مقدمه:

مطالبی که در حال مطالعه ی آن هستید بخش چهارم از کتاب" مرجع کامل میکروکنترلرهای سری AT91SAM شرکت ATMEL" است که در آن به بررسی نحوه ی راه اندازی و استفاده از پروتکل های ارتباطی موجود در میکروکنترلر های سری AT91SAM شرکت ATMEL ، شامل پروتکل USART ، پروتکل SPI ، پروتکل I2C و پروتکل I2S پرداخته شده است .

ویرایش قبلی این مطالب این بخش قبلا در مجله ی PMM (مجله میکروکنترلر فارسی – Persian Microcontroller Magazine) شماره 6–7 و8 منتشر بود .

قبل از مطالعه مطالب این بخش شما باید بخش های 1 و 2 و 3 این کتاب که دارای مطالب زیر هستند را مطالعه کرده باشید:

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 . (نویسنده :1nafar)

2- راه اندازی منابع تامین کلاک : در این بخش شما با نحوه ی پیکربندی منابع کلاک میکروکنترلر آشنا می شوید .(نویسنده : آرمین غنی)

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 .(نویسنده : آرمین غنی)

برای کسب اطلاعات بیشتر د ر مورد این کتاب پست شماره 11 تاپیک زیر را مطالعه کنید :

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

توجه داشته باشید که مطالب این بخش ممکن است ناقص/اشتباه یا تاریخ گذشته باشند .

فهرست مطالب:

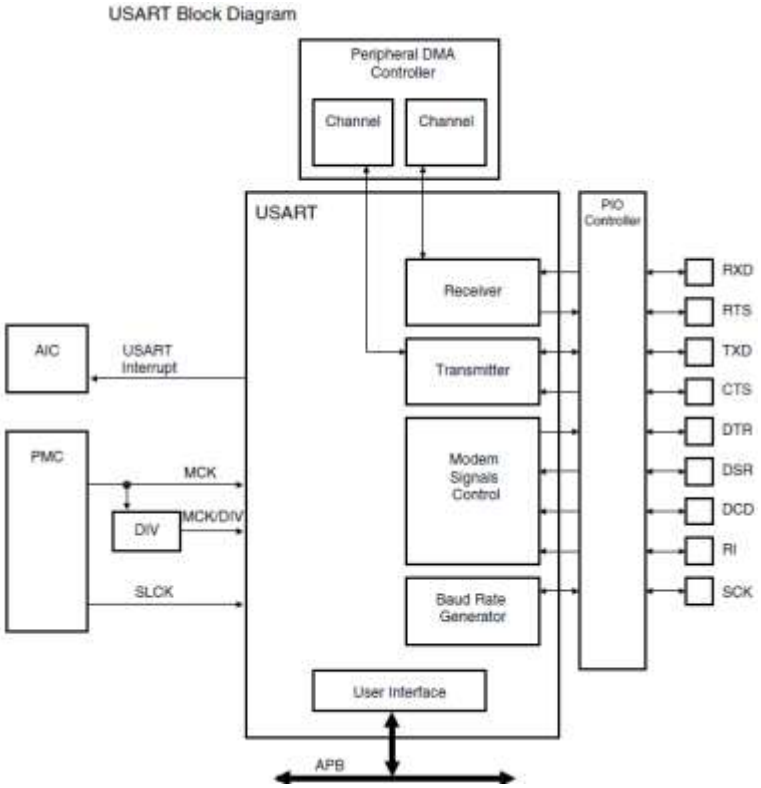
پورت سریال در سری AT91SAM و کتابخانه ی usart.h

میکرو کنترلر های سری at91sam معمولا دارای یک تا 4 عدد پورت usart (universal synchronous asynchronous serial link.) یا به نوعی همان سریال می باشند .این پورت ها علاوه بر پشتیبانی از ارتباط سریال هم زمان و غیر هم زمان ، از مد handshaking ، ISO7816 (اسمارت کارت) ، modem ports (مودم سریال) ، پروتکل RS485 و IrDA Transceivers (پورت اینفرارد) نیز پشتیبانی می کند . لیست پین های مورد استفاده در این پروتکل را در زیر مشاهده می کنید :

نام پایه	Description	توضیح	Type	Active Level
SCK	Serial Clock	خروجی کلاک برای ارتباط همزمان	I/O	
TXD	Transmit Serial Data	خروجی داده	I/O	
RXD	Receive Serial Data	ورودی داده	Input	
RI	Ring Indicator	نشان دهنده حلقه	Input	Low
DSR	Data Set Ready	آمادگی مجموعه داده ها	Input	Low
DCD	Data Carrier Detect	تشخیص حامل	Input	Low
DTR	Data Terminal Ready	آمادگی ترمینال داده	Output	Low
CTS	Clear to Send	ترخیص به ارسال	Input	Low
RTS	Request to Send	تقاضای ارسال	Output	Low

هر کدام از پین های بالا برای کاربردی خاص استفاده می شوند ، مثلا پایه های TXD و RXD برای ایجاد ارتباط سریال غیر همزمان میان دو میکرو کنترلر به کار می رود ، در صورتی که بخواهید ارتباط به صورت همزمان باشد ، پایه SCK نیز باید استفاده شود .

در تصویر زیر ، بلوک دیاگرام داخلی واحد usart درمیکرو کنترلر های اتمل را مشاهده می کنید :



در این واحد بخش Peripheral DMA Controller برای کنترل کردن ارسال و دریافت ایفای نقش میکند و بخش AIC برای دریافت وقفه ی USART می باشد ، شما می توانید وقفه را برای دریافت یا ارسال داده یا تشخیص بیت های نقلی و کلاک و... فعال کنید .

واحد PMC میتواند با فعال کردن یا غیر فعال کردن کلاک واحد USART ، آن را راه اندازی کرده یا به صورت کامل از کار بیندازد ، USART نیز می تواند مانند دیگر بخش های میکرو کنترلر در انواع مد های کم مصرفی پیکربندی شود . شما قبل از کار با واحد usart و مقدار دهی رجیستر های آن باید واحد PMC را برای تولید کلاک مورد نیاز فعال سازید .

در نهایت این واحد از طریق باس APB با CPU و سایر واحد های جانبی ارتباط برقرار می کند .

بررسی رجیستر ها

در این بخش به بررسی برخی از رجیستر های تعیین شده برای واحد USART پرداخته ایم ، نحوه ی پیکربندی بخش های مختلف در میان مطالب مربوط به رجیستر ها آورده شده است . شما میتوانید سایر رجیستر ها را در صفحات 332 تا 351 AT91SAM7X512/256/128 Preliminary مشاهده کنید .

- رجیستر US_CR (USART Control Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	RTSDIS	RTSEN	DTRDIS	DTREN
15	14	13	12	11	10	9	8
RETTO	RSTNACK	RSTIT	SENDA	STTTO	STPBRK	STTBRK	RSTSTA
7	6	5	4	3	2	1	0
TXDIS	TXEN	RXDIS	RXEN	RSTTX	RSTRX	–	–

رجیستر US_CR عموما برای فعال یا غیر فعال کردن عملیات های ارسال و دریافت داده استفاده میشود . با مقدار دهی این رجیستر میتوانید پایه های مربوط به واحد usart را از حالت I/O عمومی به ورودی و خروجی داده ی سریال تبدیل کنید :

بیت RSTRX (Reset Receiver):

0: بدون اثر

1: بافر دریافت داده ریست میشود .

بیت RSTTX (Reset Transmitter) :

0: بدون اثر

1: بافر ارسال داده ریست میشود .

بیت RXEN (Receiver Enable):

0: بدون اثر

1: در صورتی که بیت RXDIS صفر باشد ، usart میتواند داده ی موجود بر روی باس را دریافت کند

بیت RXDIS (Receiver Disable):

0: بدون اثر

1: دریافت داده غیر فعال میشود .

بیت TXEN (Transmitter Enable) :

0: بدون اثر

1: در صورتی که بیت TXDIS صفر باشد ، usart میتواند داده ی خود را به باس ارسال کند

بیت TXDIS (Transmitter Disable) :

0: بدون اثر

1: ارسال داده غیر فعال میشود

بیت RSTSTA (Reset Status Bits):

0: بدون اثر

1: باعث ریست شدن بیت های PARE, FRAME, OVRE, و RXBRK در رجیستر US_CSR می گردد .

بیت STTBRK (Start Break):

0: بدون اثر

1: Starts transmission of a break after the characters present in US_THR and the Transmit Shift Register have been transmitted.

No effect if a break is already being transmitted.

بیت STPBRK (Stop Break):

0: بدون اثر

1: Stops transmission of the break after a minimum of one character length and transmits a high level during 12-bit periods.

No effect if no break is being transmitted.

بیت STTTO (Start Time-out):

0: بدون اثر

1: Starts waiting for a character before clocking the time-out counter. Resets the status bit TIMEOUT in US_CSR.

بیت SENDA(Send Address) :

0: بدون اثر

1: In Multidrop Mode only, the next character written to the US_THR is sent with the address bit set.

بیت RSTIT (Reset Iterations):

0: بدون اثر

1: Resets ITERATION in US_CSR. No effect if the ISO7816 is not enabled.

بیت RSTNACK (Reset Non Acknowledge):

0: بدون اثر

1: بیت NACK در رجیستر US_CSR ریست میشود .

بیت RETTO (Rearm Time-out):

0: بدون اثر

1: Restart Time-out (از تاخیر زمانی چشم پوشی میشود)

بیت DTREN (Data Terminal Ready Enable) :

0: بدون اثر

1: پایه ی DTR فعال شده و به وضعیت صفر منطقی میرود .

بیت DTRDIS (Data Terminal Ready Disable):

0: بدون اثر

1: پایه ی DTR غیر فعال شده و به وضعیت یک منطقی میرود .

بیت RTSEN (Request to Send Enable):

0: بدون اثر

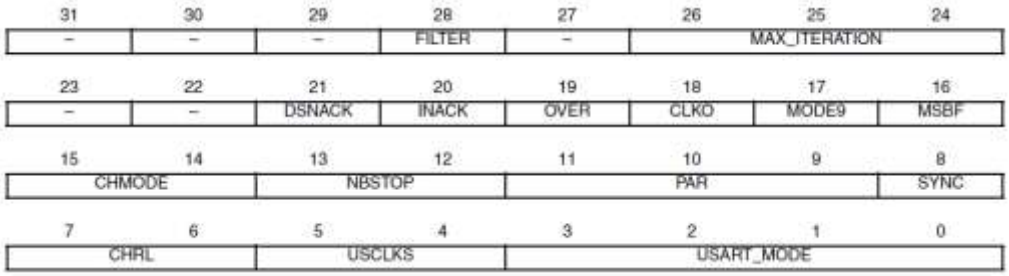
1: پایه ی RTS فعال شده و به وضعیت 0 منطقی میرود .

بیت RTSDIS(Request to Send Disable) :

0: بدون اثر

1: پایه ی RTS غیر فعال شده و به وضعیت یک منطقی میرود .

رجیستر US_MR (USART Mode Register):



با مقدار دهی بیت های این رجیستر میتوانید مد کاری واحد usart را مشخص کنید و ویژگیهای ارسال و دریافت داده را تعیین نمایید . در ادامه مد های مختلف این باس تشریح شده است .

بیت های USART_MODE :

مقدار بیت های USART_MODE				مد کاری	توضیحات
0	0	0	0	Normal	توضیحات شماره ی 1
0	0	0	1	RS485	توضیحات شماره ی 2
0	0	1	0	Hardware Handshaking	توضیحات شماره ی 3
0	0	1	1	Modem	توضیحات شماره ی 4
0	1	0	0	IS07816 Protocol: T = 0	توضیحات شماره ی 5
0	1	0	1	Reserved	توضیحات شماره ی 6
0	1	1	0	IS07816 Protocol: T = 1	توضیحات شماره ی 5
0	1	1	1	Reserved	توضیحات شماره ی 6
1	0	0	0	IrDA	توضیحات شماره ی 7
1	1	x		1 1 x x Reserved	توضیحات شماره ی 6

بیت های USCLKS (Clock Selection) :

با مقدار دهی این بیت ها مطابق جدول مقابل میتوانید منبع تامین کننده ی کلاک usart را مشخص نمایید :

USCLKS		Selected Clock
0	0	MCK
0	1	MCK/DIV (DIV = 8)
1	0	Reserved
1	1	SCK

توجه داشته باشید که مقدار کلاک وارد شده به usart در تعیین نرخ انتقال داده (Baud Rate) تاثیر مستقیم دارد . همان طور که در جدول مشاهده میکنید ، کلاک این باس میتواند به صورت مستقیم از کلاک اصلی سیستم ، یا کلاک اصلی سیستم / 8 یا پایه ی sck تامین شود .

بیت های CHRL (Character Length) :

واحد usart موجود در سری At91sam میتواند داده را در قالب های 5 و 6 و 7 و 8 بیتی ارسال و دریافت کند ، طول داده ی ارسالی با مقدار دهی این بیت ها و مطابق جدول مقابل تعیین میشود :

CHRL		Character Length
0	0	5 bits
0	1	6 bits
1	0	7 bits
1	1	8 bits

بیت SYNC(Synchronous Mode Select)

1:یوزارت در مد Synchronous فعال میشود .

0: usart در مد Asynchronous فعال میشود .

بیت های PAR (Parity Type):

بیت های توازن باعث تشخیص خطا در داده ی ارسالی و دریافتی میگردد ، شما میتوانید نوع بیت توازن را با مقدار دهی بیت های par مطابق جدول زیر مقایسه مشخص کنید :

PAR			Parity Type
0	0	0	Even parity
0	0	1	Odd parity
0	1	0	Parity forced to 0 (Space)
0	1	1	Parity forced to 1 (Mark)
1	0	x	No parity
1	1	x	Multidrop mode

بیت توازن یک بیت 0 یا 1 است که معمولا به انتهای داده ی ارسالی افزوده میشود . هنگامی که داده توسط گیرنده دریافت میشود ، ابتدا تعداد بیت های یک آن شمرده میشود . درصورتی که فرضا شما از بیت خطای زوج استفاده کرده باشید ، تعداد یک های موجود در داده باید زوج باشد ، در غیر اینصورت داده معیوب بوده و گیرنده باید مجددا آن را درخواست کند ، جدول زیر شما را در درک این مطلب یاری میکند :

Parity Bit Examples

Character	Hexa	Binary	Parity Bit	Parity Mode
A	0x41	0100 0001	1	Odd
A	0x41	0100 0001	0	Even
A	0x41	0100 0001	1	Mark
A	0x41	0100 0001	0	Space
A	0x41	0100 0001	None	None

بیت های NBSTOP(Number of Stop Bits) :

با مقدار دهی این بیت ها میتوانید ، تعداد بیت های STOP در هنگام پایان عملیات ارسال داده را مطابق جدول زیر تعیین کنید :

NBSTOP		Asynchronous (SYNC = 0)	Synchronous (SYNC = 1)
0	0	1 stop bit	1 stop bit
0	1	1.5 stop bits	Reserved
1	0	2 stop bits	2 stop bits
1	1	Reserved	Reserved

بیت های CHMODE (Channel Mode):

CHMODE		Mode Description
0	0	Normal Mode
0	1	Automatic Echo. Receiver input is connected to the TXD pin.
1	0	Local Loopback. Transmitter output is connected to the Receiver Input.
1	1	Remote Loopback. RXD pin is internally connected to the TXD pin.

توسط این بیت ها میتوانید وضعیت Echo و Loopback را برای واحد USART تعیین کنید ، در صورت فعال بودن Echo داده ی وارد شده به باس به صورت خودکار در پایه ی TXD ظاهر میشود . در حالت Local Loopback خروجی داده به ورودی داده متصل میشود .

بیت MSBF (Bit Order) :

0: Least Significant Bit is sent/received first.

1: Most Significant Bit is sent/received first.

بیت MODE9 (9-bit Character Length) :

0: CHRL defines character length.

1: 9-bit character length.

بیت CLKO (Clock Output Select) :

0: The USART does not drive the SCK pin.

1: The USART drives the SCK pin if USCLKS does not select the external clock SCK.

بیت OVER (Oversampling Mode) :

0: 16x Oversampling.

1: 8x Oversampling.

بیت INACK (Inhibit Non Acknowledge) :

0: The NACK is generated.

1: The NACK is not generated.

بیت DSNACK (Disable Successive NACK) :

0: NACK is sent on the ISO line as soon as a parity error occurs in the received character (unless INACK is set).

1: Successive parity errors are counted up to the value specified in the MAX_ITERATION field. These parity errors generate

a NACK on the ISO line. As soon as this value is reached, no additional NACK is sent on the ISO line. The flag ITERATION is asserted.

بیت MAX_ITERATION

Defines the maximum number of iterations in mode ISO7816, protocol T= 0.

بیت FILTER (Infrared Receive Line Filter) :

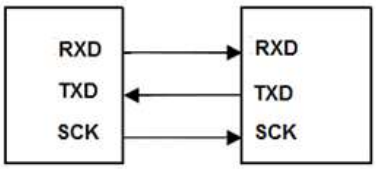
0: The USART does not filter the receive line.

1: The USART filters the receive line using a three-sample filter (1/16-bit clock) (2 over 3 majority).

❖ توضیحات شماره ی 1

هنگامی که usart در مد Normal راه اندازی میشود ، میتواند داده را به دو صورت synchronous یا Asynchronous که توسط بیت SYNC در رجیستر US_MR تعیین میشود ، ارسال و دریافت کند ، در مجله ی شماره ی 4 نحوه ی ارسال و دریافت داده به صورت سنکرون و آسنکرون به صورت مفصل بررسی شد .

در مد synchronous یا همزمان ، باید اتصالات زیر را میان دو سخت افزار موجود برقرار نمایید :

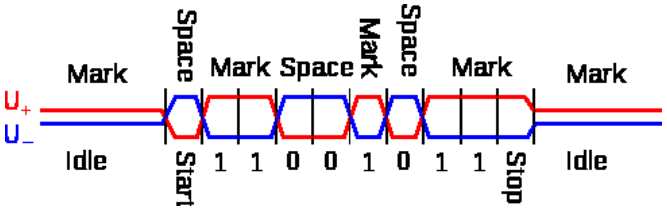


برای میکرو کنترلی که کلاک به آن وارد میشود ، باید بیت های USCLKS را با مقدار 11 جایگزین کنید ، تا کلاک باس USART این میکرو کنترلر از طریق پایه ی SCK و میکرو کنترلر دیگر تامین شود .

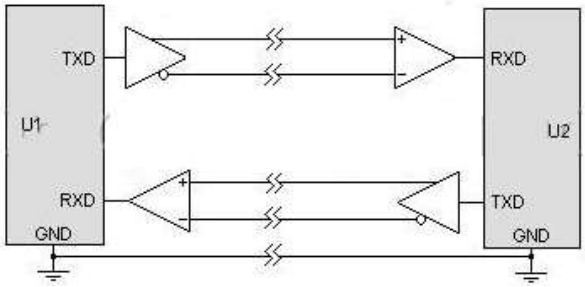
در مد Asynchronous یا غیر همزمان نیازی به اتصال پایه ی SCK وجود ندارد و داده از طریق دو پایه ی دیگر منتقل میشود .

❖ توضیحات شماره ی 2

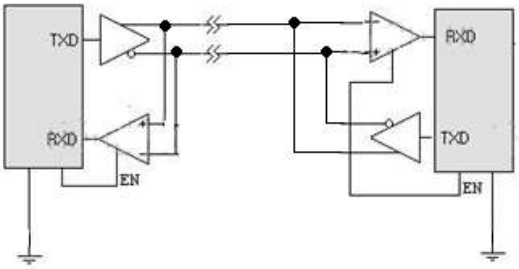
Rs485 یک پروتکل سریال برای انتقال داده میباشد ، این پروتکل که ارتقاع یافته ی rs232 است بیشتر در مقاصد صنعتی استفاده میشود و تشابه زیادی با rs232 دارد . در پروتکل rs485 از خطوط دیفرانسیلی استفاده میشود . در حالت دیفرانسیلی ، داده ی موجود از طریق دو خط ارسال میشود و در صورتی که نویزی در محیط وجود داشته باشد ، بر روی داده موجود در هر دو خط تاثیر گذاشته و در عمل نمیتواند آن را تغییر دهد :



در این حالت ما به 4 سیم برای انتقال داده نیاز خواهیم داشت ، دو سیم برای ارسال و دو سیم برای دریافت (پروتکل rs433) :



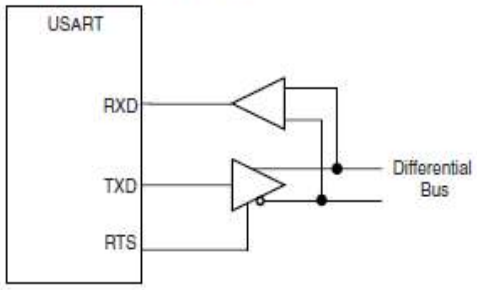
با حذف کردن دو سیم و سوییچ کردن ارسال و دریافت بر روی دو سیم دیگر پروتکل rs485 بوجود میاید .



در rs485 به دلیل استفاده از خطوط دیفرانسیلی میتوان فاصله ی دو وسیله را تا 1200 متر افزایش داد ، در این حالت حداکثر سرعت انتقال داده برابر با 100 کیلو بیت بر ثانیه است (در فاصله های کم (مثلا 10 متر) سرعت تا 3.5 مگابیت بر ثانیه افزایش میباید .

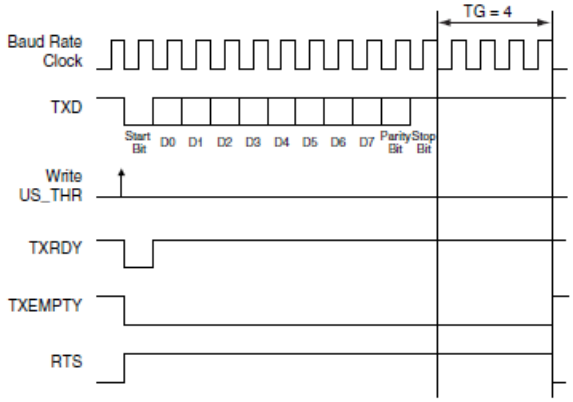
استفاده از واحد usart در سری at91sam به سادگی و با نوشتن عدد 0x1 به جای بیت های USART_MODE رجیستر US_MR میسر است . در این شرایط واحد usart در حالتی میان حالت سنکرون و آسنکرون راه اندازی میشود ، در این حالت پایه ی RTS در نقش پایه ی EN که در مدار بالا آورده شده ظاهر گشته و نقش سوییچ کردن مبدل در حالت های ارسال و دریافت را به عهده میگیرد:

Typical Connection to a RS485 Bus

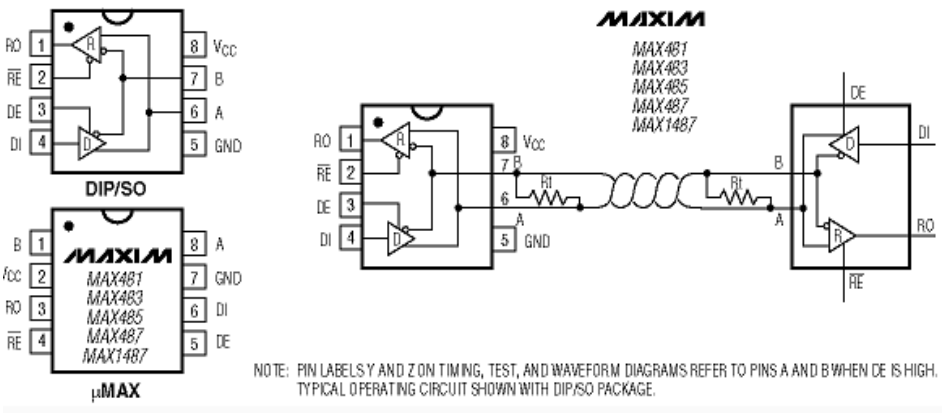


در حالت RS485 نیز مانند سایر حالت های قبلی ، بیت اول مربوط به شروع کار ، 8 بیت بعدی داده ی موجود روی باس و بیت اخر ، بیت تشخیص خطا میباشد :

Example of RTS Drive with Timeguard

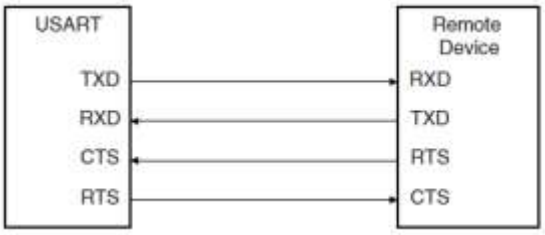


برای درایو کردن پورت RS485 قطعات مختلفی ارائه شده است که در اینجا میتوانیم به تراشه های MAX481, MAX483, MAX485 , MAX487 , MAX1487 اشاره کنیم .

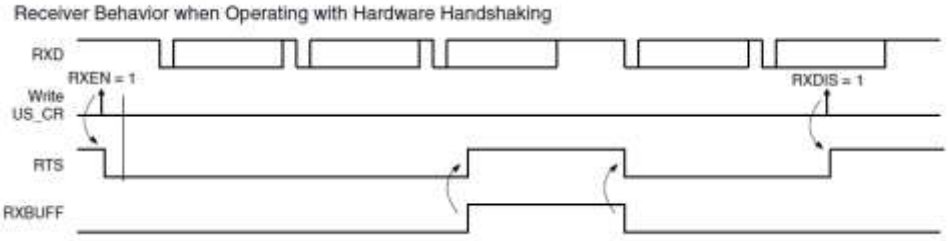


❖ توضیحات شماره ی 3 :

با نوشتن عدد 0x2 به جای بیت های USART_MODE رجیستر US_MR واحد usart در مد handshaking (دست تکانی) راه اندازی میشود ، در این مد عملیات انتقال داده با کنترل کردن سیگنال های RTS و CTS انجام میشود ، در این حالت پایه های RTS و CTS نیز در کنار پین های TXD و RXD برای کنترل و انتقال داده به کار میروند :

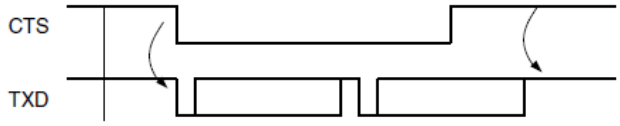


در حالت عادی و هنگامی که پایه ی CTS دستگاه کنترل شده در وضعیت منطقی یک قرار دارد (این وضعیت توسط پایه ی RTS یوزارت بوجود آمده است) ، دستگاه کنترل شده نمیتواند داده ای را به میکرو کنترلر ارسال کند ، هنگامی که پایه ی RTS پورت USART صفر میشود ، عملیات انتقال داده آغاز میشود .



حالت بالا در هنگام انتقال داده از باس usart به دستگاه کنترل شده نیز صادق است ، یعنی میکرو کنترلر فقط در حالتی قادر به ارسال داده است ، که پایه ی CTS آن توسط پایه ی RTS دستگاه کنترل شده صفر شود .

Transmitter Behavior when Operating with Hardware Handshaking



❖ توضیحات شماره ی 4

پورت سریال در سال 1967 و برای ارتباط کامپیوتر با مودم ساخته شد و بعد از گذشت زمان و تکامل پروتکل های ارتباطی ، مواردی همچون پروتکل RS232 و مد Handshaking به آن اضافه گردید (البته در میکرو کنترلر به دلیل نیاز بیشتر ، پروتکل های همچون RS485 ، IRDA و... نیزبه آن پورت اضافه شد) .

در این بخش منظور از مودم وسیله ای است که میتواند داده یا فرمان موجود در محیط اطراف را به داده ی قابل فهم کامپیوتر تبدیل کند .

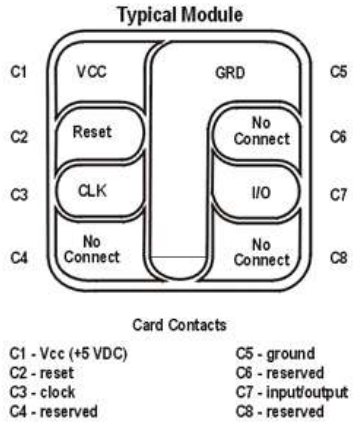
در Modem Mode از پایه های زیر جهت اتصال میکرو کنترلر به کامپیوتر استفاده میشود ، همچنین میکرو با نوشتن عدد 0x3 به جای بیت های USART_MODE رجیستر US_MR در این مد پیکربندی میگردد .

USART Pin	V24	CCITT	Direction
TXD	2	103	From terminal to modem
RTS	4	105	From terminal to modem
DTR	20	108.2	From terminal to modem
RXD	3	104	From modem to terminal
CTS	5	106	From terminal to modem
DSR	6	107	From terminal to modem
DCD	8	109	From terminal to modem
RI	22	125	From terminal to modem

در این مد ، یک ارتباط کاملا ایمن و کنترل شده میان میکرو کنترلر و کامپیوتر بوجود میاید . که شما میتوانید توسط این رابط انواع داده و فرمان را به کامپیوتر ارسال کنید ، جهت اطلاعات بیشتر میتوانید به راهنمایی هایپر ترمینال ویندوز مراجعه نمایید .

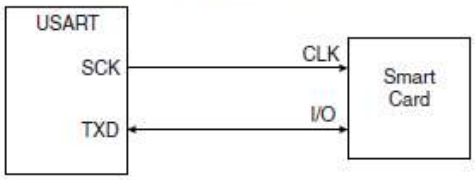
❖ توضیحات شماره ی 5

ISO7816 پروتکلی برای انتقال داده به صورت سریال است که یک از موارد استفاده ان در کارت های هوشمند یا smart cards میباشد . در این پروتکل با استفاده از دو خط کلاک و داده میتوان محتویات موجود در حافظه ی کارت را خواند و نصبت به آن عملیات مناسب را انجام داد .

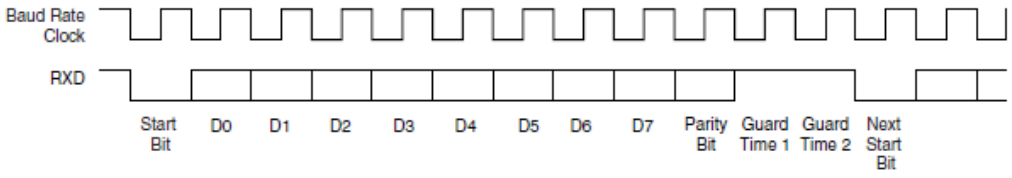


در سری at91sam ، با نوشتن مقادیر 0x4 و 0x5 به جای بیت های USART_MODE رجیستر US_MR میتوان واحد usart را برای راه اندازی این کارت ها پیکربندی کرد . مقادیر 0x4 و 0x5 به ترتیب برای راه اندازی کارت های نوع t0 و نوع t1 به کار میروند

Connection of a Smart Card to the USART



همانگونه که در تصویر بالا مشخص است ، برای اتصال کارت به میکرو به دو پایه ی SCK و TXD نیاز داریم ، پایه ی SCK وظیفه ی تایمین کلاک مورد نیاز کارات را بر عهده دارد ، همچنین پایه ی TXD در ابتدا داده ای را به کارت منتقل کرده و بعد از فعال شدن کارت ، اطلاعات موجود در آن را به میکرو منتقل میکند .



در تصویر بالا ، اطلاعاتی که از طرف کارت به میکرو میرود ، اورده شده است .

برای راه اندازی واحد USART در مد ISO7816 علاوه مقدار دهی رجیستر US_MR ، باید رجیستر های US_NER و US_CR و US_MR را نیز مقدار دهی نمایید .

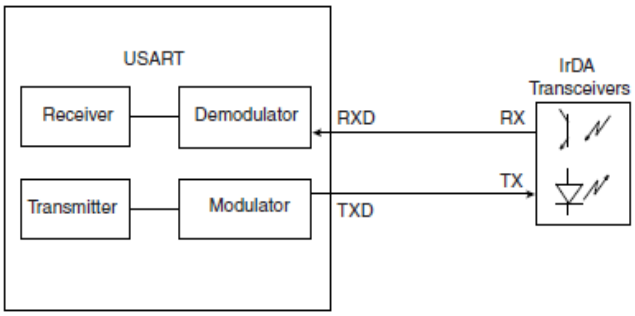
❖ توضیحات شماره ی 6

این وضعیت برای حالت های بعدی رزور شده است و کاربر نیمتواند آنها را مقدار دهی نمایید .

❖ توضیحات شماره ی 7

احتمالا تا کنون پورت Infrared یا به اصطلاح مادون قرمز را بر روی گوشی موبایل خود دیده اید ، این بخش از یک فرستنده / گیرنده مادون قرمز تشکیل شده است و شما میتوانید توسط آن هر چیزی را از گوشی خود به گوشی دیگر منتقل کنید .

Led های فرستنده/ گیرنده موجود در این بخش ، در یک بسته بندی قرار گرفته اند (یک قطعه مجزا) و اتصال آنها به میکرو کنترلر مانند شکل زیر است (یک نمونه مشهور از این قطعات سری tfd4xxx میباشند ، توجه داشته باشید که استفاده از led های مادون قرمز معمولی به جایی این قطعه باعث کاهش کیفت داده ی ارسالی و بالا رفتن خطا میشود .



این قطعه داده ی خروجی از باس usart میکرو کنترلر فرستنده (که متناسب با ورودی قطعه پیکرپندی شده است) را دریافت کرده (داده ی خروجی میکرو بر حسب استاندارد IrDA version 1.1 با یک پالس دیگر مدوله میشود)و آن را از طریق فرستنده ی مادون قرمز در محیط اطراف منتشر میکند .

در آن طرف گیرنده داده منتشر شده را دریافت کرده و آن را به میکرو تحویل میدهد و میکرو بعد از کدگشایی داده ، آن را به بخش های بعدی که در برنامه معین میشوند ، ارسال میکند .

کار با بخش IrDA بسیار آسان و تقریبا شبیه به کار با واحد usart است ، برای راه اندازی این بخش کافی است کار های زیر را انجام دهید :

- برای ارسال داده ، واحد usart را راه اندازی کنید ، مقدار نرخ انتقال داده در ادامه اورده شده است . شما میتوانید از usart0 یا usart1 برای کار با این پروتکل استفاده کنید (امکان اتصال دو قطعه به صورت همزمان به یک میکرو وجود دارد ، هر قطعه به یکی از واحد usart متصل میشود)
- در پروتکل IrDA® Infrared قطعه در یک زمان نمیتواند هم داده را ارسال و هم دریافت کند ، به همین دلیل ابتدا باید پایه ی tx را به صفر منطقی ببرید تا led فرستنده خاموش شود ، سپس بافر ورودی را مدام چک کنید تا اگر دستگاهی قصد ارسال داده به میکرو کنترلر را داشت ، میکرو متوجه شده و داده را دریافت کند .
- سرعت انتقال داده در این پروتکل از 2.4 Kb/s تا 115.2 Kb/s میباشد . همچنین برای راه اندازی واحد usart در این مد کافی است قبل از هر چیز ، مقدار 0x8 را به جای بیت های USART_MODE در ریسجتر US_MR قرار دهید تا بخش های مدولاتور و دمولاتور فعال شوند .در این حالت داده خروجی usart به صورت خودکار به داده ی سازگار با پروتکل irda تبدیل میشود .

رجیستر US_CSR (USART Channel Status Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
CTS	DCD	DSR	RI	CTSIC	DCDIC	DSRIC	RIIC
15	14	13	12	11	10	9	8
–	–	NACK	RXBUFF	TXBUFE	ITERATION	TXEMPTY	TIMEOUT
7	6	5	4	3	2	1	0
PARE	FRAME	OVRE	ENDTX	ENDRX	RXBRK	TXRDY	RXRDY

رجیستر US_CSR وضعیت بیت های رجیسترهای مختلف واحد usart را در خود ذخیره میکند ،شما میتوانید با خواندن این بیت ها از فعال / غیر فعال بودن بخش های مختلف اگاه شده و عملیات های متناظر را انجام دهید :

• RXRDY: Receiver Ready

0: No complete character has been received since the last read of US_RHR or the receiver is disabled. If characters were being received when the receiver was disabled, RXRDY changes to 1 when the receiver is enabled.

1: At least one complete character has been received and US_RHR has not yet been read.

• TXRDY: Transmitter Ready

0: A character is in the US_THR waiting to be transferred to the Transmit Shift Register, or an STTBRK command has been requested, or the transmitter is disabled. As soon as the transmitter is enabled, TXRDY becomes 1.

1: There is no character in the US_THR.

• RXBRK: Break Received/End of Break

0: No Break received or End of Break detected since the last RSTSTA.

1: Break Received or End of Break detected since the last RSTSTA.

• ENDRX: End of Receiver Transfer

0: The End of Transfer signal from the Receive PDC channel is inactive.

1: The End of Transfer signal from the Receive PDC channel is active.

• ENDTX: End of Transmitter Transfer

0: The End of Transfer signal from the Transmit PDC channel is inactive.

1: The End of Transfer signal from the Transmit PDC channel is active.

• OVRE: Overrun Error

0: No overrun error has occurred since the last RSTSTA.

1: At least one overrun error has occurred since the last RSTSTA.

• FRAME: Framing Error

0: No stop bit has been detected low since the last RSTSTA.

1: At least one stop bit has been detected low since the last RSTSTA.

• PARE: Parity Error

0: No parity error has been detected since the last RSTSTA.

1: At least one parity error has been detected since the last RSTSTA.

• TIMEOUT: Receiver Time-out

0: There has not been a time-out since the last Start Time-out command (STTTO in US_CR) or the Time-out Register is 0.

1: There has been a time-out since the last Start Time-out command (STTTO in US_CR).

• TXEMPTY: Transmitter Empty

0: There are characters in either US_THR or the Transmit Shift Register, or the transmitter is disabled.

1: There are no characters in US_THR, nor in the Transmit Shift Register.

• ITERATION: Max number of Repetitions Reached

0: Maximum number of repetitions has not been reached since the last RSIT.

1: Maximum number of repetitions has been reached since the last RSIT.

• TXBUFE: Transmission Buffer Empty

0: The signal Buffer Empty from the Transmit PDC channel is inactive.

1: The signal Buffer Empty from the Transmit PDC channel is active.

• RXBUFF: Reception Buffer Full

0: The signal Buffer Full from the Receive PDC channel is inactive.

1: The signal Buffer Full from the Receive PDC channel is active.

• NACK: Non Acknowledge

0: No Non Acknowledge has not been detected since the last RSTNACK.

1: At least one Non Acknowledge has been detected since the last RSTNACK.

• RIIC: Ring Indicator Input Change Flag

0: No input change has been detected on the RI pin since the last read of US_CSR.

1: At least one input change has been detected on the RI pin since the last read of US_CSR.

• DSRIC: Data Set Ready Input Change Flag

0: No input change has been detected on the DSR pin since the last read of US_CSR.

1: At least one input change has been detected on the DSR pin since the last read of US_CSR.

• DCDIC: Data Carrier Detect Input Change Flag

0: No input change has been detected on the DCD pin since the last read of US_CSR.

1: At least one input change has been detected on the DCD pin since the last read of US_CSR.

• CTSIC: Clear to Send Input Change Flag

0: No input change has been detected on the CTS pin since the last read of US_CSR.

1: At least one input change has been detected on the CTS pin since the last read of US_CSR.

• RI: Image of RI Input

0: RI is at 0.

1: RI is at 1.

• DSR: Image of DSR Input

0: DSR is at 0

1: DSR is at 1.

• DCD: Image of DCD Input

0: DCD is at 0.

1: DCD is at 1.

• CTS: Image of CTS Input

0: CTS is at 0.

1: CTS is at 1.

رجیستر US_RHR (USART Receive Holding Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
RXSYNH	–	–	–	–	–	–	RXCHR
7	6	5	4	3	2	1	0
RXCHR							

داده ی دریافتی از باس USART در بیت های RXCHR این رجیستر ذخیره میشود . توجه داشته باشید که بیت نهم Parity Bit میباشد .

بیت RXSYNH (Received Sync):

0: آخرین کارکتر دریافت شده ، داده میباشد .

1: آخرین کارکتر دریافت شده یک دستور است .

رجیستر US_THR (USART Transmit Holding Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
TXSYNH	–	–	–	–	–	–	TXCHR
7	6	5	4	3	2	1	0
TXCHR							

داد ه ای که باید به پورت سریال ارسال شود ، در این رجیستر قرار می گیرد . توجه داشته باشید که بیت شماره ی 8 این رجیستر برای Parity Bit رزور شده و کاربر نمیتواند در آن چیزی بنویسد ، برای ارسال داده ی بیشتر از 8 بیت ، ان را دستور ات شیفت بشکنید .

بیت TXSYNH (Sync Field to be transmitted) :

0 : کارکتری بعدی که قرار است ارسال شود ، داده میباشد .

1: کارکتری بعدی که قرار است ارسال شود ، دستور است .

رجیستر US_BRGR (USART Baud Rate Generator Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	FP		
15	14	13	12	11	10	9	8
CD							
7	6	5	4	3	2	1	0
CD							

شاید یکی از مهم ترین بخش های راه اندازی واحد USART مقدار دهی این رجیستر باشد ، Baud Rate یا نرخ انتقال داده یکی از عوامل مهم در پایداری ارتباط سریال میباشد ، در صورتی که مقادیر این رجیستر به درستی انتخاب نشوند ، عملیات انتقال داده با شکست روبرو خواهد شد .

Baud Rate در مد Asynchronous :

هنگامی که واحد USART در مد آسنکرون برنامه ریزی میشود ، منبع کلاک تعیین شده توسط بیت های USCLKS رجیستر US_MR ابتدا به مقدار بیت های CD که در رجیستر US_BRGR مقدار دهی شده است تسیم شده و سپس به مقادیر 8 یا 16 که

توسط بیت OVER در رجیستر US_MR مشخص شده تقسیم میگردد . رابطه ی زیر برای بدست اوردن *Baud Rate در این مد به کار میرود :*

$$Baud\ Rate = \frac{Selected\ Clock}{(8(2-over)CD)}$$

در رابطه ی بالا over ، همهن بیت over در رجیستر US_MR است که میتواند مقدار 0 یا 1 داشته باشد .

مقدار خطای انتقالی USART از رابطه ی زیر محاسبه میشود ، در صورتی که مقدار خطا از 5% بیشتر باشد ، باس کار نخواهد کرد . جدول 2-30 صفحه ی 302 دیتا شیت ، شما را در انتخاب مقادیر مناسب یاری خواهد نمود :

Error=1-(ExpectedBaudRate /ActualBaudRate)

Baud Rate در مد Synchronous :

در این مد مقدار Baud Rate از رابطه ی زیر محاسبه میشود :

BaudRate = SelectedClock / CD

در صورتی که منبع کلاک تعیین شده توسط بیت های USCLKS رجیستر US_MR خروجی پین SCK باشد ، مقدار دهی این رجیستر بی اثر است و *Baud Rate معتبر نمیباشد .*

خلاصه ی نوشته های بالا در جدول زیر آورده شده است :

CD	USART_MODE ≠ ISO7816			USART_MODE = ISO7816
	SYNC = 0		SYNC = 1	
	OVER = 0	OVER = 1		
0	Baud Rate Clock Disabled			
1 to 65535	Baud Rate = Selected Clock/16/CD	Baud Rate = Selected Clock/8/CD	Baud Rate = Selected Clock /CD	Baud Rate = Selected Clock/CD/FI_DI_RATIO

نحوه ی محاسبه ی Baud Rate در مد ISO7816 در ادامه آورده شده است .

بیت های FP (Fractional Part) این بیت ها مقدار دقت Baudrate را تعیین میکند: .Baudrate resolution=FP x 1/8

رجیستر US_RTOR (USART Receiver Time-out Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
TO							
7	6	5	4	3	2	1	0
TO							

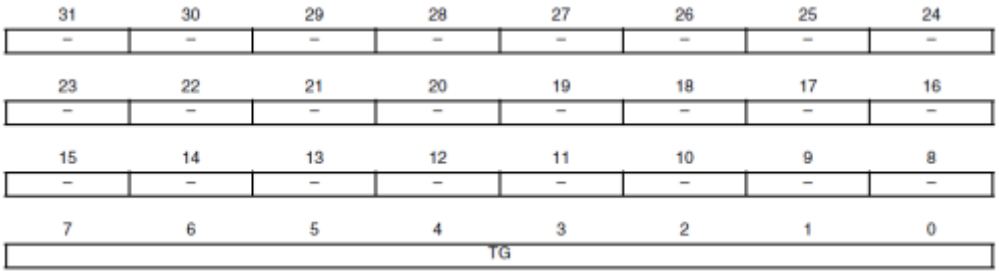
توسط این رجیستر میتوانید ، میتوانید اتصالات معیوب به پایه ی RXD میکرو کنترلر مورد نظر خود را تشخیص دهید

در صورتیکه به جای بیت های TO رقم صفر قرار دهید ، Time-out غیر فعال میشود ، قرار دادن مقادیر 1-65535 (به صورت هگز یا باینری) به جای این بیت ها باعث میشود که در هر دوره ی چک کردن بافر داده ی ورودی (رجیستر US_RHR) میکرو به اندازه ی زمان TO * Bit Period منتظربماند ، در صورتی که با صفر شدن شمارنده (زمان صفر شود) داده ای به میکرو وارد نشود ، بیت TIMEOUT در رجیستر US_CSR (USART Channel Status Register) یک میشود و شما میتوانید با چک کردن آن از عدم ارسال داده توسط وسیله ی دیگر آگاه شوید (با وارد شدن داده ، تایمر ریست شده و شمارش مجددا آغاز میشود) و یکی از موراد زیر را انجام دهید :

- کلاک مخصوص این تایمر را با یک کردن بیت STTTO رجیستر US_CR متوقف کنید ، در این حالت تایمر تا ورود یک داده ی جدید منتظر مانده و بعد از کار خود را شروع میکند ، این مورد برای حالتی که زمان شروع ارسال داده توسط دستگاه دیگر مشخص نیست ، استفاده میشود .
- بیت RETTO رجیستر US_CR را یک کنید ، در این حالت Time-out همیشه فعال است و با آمدن هر کارکتر جدید مجددا مقدار دهی میشود در صورتی که تا صفر شدن شمارنده داده ای به میکرو وارد نشود ،بیت TIMEOUT رجیستر US_CSR یک شده و شما میتوانید با چک کردن آن عملیات دلخواه را انجام دهید . این حالت برای دستگاه های که به ارتباطی پیوسته دارند استفاده میشود .

جدول مربوط به زمان بندی این تایمر در جدول 8-30 صفحه ی 319 دیتا شیت آورده شده است .

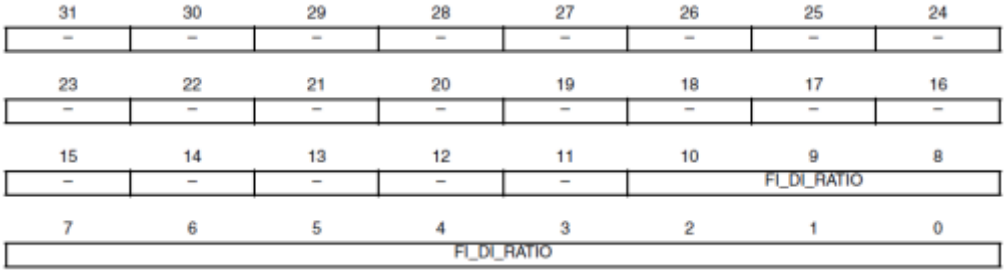
رجیستر US_TTGR (USART Transmitter Timeguard Register):



این رجیستر به شما این امکان را میدهد ، که میان دو کارکتر ارسالی یک تاخیر زمانی را ایجاد نمایید ، این حالت معمولا در هنگام ارسال داده به دستگاه های که ارتباط پیوسته ندارند یا سرعت پردازش آنها کم است استفاده میشود .

عمل کردن این تاخیر مانند یک stop bit طولانی است ، با قردادن رقم صفر به جای بیت ها TG این وضعیت غیر فعال میشود ، مقادیر 1تا 255 باعث ایجاد تاخیر TG x Bit Period خواهد شد .

مقادیر Timeguard و Bit Period در جدول شماره ی 7-30 صفحه ی 318 دیتا شیت آورده شده است .



رجیستر US_FIDI (USART FI DI RATIO Register) :

در پروتکل ISO7816 باید مقدار bit rate (فرکانس خروجی SCK) را با مقدار دهی بیت های FI_DI_RATIO (FI Over DI Ratio Value) تعیین کنیم . این مقدار از رابطه ی زیر بدست میاید :

bit rate=(bit-rate adjustment factor / clock frequency division factor) * ISO7816 clock frequency

مقادیر مجاز bit-rate adjustment factor در جدول زیر آورده شده است :

DI field	0001	0010	0011	0100	0101	0110	1000	1001
Di (decimal)	1	2	4	8	16	32	12	20

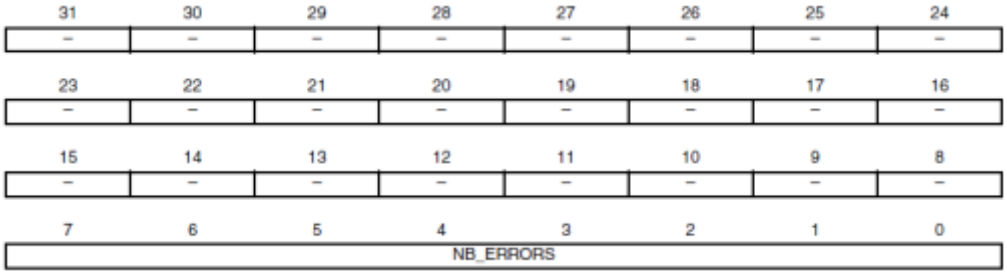
مقادیر مجاز clock frequency division factor در جدول زیر آورده شده است :

FI field	0000	0001	0010	0011	0100	0101	0110	1001	1010	1011	1100	1101
Fi (decimal	372	372	558	744	1116	1488	1860	512	768	1024	1536	2048

هنگامی که باس USART در مد ISO7816 پیکربندی میشود ، منبع کلاک تعیین شده توسط بیت های USCLKS رجیستر US_MR ابتدا به مقدار بیت های CD که در رجیستر US_BRGR مقدار دهی شده است ، تقسیم میگردد . در این حالت یک پالس کلاک بر روی پایه ی SCK جهت تامین کردن پالس همزمانی کارت ایجاد میشود . با تقسیم شدن مقدار بدست آمده به مقدار بیت های FI_DI_RATIO در رجیستر US_FIDI مقدار ISO7816 clock frequency برای رابطه ی بالا بدست می آید .

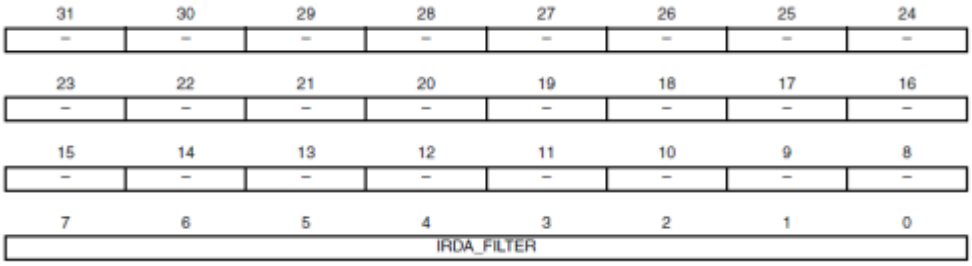
توجه داشته باشید که نرخ انتقال کم باعث کم شردن سرعت و بالا رفتن دقت و نرخ انتقال داده زیاد باعث افزایش سرعت و کم شدن دقت میگردد .

رجیستر US_NER (USART Number of Errors Register) :



بیت های NB_ERRORS (Number of Errors): تعداد خطای رخ داده در هنگام انتقال داده ی ISO7816 در این رجیستر ذخیره میشود ، مقدار این رجیستر بعد از خوانده شدن پاک میشود .

رجیستر US_IF (USART IrDA FILTER Register) :



بیت های IRDA_FILTER (IrDA Filter): با مقدار دهی این بیت ها میتوانید میزان دقت IrDA demodulator filter را مشخص کنید .

برای واحد USART سه رجیستر US_IMR (Interrupt Mask Register) و US_IER (Interrupt Enable Register) و US_IDR (Interrupt (Disable Register جهت فعال سازی و استفاده از وقفه ی بخش های مختلف در نظر گرفته شده است که ما بررسی آنها را به بخش های بعدی و بعد از تشریح واحد AIC موکول خواهیم کرد .

کتابخانه ی usart.h :

به دلیل گسترده بودن رجیستر های تعیین شده برای این واحد و نیاز به مقدار دهی آنها برای ارسال و دریافت داده ، کار با واحد usart آنهم فقط با یک هدر ، اندکی دشوار به نظر میرسد . در این بخش ما مجموعه دستورات مورد نیاز برای راه اندازی این باس در مد های سنکرون و آسنکرون را در یک فایل هدر جمع آوری نموده ایم ، شما میتوانید با مطالعه ی این فایل با نحوه ی مقدار دهی رجیستر ها و پیکربندی هر یک آشنا شده و سایر بخش ها را به توجه به آن راه اندازی نمایید .

قبل از باز کردن نرم افزار keil فایل usart.h را در مسیر زیر کپی کنید :

Program Files\Keil\ARM\INC\Atmel\ SAM7X

برای استفاده از این کتابخانه باید آن را در برنامه فواخوانی کنید :

```
#include "usart.h"
```

با فرا خوانی کتابخانه دستورات زیر به مجموعه دستورات keil افزوده می شود :

```
usartX_init(mode , Baud_Rate);
```

در این دستور MODE ، مد کاری واحد USART میباشد و به جای Baud_Rate مقدار نرخ انتقال داده که می تواند 1200 یا 2400یا 9600 یا سایر مقادیر استاندارد باشد قرار می گرد . همچنین X میتواند 0 برای واحد USART0 و 1 برای واحد USART1 باشد . با این کتابخانه واحد USART میتواند در مد های سنکرون (sync) و آسنکرون (Async) راه اندازی شود . مثال :

```
usart0_init(Async, 9600);
```

در این دستور واحد USART0 در مد آسنکرون با باود 9600 راه اندازی شده است .

توجه :

- ✓ برای مد سنکرون (در حالتی که کلاک از پایه ی sck تامین میشود) ، مقدار Baud_Rate میتواند هر چیزی باشد ، چون برای این حالت Baud_Rate تعریف نشده و عدد موجود فقط جهت تکمیل دستور است .
- ✓ در این هدر ، مواردی مانند بیت های خطا ، منبع تامین کلاک ، Channel Mode و.. در حالت پیش فرض تنظیم شده و دستور خاصی جهت دست یابی کاربر به آنها ایجاد نشده است ، در صورتی که قصد دارید این تنظیمات را تغییر دهید ، فایل هدر را ویرایش نمایید .

```
#define M_crystal 18432000
```

- ✓ این دستور در پنجمین خط فایل هدر موجود میباشد ، در این دستور 18432000 مقدار فرکانس کاری میکرو کنترلر یا همان کریستال است ، در صورتی فرکانس کریستال توسط واحد pll در مقادیر دلخواه ضرب میشود ، شما باید حاصل نهایی را به جای آن بنویسید .

اکنون شما می توانید با استفاده از دستورات زیر داده را به پورت ارسال یا از آن دریافت کنید :

```
printf ( ..... );
```

با این دستور می توانید یک کارکتر یا عدد یا رشته یا .... را به پورت سریال ارسال کنید ، این دستور به فرم های های مختلفی استفاده می شود ، در زیر فرم های مختلف در مثال های گوناگون اورده شده است :

```
printf ( "string");
```

با این دستور رشته ی string به پورت سریال ارسال می شود ، رشته ی string می تواند مجموعه ای از اعداد وحروف که در می آن دو عدد " قرار گرفته اند باشد

```
printf ("%d ",x);
```

با این دستور متغیر یا عدد ثابت x به پورت سریال ارسال می شود ، به جای d می توان از نماد های دیگر همچون c (برای ارسال یک متغیر از نوع char) یا s (برای ارسال یک رشته) یا .... استفاده نمود)

```
printf ("%d cubed = %d",n,(n*n));
```

با دستور بالا ابتدا متغیر n که از نوع int است به پورت ارسال می شود ، سپس عبارت cubed = ارسا می شود و در نهایت حاصل n*n ارسال خواهد شد ، یعنی شما در گیرنده با فرض اینکه مقدار اولیه n برابر با 3 باشد ، عبارت های زیر را دریافت خواهید کرد :

```
3 cubed = 9
```

کارکتر های خط بعد و فاصله : در دستور print دو کاکتر \n و \t به ترتیب برای رفتن به خط بعد (enter) و فاصله (8بار space) در نظر گرفته شده اند . برای درک بهتر موضوع مثال را ببنید .

```
sendchar (char);
```

با استفاده از دستور sendchar می توانید یک داده یا متغیر از نوع char را به پورت usart ارسال کنید . char باید یک حرف یا کارکتر \n باشد .

دریافت اطلاعات از پورت سریال :

```
scanf("%s",&x);
```

از دستور scanf برای دریات داده از پورت سریال استفاده می شود ، در این دستور نیز مانند دستور printf باید نوع متغیر دریافتی مشخص شود .

به جای s می توانید از نماد های دیگر همچون c (برای دریافت یک متغیر از نوع char) یا d (برای دریافت یک متغیر از نوع عدد صحیح) یا .... استفاده نمایید .

```
x= getkey ();
```

با استفاده از دستور getkey می توان داده موجود بر روی پورت usart را در یافت کرد و در متغیر مناسب مثالا x قرار داد .

دستور getkey قادر به دریافت یک رقم یا یک حرف می باشد .

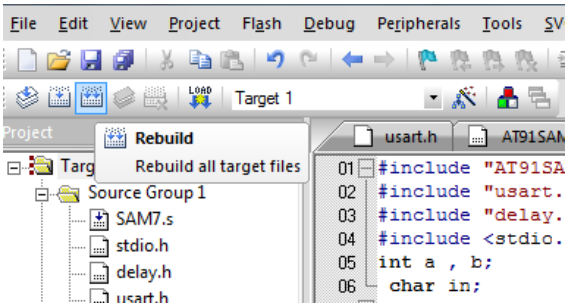
پروژه : شبیه سازی پورت سریال در keil :

در این مرحله قصد داریم برنامه ی زیر را در نرم افزار keil شبیه سازی کنیم :

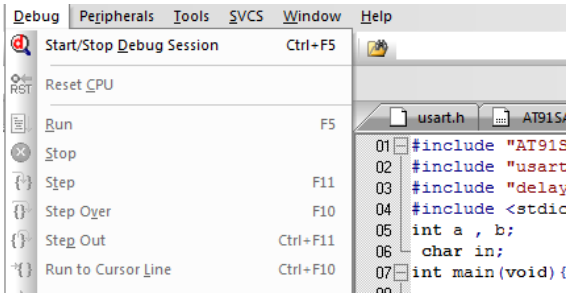
```
#include "AT91SAM7x256.h"
#include "usart.h"
#include "delay.h"
int a , b;
char inPUT;
```

```
int main( void ){
 usart0_init(Async, 9600);
printf("1234567\t");
while(1){
printf("please wait...\n");
delay_s(1);
++a;
printf("press a key\n");
scanf("%c",&inPUT);
printf("a is: %d ****You have entered: %c\n",a,inPUT);
printf("press a key\n");
b=getkey();
sendchar (b);
} }
```

قبلا با مراحل شبیه سازی برنامه آشنا شدیم و بخش های همچون پورت ها ، adc ، تایمر واچ داگ و.... را با شبیه ساز kiel آزمایش کردیم ، در اینجا نیز مطابق مراحل قبل برنامه را کامپایل کنید :

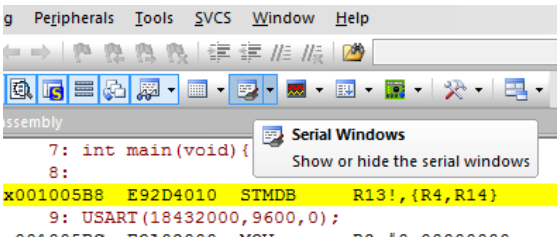


برای کامپایل کردن برنامه ، گزینه ی Rebuild را از منو بار انتخاب کنید ، این گزینه در منو ی projectنیز موجود می باشد . بعد از کامپایل کردن برنامه از منوی debug گزینه ی start/stop debug session را انتخاب کنید تا وارد محیط شبیه سازی شوید .



برای شبیه سازی پورت usart می توآنید از ترمی نال برای مشاهده ی داده ارسالی و دریافتی و از بخش Peripherals برای مشاهده ی وضعیت ریسجتر های usart استفاده کنید .

برای مشاهده ی داده های ارسالی و دریافتی از تولبار اجرا گزینه ی serial windows را انتخاب کنید .



مشاهده می کنید که در پایین نرم افزار پنجره ای به نام uart #1 باز می شود ،در این پنجره می توانید داده ارسالی و دریافتی به پورت usart0 را مشاهده کنید :

در تولبار اجرا گزینه ی run را انتخاب کنید ، و در وسط پنجره ی uart #1 کلیک نمایید :

با اجرا شدن دستور :

```
printf("123456\t");
```

عبارت 123456به پورت ارسال می شود ، بنابراین این عبارت در پنجره ی uart #1 به نمایش در می آید ، کارکتر \t باعث ایجاد فاصله به اندازی 5 کاکتر می شود .

دستور :

```
printf("please wait...\n");
```

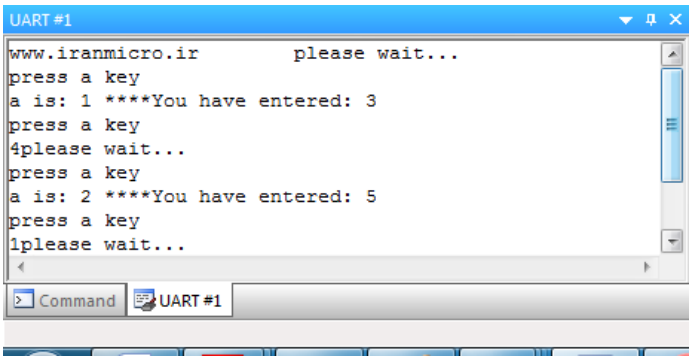
عبارت please wait... را به پورت ارسال می کند ، کارکتر \n که همان enter است باعث رفتن به خط بعد می شود (در میکرو کنترلر ها کد اسکی کلید enter ارسال می شود)

هنگامی که cpu به دستور :

```
scanf("%c",&in);
```

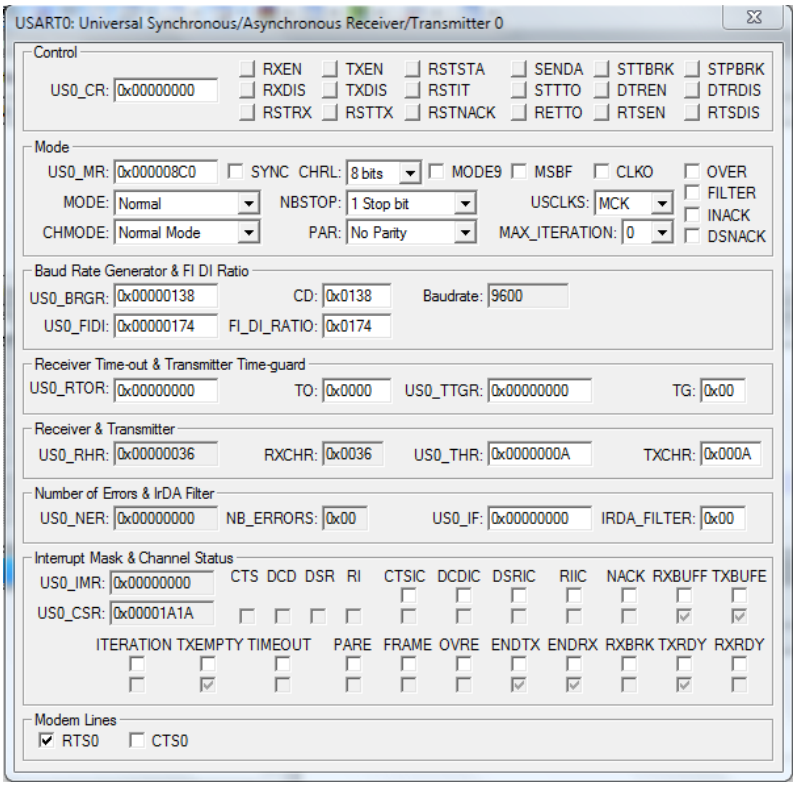
رسید منتظر می ماند تا کلیدی فشرده شود ، کد مربوط به کلید فشرده شده در متغیر in ذخیره می شود ، هنگامی که شما کلید enter را فشار می دهید (کد مربوط به کلید enter را برای میکرو ارسال می کنید) ، cpu به خطی بعدی می رود :

```
printf("a is: %d ****You have entered: %c\n",a,in);
```



در این خط مقدار متغیر a که هر بار یک واحد به آن افزوده می شود و عبارت :You have entered****و مقدار موجود در متغیر in به ترمینال ارسال می شود . دستور getkey کد دریافت شده از پورت را در متغیر b می ریزد ، مقدار متغیر b با دستور sendchar به پورت ارسال می شود .

برای مشاهده ی ریجستر های مربوط به usart از منوی Peripherals و زیر منوی usart گزینه ی 1 usart را انتخاب کنید :



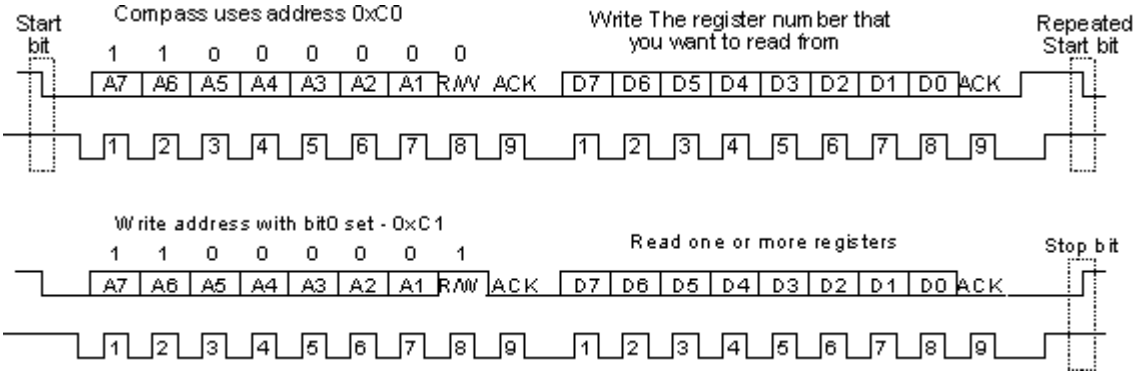
با اجرا کردن برنامه می توانید تغییراتی که بر روی رجستر ها ایجاد می شود را مشاهده کنید . همانطور که میبینید ، وضعیت کلیه رجیستر های بررسی شده در این بخش وجود دارد .

I2C و نحوه ی کار با آن

یکی از پروتکل های پرکاربرد در میکروکنترلرها پروتکل I2C است که کاربران توسط آن میتوانند قطعات دارای این پروتکل را فقط با استفاده از دو سیم به میکروکنترلر متصل کنند، در این حالت میکروکنترلر در مد مستر (Master)و سایر قطعات متصل شده به باس در مد اسلیو (slave) پیکربندی میشوند .

در پروتکل I2C به هر وسیله ی جانبی که باس متصل میشود یک آدرس تعلق میگیرد، معمولا آدرس توسط سه پایه ی موجود بر روی قطعه که با نام های A0 تا A2 مشخص میشوند ، معین میشود .(البته بعضی از قطعات دارای آدرس داخلی هستند ، که در دیتاشیت آنها مشخص گردیده است یا آدرس دهی برخی از قطعات مانند میکروکنترلرها در برنامهی نوشته شده برای آنها مشخص میشود) .

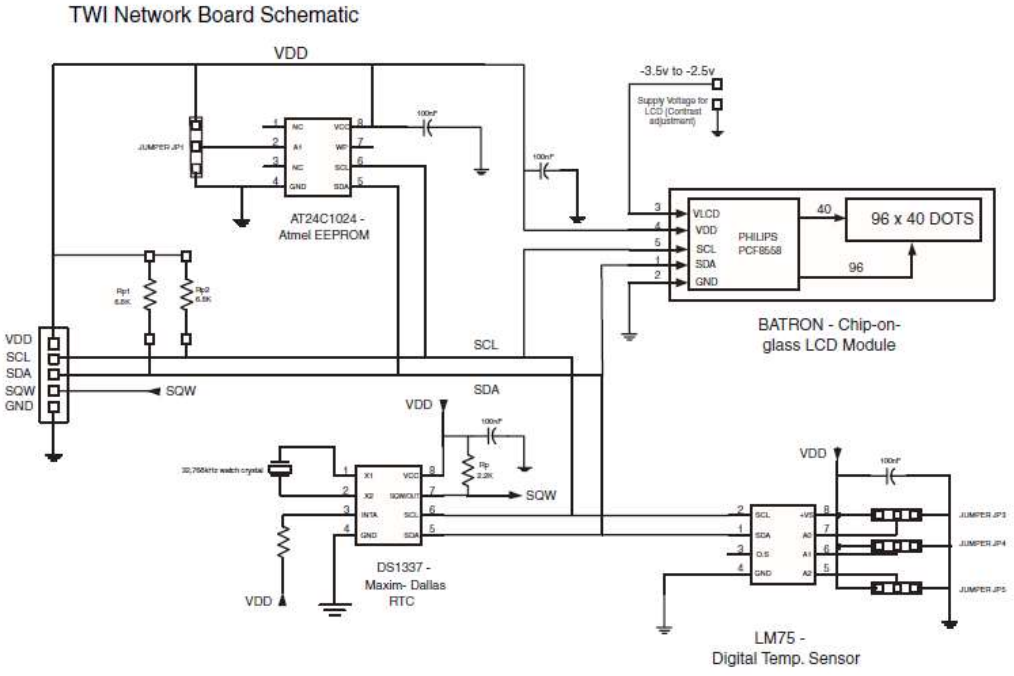
آدرس قطعه میتواند از صفر تا 127 دسیمال یا 0 تا 0F هگز باشد ، در این حالت میتوان تا 128 قطعه را در یک باس I2C با هم شبکه کرد (البته به شرط رعایت ملزومات مورد نیاز در طراحی باس I2C). هنگامی که میکرو کنترلر قصد دارد تا با یکی از دستگاه های جانبی ارتباط بر قرار کند ، ابتدا پایه SDA (data) را به یک منطقی میبرد و بعد از گذشت یک پالس کلاک ، آدرس دستگاه جانبی مورد نظر را بر روی باس میفرستد :



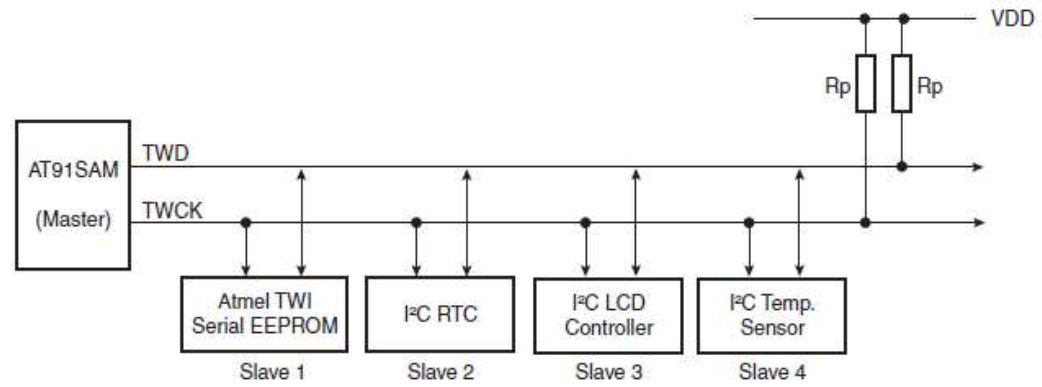
آدرس ارسال شده به باس شامل 8 بیت است که 7 بیت اول آن آدرس قطعه و بیت آخر بیت R/W است (R یعنی قصد داریم اطلاعات قطعه را بخوانیم و W یعنی قصد داریم اطلاعات را به قطعه بدهیم) :

کلیه دستگاه های جانبی آدرس را از روی باس خوانده و فقط دستگاهی که آدرسش با آدرس موجود بر روی خط یکی است ، داده های بعدی را دریافت میکند .

در باس I2C تنها یک مستر وجود دارد و مستر قطعه ای است که آغاز کننده ی عملیات ارسال و دریافت اطلاعات میباشد . در این باس سایر اسلیو ها نمیتوانند اطلاعاتی را به سایر قطعات ارسال کنند . (به بیان ساده تر میتوان گفت که در این باس فقط یک قطعه ی مستر وجود دارد که میتواند اطلاعات را به سایر اسلیوها ارسال کرده یا اطلاعات آنها را بخواند ، در این باس سایر اسلیوها نمیتوانند آغازگر تبادل داده شده و با هم تبادل اطلاعات کنند .) در این حالت با شروع تبادل داده سیگنال کلاک بر روی پایه ی SCL (سریال کلاک) توسط مستر ایجاد شده و سپس تبادل اطلاعات مانند تصویر بالا آغاز میشود . به این ترتیب برای ارتباط با دستگاه های جانبی به دو خط SDA (انتقال داده) و SCL (خط کلاک) نیاز خواهد بود. از خط اول برای ارسال پالس همزمانی (clock) و از خط دوم برای انتقال داده استفاده میشود ، همچنین گراند کلیه لوازم متصل شده به باس باید مشترک باشد .



تصویر بالا مربوط به اتصال چهار دستگاه جانبی به باس i2c است . همانطور که مشاهده میکنید کلیه خطوط داده و کلاک دستگاه های موجود به یکدیگر متصل شده و بعد از pullup شدن توسط مقاومت های 10 یا 2.2 کیلو به میکروکنترلر (ترمینال) متصل شده اند .



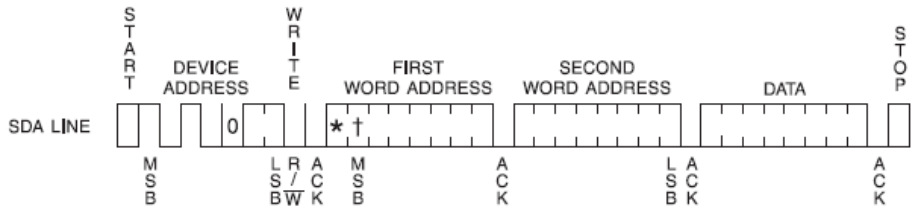
Pull up value as given by the I²C Standard

در باس I2C حداکثر فاصله ی میان میکرو کنترلر و وسیله ی جانبی نباید بیشتر از 80 سانتی متر شود ، همچنین وجود مقاومت های pullup که مقدار آنها 2.2 یا 10 کیلو میباشد ، الزامی است .

✓ ویژگی های باس i2c در میکرو کنترلر های اتمل :

سرعت باس	پشتبانی از مد های Standard Mode Speed (100 KHz) و Fast Mode Speed (400 KHz)
حداقل و حداکثر آدرس	0 تا ff هگز ، قابلیت اتصال 255 وسیله ی جانبی
مد های ارسال داده	پشتیبانی از ACK and NACK ، ادرس دهی 10 بیتی یا 7 بیتی برای بعضی از لوازم
سایر ویژگی ها	دارای فیلتر داخلی(برای تثبیت پالس کلاک) ، سطح داده ورودی میتواند 0 و 3.3 یا 0 و 5 ولت باشد ، پایدار شدن باس فقط یک میکرو ثانیه زمان میرد ، و....

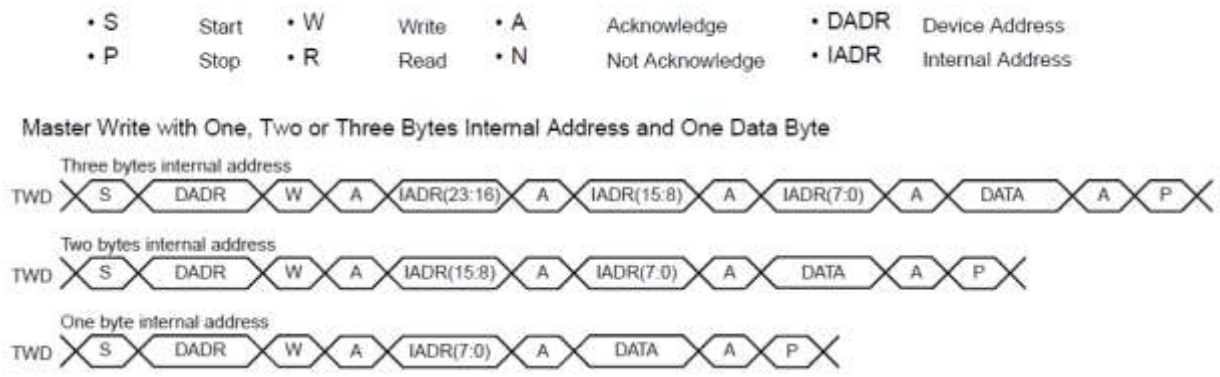
در میکروکنترلر های شرکت اتمل باس I2C با نام باس TWI شناخته میشود . اصول کار با TWI دقیقا مشابه با I2C است ،با این تفاوت که در این باس امکان ارسال آدرس داخلی قطعه ی جانبی نیز به صورت همزمان وجود دارد :



در باس I2C علاوه بر رجیسترهای مربوط به پیکربندی باس یک رجیستر نیز جهت ارسال و دریافت داده وجود دارد؛ تصویر بالا داده های ارسالی به یک حافظه ی EEPROM سریال جهت نوشتن داده در آن را نمایش میدهد برای خواندن ما باید ابتدا با یک دستور آدرس قطعه را به باس ارسال کنیم ، سپس با یک دستور دیگر آدرس خانه ی حافظه (مکانی که قصد خواندن آن را داریم) را ارسال کنیم و بعد داده های موجود در خانه ی مورد نظر را توسط یک دستور دیگر بخوانیم :

```
i2c_start();
  i2c_write(0xa0);
  i2c_write(address>>8);
  i2c_write(address);
  i2c_write(data);
```

در حالی که در باس TWI علاوه بر رجیستر های پیکربندی و رجیسترارسال و دریافت داده ، دو رجیستر دیگر نیز برای ذخیره سازی آدرس اصلی قطعه و آدرس داخلی آن وجود دارد ، در این حالت کاربر میتواند تنها با یک دستور آدرس قطعه را به باس ارسال کرده و بعد از ارسال آدرس داخلی آن ، محتوای خانه ی مورد نظر را خوانده یا آن را تغییر دهد :



در بخش مثالها در مورد نحوه ی انتقال داده و عملکرد این باس بیشتر توضیح می دهیم .

در ادامه به بررسی رجیستر های مربوط به این باس در کامپایلر KEIL پرداخته ایم ، بعد از مطالعه ی این رجیستر ها میتوانید این باس را راه اندازی نموده و اطلاعات بعدی را بهتر بفهمید .

ریجستر های مربوط به i2c:

برای واحد I2C نیز همچون سایر بخش های جانبی رجیستر های در نظر گرفته شده است که نام و کاربرد آنها را در جدول زیر مشاهده میکنید :

نام ریجستر	نام کامل	توصیف
TWI_CR	Control Register	تنظیمات اصلی باس i2c با مقدار دهی این رجیستر انجام میشود .
TWI_MMR	Master Mode Register	در صورتی که باس در حالت مستر پیکربندی شود ، با مقدار دهی این رجیستر میتوان موارد همچون آدرس داخلی داخلی و نحوی ارسال و... را تعیین کرد .
TWI_IADR	Internal Address Register	در صورتی که باس در حالت اسلیو پیکربندی شود ، آدرس قطعه با مقدار این رجیستر تعیین میشود .
TWI_CWGR	Clock Waveform Generator Register	تنظیمات مربوط به فرکانس و ... کلاک باس با مقدار دهی این رجیستر انجام میشود .
TWI_SR	Status Register	وضعیت بخش های مختلف باس در این رجیستر ذخیره میشود .
TWI_IER	Interrupt Enable Register	با مقدار دهی این رجیستر میتوان وقفه ی بخش های مختلف باس را فعال کرد .
TWI_IDR	Interrupt Disable Register	با مقدار دهی این رجیستر میتوان وقفه ی بخش های مختلف باس را غیر فعال کرد.
TWI_IMR	Interrupt Mask Register	نوع پوشش وقفه با این رجیستر مشخص میشود .
TWI_RHR	Receive Holding Register	داده ای که از باس دریافت میشود ، در این رجیستر قرار میگیرد .
TWI_THR	Transmit Holding Register	داده ای که قرار است به باس ارسال شود ، در این رجیستر قرار میگیرد .

رجیستر TWI_CR (TWI Control Register) :

TWI_CR یک رجیستر 32 بیتی است که ما از 5 بیت آن برای فعال سازی و پیکربندی باس I2C استفاده میکنیم ، بقیه بیت ها برای مقاصد آینده رزرو شده و مقدار دهی آنها تاثیری در عملکرد باس ندارد .

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
SWRST	–	–	–	MSDIS	MSEN	STOP	START

بیت START (Send a START Condition):

0 = بی اثر

1= توضیحات 1

بیت STOP (Send a STOP Condition):

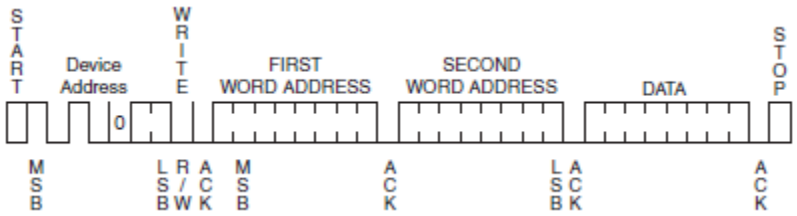
0 = بی اثر

1= توضیحات 1

توضیحات 1 :

همانطور که در تصویر زیر مشاهده میکنید ، هنگامی که میکرو کنترلر قصد دارد ارتباط خود را دستگاه دیگر شروع کند ، با یک بیت start خط sda (TWD) را یک میکند ، در این حالت کلیه دستگاه های جانبی متوجه اشغال شدن خط شده و خود را برای دریافت آدرس آماده میکنند و...

بعد از اینکه ارتباط برقرار شد و داده مورد نظر ارسال گردید ، میکرو مجددا با یک بیت stop ارتباط خود را قطع میکند .



با مقدار دهی بیت های بالا در هنگام شروع ارسال داده و اتمام ارسال میتوانید بیت های start و stop را ایجاد نمایید ، توجه داشته باشید که :

ارسال بیت START برای آگاه سازی سایر لوازم موجود بر روی باس الزامی است . همچنین در صورتی که بیت STOP ارسال نشود ، بعد از ارسال 8 بیت داده ی آخر ، بیت بعدی میتواند به عنوان بیت ACK یا STOP در نظر گرفته شود که در این حالت در عمل کرد باس خطا ایجاد شده و کلیه داده های بعدی نامعتبر خواهد بود .

- ✓ هنگام خواندن یک بایت داده از مستر ، هر دو بیت stop و start باید یک شوند
- ✓ هنگام خواندن چندین باید داده ، بیت stop باید بعد از دریافت آخرین داده یک شود .
- ✓ در مد خواند مستر ، اگر یک بیت NACK دریافت شود ، بیت stop به صورت خودکار تولید میگردد .
- ✓ در هنگام نوشتن چندین بایت داده اگر بافر ارسال داده و رجیستر THR خالی باشند ، یک پالس stop به صورت خودکار ارسال میشود .
- ✓ بیت stop و start فقط باید توسط مستر ارسال شود .
- ✓ در ادامه این وضعیت ها را در مثال علمی بررسی نموده ایم .

بیت MSEN (TWI Master Transfer Enabled):

0 = بی اثر

1= اگر بیت MSDIS صفر باشد ، انتقال داده در مد مستر فعال میشود .

بیت MSDIS (TWI Master Transfer Disabled):

0 = بی اثر

1= مد مستر غیر فعال میشود و میکروکنترلر میتواند در مد اسلیو راه اندازی شود .

بیت SWRST (Software Reset):

0 = بی اثر

1= با یک شدن این بیت ، باس ریست میشود .

رجیستر TWI_MMR (TWI Master Mode Register):

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	DADR						
15	14	13	12	11	10	9	8
–	–	–	MREAD	–	–	IADRSZ	
7	6	5	4	3	2	1	0
–	–	–	–	–	–	–	–

بیت های IADRSZ : این بیت ها اندازه ی آدرس داخلی قطعه را مطابق جدول زیر تعیین میکنند :

مقدار آدرس	IADRSZ[9:8]	
آدرس داخلی غیر فعال میشود .	0	0
آدرس داخلی قطعه برابر با یک بایت است .	1	0
آدرس داخلی قطعه برابر با دو بایت است .	0	1
آدرس داخلی قطعه برابر با سه بایت است .	1	1

از این بیت ها بیشتر برای ارتباط با قطعات جانبی نظیر حافظه های EEPROM ، تراشه های DAC و ADC (مبدل های دیجیتال به آنالوگ و آنالوگ به دیجیتال) و ... که دارای آدرس داخلی هستند استفاده می شود ، در حالت عادی و بدون مقدار دهی این بیت ها اگر قصد داشته باشید با یک EEPROM ارتباط برقرار کنید ، ابتدا باید ادرس قطعه را به باس بفرستید و سپس ، ادرس خانه ای که قصد خواندن یا نوشتن ان را دارید ارسال کنید ، با استفاده از این بیت ها میتوانید مقدار آدرس داخلی قطعه را مشخص کرده و سپس توسط بیت های IADR مستقیما آدرس داخلی قطعه (آدرس خانه ی مورد نظر) را بخوانید . در بخش مثال ها اطلاعاتی بیشتری در این باره اورده شده است .

بیت MREAD (Master Read Direction):

0= مستر آماده ی نوشتن میشود .

1= مستر آماده ی خواندن میشود .

هنگامی که مستر قصد دارد داده ای را به سمت اسلیو ارسال کند (توجه داشته باشید که در اینجا منظور از مستر ، قطعه ای است که اول ارتباط را شروع میکند و منظور از اسلیو قطعه ای است که به ارتباط ایجاد شده ملحق میشود) بعد از ارسال بیت start ، داده ی 7 بیتی مربوط به آدرس اسلیو را به خط میفرستد و...

بیت های DADR (Device Address) :

در صورتی که میکروکنترلر در مد مستر پیکربندی شود ، آدرس قطعه ی slave در این بیت ها قرار میگیرد و در صورتی که میکروکنترلر به صورت اسلیو راه اندازه شود ، مقدار درج شده در این بیت ها ادرس ان در باس خواهد بود

رجیستر TWI_IADR (TWI Internal Address Register):

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
IADR							
15	14	13	12	11	10	9	8
IADR							
7	6	5	4	3	2	1	0
IADR							

بیت های IADR (Internal Address):

در رجیستر قبلی با استفاده از بیت های IADRSZ اندازه ی آدرس داخلی قطعه را مشخص کردیم ، همکنون با استفاده از این رجیستر میتوانید آدرس داخلی قطعه را مشخص نمایید . آدرس داخلی قطعه میتوانـد 8 یـا 16 یـا 24 بیـت باشـد (مـثلا 0x05)

رجیستر TWI_CWGR (TWI Clock Waveform Generator Register):

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	CKDIV		
15	14	13	12	11	10	9	8
CHDIV							
7	6	5	4	3	2	1	0
CLDIV							

همانطور که قبلا گتفیم میکرو کنترلر های سری at91sam میتوانند از دو سرعت 100 و 400 کیلو هرتـز پشـتیبانی کننـد ، در کامپایلر keil سرعت باس i2c برای مد های کم سرعت و پر سرعت به ترتیب با مقدار دهی بیت های CLDIV (Clock Low Divider) و CHDIV (Clock High Divider) انجام میشود ، مقدار فرکانس باس برای هر سرعت از فرمول های زیر بدست می آید :

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$T_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 3) \times T_{MCK}$$

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$T_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 3) \times T_{MCK}$$

مقدار فرکانس باس برای قطعات مستر ، باید کمتر یا برابر با فرکانس باس قطعات slave باشد . در ادامه در مورد فرمول های بالا بیشتر توضیح داده ایم .

رجیستر TWI_SR (TWI Status Register):

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	NACK
7	6	5	4	3	2	1	0
–	–	–	–	–	TXRDY	RXRDY	TXCOMP

با استفاده از رجیستر فقط خواندنی TWI_SR میتوانید وضعیت بخش های مختلف باس i2c را دریافت کنید .

بیت TXCOMP (Transmission Completed) : هنگامی که این بیت 0 باشد ، به این مفهموم است که باس در حال انجام دادن یک عملیات (خواندن یا نوشتن) است ، یک بودن این بیت به مفهوم خالی بودن بافر ها (ارسال و دریافت داده) و شیف رجیستر یا رخ دادن بیت stop یا یک شدن بیت MSEN است .

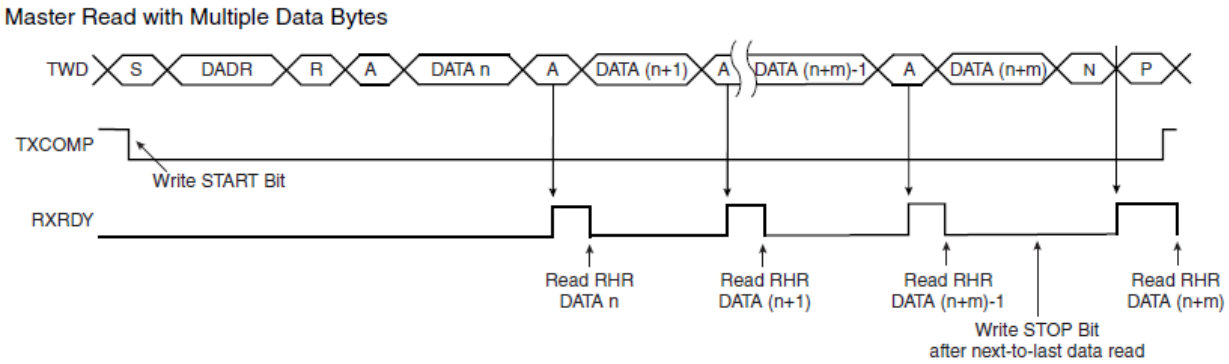
بیت RXRDY (Receive Holding Register Ready) : این بیت وضعیت رجیستر دریافت داده (TWI_RHR) را مشخص میکند ، هنگامی که باس داده ای را دریافت کند بیت RXRDY یک شده و هنگامی که رجیستر مذکور خالی باشد ، بیت RXRDY صفر میشود .

بیت TXRDY (Transmit Holding Register Ready) : این بیت وضعیت رجیستر ارسال داده (TWI_RHR) را مشخص میکند ، هنگامی که قصد داریم داده ای را ارسال کنیم ، باید آن را در رجیستر TWI_THR بریزیم با مقدار دهی این رجیستر داده ی موجود به یک shift register وارد شده و سپس به صورت سریال به دستگاه دیگر ارسال میشود . خالی بودن بافر داده و شیفت رجیستر باعث صفر شدن بیت TXRDY و پر بودن آنها باعث یک شدن آن میشود . همچنین یک شدن هر یک از بیت های TXCOMP و NACK و MSEN نیز این بیت را یک میکند .

بیت NACK (Not Acknowledged) :

همانطور که قبلا نیز گفتیم ، رجیستر های ارسال و دریافت داده در باس i2c هشت بیتی هستند ، و میتوانند از 0 تا ff را در خود ذخیره کنند . در صورتی که قصد داشته باشیم داده ای با طول بیشتر از 8 بیت را از طریق باس ارسال کنیم ، کافی است داده را در بخش های 8 بیتی در رجیستر ارسال داده قرار دهیم . در این حالت شیفت رجیستر ارسال داده بعد از ارسال هر بخش بیتی به نام ack (acknowledged) را به قطعه ی دیگر ارسال می کند . در قطعه ی دیگر با دریافت شدن این بیت ، تداوم داشتن ارسال داده تصدیق شده و میکرو کد های بعدی را به کد دریافت شده میچسباند .

هنگامی که قطعه ی فرستنده ی تمامی بخش های موجود را ارسال کرد بیت دیگری به نام NACK (Not Acknowledged) را به قطعه ی گیرنده ی ارسال کرده و آن را از پایان یافتن ارسال داده مطلع میکند .



هنگامی که داده ی بدون بیت acknowledged از slave دریافت می شود یا بیت TXCOMP یک میشود ، بیت NACK یک میگردد ، برای سایر حالات این بیت صفر است .

رجیستر TWI_RHR (TWI Receive Holding Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
RXDATA							

داده ای که از باس i2c دریافت میشود ، در 8 بیت اول این رجیستر (بیت های RXDATA) قرار میگیرد . شما باید محتوای این رجیستر را در یک متغیر مناسب ریخته و در برنامه خود استفاده نمایید .

رجیستر TWI_THR (TWI Transmit Holding Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
TXDATA							

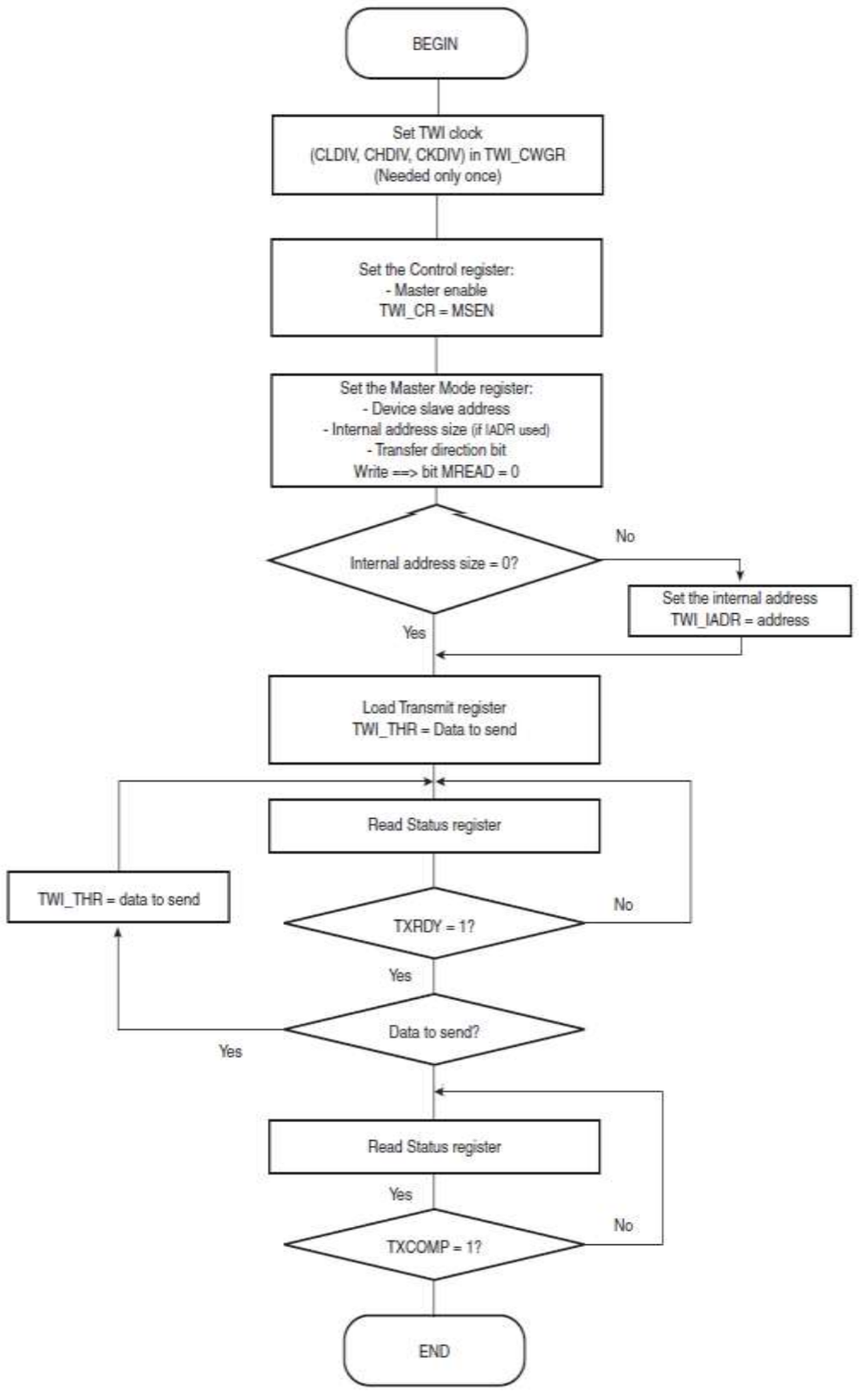
داده ای که قصد داریم به دستگاه دیگر ارسال کنیم ، باید در این رجیستر ریخته شود .

برای باس i2c سه رجیستر TWI_IER و TWI_IDR و TWI_IMR برای فعال و غیر فعال و کار با وقفه ی بخش های مختلف باس در نظر گرفه است ، این رجیستر ها در یک برگه ی اطلاعاتی دیگر(که در مورد واحد کنترل وقفه نوشته میشود) بررسی خواهد شد .

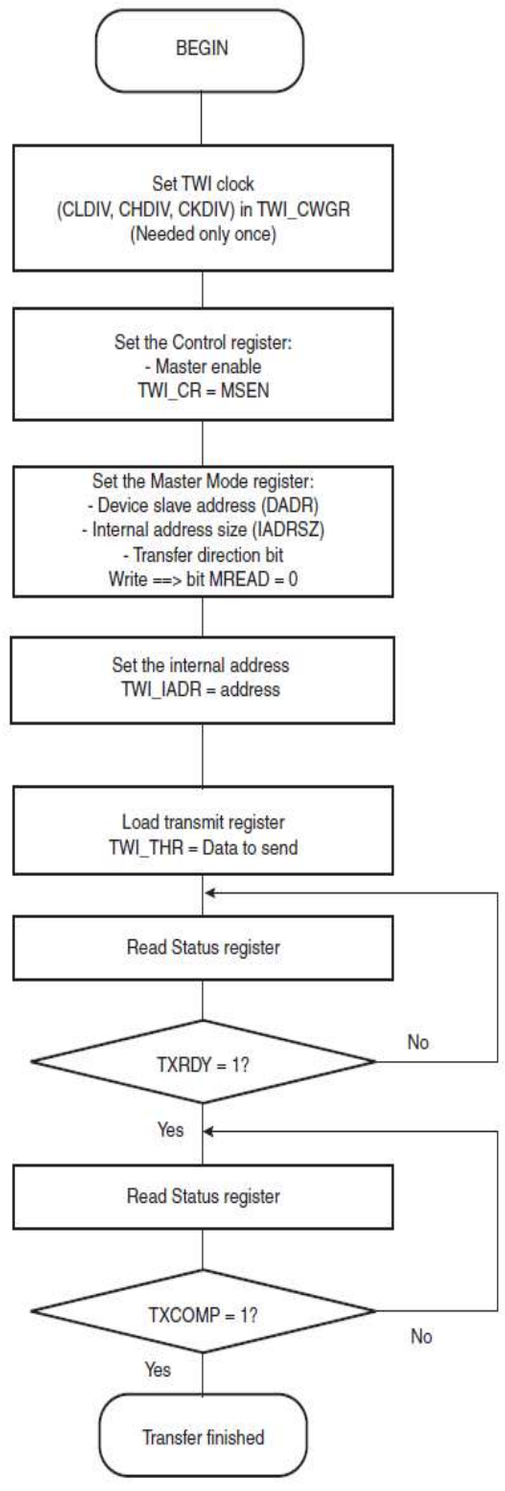
پیکربندی باس I2C در سری AT91SAM :

در صفحه ی بعد تعدادی بلوگ دیاگرام مروبط به راه اندازی باس TWI آورده شده است ، مطالعه ی این بلوک دیاگرام ها شما را در درک مطالب فوق یاری میدهد .

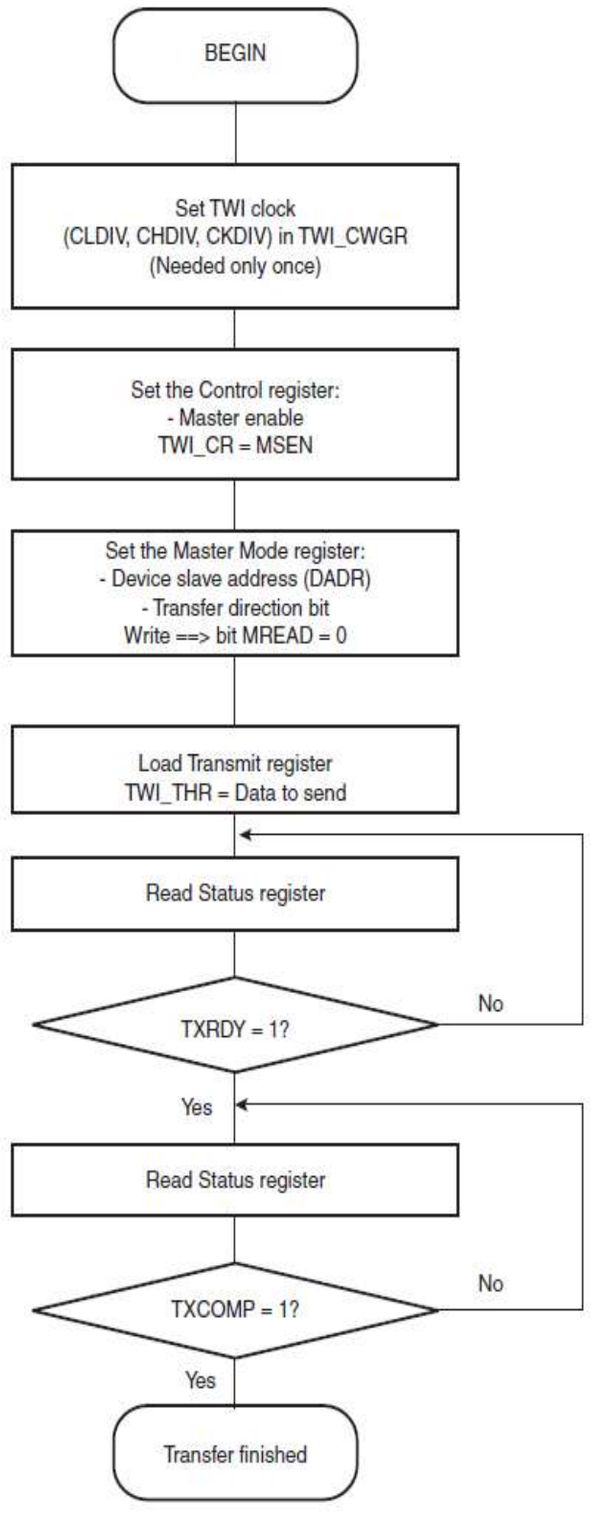
TWI Write Operation with Multiple Data Bytes with or without Internal Address



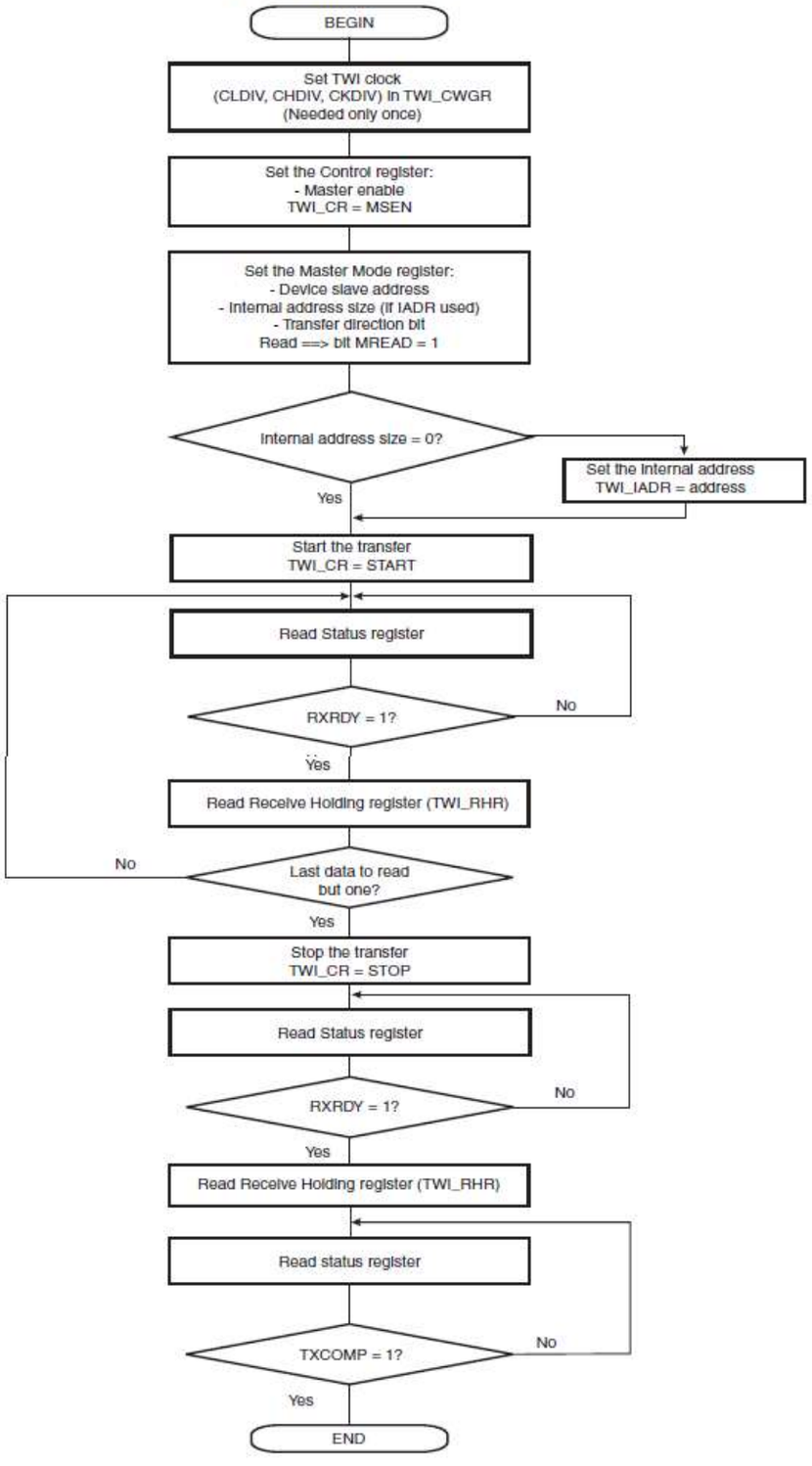
TWI Write Operation with Single Data Byte and Internal Address



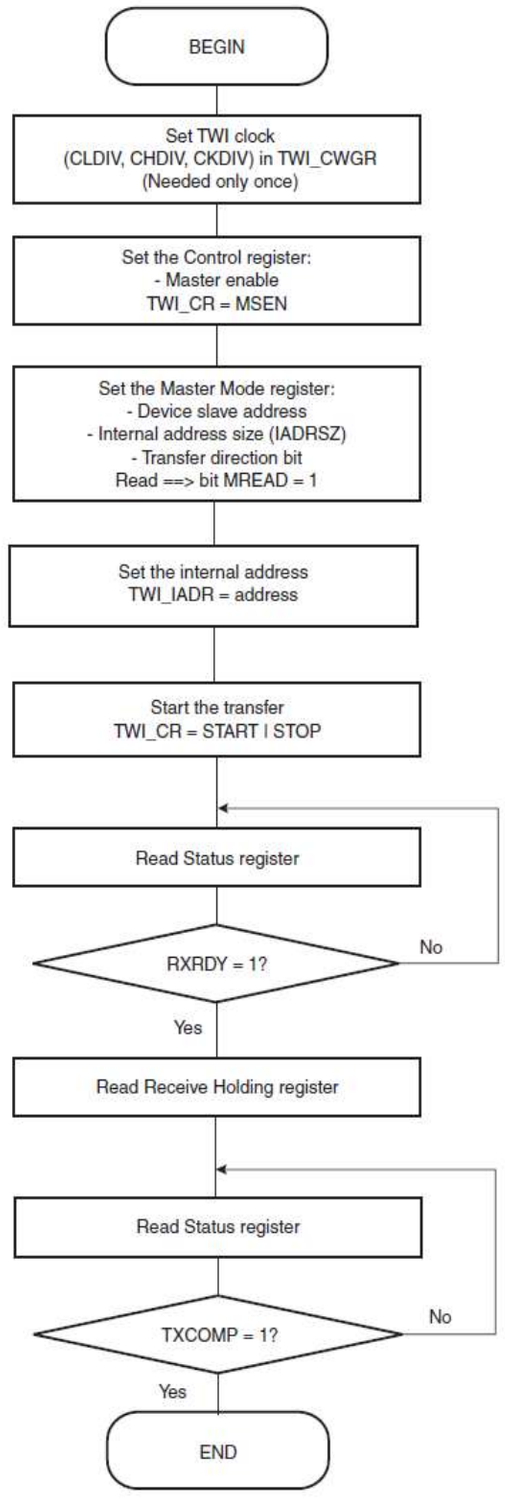
TWI Write Operation with Single Data Byte without Internal Address



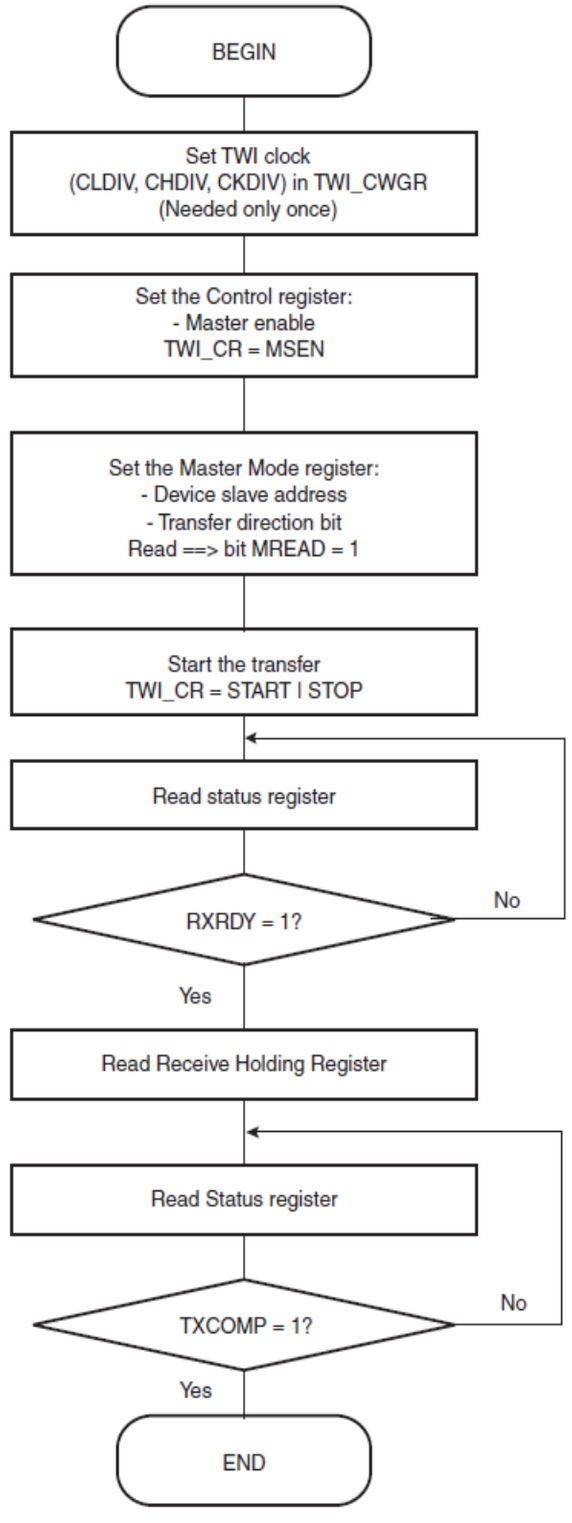
TWI Read Operation with Multiple Data Bytes with or without Internal Address



TWI Read Operation with Single Data Byte and Internal Address



TWI Read Operation with Single Data Byte without Internal Address



برای پیکربندی باس I2C یا TWI در میکروکنترلرهای سری At91sam از کتابخانه ی twi.c استفاده می شود ، این کتابخانه توسط خودم نوشته شده و برای آشنایی شما با نحوه ی نوشتن کتابخانه در زیر به بررسی مراحل نوشتن آن پرداخته شده است .

نوشتن کتابخانه برای باس TWI :

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)

پیکربندی واحد (باس I2C) :

در میکروکنترلر های سری AT91SAM اولین قدم برای راه اندازی هر بخش ، فعال سازی منبع کلاک آن در واحد PMC است .

مطابق اطلاعات موجود در صفحه ی 194 دیتاشیت میکروکنترلر AT91SAM7X256 ما باید بیت نهم رجیستر PMC_PCER را یک کنیم تا کلاک واحد TWI فعال شود :

```
*AT91C_PMC_PCER=0X 100;
```

(با دستور بالا عدد 100 در مبنای هگز یا 100000000 در مبنای باینری در رجیستر PCER_ قرار داده میشود)

در صورتی که واحد مورد نظر دارای پایه ورودی/ خروجی باشد ، باید با مقدار دهی رجیستر های مربوط به واحد PIO ، ورودی / خروجی های مورد نیازش را فعال کنیم :

```
*AT91C_PIOA_PDR = (1<<10)|(1<<11);
```

```
*AT91C_PIOA_MDER= (1<<10)|(1<<11);

*AT91C_PIOA_ASR = (1<<10)|(1<<11);
```

مطابق فوچارت های موجود در صفحه ی قبل ، در مرحله ی دوم باید مقدار کلاک باس TWI را مشخص کنیم که این کار با مقدار دهی رجیستر TWI_CWGR انجام میشود :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	CKDIV		
15	14	13	12	11	10	9	8
CHDIV							
7	6	5	4	3	2	1	0
CLDIV							

همانطور که قبلا اشاره شد ، میکرو کنترلر های سری at91sam میتوانند از دو سرعت 100 و 400 استاندارد کیلو هرتز پشتیبانی کنند . که با استفاده از این رجیستر میتوان تعیین کرد که زمان تناوب پالس کلاک در حالت صفر و یک چقدر باشد :

- **CLDIV: Clock Low Divider**

The SCL low period is defined as follows:

$$T_{low} = ((\text{CLDIV} \times 2^{\text{CKDIV}}) + 3) \times T_{MCK}$$

- **CKDIV: Clock Divider**

The CKDIV is used to increase both SCL high and low periods.

- **CHDIV: Clock High Divider**

The SCL high period is defined as follows:

$$T_{high} = ((\text{CHDIV} \times 2^{\text{CKDIV}}) + 3) \times T_{MCK}$$

فعلا مقادیر CLDIV و CHDIV را برای سرعت 400 کیلو هرتز محاسبه میکنیم و در ادامه با یک رابطه ریاضی این مقادیر را برای هر سرعتی محاسبه خواهیم کرد : (توجه داشته باشید که فرکانس کاری میتواند برای هر مقدار دلخواهی محاسبه شود ، اعداد ذکر شده ، فرکانس های استانداردی هستند که اکثر قطعات I2C در آنها کار میکنند .)

چون پالس کلاک مربعی است ، پس مقادیر CLDIV و CHDIV با هم برابر خواهد بود :

$$T = \frac{1}{400000} = 0.0000025 \rightarrow T_{\text{HIGH}} = T_{\text{LOW}} = \frac{0.0000025}{2} = 0.00000125$$

ما باید CLDIV یا CLDIV را طوری تعیین کنیم که مقدار THIGH یا TLOW برابر با عدد بدست آمده شود . با توجه به اینکه ممکن است به ازای مقادیر مختلف MCK اعداد بدست آمده دارای رقم اعشار باشند ، شرکت اتمل ضریب CKDIV را در فرمول بالا لحاظ کرده است ، این ضریب در شرایط عادی باید برابر یک باشد ، در صورتی که با این مقدار ، زمان تناوب بدست آمده به مقدار مورد نظر نرسید میتوان آن را یک رقم افزایش داد و مجددا محاسبات را انجام داد ، مطابق دیتا شیت مقدار $CLDIV \times 2^{CKDIV}$ باید کوچکتر یا مساوی 8191 باشد . با قرار دادن عدد یک به جای CLDIV فرمول زیر بدست می آید :

$$\text{CHDIV} = \frac{\left(\frac{Thigh * 2}{Tmck} - 3\right)}{2^{\text{CKDIV}}} \; or \; \text{CHDIV} = \frac{\left(\frac{MCK}{Ftwi * 2} - 3\right)}{2^{\text{CKDIV}}}$$

در این فرمول Tmck زمان تناوب کلاک اصلی سیستم است که از رابطه ی 1/MCK بدست می آید .

در صورتی که میکروکنترلر با کلاک 48 مگا هرتز کار کند ، مقدار Tmck برابر با 2.083333333333333333333333333333e-8 خواهد بود و نتیجه ی مخرج کسر برابر با 57 میشود . که با تقسیم کردن این عدد به دو ، عدد مورد نیاز برای CLDIV یا CLDIV بدست می آید که برابر با 28 میشود (عدد اعشاری گرد میشوند)

```
*AT91C_TWI_CWGR=(0x10000|0x1C00|0x1C);
```

توجه داشته باشید که عدد 48 مگاهرتز فرکانس اصلی سیستم بوده و برابر با فرکانس کریستال متصل شده به میکروکنترلر نیست ، برای کسب اطلاعات بیشتر در فصل قبلی ، مطالب بخش پیکربندی واحد PLL را بخوانید .

مطابق فلوچارت ، در مرحله ی بعد باید رجیسترهای Control Register و Master Mode Register را مقدار دهی کنیم :

در رجیستر اول باید مشخص کنیم میکروکنترلر در باس ، مستر است یا اسلیو :

```
*AT91C_TWI_CR=0x4;
```

با یک کردن بیت سوم این رجیستر (مقدار دهی با 100 باینری) ، میکروکنترلر در حالت مستر راه اندازی میشود ، می توانیم با یک کردن بیت چهارم ، میکرو را در حالت اسلیو پیکربندی کنیم .

در پروتکل i2c هر قطعه به جز قطعه ی مستر باید دارای یک آدرس باشد . در صورتی که میکروکنترلر در مد اسلیو پیکربندی میشد ، با مقدار دهی بیت های DADR آدرس آن در باس مشخص میگردید .

در مد مستر بیت های DADR مشخص کننده ی آدرس دستگاهی است که قصد ارتباط با آن را داریم ،فعلا من برای قطعه ی جانبی ادرس 1 را در نظر میگیرم :

```
AT91C_TWI_MMR=0x 10000;
```

مقدار دهی بیت های دیگر را در ادامه انجام خواهیم داد .

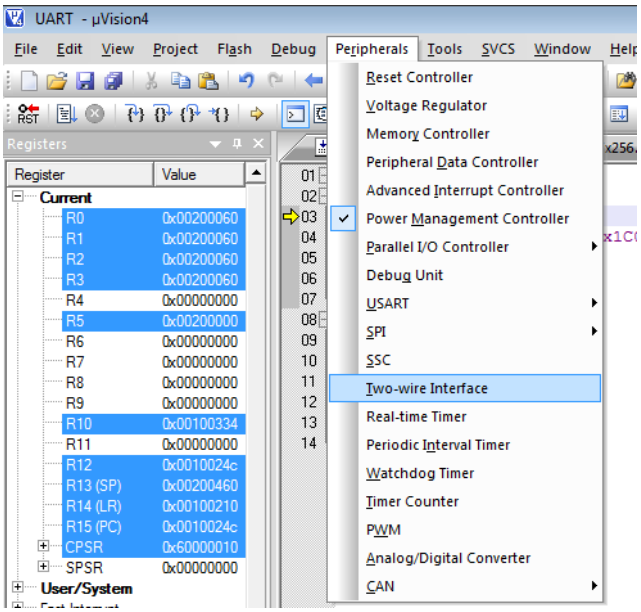
تا اینجا ما دستورات زیر را نوشته ایم ، بعد از افزودن کتابخانه ی میکروکنترلر و حلقه ی MAIN در نرم افزار KEIL یک پروژه ی جدید ایجاد کرده و دستورات را در آن کنید :

```
#include <AT91SAM7x256.h>

int main(void){

*AT91C_PMC_PCER=0X100;

        *AT91C_PIOA_PDR = (1<<10)|(1<<11);

        *AT91C_PIOA_MDER= (1<<10)|(1<<11);

        *AT91C_PIOA_ASR = (1<<10)|(1<<11);

*AT91C_TWI_CWGR=(0x10000|0x1C00|0x1C);

*AT91C_TWI_CR=0x4;

*AT91C_TWI_MMR=0x10000;

}
```

شبیه سازی برنامه :

برای رفع خطا های احتمالی برنامه فعلی را شبیه سازی میکنیم تا ببینیم پیکربندی باس به درستی انجام شده است یا نه ؟

با انتخاب گزینه ی Two-Wire Interface از منوی Peripherals میتوانید رجیستر های مربوط به TWI را در محیط KEIL مشاهده کنید ، برای کسب اطلاعات بیشتر در مورد نحوه ی شبیه سازی میتوانید به بخش اول کتاب مراجعه کنید .



با شروع شدن شبیه سازی ، رجیستر ها در پنجره ی Two-Wire Interface مقدار دهی میشوند :

توجه داشته باشید که رجیستر TWI_CR از نوع " فقط نوشتنی " بوده و مقدار آن در این بخش نمایش داده نمیشود .

در فایل AT91SAM7x256.h و سایر هدر های میکروکنترلر مجموعه ای از دستورات استاندارد وجود دارد که میتوان از آنها برای مقدار دهی رجیستر ها استفاده نمود ، مثلا برنامه ی بالا را به صورت نیز میتوانیم بنویسیم :

```
#include <AT91SAM7x256.h>

int main(void){

*AT91C_PMC_PCER=(1 << AT91C_ID_TWI);

*AT91C_PIOA_PDR |= AT91C_PA10_TWD|AT91C_PA11_TWCK;

        *AT91C_PIOA_MDER|= AT91C_PA10_TWD|AT91C_PA11_TWCK;

        *AT91C_PIOA_ASR |= AT91C_PA10_TWD|AT91C_PA11_TWCK;

*AT91C_TWI_CWGR=(28<<16)|(28 << 8)|28 ;

*AT91C_TWI_CR=AT91C_TWI_MSEN;

*AT91C_TWI_MMR=(1<<16);

}
```

در این برنامه به جای محاسبه ی اعداد برای رجیسترها ، آنها را با دستور شیف (<<) به آدرس مورد نظر شیفت داده ایم ، مثلا برای رجیستر TWI_MMR باید بیت شانزدهم را یک می کردیم ، در برنامه ی قبلی من در جلوی عدد 1 تعداد 16 عدد صفر گذاشتم و این عدد باینری را به هگز تبدیل کردم ، در حالی که با این دستور ، شیف دادن توسط کامپایلر انجام شده و خواندن برنامه برای افرادی که از آن استفاده خواهند کرد ، ساده تر خواهد بود .

ایجاد اولین تابع از کتابخانه :

تا اینجا ما بخش پیکربندی باس را ایجاد کردیم ، در این پیکربندی ، همه چیز ثابت است ، ما میتوانیم مجموعه دستورات بالا را به صورت یک تابع در آورده و مقادیر مورد نیاز جهت پیکربندی باس ، که در زیر اورده شده اند را به آن ارسال کنیم :

- مقدار فرکانس باس TWI
- مستر یا اسلیو بودن میکروکنترلر

```
void TWI_Init( unsigned int TwiClock,unsigned int mck,unsigned char mode);
```

به جای TWI_Init میتوانیم هر واژه ی دلخواه دیگری را استفاده کنیم ، واژه های مثل TWI_Configure، twi_setting و...

این تابع به دو مقدار برای پیکربندی رجیستر ها نیاز دارد ، همچنین ما باید مقادیر مورد نیاز برای باید CLDIV یا CLDIV رجیستر TWI_CWGR را در برنامه حساب کنیم :

من از حلقه ی زیر برای محاسبه ی CLDIV و CLDIV و CKDIV استفاده میکنم :

```
while ( ( cldiv = ( (MCK/(2*TwiClock))-3 ) / (1 << ckdiv)) > 255 )
  ckdiv++ ;
*AT91C_TWI_CWGR= =(ckdiv<<16)|((unsigned int)cldiv << 8)|(unsigned int)cldiv  ;
```

در مجموعه دستورات بالا به فرکانس اصلی (MCK) برای محاسبه کلاک باس نیاز داریم ، همچنین با دستور زیر رجیستر TWI_CR مقدار دهی میشود :

```
if(mode==0)
 *AT91C_TWI_CR = AT91C_TWI_MSEN;
 else
 *AT91C_TWI_CR = AT91C_TWI_MSDIS;
```

در اینجا اگر mode برابر با 0 باشد میکروکنترلر در حالت مستر و در غیر اینصورت در حالت اسلیو پیکربندی میشود (مثلا اگر mode برابر یک باشد) .

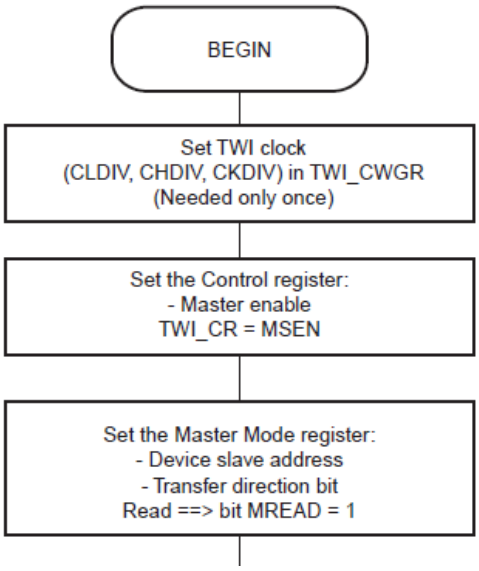
```
#include <AT91SAM7x256.h>
#define MCK 48000000
 void TWI_Init ( unsigned int twck, unsigned char mode){
 unsigned int cldiv,ckdiv=1 ;
*AT91C_PMC_PCER=(1 << AT91C_ID_TWI);
*AT91C_PIOA_PDR |= AT91C_PA10_TWD|AT91C_PA11_TWCK;
        *AT91C_PIOA_MDER|= AT91C_PA10_TWD|AT91C_PA11_TWCK;
        *AT91C_PIOA_ASR |= AT91C_PA10_TWD|AT91C_PA11_TWCK;
 while ( ( cldiv = ( (MCK/(2*twck))-3 ) / (1 << ckdiv)) > 255 )
  ckdiv++ ;
*AT91C_TWI_CWGR=(ckdiv<<16)|((unsigned int)cldiv << 8)|(unsigned int)cldiv  ;
```

```
if(mode==0)
*AT91C_TWI_CR = AT91C_TWI_MSEN;
else
*AT91C_TWI_CR = AT91C_TWI_MSDIS;
}
int main(void){
TWI_Init (200000,1,22,0) ;
*AT91C_TWI_MMR= (address_size << 8) | (iaddress << 16);
}
```

با تغییر دادن مقادیر تابع TWI_Init (200000 ,0) ; مشاهده میکنید که پیکربندی باس تغییر میکند .

مطابق فلوچارت ما تا ایجا مراحل پیکربندی باس را انجام دادیم و اکنون نوبت به خواندن و نوشتن اطلاعات میرسد.

(اگر دقت کنید می بینید که این مراحل در هر شش فلوچارت بالا ثابت بود) .



ایجاد تابع نوشتن :

برای ارسال اطلاعات به باس TWI ابتدا باید بیت MREAD در رجیستر TWI_MMR را صفر کنیم (در شروع کار این بیت صفر است و این کد در این مرحله تاثیری در برنامه ندارد و بیشتر برای تکمیل شدن پروسه است (حتی اگر بیت MREAD یک هم باشد کد زیر به تنهایی در آن تاثیری ندارد)) :

```
*AT91C_TWI_MMR|= (0x0 << 12);
```

استفاده از دستور =| باعث میشود تا اطلاعات موجود در رجیستر با داده های جدید OR شده و از بین نروند .

در مرحله ی بعدی و مطابق استاندارد I2C باید فرمان استارت را به باس ارسال کرده و بعد از آن آدرس قطعه ی جانبی را ارسال کنیم ، من جهت کاربردی تر بودن آموزش اطلاعات را به یک ای سی EEPROM به شماره ی AT24C256 ارسال میکنم :

Figure 7. Device Address

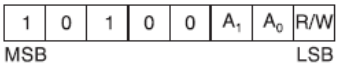
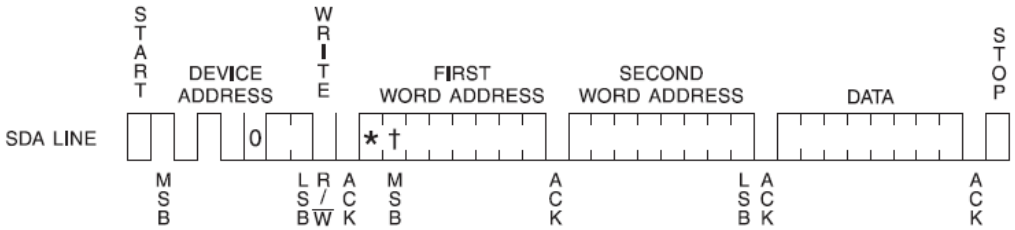


Figure 8. Byte Write



مطابق اطلاعات موجود در دیتاشیت قطعه ، آدرس دهی قطعه مطابق وضعیت پایه های A1 و A0 (تصویر 7) انجام میشود ، مثلا در صورتی که این پایه به زمین (صفر ولت) متصل شوند ، آدرس قطعه در باس برابر A0 خواهد بود .

```
*AT91C_TWI_CR = AT91C_TWI_START;

*AT91C_TWI_MMR|=  (0xA0 << 16);

*AT91C_TWI_MMR= (address_size << 8) | (iaddress << 16);
```

قبل از ارسال داده باید بیت TXRDY در رجیستر TWI_SR را چک کنیم تا از آزاد بودن باس مطمئن شویم :

```
while(!(*AT91C_TWI_SR & AT91C_TWI_TXRDY) == AT91C_TWI_TXRDY);
```

با دستور بالا یک حلقه ایجاد شده و در آن مدام وضعیت بیت TXRDY خوانده میشود (با دستور AT91C_TWI_SR & AT91C_TWI_TXRDY) ، در صورتی که وضعیت این بیت برابر با یک بود ، شرط حلقه نقض شده و CPU دستور بعدی که دستور زیر است را اجرا میکند :

```
*AT91C_TWI_THR=0x00;
```

من میخواهم داده را در ادرس 0 بنویسم ، پس عدد صفر را به باس ارسال میکنم ، از انجا که این قطعه دارای دو بایت ادرس داخلی است پس عدد صفر باید دو بار به باس ارسال شود :

```
while(!(*AT91C_TWI_SR & AT91C_TWI_TXCOMP) == AT91C_TWI_TXCOMP) ;

        *AT91C_TWI_THR=0x00;
```

با چک کردن بیت TXCOMP در رجیستر TWI_SR میتوان از ارسال یا عدم ارسال داده با خبر شویم.

بعد از ارسال 2 بایت آدرس باید داده را ارسال کنیم ، ارسال داده نیز مشابه با ارسال ادرس با مقدار دهی رجیستر TWI_THR انجام میشود :

```
while(!(*AT91C_TWI_SR & AT91C_TWI_TXCOMP) == AT91C_TWI_TXCOMP) ;

        *AT91C_TWI_THR=0x22;
```

با دستور بالا عدد 22 هگز به باس ارسال شده و در خانه ی صفر حافظه نوشته میشود . بعد از انجام عملیات نوشتن باید بیت STOP در رجیستر TWI_CR را صفر کنیم تا باس برای استفاد های بعدی آزاد شود ، در زیر ورژن جدید برنامه آورده شده است :

```
#include <AT91SAM7x256.h>
#define MCK 48000000
 void TWI_Init ( unsigned int twck, unsigned char address_size,unsigned int iaddress,unsigned char mode){
 unsigned int cldiv,ckdiv=1 ;
          *AT91C_PMC_PCER = (1 << AT91C_ID_TWI);
 while ( ( cldiv = ( (MCK/(2*twck))-3 ) / (1 << ckdiv)) > 255 )
   ckdiv++ ;
*AT91C_TWI_CWGR=(ckdiv<<16)|((unsigned int)cldiv << 8)|(unsigned int)cldiv  ;
 if(mode==0)
 *AT91C_TWI_CR = AT91C_TWI_MSEN;
 else
 *AT91C_TWI_CR = AT91C_TWI_MSDIS;
 *AT91C_TWI_MMR= (address_size << 8) | (iaddress << 16);
}
int main(void){
TWI_Init (200000,0,22,0) ;
*AT91C_TWI_MMR|= (0x0 << 12);
*AT91C_TWI_CR |= AT91C_TWI_START;
*AT91C_TWI_MMR|=  (0xA0 << 16);

 while(!(*AT91C_TWI_SR & AT91C_TWI_TXRDY) == AT91C_TWI_TXRDY);
          *AT91C_TWI_THR=0x00;
            while(!(*AT91C_TWI_SR & AT91C_TWI_TXCOMP) == AT91C_TWI_TXCOMP) ;
          *AT91C_TWI_THR=0x00;
            while(!(*AT91C_TWI_SR & AT91C_TWI_TXCOMP) == AT91C_TWI_TXCOMP) ;
          *AT91C_TWI_THR=0x22;
```

```
while(!(*AT91C_TWI_SR & AT91C_TWI_TXCOMP) == AT91C_TWI_TXCOMP) ;
        *AT91C_TWI_CR = AT91C_TWI_STOP;
}
```

در برنامه ی بالا از باس i2c را در مد i2c استفاده کردیم ، در صورتی که بخواهیم از این باس در مد twi استفاده کنیم باید در رجیستر TWI_MMR ، بیت های IADRSZ را با عدد 10 باینری مقدار دهی کنیم در این حالت مقدار آدرس برابر با 2 بایت تعیین میشود ، سپس آدرس خانه ی مورد نظر را به جای ارسال به صورت دستی در رجیستر TWI_IADR قرار دهیم ، در این حالت میکروکنترلر تمامی کار های مربوط به آدرس دهی را انجام داده و کافی است ما داده ی مورد نظر خود را در رجیستر TWI_THR بنویسیم ، در زیر برنامه ی بالا با آدرس دهی داخلی را مشاهده میکنید :

```
#include <AT91SAM7x256.h>

#define MCK 48000000

 void TWI_Init ( unsigned int twck, unsigned char address_size,unsigned int iaddress,unsigned char mode){

 unsigned int cldiv,ckdiv=1 ;

          *AT91C_PMC_PCER = (1 << AT91C_ID_TWI);

 while ( ( cldiv = ( (MCK/(2*twck))-3 ) / (1 << ckdiv)) > 255 )

   ckdiv++ ;

*AT91C_TWI_CWGR=(ckdiv<<16)|((unsigned int)cldiv << 8)|(unsigned int)cldiv  ;

 if(mode==0)

 *AT91C_TWI_CR = AT91C_TWI_MSEN;

 else

 *AT91C_TWI_CR = AT91C_TWI_MSDIS;

 *AT91C_TWI_MMR= (address_size << 8) | (iaddress << 16);

}

int main(void){

TWI_Init (200000,2,22,0) ;
```

```
*AT91C_TWI_MMR|= (0x0 << 12)|(0x2 << 12);

*AT91C_TWI_IADR=0x01;

*AT91C_TWI_CR |= AT91C_TWI_START;

  while(!(*AT91C_TWI_SR & AT91C_TWI_TXRDY) == AT91C_TWI_TXRDY);

  *AT91C_TWI_THR=0x22;

            while(!(*AT91C_TWI_SR & AT91C_TWI_TXCOMP) == AT91C_TWI_TXCOMP) ;

          *AT91C_TWI_CR = AT91C_TWI_STOP;

}
```

در برنامه ی بالا عدد 22 در خانه ی 1 حافظه ....

معرفی کتابخانه باس TWI :

توابع و دستورات مربوط به پیکربندی واحد TWI :

```
void TWI_ConfigureMaster(AT91S_TWI *pTwi, unsigned int twck, unsigned int mck);
```

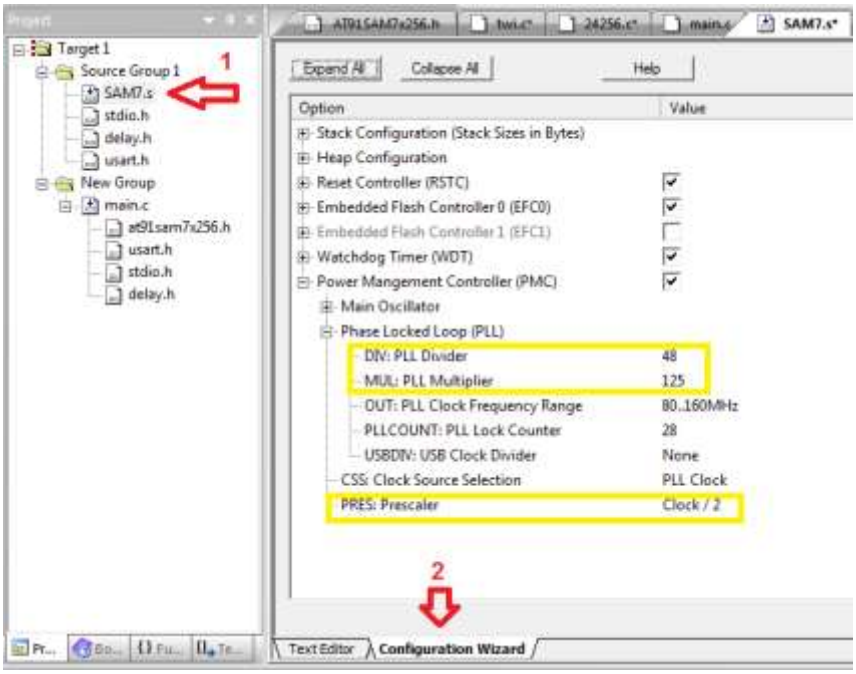
از این تابع برای پیکربندی باس twi در مد مستر استفاده میشود ، در برنامه اصلی سه متغیر به شرح زیر باید به این تابع ارسال شود :

*pTwi AT91S_TWI : نام باس TWI که قصد کار با آن را داریم ، در اینجا میتوانیم از AT91C_ID_TWI برای میکروکنترلر های که دارای یک باس هستند استفاده کنیم ، برای سایر میکروکنترلر ها باید ID (شماره) مربوط به باس را در این مکان قرار دهیم (مثلا AT91C_ID_TWI2 برای راه اندازی واحد TWI شماره 2 در میکروکنترلر های که دارای 2 باس هستند)

unsigned int twck : مقدار کلاک باس TWI به جای این دستور نوشته میشود ، باس TWI از دو سرعت 100000 هرتز (100 کیلو هرتز) و 400000 هرتز (400 کیلو هرتز) پشتیبانی میکند .

unsigned int mck: مقدار کلاک CPU اصلی سیستم به جای این دستور قرار داده میشود ، به عنوان مثال در صورتی که کریستال 18.432 مگاهرتز به میکروکنترلر متصل شده باشد و تنظیمات PLL مطابق تصویر باشد ، عدد 48000000 (

کلاک CPU بر حسب هرتز) به جای این دستور نوشته شود . شما میتوانید با مراجعه به بخش منابع کلاک اطلاعات بیشتری در م ودر نحوه ی محاسبه این عدد بدست آورید :



مثال :

```
TWI_ConfigureMaster(AT91C_BASE_TWI, 400000, 48000000);
```

در مثال بالا باس TWI میکروکنترلر AT91SAM7X256 با فرکانس کاری 40 کیلو هرتز پیکربندی شده است ، در این مثال فرکانس کاری میکروکنترلر برابر با 48 مگاهرتز است .

در صورتی که قصد داشتید باس را در حالت اسلیو پیکربندی کنید ، باید از دستور زیر استفاده نمایید :

```
void TWI_ConfigureSlave(AT91S_TWI *pTwi, unsigned char slaveAddress);
```

در این دستور AT91S_TWI *pTwi مانند حالت قبلی نام باس TWI است که قصد کار با آن را داریم ، همچنین به جای unsigned char slaveAddress عددی بین یک تا 255 که مشخص کننده ی آدرس میکروکنترلر در باس است قرار داده میشود .

مثال :

```
TWI_ConfigureSlave(AT91C_BASE_TWI ,23);
```

با دستور بالا واحد TWI میکروکنترلر در حالت اسلیو پیکربندی میشود و سایر دستگاه های موجود در باس میتوانند با ارسال داده به آدرس 23 با آن تبادل داده کنند .

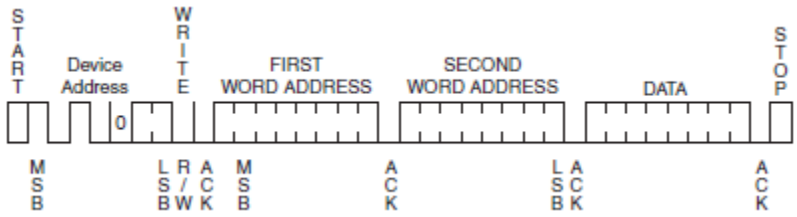
دستورات و توابع مربوط به انتقال داده :

قبل از ارسال داده ، باید با دستور زیر باس TWI را استارت کنیم :

```
void TWI_StartRead( AT91S_TWI *pTwi, unsigned char address, unsigned int iaddress,unsig ned char isize)

void TWI_StartWrite( AT91S_TWI *pTwi,unsigned char address,unsigned int iaddress,unsigned char isize,unsigned
char byte)
```

همانطور که در تصویر زیر مشاهده میکنید ، هنگامی که میکرو کنترلر قصد دارد ارتباط خود را دستگاه دیگر شروع کند ، با یک بیت start خط sda (TWD) را یک میکند ، در این حالت کلیه دستگاه های جانبی متوجه اشغال شدن خط شده و خود را برای دریافت داده آماده میکنند .(معمولا اولین داده ، آدرس دستگاهی است که قصد ارتباط با ان را داریم)

بعد از اینکه ارتباط برقرار شد و داده مورد نظر ارسال گردید ، میکرو مجددا با یک بیت stop ارتباط خود را قطع میکند .



با دستور بالا هنگام شروع ارسال داده بیت start به باس ارسال میشود . توجه داشته باشید که ارسال بیت START برای آگاه سازی سایر لوازم موجود بر روی باس الزامی است .

در این تابع AT91S_TWI *pTwi مانند حالت قبلی نام باس TWI است که قصد کار با آن را داریم ، همچنین به جای unsigned char address آدرس دستگاه جانبی و به جای unsigned int iaddress آدرس میکروکنترلر در باس قرار داده میشود .

در میکروکنترلر های سری At91sam آدرس دهی در باس twi میتواند در سه فرمت مطابق جدول زیر انجام شود :

عملکرد	مقدار isize
آدرس داخلی غیر فعال میشود .	0
آدرس داخلی قطعه برابر با یک بایت است .	1
آدرس داخلی قطعه برابر با دو بایت است .	2
آدرس داخلی قطعه برابر با سه بایت است .	3

با قرار دادن مقادیر 1 تا 3 به جای unsigned char isize میتوان تعداد بیت های آدرس را مشخص نمود .

توجه داشته باشید که :

باس twi معمولا دارای ادرس یک بایت (یا 8 بیتی) است ، اما در میکروکنترلر های at91sam امکان آدرس دهی به صورت 2 یا 3 بایتی نیز وجود دارد .با افزایش سایر آدرس میتوان دستگاه های بیشتری را در شبکه ی twi قرار داد ، مثلا اگر آدرس داخلی دارای 3 بایت باشد حداکثر میتوان تا 16777216 عدد دستگاه را در یک باس TWI با هم شبکه کرد

توجه داشته باشید که در پیکربندی در مد اسلیو داده ی انتسابی به unsigned char address عملکردی نخواهد داشت و این متغیر تنها برای یکپارچه سازی دستورات در این تابع برای مد اسلیو قرار داده شده است (ما میتوانستیم یک تابع دیگر بدون وجود این دستور برای مد اسلیو ایجاد کنیم)

unsigned char byte متغیری اختیاری است که میتوان در صورت نیاز ، با استارت شدن باس آن را به دستگاه جانبی ارسال نمود .

```
void TWI_Stop(AT91S_TWI *pTwi)
```

با دستور بالا فرمان توقف به باس ارسال میشود ، در این حالت باس تا ارسال بیت استارت بعدی توسط میکروکنترلر یا سایر لوزام موجود در باس ، آزاد میشود .

در صورتی که بیت STOP ارسال نشود ، بعد از ارسال 8 بیت داده ی آخر ، بیت بعدی میتواند به عنوان بیت ACK در نظر گرفته شود ، این حالت برای ارسال داده های با طول بیشتر از 8 بیت استفاده میشود . توجه داشته باشید که عدم ارسال بیت توقف باعث ایجاد خطا در عمکرد باس شده و کلیه داده های بعدی نامعتبر خواهد بود .

برای انتقال داده در باس TWI از دو تابع زیر استفاده میشود :

```
void TWI_WriteByte(AT91S_TWI *pTwi, unsigned char byte);

unsigned char TWI_ReadByte(AT91S_TWI *pTwi);
```

از دستور اول برای ارسال داده به باس و از دستور دوم برای خواند داده از باس استفاده میشود ، توجه داشته باشد که تمامی داده های ارسالی و دریافتی از باس حداکثر میتوانند 8 بیت طول داشته باشند (از 0 تا 255 باشند) برای خواندن یا نوشتن داده های با طول بیشتر باید انها را در بخش های 8 بیتی ارسال کنید . در این دستورات AT91S_TWI *pTwi مانند حالت قبلی نام باس TWI است که قصد کار با آن را داریم و unsigned char byte متغیر یا عددی است که به باس ارسال میشود ، دستور TWI_ReadByte حاوی مقدار دریافتی از باس است ، مثال :

```
TWI_ConfigureMaster(AT91C_BASE_TWI, 400000, 48000000);

TWI_StartWrite(AT91C_BASE_TWI,45,23,1,0);

TWI_WriteByte(AT91C_BASE_TWI ,15);

TWI_Stop(AT91S_TWI *pTwi);
```

در مثال بالا باس TWI میکروکنترلر AT91SAM7X256 با فرکانس کاری 40 کیلو هرتز پیکربندی شده است ، در این مثال فرکانس کاری میکروکنترلر برابر با 48 مگاهرتز است .

با دستور TWI_StartRead ، ادرس داخلی قطعه برابر با 32 و آدرس قطعه ی جانبی برابر با 23 در نظر گرفته میشود ، همچنین حداکثر طول آدرس برابر با یک بایت است .

با دستور TWI_WriteByte(AT91C_BASE_TWI ,15); عدد 15 توسط واحد TWI به باس ارسال میشود .

مثال :

```
TWI_ConfigureSlave(AT91C_BASE_TWI ,23);

TWI_StartRead(AT91C_BASE_TWI, 23, 32,1);

Int a;

a= TWI_ReadByte(AT91C_BASE_TWI);

TWI_Stop(AT91S_TWI *pTwi);
```

در مثال بالا میکروکنترلر در مد اسلیو پیکربندی شده است ، در این حالت با دستور TWI_ReadByte داده ی دریافتی از باس در متغیر a ریخته میشود .

سایر دستورات و توابع موجود در کتابخانه TWI.H :

در این کتابخانه دستورات دیگری نیز به شرح زیر وجود دارد ، که کاربران در صورت نیاز میتوانند از انها استفاده کنند :

```
extern unsigned char TWI_ByteReceived(AT91S_TWI *pTwi);

extern unsigned char TWI_ByteSent(AT91S_TWI *pTwi);

extern unsigned char TWI_TransferComplete(AT91S_TWI *pTwi);

TWI_EnableIt(AT91S_TWI *pTwi, unsigned int sources);

TWI_DisableIt(AT91S_TWI *pTwi, unsigned int sources);

TWI_GetStatus(AT91S_TWI *pTwi);
```

با دستور بالا وضعیت بیت های مختلف واحد TWI را میتوان مشاهده کرد ، وضعیت های متناظر با عدد برگشتی تابع بالا در زیر آورده شده است :

مقدار برگشتی	وضعیت

TWI_GetMaskedStatus(AT91S_TWI *pTwi);

با دستور بالا وضعیت بیت های مختلف واحد TWI را میتوان مشاهده کرد ، وضعیت های متناظر با عدد برگشتی تابع بالا در زیر آورده شده است :

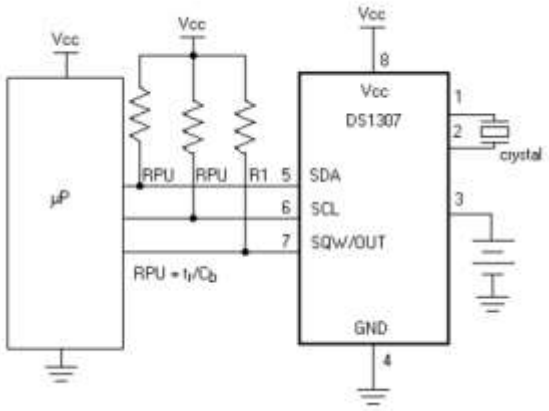
extern void TWI_SendSTOPCondition(AT91S_TWI *pTwi);

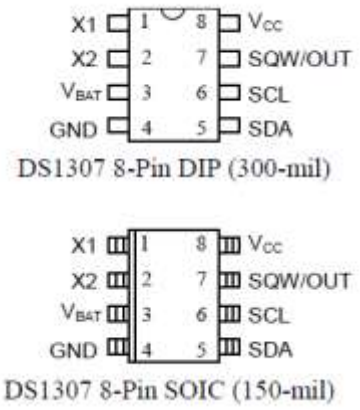
با دستور بالا فرمان توقف به باس ارسال میشود ، در این حالت باس تا ارسال بیت استارت بعدی توسط میکروکنترلر یا سایر لوزام موجود در باس ، آزاد میشود . (این دستور دقیقا مانند دستور TWI_Stop(AT91S_TWI *pTwi) است)

مروری بر DS1307:

DS1307 یک چیپ RTC (Real-time clock) میباشد که میتواند به صورت خودکار ثانیه ، دقیقه ، ساعت ، روز ، ماه ، روز هفته و سال را شمارش کند ، این ایسی بسیار کم مصرف بوده و وجود قابلیت های همچون 56 بایت RAM داخلی برای ذخیره ی اطلاعات ، قابلیت ارسال اطلاعات به صورت رابط سریال Two-wire (I2C) و قابلیت اتصال باطری بک

آپ و مصرف جریان کمتر از 500 نانو آمپر از آن و... باعث محبویت آن در ساخت انواع ساعت های دیجیتال و ... شده است . در زیر شکل قطعه و مدار مورد نیاز برای اتصال آن به میکرو را مشاهده میکنید :





در تصویر بالا پایه ی SDA ، پایه ی انتقال داده است که باید به پایه ی انتقال داده ی میکرو کنترلر (TWD) متصل شود ، همچنین پایه ی کلاک قطعه که با نام SCL مشخص شده است باید به پایه ی کلاک میکرو (TWCK) متصل شود . در باس I2C این قطعه به عنوان slave استفاده میشود .

پایه ی SQW/OUT یکی از پین های اختیاری است که توسط کاربر میتواند به عنوان خروجی کلاک پیکربندی شده و پالسی با فرکانس 1Hz, 4kHz, 8kHz, 32kHz روی آن ایجاد شود .

پایه های VCC و GND تامین کننده ی تغذیه ی قطعه بوده و باید به 5 ولت متصل شوند . در این حالت قطعه در حالت عادی کار کرده و میکرو میتواند از طریق باس داده ی آن را بخواند . هنگامی که تغذیه از VCC قطع شود ، قطعه توان مورد نیاز خود را از باطری 3 ولتی که به پایه ی VBAT متصل شده است تامین میکند ، در این حالت میکرو دیگر نمیتواند با قطعه ارتباط برقرار کنند و تنها عملیات شمارش زمان و تاریخ در قطعه انجام میشود .

به پایه های X1 و X2 باید یک کریستال 32.768kHz متصل شود ، این کریستال وظیفه ی تامین کردن کلاک مورد نیاز قطعه برای شمارش زمان را به عهده خواهد داشت .

باس SPI و نحوه ی کار با آن

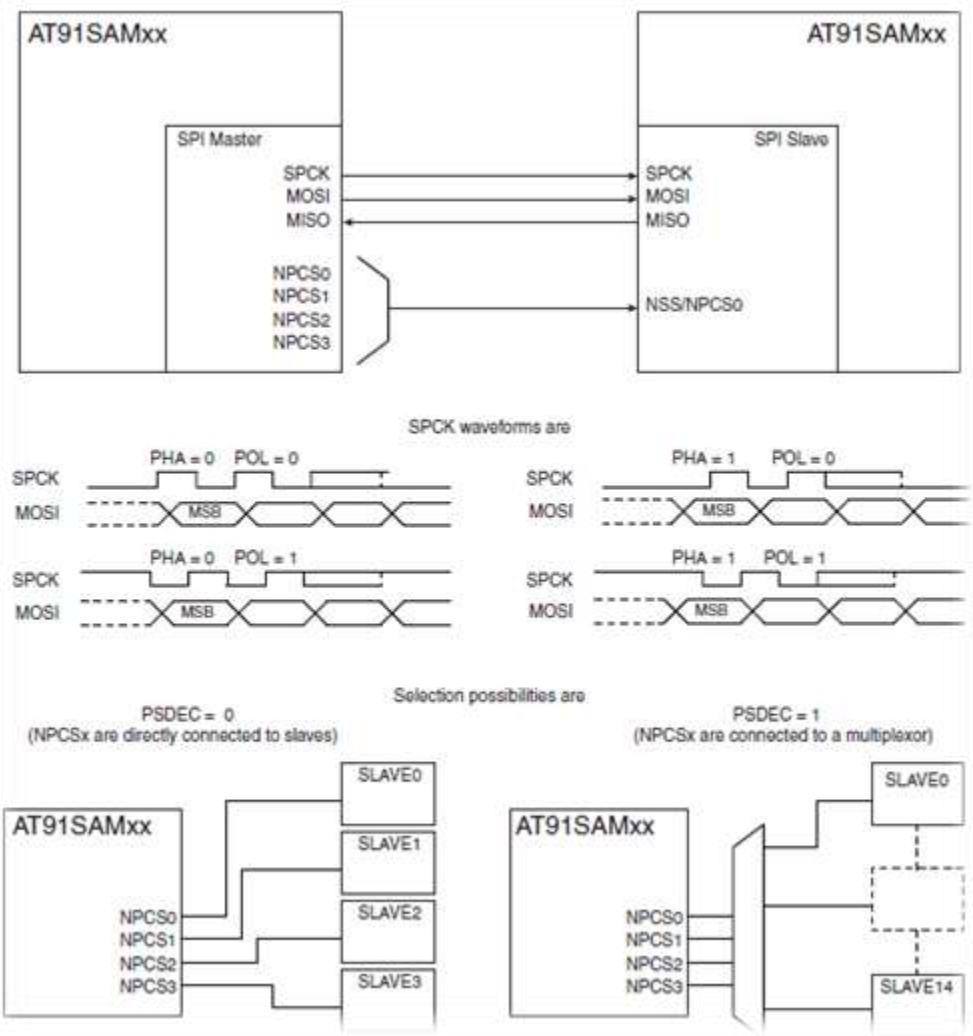
Serial Peripheral Interface یا SPI یک رابط سریال همزمان میباشد که توسط آن میتوانید انواع لوزام جانبی نظیر حافظه های mmc/sd ، eeprom های پر سرعت ، مبدل های دیجتال به آنالوگ و آنالوگ به دیجیتال و... را به میکرو کنترلر متصل کنید . از این باس برای ایجاد شبکه های میکرو کنترلری نیز استفاده میشود .

میکرو کنترلر های arm اتمل معمولا دارای 1 یا دو باس spi با مشخصات زیر هستند :

- ✓ دارای 4 پایه ی انتخاب کننده (Chip Selects) مستقل برای انتخاب کردن لوازم جانبی با قابلیت پشتیبانی از قطعات انکودر (با استفاده از یک انکودر 4 به 16 میتوانید 15 قطعه جانبی را بدون هیچ مشکلی در سرعت یا همزمان سازی کنترل کنید).
- ✓ ارتباط ساده با حافظه های سریال نظیر DataFlash و حافظه های eeprom سه سیمه (3-wire EEPROMs)
- ✓ قابلیت ارتباط با کلیه قطعات سازگار با spi نظیر انواع adc و dac سریال ، کنترل کننده های can و شبکه و lcd ، سنسورها ،....
- ✓ برنامه ریزی طول داده انتقالی بین 8 تا 16 بیت .

- ✓ کنترل ساده ی نوبت و لبه ی (صفر به یک یا یک به صفر) پایه های cs.
- ✓ برنامه ریزی فرکانس خروجی از میکرو ، تاخیر در لبه های sc ، تاخیر در تبدیل و... .
- ✓ پیکربندی در دو مد مستر و اسلیو.
- ✓ دارای آشکارساز خطای توسعه یافته .
- ✓ و...

در تصویر زیر نحوه ی اتصال چند وسیله از طریق باس spi آورده شده است ، همانطور که قبلا نیز گفته شد در این باس خطوط mosi و miso و sck بین تمامی قطعات مشترک هستند و این خطوط NPCSx هستند که تعیین میکنند کدام میکرو اطلاعات را از مستر بگیرد یا برای ان ارسال کند :



در این باس میکرو کنترلر master میتواند به صورت همزمان به چند وسیله متصل شده و اطلاعاتی را برای آنها ارسال نماید ، این کار با یک کردن پایه های cs (chip select) دستگاه های مذکور انجام می شود . برای خواندن اطلاعات خروجی یک قطعه ، master باید پایه ی cs آن را یک کرده و اطلاعات موجود در بافر خروجی آن را بخواند ، در هنگام خواندن اطلاعات master می تواند فقط به یک وسیله متصل شود .

بررسی رجیستر های باس SPI

برای کار با واحد SPI نیز همچون سایر بخش های داخلی ، رجیستر های در نظر گرفته شده است ، این رجیستر ها به شرح زیر است :

نام ریسجتر	نام کامل	توصیف
SPI_CR	Control Register	رجیستر کنترل واحد SPI ، تنظیمات اصلی باس با این رجیستر انجام میشود.
SPI_MR	Mode Register	رجیستر تنظیمات ، تنظمات جانبی باس (مستر یا اسلیو و...) با این رجیستر انجام میشود
SPI_RDR	Receive Data Register	رجیستر دریافت داده ، داده ی دریافتی از باس در این رجیستر ذخیره میشود .
SPI_TDR	Transmit Data Register	رجیستر ارسال داده ، داده ی که باید به باس ارسال شود در این رجیستر ریخته میشود .
SPI_SR	Status Register	رجیستر وضعیت ، وضعیت بخش های مختلف باس از طریق این رجیستر قابل خواندن است .
SPI_IER	Interrupt Enable Register	رجیستر فعال ساز وقفه ، برای فعال کردن وقفه ی بخش های مختلف از این رجیستر استفاده میشود
SPI_IDR	Interrupt Disable Register	رجیستر غیر فعال کننده ی وقفه ، برای غیر فعال کردن وقفه ی بخش های مختلف از این رجیستر استفاده میشود
SPI_IMR	Interrupt Mask Register	رجیستر تعیین کننده ی پوشش وقفه ، تعیین عملیات متناظر با نوع وقفه در این رجیستر صورت میگیرد .
SPI_CSR0	Chip Select Register 0	رجیستر انتخاب کننده ی قطعه (chip select) ، وضعیت پایه ی NPCS0
SPI_CSR1	Chip Select Register 1	رجیستر انتخاب کننده ی قطعه (chip select) ، وضعیت پایه ی

		NPCS1
SPI_CSR2	Chip Select Register 2	رجیستر انتخاب کننده ی قطعه (chip select) ، وضعیت پایه ی NPCS2
SPI_CSR3	Chip Select Register 3	رجیستر انتخاب کننده ی قطعه (chip select) ، وضعیت پایه ی NPCS3

در ادامه به بررسی نحوه ی مقدار دهی رجیستر ها و بیت های موجود در هر یک پرداخته ایم .

رجیستر SPI_CR (SPI Control Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	LASTXFER
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	–
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
SWRST	–	–	–	–	–	SPIDIS	SPIEN

بیت SPIEN (SPI Enable) : نوشتن مقدار یک در این بیت باعث فعال شدن باس spi میشود ، نوشتن مقدار دیگر تاثیری در عملکرد باس ندارد .

بیت SPIDIS (SPI Disable) : نوشتن رقم یک در این بیت باعث غیر فعال شدن باس میگردد ، در این حالت تمامی پایه ها در وضعیت ورودی قرار گرفته و داده ای در باس مبادله نمیشود . (در صورتی که در هنگام انتقال داده این بیت یک شود ، spi بعد از انتقال اطلاعات موجود غیر فعال میشود . نوشتن مقدار دیگر تاثیری در عملکرد باس ندارد .

بیت SWRST (SPI Software Reset) : با نوشتن یک در این بیت ، باعث Spi ریست میشود و در مد slave پیکربندی میشود . برای باز گشت به مد قبلی ، رجیستر SPI_MR باید مجددا مقدار دهی شود . نوشتن مقدار دیگر تاثیری در عملکرد باس ندارد .

بیت LASTXFER (Last Transfer): در حالت عادی پایه های CS (انتخاب قطعه) بعد از ارسال داده توسط MASTER در حالت خود میماند (با توجه به نوع قطعه و سطحی که فعال میشود ، ممکن است این پایه به سطح صفر یا یک منتطقی برود) . با مقدار دهی این بیت و بیت CSAAT (Chip Select Active After Transfer) با مقدار یک ، هنگامی که ارسال داده به پایان رسید ، پایی CS مربوطه تغییر وضعیت داده و قطعه ی جانبی را از خط خارج میکند .

رجیستر SPI_MR (SPI Mode Register):

31	30	29	28	27	26	25	24
DLYBCS							
23	22	21	20	19	18	17	16
–	–	–	–	PCS			
15	14	13	12	11	10	9	8
–	–	–	–	–	–	–	–
7	6	5	4	3	2	1	0
LLB	–	–	MODFDIS		PCSDEC	PS	MSTR

بیت MSTR (Master/Slave Mode) : با مقدار دهی این بیت میتوانید مشخص کنید که میکرو در چه مدی پیکربندی شود ، MSTR=0 باس را در مد Slave و MSTR=1 میکرو را در مد Master پیکربندی میکند .

هنگامی که باس spi در حالت Master پیکربندی میشود ، پایه های NPCS0-3 در حالت خروجی پیکربندی میشوند . در این حالت واحد SPI با استفاده از تنظیماتی که در ادامه به بررسی آنها میپردازیم پایه های SPCK را به عنوان خروجی پیکربندی کرده و پالس کلاکی را بر روی آن ایجاد میکند ، در این حالت پایه های MOSI به عنوان خروجی اطلاعات وظیفه ی دریافت اطلاعات از خروجی شیفت رجیستر و پایه ی MISO وظیفه ی دریافت اطلاعات از روی باس را به عهده میگیرد .

در هنگام پیکربندی باس SPI در حالت Slave نیز حالتی مشابه پیکربندی در مد Master رخ میدهد ، در مد Slave پایه های پایه های MOSI به عنوان خروجی اطلاعات وظیفه ی دریافت اطلاعات از خروجی شیفت رجیستر و پایه ی MISO وظیفه ی دریافت اطلاعات از روی باس را به عهده میگیرد . همچنین پایه ی SPCK و NPCS0 در حالت ورودی قرار گرفته و به ترتیب وظیفه ی دریافت پالس همزمانی و سیگنال CS (انتخاب قطعه) از Master را به عهده میگیرند . در مد Slave از پایه های NPCS1-3 استفاده نمیشود و شما میتوانید از آنها برای کاربرد های دیگر استفاده نمایید ، همچنین در این مد قطعه فقط در صورتی که سیگنال CS را دریافت کند وارد شبکه میشود (به Master متصل میشود)

بیت PS (Peripheral Select) : انتخاب رقم یک برای این بیت ، باس را برای ارتباط با چندین وسیله پیکربندی میکند ، انتخاب رقم صفر باعث میشود تا باس برای ارتباط با یک وسیله ی جانبی پیکربندی شود .

بیت PCSDEC (Chip Select Decode) :

اگر این بیت صفر باشد ، باس spi برای ارتباط با 4 وسیله ی جانبی پیکربندی میشود ، این بدین مفهموم است که پایه های cs باید مستقیما به ورودی cs دستگاه های جانبی متصل شوند . در صورتی که این بیت 1 شود ، واحد spi برای

ارتباط با یک .decoder چهار به 16 بیت پیکربندی میشود در این حالت میتوانیم تا 16 وسیله ی جانبی را با مقدار دهی بیت های SPI_CSR0-3 (که در ادامه به بررسی آنها خواهیم پرداخت) آدرس دهی نماییم .

بیت MODFDIS (Mode Fault Detection) :

از این بیت برای فعال کردن مد تشخیص خطا (با مقدار دهی صفر) و غیر فعال کردن آن استفاده میشود (با مقدار دهی یک) .

بیت LLB (Local Loopback Enable) :

یک بودن این بیت باعث فعال شدن حلقه ی فیدبک داخلی و صفر بودن آن باعث غیر فعال شدن حلقه میشود .

هنگامی که حلقه ی فیدبک داخلی فعال باشد ، اطلاعات خروجی شیفت رجیستر به پایه ی MISO اعمال میشود . در این حالت باس می تواند صحت اطلاعات خروجی را بررسی نیمده و در صورت وجود خطا آنها را مجددا ارسال کند (این حالت فقط در مد Master فعال میشود)

بیت ها PCS (Peripheral Chip Select) :

از این بیت ها در حالتی که بیت PS (Peripheral Select) صفر باشد استفاده میشود (میکرو فقط به یک وسیله ی جانبی متصل باشد) ، با استفاده از این بیت ها میتوانید وضعیت پایه های NPCS0-3 را کنترل نمایید :

If PCSDEC = 0:
PCS = xxx0 NPCS[3:0] = 1110
PCS = xx01 NPCS[3:0] = 1101
PCS = x011 NPCS[3:0] = 1011
PCS = 0111 NPCS[3:0] = 0111
PCS = 1111 forbidden (no peripheral is selected)
(x = don't care)
If PCSDEC = 1:
NPCS[3:0] output signals = PCS.

بیت های DLYBCS (Delay Between Chip Selects) :

بیت های DLYBCS مقدار تاخیر میان فعال شدن هر پایه ی NPCS را طبق فرمول زیر مشخص میکنند :

$$\text{Delay Between Chip Selects} = \frac{DLYBCS}{MCK}$$

وجود تاخیر میان فعال شدن پایه های cs امکان وارد متصل شدن دو وسلیه ی slave به صورت همزمان به مستر را صفر میکند .

رجیستر SPI_RDR (SPI Receive Data Register) :

از این رجیستر برای دریافت اطلاعات از باس spi استفاده میشود .

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	PCS			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
RD							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
RD							

داده های دریافت شده از باس spi در 16 بیت اول این رجیستر (به جای بیت های RD (Receive Data)) ذخیره میشوند ، همچنین بیت های PCS (Peripheral Chip Select) در مد Master برای ذخیره کردن وضعیت پایه های NPCS استفاده میشود (میکرو با توجه به وضعیت این بیت ها متوجه میشود داده ی دریافتی از کدام وسیله ی چانبی دریافت شده است .

رجیستر SPI_TDR (SPI Transmit Data Register) :

از این رجیستر برای ارسال اطلاعات به باس استفاده میشود

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	LASTXFER
23	**22**	**21**	**20**	**19**	**18**	**17**	**16**
–	–	–	–	PCS			
15	**14**	**13**	**12**	**11**	**10**	**9**	**8**
TD							
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**
TD							

اطلاعاتی که باید به باس SPI ارسال شوند در این رجیستر و به جای بیت های TD (Transmit Data) نوشته میشوند ، همچنین بیت های PCS (Peripheral Chip Select) در حالتی که میکرو با چندین دستگاه جانبی ارتباط دارد استفاده میشود (بیت ps در رجیستر SPI_MR برابر 1 باشد) ، در این حالت :

If PCSDEC = 0:

PCS = xxx0 NPCS[3:0] = 1110

PCS = xx01 NPCS[3:0] = 1101

PCS = x011 NPCS[3:0] = 1011

PCS = 0111 NPCS[3:0] = 0111

PCS = 1111 forbidden (no peripheral is selected)

(x = don't care)

If PCSDEC = 1:

NPCS[3:0] output signals = PCS

قبلا گفتیم که یک بودن PCSDEC به این مفهوم است که خطوط cs به یک دیکودر 4 به 16 متصل شده اند و میکرو مستر باید تا 16 وسلیه ی جانبی را کنترل کند و همچنین صفر بودن به مفهوم اتصال مستقیم خطوط cs به لوازم جانبی میباشد (کنترل تا 4 وسلیه ی جانبی) .

با مقدار دهی بیت های PCS میتوانید سطح منطقی پایه های NPCS0-3 را کنترل نمایید ، برای مثال هنگامی که این بیت ها با رقم 0011 یا 1011 مقدار دهی شوند ، پایه ی NPCS2 در سطح منطقی یک و سایر پایه ها در سطح منطقی یک قرار خواهند گرفت .

بیت LASTXFER (Last Transfer) : در حالت عادی پایه های CS (انتخاب قطعه) بعد از ارسال داده توسط MASTER در حالت خود میماند (با توجه به نوع قطعه و سطحی که فعال میشود ، ممکن است این پایه به سطح صفر یا یک منتطقی برود) . با مقدار دهی این بیت و بیت CSAAT (Chip Select Active After Transfer) با مقدار یک ، هنگامی که ارسال داده به پایان رسید ، پایی CS مربوطه تغییر وضعیت داده و قطعه ی جانبی را از خط خارج میکند .

این بیت فقط در حالتی که مستر با چندین دستگاه جانبی ارتباط بر قرار میکند ، کابرد دارد (PS = 1)

رجیستر SPI_SR (SPI Status Register) :

31	30	29	28	27	26	25	24
–	–	–	–	–	–	–	–
23	22	21	20	19	18	17	16
–	–	–	–	–	–	–	SPIENS
15	14	13	12	11	10	9	8
–	–	–	–	–	–	TXEMPTY	NSSR
7	6	5	4	3	2	1	0
TXBUFE	RXBUFF	ENDTX	ENDRX	OVRES	MODF	TDRE	RDRF

از این رجیستر برای دریافت وضعیت بخش های مختلف باس SPI استفاده میشود . در ادامه به بررسی بیت های این رجیستر پرداخته ایم ، در این بخش ":0" به مفهوم وجود مقدار صفر در بیت و ":1" به مفهوم یک بودن بیت است .

بیت RDRF (Receive Data Register Full) :

0 : اطلاعات جدید در رجیستر SPI_RDR (SPI Receive Data Register) وجود ندارد .

1: اطلاعات جدید در رجیستر SPI_RDR (SPI Receive Data Register) وجود دارد .

بیت TDRE(Transmit Data Register Empty):

0: داده ی نوشته ی شده در رجیستر SPI_TDR هنوز ارسال نشده است (این رجیستر پر است) .

1: داده ی نوشته ی شده در رجیستر SPI_TDR هنوز ارسال شده است (این رجیستر خالی است) .

بیت MODF(Mode Fault Error):

0: خطایی در انتقال داده رخ نداده است .

1: در انتقال داده خطایی رخ داده است .

بیت OVRES(Overrun Error Status):

بعضی مواقع ممکن است رجیستر SPI_RDR (دریافت داده) ، داده ای بیشتر از حجمش را دریافت کند (مثلا طول داده برابر 17 بایت باشد یا داده ای چندین بار به صورت متوالی دریافت شود ، در این حالت خطایی به نام overrun رخ میدهد که با خواندن این بیت میتوان رخ دادن آن را چک کرد .

0: خطای overrun رخ نداده است

1 : خطای overrun رخ داده است .

بیت ENDRX(End of RX buffer):

0 = The Receive Counter Register has not reached 0 since the last write in SPI_RCR(1) or SPI_RNCR(1).

1 = The Receive Counter Register has reached 0 since the last write in SPI_RCR(1) or SPI_RNCR(1).

بیت ENDTX (End of TX buffer):

0 = The Transmit Counter Register has not reached 0 since the last write in SPI_TCR(1) or SPI_TNCR(1).

1 = The Transmit Counter Register has reached 0 since the last write in SPI_TCR(1) or SPI_TNCR(1).

بیت RXBUFF (RX Buffer Full):

0: بیت های SPI_RCR یا SPI_RNCR صفر نیستند .

1: بیت های SPI_RNCR و SPI_RCR صفر هستند .

بیت TXBUFE (TX Buffer Empty):

0: بیت های SPI_TCR یا SPI_TNCR صفر نیستند .

1: بیت های SPI_TCR یا SPI_TNCR صفر هستند .

بیتNSSR (NSS Rising):

0 = No rising edge detected on NSS pin since last read.

1 = A rising edge occurred on NSS pin since last read.

بیتTXEMPTY (Transmission Registers Empty)

0 = As soon as data is written in SPI_TDR.

1 = SPI_TDR and internal shifter are empty. If a transfer delay has been defined, TXEMPTY is set after the completion

of

such delay.

بیت SPIENS (SPI Enable Status)

0: باس SPI غیر فعال است .

1: باس SPI فعال است .

رجیستر های SPI_CSR0... SPI_CSR3 (SPI Chip Select Register)

31	30	29	28	27	26	25	24
DLYBCT							
23	22	21	20	19	18	17	16
DLYBS							
15	14	13	12	11	10	9	8
SCBR							
7	6	5	4	3	2	1	0
BITS				CSAAT	–	NCPHA	CPOL

· CPOL: Clock Polarity
0 = The inactive state value of SPCK is logic level zero.
1 = The inactive state value of SPCK is logic level one.
CPOL is used to determine the inactive state value of the serial clock (SPCK). It is used with NCPHA to produce the
required clock/data relationship between master and slave devices.
· NCPHA: Clock Phase
0 = Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.
1 = Data is captured on the leading edge of SPCK and changed on the following edge of SPCK.
NCPHA determines which edge of SPCK causes data to change and which edge causes data to be captured. NCPHA is
used with CPOL to produce the required clock/data relationship between master and slave devices.
· CSAAT: Chip Select Active After Transfer
0 = The Peripheral Chip Select Line rises as soon as the last transfer is achieved.
1 = The Peripheral Chip Select does not rise after the last transfer is achieved. It remains active until a new transfer is
requested on a different chip select.
· BITS: Bits Per Transfer
The BITS field determines the number of data bits transferred. Reserved values should not be used.

BITS	Bits Per Transfer		BITS	Bits Per Transfer
0000	8		1000	16
0001	9		1001	رزور شده برای مصارف بعدی
0010	10		1010	رزور شده برای مصارف بعدی
0011	11		1011	رزور شده برای مصارف بعدی
0100	12		1100	رزور شده برای مصارف بعدی
0101	13		1101	رزور شده برای مصارف بعدی
0110	14		1110	رزور شده برای مصارف بعدی
0111	15		1111	رزور شده برای مصارف بعدی

· SCBR: Serial Clock Baud Rate
In Master Mode, the SPI Interface uses a modulus counter to derive the SPCK baud rate from the Master Clock MCK. The
Baud rate is selected by writing a value from 1 to 255 in the SCBR field. The following equations determine the SPCK baud
rate:

$$\text{SPCK Baudrate} = \frac{MCK}{SCBR}$$

Programming the SCBR field at 0 is forbidden. Triggering a transfer while SCBR is at 0 can lead to unpredictable results.
At reset, SCBR is 0 and the user has to program it at a valid value before performing the first transfer.
· DLYBS: Delay Before SPCK
This field defines the delay from NPCS valid to the first valid SPCK transition.
When DLYBS equals zero, the NPCS valid to SPCK transition is 1/2 the SPCK clock period.
Otherwise, the following equations determine the delay:

$$\text{Delay Before SPCK} = \frac{DLYBS}{MCK}$$

· DLYBCT: Delay Between Consecutive Transfers
This field defines the delay between two consecutive transfers with the same peripheral without removing the chip select.
The delay is always inserted after each transfer and before removing the chip select if needed.
When DLYBCT equals zero, no delay between consecutive transfers is inserted and the clock keeps its duty cycle over the
character transfers.
Otherwise, the following equation determines the delay:

$$\text{Delay Between Consecutive Transfers} = \frac{32 \times DLYBCT}{MCK}$$

برای باس SPI سه رجیستر SPI_IER و SPI_IDR و SPI_IMR برای فعال و غیر فعال و کار با وقفه ی بخش های مختلف باس در نظر گرفته است ، ما بررسی این رجیستر ها را به فصل آخر و بعد از بررسی نحوه ی کار با بخش ACI موکول میکنیم .

SSC و نحوه ی راه اندازی آن در keil

SSC یکی از امکانات موجود در میکروکنترلر های مبتنی بر هسته ی 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 .

همانطور که گفتیم در این سیستم از سه سیم برای انتقال داده استفاده می شود ، هر یک از این خطوط دارای وظیفه خاص خود است:

SD (Serial Data)line :

برای انتقال داده بین گیرنده و فرستنده به دو خط داده نیاز داریم ، خط اول صدا را از DAC یا وسیله مبدل به پردازنده می برد ، خط دوم می تواند دستورات مربوط به صدای دیجیتال (که قرار است بعد تبدیل شدن به ولتاژ انالوگ از بلندگو بخش شود) یا سایر اطلاعات را از پردازنده به DAC یا وسیله جانبی منتقل کند .

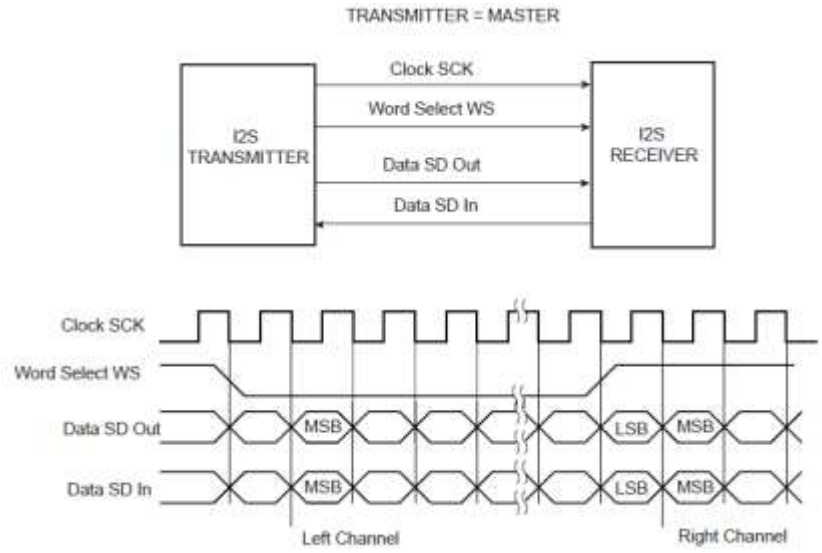
WS(a left/right channel Word Select) :

این خط می تواند وسیله سمت راست یا چپ را انتخاب کند ، حتما تا بحال به صدای استریو گوش داده اید....

: SCK (a continuous Serial Clock)

بر روی این خط یک پالس کلاک برای همزمان سازی گیرنده و فرستنده ایجاد می شود ، ایجاد پالس توسط مستر یا پردازنده اصلی صورت می گیرد .

با توجه به توضیحات بالا بلوک دیاگرام مدار ما به صورت زیر خواهد بود :



در میکرو کنترلر های سری AT91SAM سخت افزاری به نام i2s وجود ندارد ، اما ما می توانیم از باس SSC به عنوان این باس استفاده کنیم و تعدادی از قطعات i2s که امکان دریافت فرمان از دیگر باس ها را دارند را به ARM متصل کنیم .

SSC چیست ؟

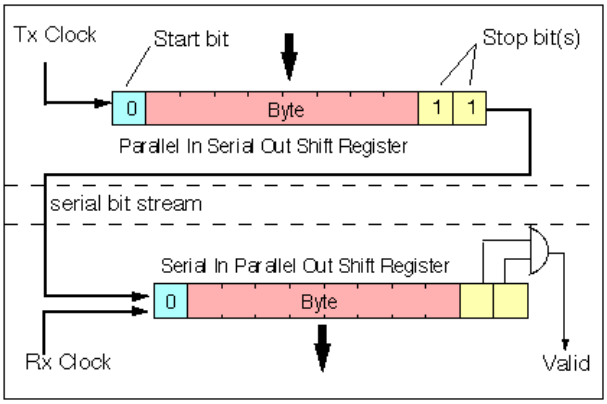
پروتکل SSC یک رابط همزمان برای ارتباط با کلیه قطعات و لوازمی می باشد که به صورت همزمان اقدام به ارسال و دریافت داده می کنند ، برای نمونه می توان به I2S, Short Frame Sync, Long Frame Sync, اشاره کرد ، کلیه این پروتکل ها داده را به صورت همزمان به خروجی ارسال می کنند ، شاید با مفهوم ارسال همزمان و غیر همزمان مشکل داشته باشید ، در زیر به معرفی این دو پرداخته ایم :

مبادله آسنکرون (Asynchronous Transmission)

در این روش ، انتقال داده به صورت غیر همزمان انجام می شود ، در واقع بین فرستنده و گیرنده همزمانی وجود ندارد . فرستنده داده موجود را ارسال می کند و گیرنده نیز آن را دریافت خواهد نمود . در این میان بیت های استارت و همزمانی ، زمان ارسال داده و صحیح بودن آن را مشخص می کنند .

در مبادله آسنکرون هر داده کاراکتری، یک بیت شروع (start bit) که مبین شروع آن کاراکتر است و همچنین یک یا دو بیت پایان که تعیین کننده پایان آن کاراکتر می باشد.

در این رابط ، داده ارسالی معمولا یک بایت (8بیت) می باشد . به دنبال بیت های داده، یک بیت توازن (parity bit) وجود دارد که دریافت کننده آن را به منظور خطایابی مورد استفاده قرار می دهد. پس از اینکه بیت Parity ارسال شد. سیگنال باید حداقل برای مدت زمان یک بیت دارای ارزش 1 شود تا پایان کاراکتر را معین کند. بیت جدید شروع (Start bit) شروع موجب می شود که وسیله دریافت کننده از آمدن یک کاراکتر داده، آگاه شود و ساعت خود را سنکرون سازد. در صورتی که بیت Parity داده قبلی صحیح باشد ، گیرنده برای دریافت ادامه اطلاعات آماه می شود ، در غیر این صورت مجددا اطلاعات قبلی را درخواست می کند :



پورت های RS232 و RS458 و... نمونه های از Asynchronous Transmission هستند .

مبادله سنکرون (Synchronous Transmission)

در این روش نیازی به بیت های شروع و پایان و... نیست و پالس کلاک وظیفه ایجاد همزمانی میان دو دستگاه را برعهده دارد .

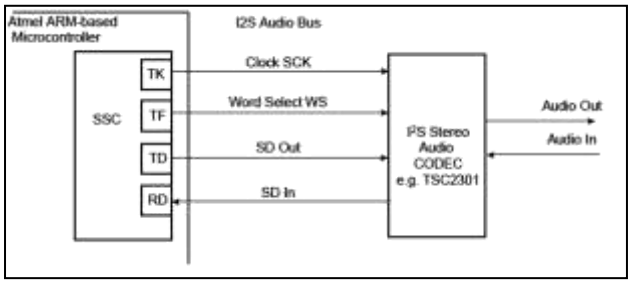
در این روش در هر پالس کلاک یک بیت از اطلاعات ارسال می شود ، بنابراین ما می توانیم با تغییر دادن فرکانس کلاک سرعت انتقال را عملا تغییر دهیم ، پروتکل SPI و i2s و.... نمونه های از Synchronous Transmission هستند .

با توجه به مطالب بالا می توانیم به کاربرد وسیع باس SSC در راه اندازی انواع قطعات جانبی پی ببریم ، این باس دارای 6 پایه به شرح زیر است :

نام پایه	Pin Description	توضیحات
RF	Receiver Frame Synchro	ورودی دستورات (از مستر)
RK	Receiver Clock Input	ورودی پالس همزمانی (از مستر)
RD	Receiver Data	ورودی داده (از مستر)
TF	Transmitter Frame Synchro	خروجی فرمان (به اسلیو)
TK	Transmitter Clock	خروجی پالس همزمانی (به اسلیو)
TD	Transmitter Data	خروجی داده (به اسلیو)

توضیحات :

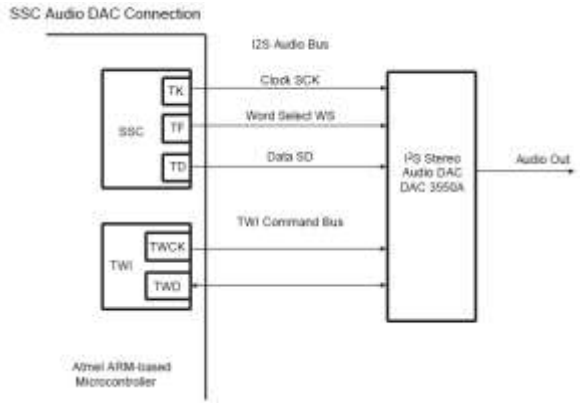
- از این باس می توان در حالت یک طرفه یا دوطرفه استفاده کرد ، مثلا با تراشه. TSC2301 میتوانیم یک سیستم صوتی Stereo ایجاد کنیم ، این سیستم علاوه بر پخش صدا قادر به ضبط آن نیز خواهد بود .



در سیستم بالا صدای آنالوگ توسط TSC2301 به داده های دیجیتال تبدیل می شود و از طریق خط RD به میکرو وارد می شود ، همچنین اطلاعات صدای دیجیتال که توسط میکرو از mmc یا ... خوانده شده است به این قطعه ارسال می

شود و بعد از تبدیل شدن به ولتاژ از بلندگو بخش می شود ، خطوط tk و tf نیز به ترتیب وظیفه ایجاد پالس همزمانی و ارسال دستورات (منظور از دستورات فریم های همزمانی ، دستورات کنترلی و... می باشد) را به عهده دارد .

- از دیگر قطعات معروف که توسط باس i2s کنترل می شوند می توانیم تراشه DAC 3550A را معرفی کنیم ، در این تراشه علاوه بر تبدیل صدای دیجیتال به آنالوگ ، امکان ایجاد اعمال مختلف نظیر کم و زیاد کردن صدا ، تغییر باس و بم و.... نیز از طریق باس i2c وجود دارد ، برای اطلاعات بیشتر به دیتاشیت این قطعه مراجعه کنید



اتصال dac3550a به میکرو کنتلر arm :

DAC 3550A Hardware Connections

Microcontroller	DAC 3550A	Bus Name
SSC (TFx)	WSI	WS (I^2S)
SSC (TKx)	CLI	SCK (I^2S)
SSC (TDx)	DAI	SD (I^2S)
TWI (TWCK)	SCL	SCL
TWI (TWD)	SDA	SDA

پیکربندی و استفاده از SSC

راه اندازی باس SSC در 4 مرحله انجام میشود :

پیکربندی باس (i/o pins) :

هنگامی که از باس ssc استفاده میکنید ، باید پایه های مورد استفاده در این باس را با توجه به نوع دستگاه جانبی به در مد peripheral پیکربندی کنید ، مثلا در هنگام استفاده از یک dac (در حالتی که قرار است فقط داده ی دیجتال به انالوگ تبدیل شود) از سه پایه ی باس استفاده میشود ، زیرا ارتباط یک طرفه است و میکرو کنترلر در حالت مستر قرار میگرد. در مد peripheral پایه نه ورودی است و نه خروجی ، پس میتواند به سرعت بین ورودی و خروجی تغییر وضعیت دهد .

در این حالت از پایه های TF, TK , TD برای ارتباط با وسیله ی جانبی استفاده میشود .

```
*AT91C_PIOB_PDR= ((unsigned int) AT91C_PB7_TK1 ) |
    ((unsigned int) AT91C_PB8_TD1 ) | ((unsigned int) AT91C_PB6_TF1 );
```

در مرحله ی بعد باید در بخش PMC (Power Management Controller) کلاک این واحد را فعال کنید ، در بخش های قبلی در مورد نحوه ی فعال کردن واحد های مختلف در واحد PMC بحث شد .

در مرحله ی بعد باید ضمن ریست کردن باس ،باید مقدار PDC (شمارنده ی داده)راصفر کنید :

```
pSSC->SSC_CR = AT91C_SSC_SWRST ;

AT91F_PDC_Close((AT91PS_PDC) &(pSSC->SSC_RPR));
```

تعیین پارامتر های SSC:

در این مرحله شما باید پارامتر های همچون فرکانس صدا ی خروجی ، تعداد کانال های خروجی ، طول داده و مقدار فرکانس کاری میکرو را مشخص نمایید .

تعداد کانال های خروجی :

همانطور که قبلا نیز اشاره کردیم در ssc قابلیت پخش صدار استریو نیز وجود دارد ، شما میتوانید با قرار دادن عدد 1 در تابع بالا به جایی عدد2 تعداد کانال خروجی را یک تعیین کنید (عدد دو به معنی خروجی استریو است)

در صورتی که تعداد خروجی 2 باشد ، بر روی پایه ی tf سیگنالی ایجاد میشود که به dac میفهماند که مدام باید بین دو کانال خوجی سوییچ شود ، در این حالت پایه ی td در یک دروه ی زمانی حاوی اطلاعات گانال سمت چپ و د دوره ی زمانی دیگر حاوی اطلاعات کانال سمت راست خواهد بود .

طول داده :

باس i2c میتواند د سه مد استاندارد 16 و 24 و 32 بیت راه اندازی شود ، برای انتقال داده ی صوتی ما از مد 16 بیت استفاده خواهیم نمود ، در این حالت پهنایی باند برابر با 96 دسیبل در اختیار گیرنده قرار میگرد تا داده خود را بدون نویز و تضعیف به فرستنده بفرستد ، مقدار پهنای باند برای سایر مد ها در جدول زیر اورده شده است :

Bit Range versus Dynamic Range/Dynamic Range = 20 log(Bit Range) where bit range = 2^{bit}

Bit Range	Dynamic Range
2^{16}	96 dB
2^{24}	144 dB
2^{32}	196 dB