت این امواج باید آن را دمودله کنید و داده خود را تحویل بگیرید .



10– Modem Line چیست؟

مودم برای انتقال اطلاعات بین کامپیوترها از طریق کانال های مخابراتی به کار می رود. یک مودم در سر راه خطوط ارسالی قرار می گیرد تا بتواند پالس های دیجیتال را به سیگنال های آنالوگ تبدیل کند. سپس سیگنال های آنالوگ را می توان از طریق خطوط تلفن ، فیبر نوری ، کابل کواکسیال، مایکروویو ، ماهواره مخابراتی و غیره انتقال داد. کامپیوتری که می بایست اطلاعات را دریافت کند نیز از یک مودم دیگر استفاده می کند تا بدین وسیله سیگنال آنالوگ را مجدداً به پالس های دیجیتال تبدیل کند که این عمل را دمدوله گویند. شرکت اتمل این امکان را برای شما مهیا کرده است که به جای کامپیوتر از میکرو کنترلر arm استفاده کنید ، این میکرو اطلاعات موازی خروجی مودم را میخواند و میتواند تصمیمات لازم را اتخاذ نمایید ، این قابلیت دو طرفه میباشد (میکرو قابلیت ارسال و دریافت اطلاعات را دارد) .

11– SAM-BA ® Boot چیست؟

به نظر من یکی از بهترین امکانات میکرو کنترلر های arm وجود SAM-BA ® Boot در آنها میباشد .

توسط SAM-BA ® Boot میتوانید میکرو کنترلر را از طریق پورت usb و بدون نیاز به سخت افزار پرگرامر به راحتی پرگرام کنید ، برای اطلاعات بیشتر به آدرس زیر یا مجله ی pmm2 مراجعه کنید :

http://www.atmel.com/dyn/products/tools_card.asp?tool_id=3883

12– JTAG چیست ؟

IEEE Standard 1149.1-1990 Test Access Portand Boundary-Scan Architecture یا jtag یک پروتکل ارتباطی میباشد که توسط تعدادی از شرکت های وابسته به سازمان ieee و تحت استانداد آن به ثبت رسیده است .در پروتکل jtag دسترسی کامل به cpu و حافظه ها فراهم میباشد ، و شما میتوانید داده های پردازش شده یا در حال پردازش توسط آنها را مشاهده کنید .

رابط jtag از 4 پایه اصلی برای ارتباط با سخت افزار استفاده میکند :

هر وسیله ای که با استاندارد Jtag سازگار باشد لازم است تا پین های زیر را داشته باشد :

1– TCK (Test Clock Input) : Clock ورودی ، این پالس برای همزمانی میان دستگاه مورد تست و پروگرامر jtag میباشد .

2– TDI (Test Data In) : از طریق این پایه ، داده از پروگرامر به دستگاه در حال تست وارد میشود .

3– TDO (Test Data Out): از طریق این پایه ، داده از دستگاه مورد تست به پروگرامر میرود .

4– TMS (Test Mode Select) : از طریق این پورت حالتهای مختلف تست انتخاب می شود.

همچنین در این میان دو پایه دیگر نیز وجود دارد :

5– TRST (Test Reset Input) این پایه از پروگرامر به ریست دستگاه متصل میشود و قبل از شروع کار آن را باز نشانی میکند .

6–JTAGSEL (JTAG SELECT) : برای راه اندازی پروتکل JTAG ، این پایه باید یک شود .

برای کار با JTAG به موارد زیر نیاز دارید :

1–کابل ارتباطی

2–سخت افزار JTAG

3– نرم افزار JTAG

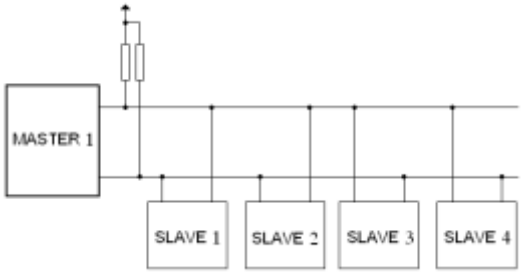
کابل ارتباطی وظیفه اتصال دستگاه مورد تست را به پروگرامر را به عهده دارد ، معمولا پروگرامر از طریق یک کابل دیگر به پورت سریال یا usb یا lpt کامپیوتر متصل میشود .

سخت افزار JTAG وظیفه کنترل داده های ارسالی و دریافتی و همچنین تغییر دادن انها به کد قابل فهم کامپیوتر را به عهده دارد .وظیفه نرم افزار JTAG ، تبدیل کد های دریافتی به علائم نمایشی و کد های قابل فهم برای انسان میباشد .

با توجه به تنوع سخت افزار و نرم افزار ، پروگرامر های JTAG زیادی برای کار با میکرو کنترلر های ARM ارئه شده است ، شما میتوانید نقشه ها و اطلاعات بیشتر را در سایت WWW.ATMEL.COM بیاید .

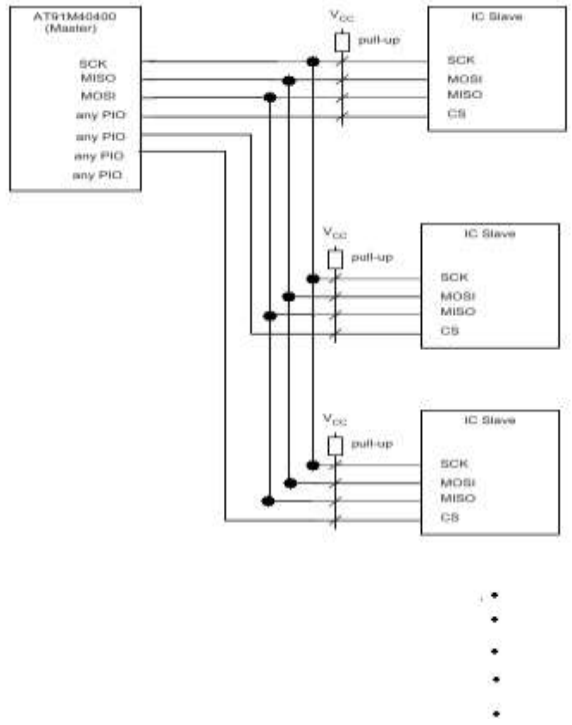
13– پروتکل Two-wire چیست ؟

پروتکل Two-wire یا ارتباط دو سیمه ، یک پروتکل ارتباطی سریال میباشد . در این پروتکل از دو سیم برای تبادل داده و کلاک ، و سیم های گراند و vcc استفاده میشود . در این پروتکل میتوان توسط دو سیم تعداد 128 وسیله را به هم متصل نمود ، هر وسیله دارای یک ادرس منحصر بفرد میباشد و توسط همان ادرس شناسایی میشود . در این پروتکل میکرو فرستنده به عنوان مستر و میکرو گیرنده اسلیو میباشد .



14–پروتکل spi چیست ؟

Serial Peripheral Interface یا پروتکل spi یک رابط سریال برای انتقال داده به صورت همزمان بین دو وسیله میباشد ، چیپ معرفی شده دارای دو عدد خط ارتباطی SPI مجزا میباشد ، SPIA از خطوط SPCKA (خروجی کلاک از مستر به اسلیو) و MISOA (داده خروجی از اسلو به مستر (دیتا به این خط وارد میشود)) و MOSIA (داده خروجی از مستر به اسلیو (داده از این خط خارج میشود) و NSSA (انتخاب کننده اسلیو 1) استفاده میکند ، برای SPIB نیز از خطوط SPCKB و MISOB و MOSIB و NSSB استفاده میشود .هنگامی که در برنامه پروتکل SPI راه اندازی میشود ، میکرو Master شروع به ارسال پالس به میکرو Slave میکند ، ارسال پالس از طریق خط SPCK انجام میشود و برای همزمانی میان دستگاه ها میباشد .با یک شدن پایه NSSA میکرو Slave انتخاب میشود و دو میکرو شروع به تبادل داده با هم میکنند .پروتکل SPI این قابلیت دارد که در هر واحد زمانی با یک میکرو به تبادل اطلاعات بپردازد ،برای فعال سازی دستگاه های بیشتر از خطوط NPCSA0 - NPCSA3 NPCSB0 - NPCSB3 استفاده میشود.



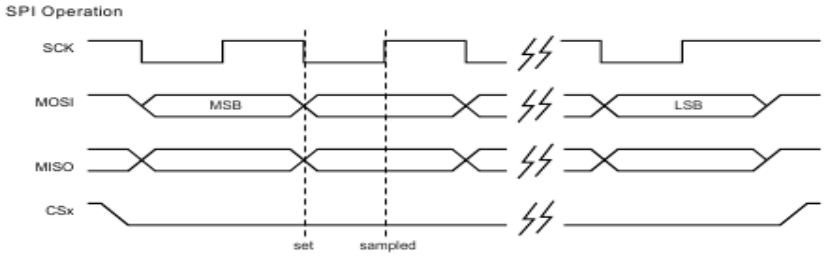
همانطور که در تصویر مشاهده میکنید ، مدار ما دارای یک میکرو مستر و تعداد سه عدد اسلیو میباشد ، با فعال شدن اولین اسلیو میتوانیم داده را به ان منتقل کنیم یا از ان بگیریم و...

فعال شدن اسلیو های از طریق پایه cs (chip select) انجام میشود ، شما میتوانید از هر پایه های به عنوان cs استفاده کنید . معمولا از پایه های NPCSA0 - NPCSA3 NPCSB0 - NPCSB3 که برای این منظور ایجاد شده اند استفاده میشود.

پروتکل spi در میکرو کنترلر های arm دارای ویژگی های خاص خود میباشد :

- Configurable number of bits to transfer
- Supports high baud rate (1M bits/sec with a 32 MHz master clock)
- MOSI line setting synchronized with the falling edge of the clock
- MISO line sampling synchronized with the rising edge of the clock
- Memory size: less than 0.5K bytes (90 Code Words - 10 Data Words in the s tack, plus the relevant buffers)

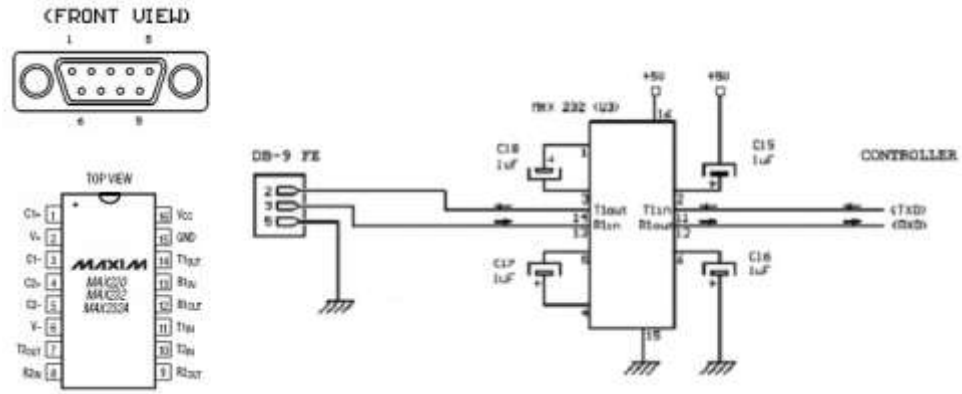
همانگونه که در تصویر مشاهده میکنید ، در هر پالس کلاک ، یک بیت از اطلاعات ارسال میشود . با این ویزگی شما قادر خواهید بود فرکانس کلاک را تا 32 مگاهرتز افزایش دهید و نرخ انتقال داده را به .5 کیلو بایت بر ثانیه برسانید :



15–پروتکل UART چیست؟

در این ارتباط از دوسیم به نام های rxd و txd استفاده میشود که خط rxd وسیله اول دیتا را از ان به بیرون منتقل میکندو خط خروجی دیتا است و به ورودی دیتا دستگاه دوم (txd) متصل میشود و خط txd ورودی دیتا است که به خروجی دیتای دستگاه دیگر (rxd) متصل میشود.

این پروتکل تقریبا در تمامی وسایل الکترونیکی استاندارد استفاده میشود ، ما در کامپیوتر ان را به نام پورت com میشناسیم . در این پروتکل ولتاژ بین 0 تا 1.5 سطح صفر منطقی و ولتاژ 3 تا 5 سطح یک منطقی میباشد . (برای میکرو کنترلر ها) از انجا که مقدار سطح ولتاژ منطقی در کامپیوتر متفاوت است ، از مدار زیر برای تطابق ولتاژ استفاده میشود .



16–Debug Unit (DBGU) چیست ؟

توسط این واحد شما میتوانید عملیات زیر را انجام دهید :

- application software

- operating systems
- hardware systems based on an ARM processor.
- The debug unit enables you to:
- stop program execution
- examine and alter processor and coprocessor state
- examine and alter memory and input/output peripheral state
- restart the processor core.

برای شروع کار باید پایه های PA2 7/PGMD15/DRXD و PA28/DT X D و gnd (پایه های 73 و 74 و زمین مدار) را به پورت com کامپیوتر خود متصل کنید . (میکرو باید برنامه ریزی شده باشد) در کلیه کامپایلر ها نرم افزار مورد نیاز برای کار با واحد دیباگ وجود دارد ، با اجرای برنامه شما به تمامی ریجستر های داخلی میکرو دسترسی دارید و میتوانید آنها در در هنگام اجرای برنامه تغییر دهید و مقدارآنها را مشاهده کنید ، شما میتوانید میکرو را ریست کنید و در کل تمامی امکانات در دسترس شما قرار دارد .

17 – وقفه چیست ؟

وقفه یا Interrupt یکی از امکانات کاربردی میکرو کنترلر ها میباشد ، با استفاده از وقفه میکرو برنامه در حال اجرا را رها کرده و به زیر برنامه وقفه میرود و برنامه دیگری را اجرا میکند . میکرو کنترلر های arm دارای دو منبع وقفه خارجی میباشند :

•FIQ - Fast Interrupt

• IRQ - Normal Interrupt

نوع اول وقفه سریع میباشد ، هنگامی که یک پالس به پایه مربوط به این وقفه اعمال میشود (سطح پالس در برنامه مشخص میشود)، cpu میکرو تمام کار های خود را رها میکند و به زیر برنامه وقفه میرود .

نوع دوم وقفه معمولی میباشد ، در این نوع وقفه cpu میکرو دستور در حال اجرا را تمام میکند و به زیر برنامه وقفه میرود .CpU بعد از انجام دادن زیر برنامه وقفه ، به حلقه اصلی برمیگردد و برنامه قبلی را ادامه میدهد . در میکروکنترلر های Arm منابع وقفه داخلی نیز وجود دارد که در آموزش برنامه نویسی به بررسی آنها پرداخته میشود .

.

ضمیمه شماره دو :راه اندازی اولیه میکروکنترلر و موارد مورد نیاز :

برای راه اندازی این میکرو کنترلر ها پیش نیاز های زیر وجود دارد :

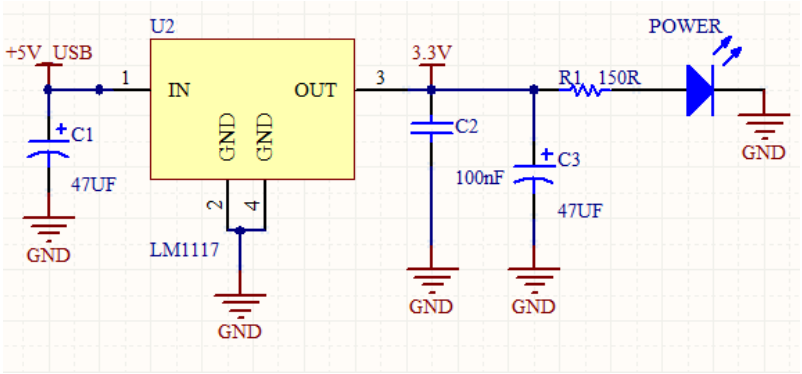
- بخش تغذیه
- بخش کلاک و pll
- بخش پروگرامر
- پیش نیاز های مربوط به امکانات جانبی

توجه داشته باشید که موارد بالا که توضیح کامل آنها در ادامه آورده شده است ، برای روشن شدن میکرو الزامی میباشد و در صورتی که موارد به درستی رعایت نشود باعث عدم کارکرد صحیح میکرو میشود .

بخش تغذیه :

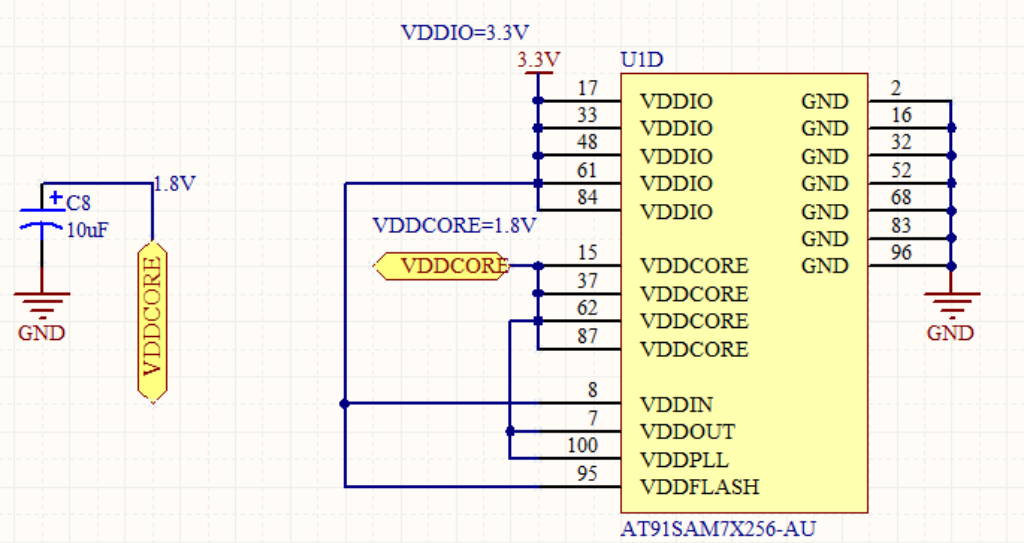
برای راه اندازی این میکرو کنترلر ها به دو ولتاژ 3.3 ولت با حداقل جریان 100 میلی آمپر برای تغذیه کردن پورت ها ، واحد های جانبی(نظیر I2C ، USART ، SPI و...) و حافظه ی فلش و به ولتاژ 1.8 ولت با حداقل جریان 50 میلی آمپر برای تغذیه کردن CPU و واحد PLL نیاز دارید .

شما میتوانید ولتاژ 3.3 را توسط رگولاتور های LF33 یا LM1117 (یا سایر رگولاتور های که خروجی 3.3 ولت تثبیت شده دارند) ، از ولتاژ USB تامین کنید :



در مدار بالا LED و مقاومت R1 صرفا برای نمایش دادن وضعیت روشن بودن برد میباشد .

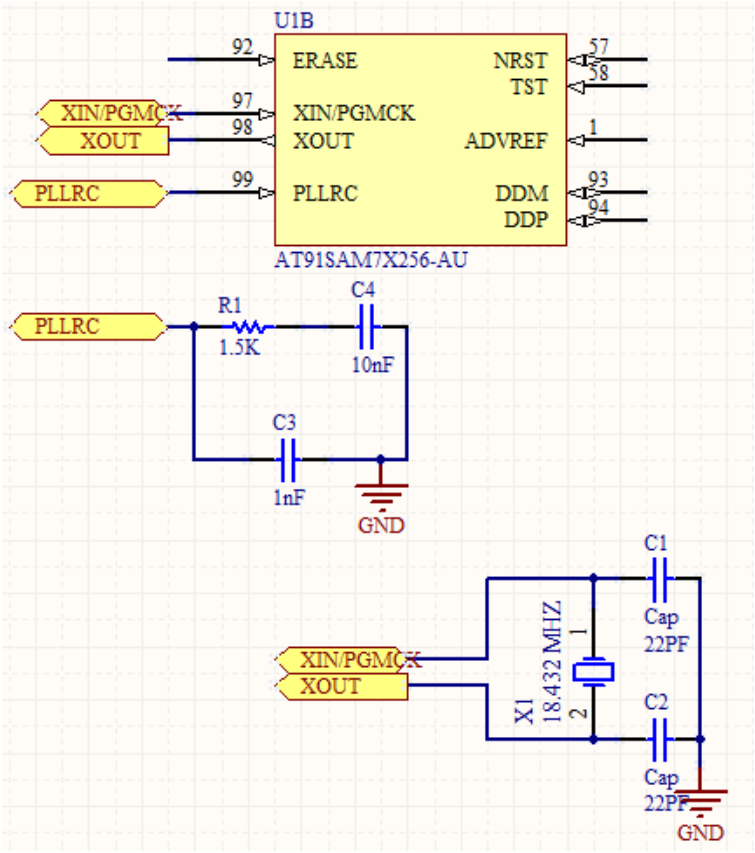
در داخل این میکرو کنترلر ها یک رگولاتور مجزا برای تامین ولتاژ 1.8 ولت در نظر گرفته شده است .ورودی این رگولاتور پایه ی 8 و خروجی آن پایه ی 7 میکرو کنترلر میباشد ، شما میتوانید با اعمال ولتاژ 3.3 ولت به ورودی ولتاژ ۱.۸ را از خروجی دریافت کنید :



بخش کلاک و PLL:

کلاک این میکرو کنترلر ها توسط کریستال موجود که بر روی دو پایه ی 97 و 98 (XIN/PGMCK و XOUT) قرار داده میشود تامین میگردد . مقدار کریستال میتواند بین 11.768 کیلو هرتز تا 55 مگاهرتز باشد . در صورتی که مایلید با استفاده از قابلیت SAM-BA میکرو کنترلر را برنامه ریز کنید ، باید مقدار کریستال را 18.432 مگاهرتز انتخاب کنید .

با استفاده از PLL میتوانید فرکانس کریستال را در مقادیر دلخواه ضرب نمایید ، واحد PLL برای انجام کار به یک اسیلاتور مجزا نیاز دارد ، که میتوان با استفاده از یک مقاومت و دو خازن ، مطابق شکل زیر آن را پیکربندی کرد . :



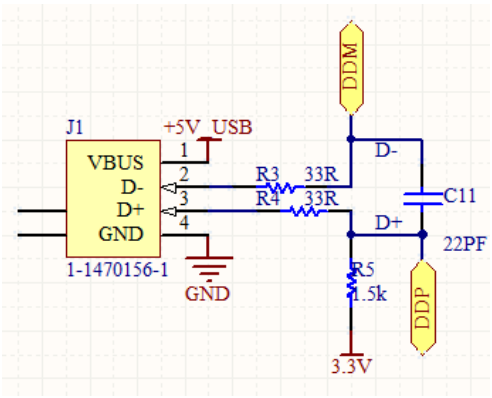
برای کسب اطلاعات بیشتر در مورد نحو ه ی راه انـدازی واحـد PLL بـه بخـش راه انـدازی واحـد هـای PMC و CLKG مراجعه کنید.

بخش پروگرامر :

برای برنامه ریزی این میکرو کنترلر های روش های SAM-BA و JTAG قابل استفاده میباشد .

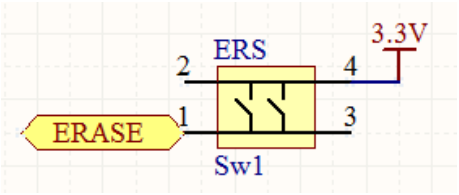
برای برنامه ریزی میکرو کنترلر از طریق پروتکل SAM-BA نیاز به هیچ سخت افزار جانبی ندارید کافی اسـت مـوارد زیـر را رعایت کنید :

1- SAM-BA برای برنامه ریزی میکرو از پورت USB استفاده میکند ، شما باید پایه ی داده + یا DDP را با یک مقاومـت 1.5 کیلو اهم به VCC متصل نمایید تا میکرو کنترلر به عنوان یک DEVISE از طرف کامپیوتر شناخته شود :

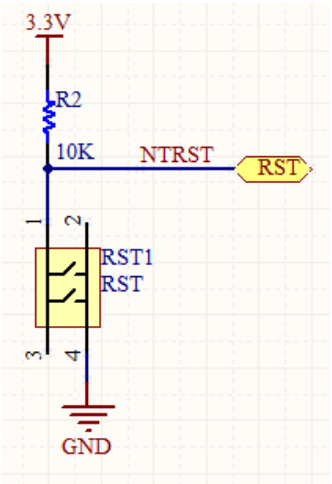


2- در حالت پیش فرض ، بر روی میکرو کنترلر برنامه ای وجود که به آن بوت لودر گفته میشود ، توسط این برنامه میکرو با پورت USB ارتباط برقرار کرده و داده ی موجود را به حافظه ی فلش منتقل میکند .

وقتی که میکرو را با SAM-BA برنامه ریزی میکنید ، برنامه ی بوت لودر به بخش های انتهای حافظه ی SRAM میرود ، با این عمل میکرو دیگر توسط کامپیوتر شناخته نمیشود . شما باید حافظه ی فلش را پاک کنید تا بوت لدر به ابتدای حافظه منتقل شده و توسط میکرو اجرا شود ، برای پاک کردن بوت لدر از پایه ی ERESE استفاده میشود. با اتصال این پایه به VCC هنگامی که برد روشن است ، حافظه ی فلش پاک میشود :

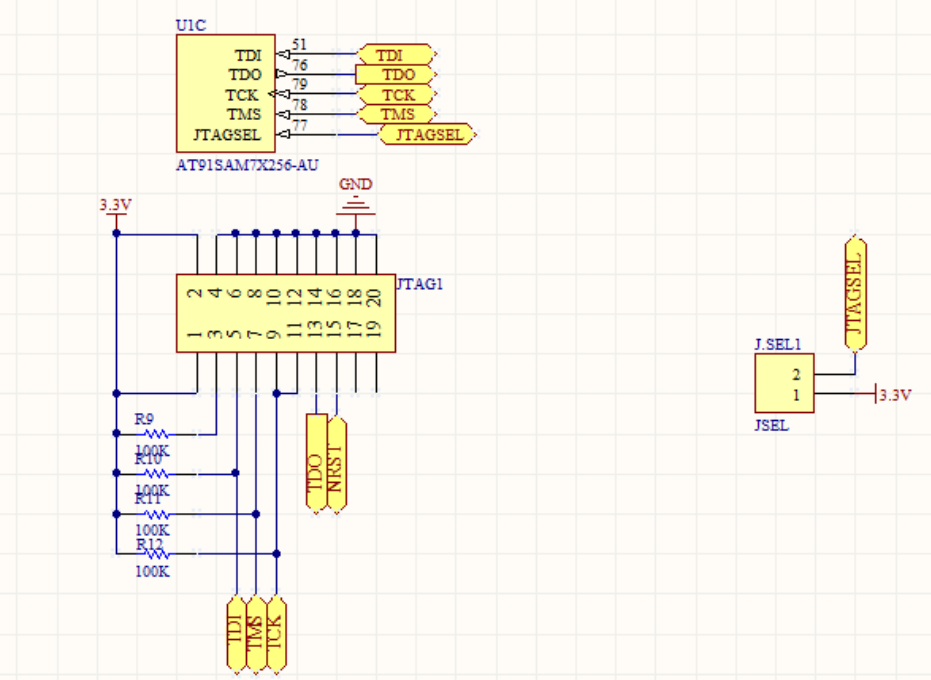


3- تمامی میکرو کنترلر ها دارای یک پایه برای باز نشانی یا ریست میباشند ، هنگامی که به این پایه یک پالس یک به صفر اعمال میشود ، CPU به خانه ی اول حافظه ی فلش رفته و برنامه را از ابتدا اجرا میکند :



4- در این میکرو کنترلر پایه ی ریست ، پایه ی 57 میباشد و با نام NTRST شناخته میشود .

برای برنامه ریزی میکرو کنترلر از طریق واسط JTAG باید پایه ی موجود در شکل زیر را پیکربندی کنید :



کانکتور JTAG1 ، یک استاندارد جهانی میباشد و کلیه پروگرامر ها و دیباگر هایی که برای ARM ارائه شده اند (نظیر HJTAG , U-LINK,JLINK و..) با آن سازگار میباشند .

برای راه اندازی واحد JTAG در این میکرو کنترلر ها لازم است که پایه ی JTAGSEL به ولتاژ تغذیه متصل شود ، برای این کار میتوانید از یک کلید فشاری (J.SEL1) استفاده کنید .

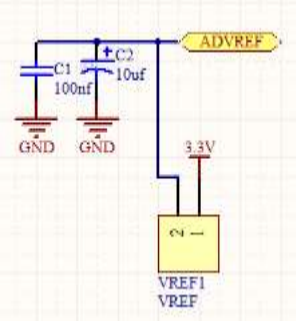
پایه ی 15 کانکتور JTAG1 باید به کلید ریست و پایه ی NTRST میکرو (پایه ی 57) متصل شود .

پیش نیاز های مربوط به امکانات جانبی :

با رعایت کردن نکات بالا میتوانید میکروکنترلر را راه اندازی کرده و با آن کار کنید ، در صورتی که نکات زیر را نیز در هنگام طراحی و روت کردن pcb رعایت کنید ، توفیق بیشتری در کار عملی با برد خواهید داشت .

1- پایه ی aref :

از این پایه برای تعیین کردن ولتاژ مرجع adc استفاده میشود ، هنگامی که این پایه آزاد باشد شما باید با تغییر دادن ریجستر ها در برنامه نویسی از ولتاژ مرجع داخلی استفاده کنید ، با متصل کردن این پایه به vcc ولتاژ مرجع برابر با vcc میشود و شما میتوانید مستقیما به سراغ خواندن خروجی adc بروید .



2- نویز :

بر روی برخی از پایه های تغذیه خازن های 100 نانو فاراد قرار دهید و در هنگام طراحی pcb ، آنها را به پایه های تغذیه نزدیک کنید ، این کار باعث از بین بردن نویز های خط میشود .