



دانشگاه صنعتی اصفهان  
دانشکده برق و کامپیوتر

## دستورکار آزمایشگاه طراحی مدارهای واسط

تهیه کننده :مهران صفایانی

۹۵ پاییز

## **فهرست مطالب :**

3.....	معرفی میکروکنترلر های ARM
6.....	آزمایش اول : آشنایی با نرم افزار KEIL uVision5
13.....	آزمایش دوم: آشنایی با پورت های ورودی / خروجی و موتورهای پله ای
19.....	آزمایش سوم: آشنایی با نمایشگرهای ماتریسی نقطه ای، ۱۶ قسمتی و ۷ قسمتی
27.....	آزمایش چهارم: آشنایی با نمایشگرهای ال سی دی کاراکتری
31.....	آزمایش پنجم : آشنایی با نمایشگرهای السیدی گرافیکی و رنگی
37.....	آزمایش ششم: آشنایی با مبدل قیاسی به رقمی و حسگر دما
42.....	آزمایش هفتم: آشنایی با مبدل رقمی به قیاسی
51.....	آزمایش نهم: آشنایی با مدولاسیون عرض پالس و موتورهای DC
56.....	آزمایش دهم: آشنایی با واحد ساعت بیدرنگ RTC
60.....	آزمایش یازدهم: آشنایی با ارتباط SPI
65.....	آزمایشدوازدهم: آشنایی با پروتکل ارتباطی (UART(RS232)
71.....	آزمایش سیزدهم: آشنایی با پروتکل I2C

## معرفی میکروکنترلر های ARM:

است که اولین پردازنده مبتنی بر معماری RISC می باشد Acron RISC Machine ARM مخفف کلمه که طراحی اولیه آن را شرکت Acron شامل ۱۲ مهندس در سال ۱۹۸۵ به عمل رسانده اند . در اواخر دهه ۱۹۸۰ شرکت های Acron و VLSI به Apple پیوستند و باعث پیشرفت چشمگیر محصولات ARM شده اند . لازم است این را بدانید که هیچ گونه تراشه ای تولید نمی کند و تنها درآمدش حاصل از فروش حق امتیاز استفاده از معماریش به شرکت های تولید کننده میکروکنترلر است .

در حال حاضر بسیاری از شرکت های تولید کننده میکروکنترلر نظیر Analog Devices , NXP , Atmel از هسته ARM در ساختار میکروکنترلرهایشان استفاده می کنند .

## ❖ مهم ترین ویژگی های پردازنده های ARM :

- صرف توان بھینه که ARM را به گزینه مناسبی جهت استفاده در تجهیزات قابل حمل تبدیل نموده است .
- استاندارد بودن تراشه ARM ، یعنی می توان برنامه های نوشته شده را بدون نیاز به تغییر و یا با تغییرات جزئی ، توسط تراشه های دیگر تولید کنندگان نیز استفاده نمود .
- معماری ساده ARM و قابلیت پیاده سازی با ترانزیستور های اندک .
- قابلیت استفاده از سیستم عامل هایی مانند Window CE و LINUX که به صورت رایگان در دسترس می باشند . مانند تبلت های موجود .
- کارایی بالا در عین ابعاد کوچک . مثلا کارایی نوعی پردازنده ARM که با فرکانس 400MHz کار می کند ، با کارایی پردازنده PENTIUM2 با فرکانس 300MHz قابل قیاس است . اما مصرف آن یک پنچاهم پردازنده پنتیوم می باشد !

## ❖ خانواده های مختلف میکروکنترلر های ARM :

به طور کلی هسته های ARM در چند خانواده ای کلی طبقه بندی می شوند که هر یک شامل گونه ها و مدل های بسیاری است . خانواده های ARM7 , ARM9 , ARM10 , ARM11 , Cortex پنج دسته ای اصلی تولیدات ARM هستند که در سرعت ، امکانات ، تعداد Pipeline ، توان تلفاتی و ... با یکدیگر متفاوتند . شرکت هایی از قبیل :

- NXP با محصولات LPC17XX , LPC13XX •
- Texas Instrument •
- ST Microelectronic با محصولات سری STM32 , STM32L •
- Toshiba با سری TX03 •

- SAM3U با محصولات سری Atmel
  - EFM32 با محصولات سری Energy Micro
- از جمله تولیدکنندگان میکروکنترلر بر پایه Cortex-M3 می باشند .

## ❖بررسی مشخصات میکروکنترلر : LPC1768 :

میکروکنترلرهایی که با معماری Cortex-M3 از شرکت NXP ارائه شده اند شامل دو گروه LPC13XX و LPC17XX است. در بین کلیه تراشه های این دو گروه، میکروکنترلر LPC1768 یکی از عمومی ترین و بهترین گزینه ها برای آموزش میکروکنترلرهای ARM می باشد . مشخصات این پردازنده عبارت اند از:

- بسته بندی این تراشه در دو نوع TFBGA100 و LQFP100 می شود که دارای 100 پین است.

- حداکثر سرعت پردازش 100MHz
- 512KB حافظه فلاش داخلی با قابلیت برنامه ریزی داخل سیستم (ISP)
- 64KB حافظه SRAM داخلی
- 70 عدد پایه ورودی خروجی (GPIO)
- ولتاژ تغذیه 3.3V
- اسیلاتور RC کالیبره شده داخلی 4MHz
- اسیلاتور خارجی با رنج کاری 1MHz تا 24MHz مجهز به واحد PLL

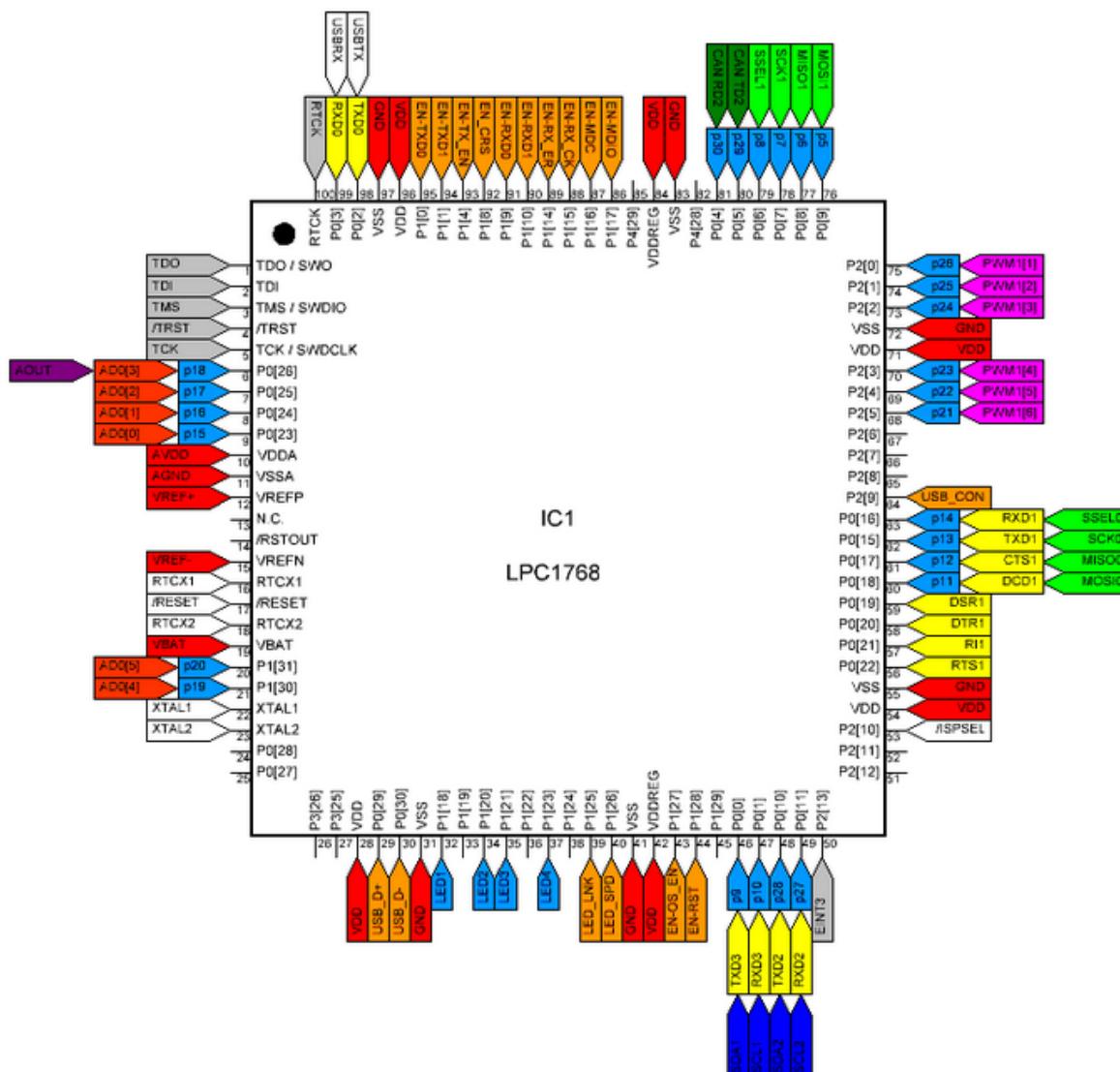
## ❖سخت افزار عمومی میکروکنترلر :LPC1768

علاوه بر ۷۰ پایه از پورت های ۰، ۱، ۲، ۳ و ۴ که به صورت GPIO در اختیار کاربر قرار گرفته ، ۳۰ پایه دیگر جهت تغذیه دیجیتال و آنالوگ و کلاک و JTAG میکروکنترلر مورد استفاده قرار می گیرد . میکروکنترلر LPC1768 دارای سخت افزار های داخلی به شرح ذیل می باشد .

- رابط شبکه Ethernet MAC
- چهار عدد بلوک ارتباط سریال UART
- رابط شبکه CAN
- رابط سریال SPI
- رابط سریال SSP
- ارتباط سریال I2C

- میدل آنالوگ به دیجیتال ۱۲ بیتی هشت کاناله
- مبدل دیجیتال به آنالوگ ۱۰ بیتی
- چهار عدد تایمر / کانتر ۳۲ بیتی
- یکی کانال PWM
- واحد راه انداز Quadrature Encoder
- ساعت همزمان (RTC) مجهر به NVRAM
- تایمر نگهبان (Watchdog)
- تایمر وقفه تکرار شدنی

شکل زیر نمای پایه های میکروکنترلر LPC1768 را نشان می دهد :



# آزمایش اول : آشنایی با نرم افزار KEIL uVision5

❖ اهداف :

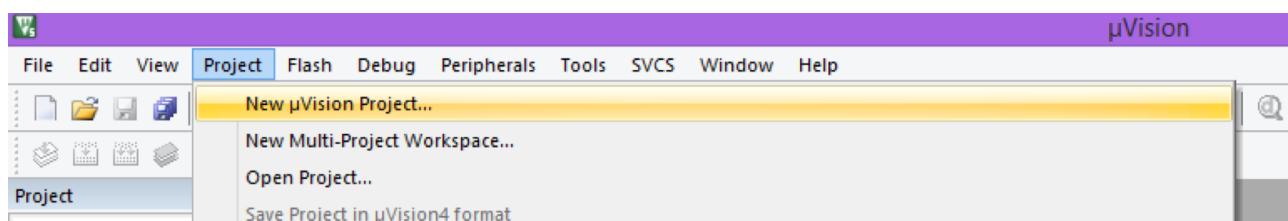
- آشنایی با نرم افزار KEIL uVision5

❖ مقدمه :

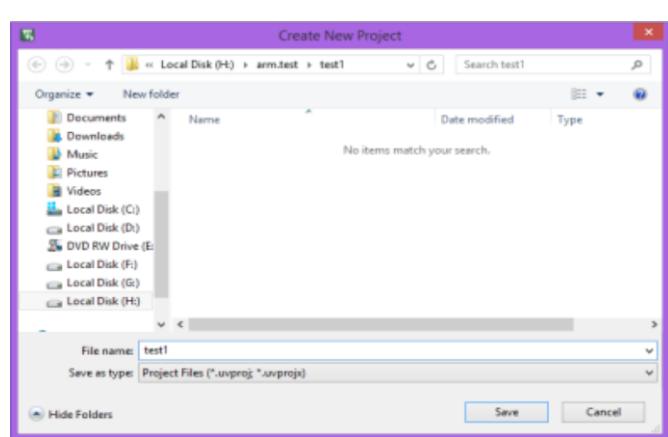
در شروع آزمایشگاه مدارهای واسط نیاز است که دانشجویان با محیط نرم افزار برنامه نویسی پردازنده هایی *LPC1768* که برای آزمایشگاه مدارهای واسط در نظر گرفته شده است، آشنا شوند. بنابراین در این آزمایش علاوه بر آشنایی با محیط نرم افزار *Keil uVision* مراحل ایجاد یک پروژه جدید به صورت گام به گام آموزش داده می شود.

## ❖ نحوه ایجاد پروژه با نرم افزار KEIL uVISION :

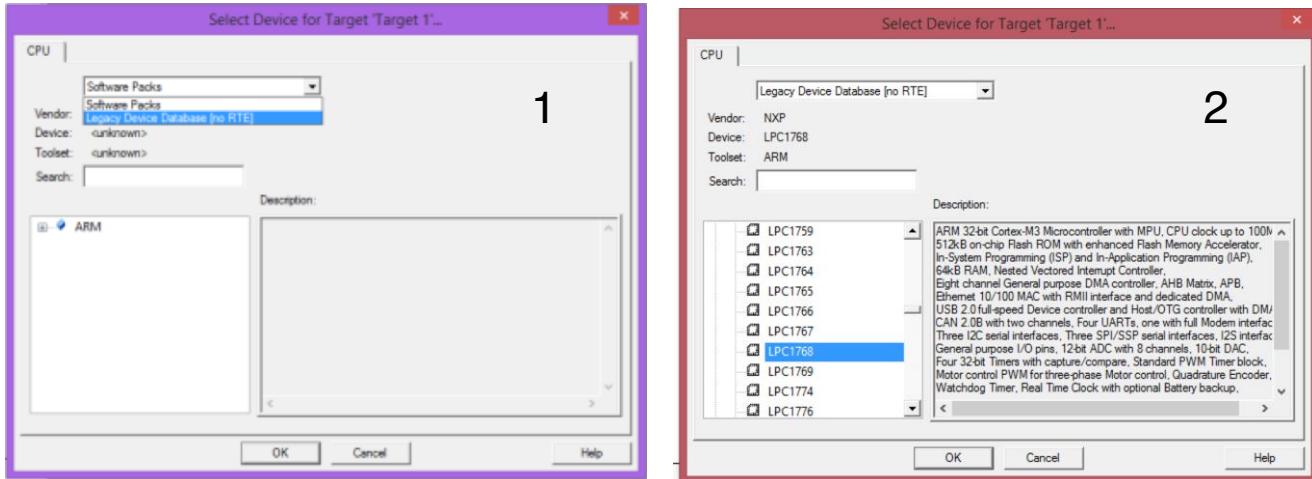
ابتدا از منوی *new uvision project* گزینه *project* انتخاب می کنیم :



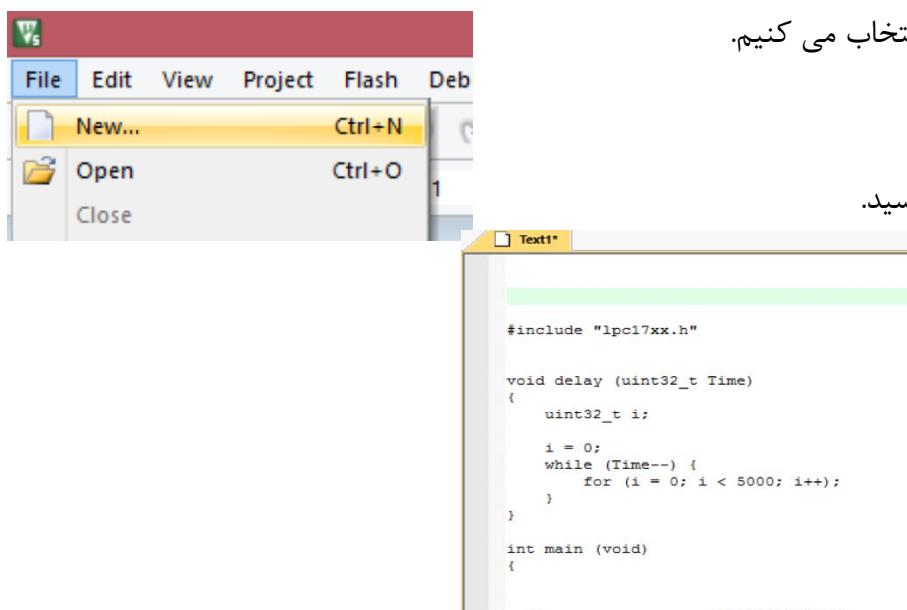
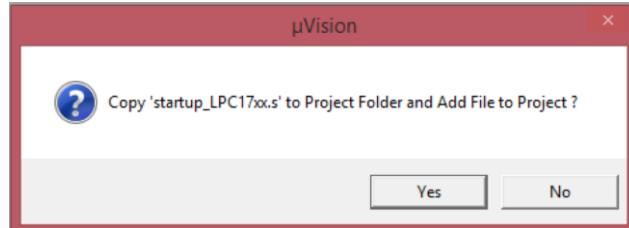
حال پنجره ای باز می شود و شما می توانید محل ذخیره پروژه و نام آن را تعیین کنید و سپس گزینه *Save* را بزنید



حال پنجره ای باز می شود و از شما می خواهد که تا شماره چیپ مورد نظر را انتخاب نمایید . از قسمت سمت چپ مجموعه تراشه های NXP را انتخاب نموده و از زیر مجموعه ای آن ، تراشه *LPC1768* را انتخاب نمایید و سپس *OK* را بزنید .



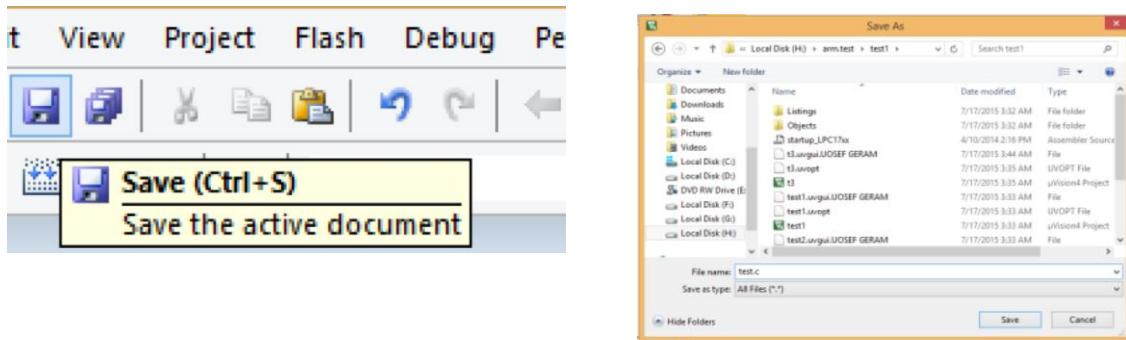
سپس از شما سوال می شود که آیا مایلید فایل *startup* میکرو به پروژه اضافه شود که باید *yes* را انتخاب کنید..



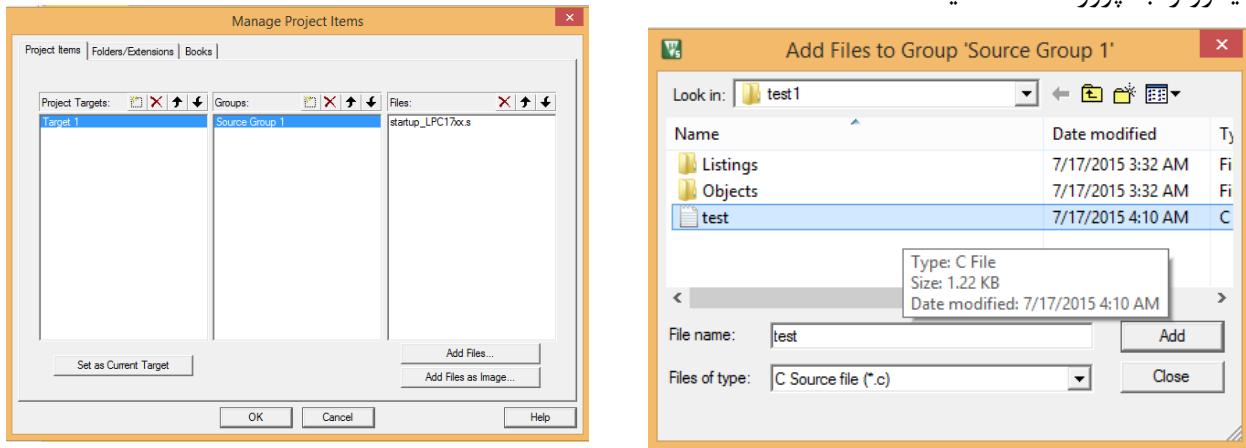
سپس از منو *New* گزینه *New File* را انتخاب می کنیم.

در صفحه مربوطه برنامه خود را بنویسید.

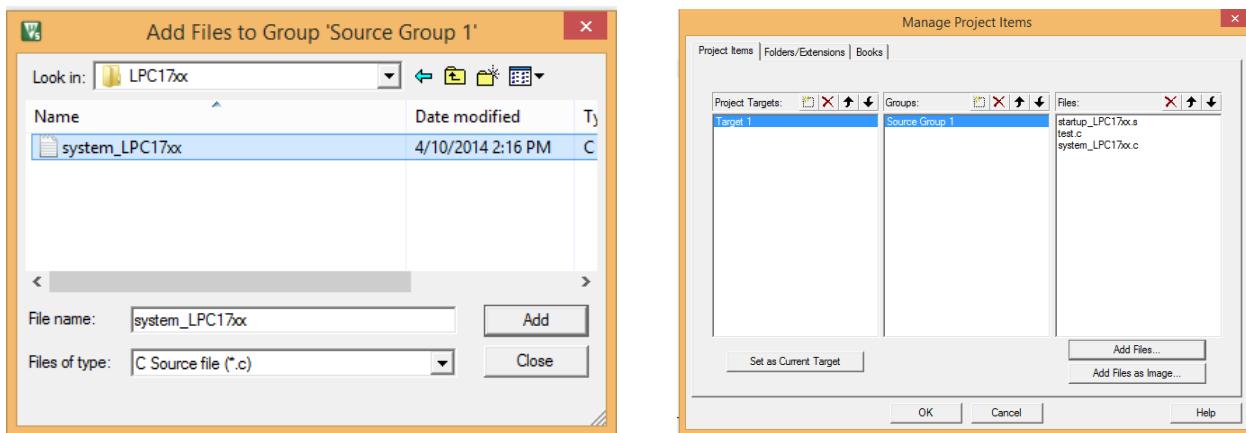
پس از نوشتن برنامه با زدن گزینه *save* فایل را با پسوند *.C* در مسیر پروژه ذخیره کنید.



پس از ذخیره کردن فایل برنامه، با انتخاب گزینه *Add Files* فایل برنامه و سپس فایل *system* میکرو را به پروژه اضافه کنید



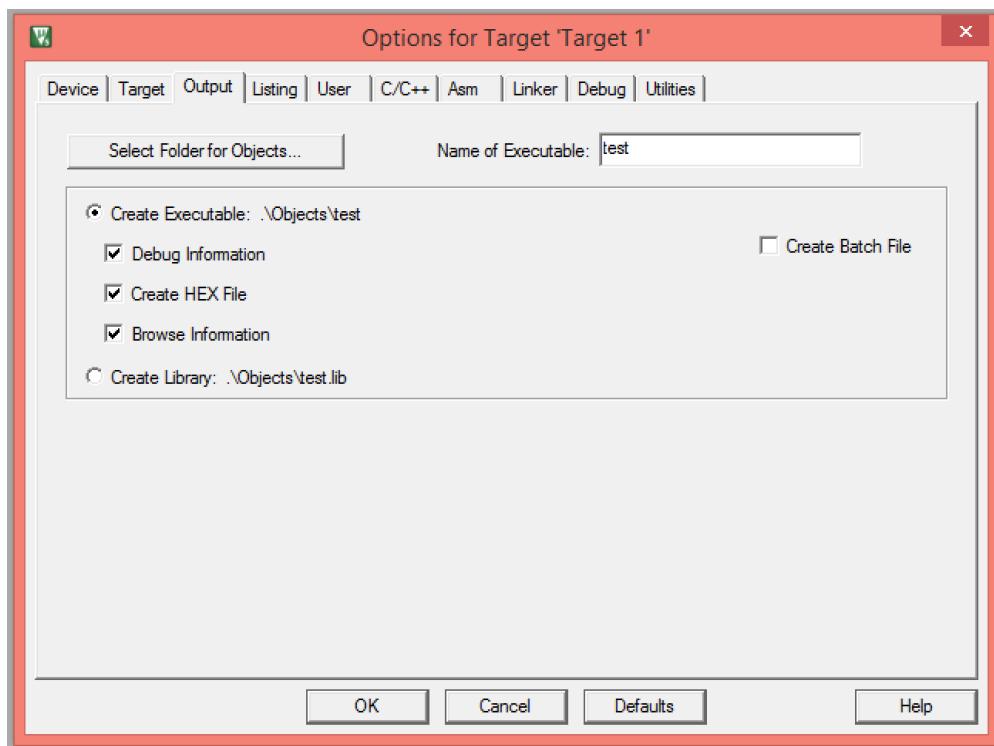
از مسیر *system* (محل نصب برنامه) فایل *C:\Keil\ARM\Startup\NXP\LPC17xx* میکرو رانیز اضافه کنید.



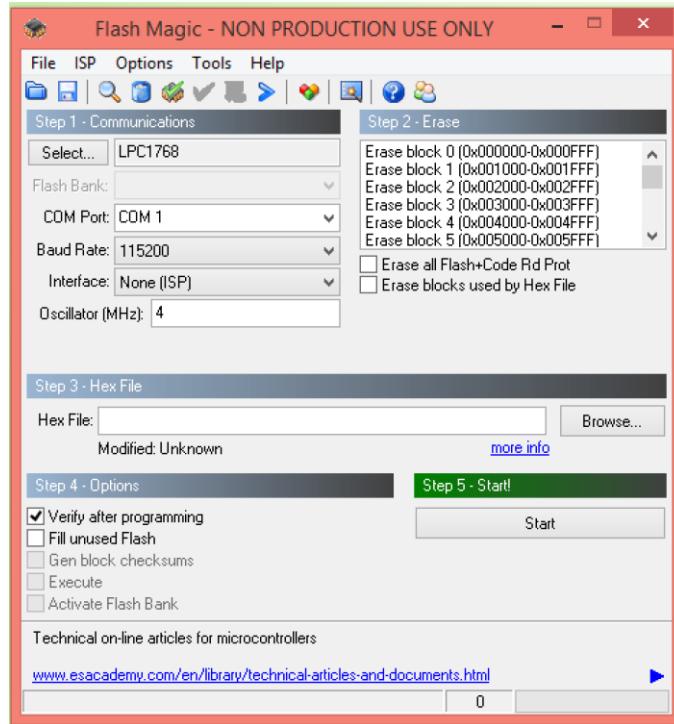
و در آخر گزینه *OK* را انتخاب کنید.

## ❖ نحوه پروگرام نمودن از طریق پورت Serial

میکروکنترلر *LPC1768* را می توان به دو طریق یکی استفاده از رابط *Jlink* و دیگری استفاده از رابط سریال پروگرام نمود که ما در این آزمایشگاه از رابط سریال استفاده می کنیم. برای پروگرام نمودن از طریق سریال نیاز به پورت سریال *RS232* و نرم افزار *FlashMagic* می باشد. قبل از آموزش نحوه پروگرام نمودن به نحوه فعال سازی ساخت فایل *.Hex* در نرم افزار *Keil* می پردازیم. برای ساخت فایل *.Hex* در نرم افزار *Keil* در تابع *Output* گزینه *Option for Target Project* را انتخاب می نماییم. در پنجره باز شده وارد تابع *Create HEX File* می شویم و گزینه *Create HEX File* را علامت می زنیم.

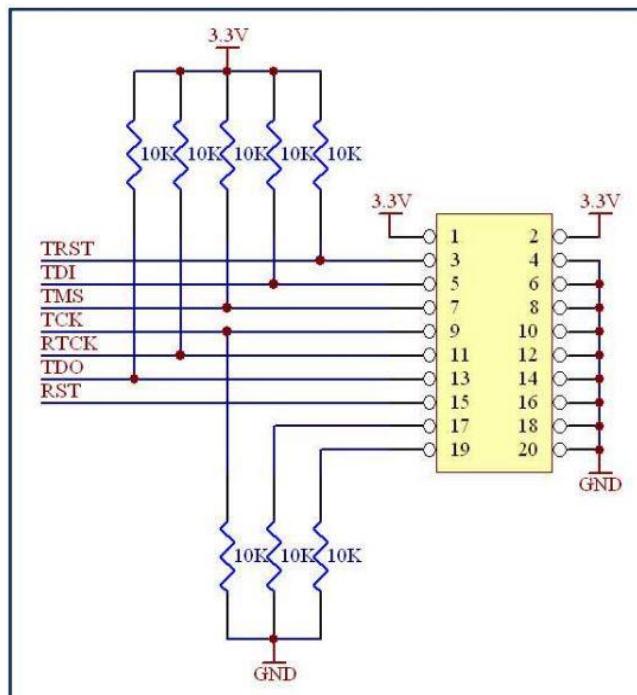


سپس پس از *build* نمودن پروژه و اطمینان از عدم وجود *Error* نرم افزار *FlashMagic* را باز نموده و طبق تصویر زیر و شماره *port* اتصالی میکرو به کامپیوتر آنرا تنظیم می نمایید و آدرس فایل *.Hex* برنامه خود را در محلی که پروژه رو ساخته اید در پوشش *Objects* وارد می نمایید. سپس میکرو را در حالت *Programming mode* قرار می دهید و دکمه *Reset* را ببروی میکرو فشار دهید. اگر همه مرحله به درستی طی شده باشد با کلیک ببروی *Start Program* خواهد شد. سپس میکرو را در حالت *Normal* قرار داده و دکمه *Reset* را دو مرتبه فشار دهید تا نتیجه را مشاهده نمایید.



## ❖ نحوه پروگرام نمودن از طریق JLink

کد HEX تولید شده می تواند از طریق پروگرامر J-Link تحت پروتکل JTAG پروگرام شود. در این صورت پروگرامر J-Link باید مطابق شکل زیر به میکروکنترلر متصل شود:



قبل از انجام هر کاری می بایست درایور J-Link آدرس دهی شود. درایور J-Link همراه با نصب نرم افزار KEIL به طور اتوماتیک بر روی کامپیوتر و در محل نصب نرم افزار KEIL نصب می شود و کافیست آن را آدرس دهی نماییم. برای این کار بعد از متصل کردن J-Link بین کامپیوتر و میکروکنترلر و در حالتی که میکروکنترلر روشن است وارد Device Manager شوید. احتمالاً گزینه J-Link به طور unknown مشخص شده است که با دو بار کلیک بر روی این گزینه و سپس انتخاب گزینه Update Deriver عملیات آدرس دهی به فایل درایور آغاز می شود. بعد از کلیک کردن بر روی گزینه Update Deriver پنجره جدیدی باز می شود که در این مرحله می بایست گزینه دوم یعنی browse my computer for device software را انتخاب نمایید. در صفحه باز شده جدید می بایست مسیر ARM\SEGGER\USB DRIVER را آدرس دهی نمایید. سپس بر روی گزینه Next کلیک نمایید. در صفحه باز شده جدید ملاحظه می کنید که درایور به طور کامل نصب شده و در Device Manager گزینه مربوط به J-Link به طور کامل شناخته شده است. حال برای پروگرام کردن میکروکنترلر می بایست بعد از تعریف هر پروژه در نرم افزار Keil uVision تنظیمات زیر را انجام دهید:

1. بعد از ایجاد پروژه به منوی Project for Target رفته و بر روی گزینه Option for Target کلیک نمایید.
2. از پنجره باز شده به سربرگ Debug رفته و گزینه Use را فعال نمایید.
3. در کشوی کنار گزینه Use، گزینه Cortex M3 J-Link را انتخاب نمایید.
4. حال به سربرگ Utilities رفته و گزینه Setting را کلیک نمایید.
5. در پنجره باز شده جدید بر گزینه Add کلیک نموده و خانواده میکروکنترلر متناسب را انتخاب نمایید.
6. بعد از انتخاب خانواده تراشه گزینه Add را کلیک نمایید تا دوباره وارد Utilities شوید. حال می بایست گزینه Cortex M3 J-Link را فعال نمایید و در کشوی مربوطه گزینه Cortex Target for Flash Programming را فعال نمایید و در کشوی مربوط گزینه M3 J-Link را انتخاب نمایی

## ❖ دستور کار:

1. طبق مراحل ذکر شده ابتدا یک پروژه ساده بسازید. کد زیر را در برنامه ای که ساخته اید کپی نمایید. سپس آن را Build کنید و از طریق امکانات منوی دیباگ به مشاهده کلاک و رجیستر های جانبی آن بپردازید و آنها را گزارش نمایید.

```
#include<lpc17xx.h>
int main() {
    LPC_GPIO2 -> FIOPIN = 0xFFFFFFFF;
    while(1) {
        }
```

.|| نقش PLL در تولید کلک چیست؟

## آزمایش دوم: آشنایی با پورت های ورودی / خروجی و موتورهای پله ای

### ❖ اهداف :

- آشنایی با پورت های ورودی و خروجی *GPIO* و رجیستر های آن
- کار با کلید های فشاری و *LED*
- کار با موتور پله ای (*Stepermotor*)

### ❖ مقدمه:

هدف از این آزمایش آشنایی با ورودی خروجی های دیجیتال ، بررسی رجیسترها مربوطه و همچنین کار با این ورودی و خروجی به صورت عملی است. بر این اساس شما داده هایی را از طریق کلیدهای فشاری به درون میکروکنترلر منتقل کرده و بر روی *LED* های موجود در بورد اطلاعاتی را نمایش می دهید همچنین با نحوه کار با موتور پله ای آشنا خواهید شد.

### ❖ بررسی رجستر های *GPIO*:

جهت کار با پورت های ورودی خروجی از پنج رجیستر استفاده می شود که در ادامه به بررسی و کار با این رجیسترها می پردازیم . در میکرو *LPC1768* می توان از پورت p0 پین های [30:0] ، پورت p1 پین های [31:0] ، پورت p2 پین های [13:0] ، پورت p3 پین های [26:25] و پورت p4 پین های [29:28] به عنوان ورودی و خروجی استفاده کرد.

**رجیستر اول *LPC\_GPIORx -> FIODIR*** : این رجیستر برای تعیین جهت پورت استفاده می شود به نحویکه با نوشتمن مقدار یک در این رجیستر پایه مربوط به عنوان خروجی و با نوشتمن مقدار صفر پایه به عنوان ورودی تعیین می شود. پورتهای میکرو *LPC1768* به طور پیش فرض در جهت ورودی می باشند و تنها برای خروجی کردن پورت نیاز به مقدار دهی داریم . به جای *0X01* در این رجیستر شماره پورت قرار می گیرد .

مثال : بیت اول رجیستر پورت را در جهت خروجی تعیین کنید.

```
FIODIR = 0X01 -> LPC_GPIO2;
```

مثال : ۸ بیت اول رجیستر پورت صفر را در جهت خروجی تعیین کنید.

```
LPC_GPIO0 -> FIODIR = 0xFF;
```

رجیستر دوم : **LPC\_GPIOx -> FIOSET** : این رجیستر جهت یک کردن پایه خروجی به کار می رود..

مثال : ۸ LED متصل به بیت کم ارزش پورت ۲ را روشن کنید.

```
LPC_GPIO2 -> FIODIR = 0xFF;
LPC_GPIO2 -> FIOSET = 0xFF;
```

رجیستر سوم : **LPC\_GPIOx -> FIOCLR** : این رجیستر جهت صفر کردن پایه خروجی به کار می رود..

مثال : ۸ LED متصل به ۸ بیت کم ارزش پورت ۲ را خاموش کنید.

```
LPC_GPIO2 -> FIODIR = 0xFF;
LPC_GPIO2 -> FIOSET = 0x00;
```

---

برنامه ای بنویسید که ۸ LED متصل به ۸ بیت کم ارزش پورت ۲ به شکل چشمک زن خاموش و روشن شوند.

```
#include <lpc17xx.h>
void delay (uint32_t Time) {
    uint32_t i;
    i = 0;
    while (Time--) {
        for (i = 0; i<5000 ; i++);
    }
}
int main(void) {
    LPC_GPIO2->FIODIR = 0xff;
    LPC_GPIO2->FIOCLR = 0xff;      //turn off leds
    while(1){
        LPC_GPIO2->FIOSET = 0xFF;
        delay(500);
        LPC_GPIO2->FIOCLR = 0xFF;
        delay(500);
    }
}
```

**رجیستر چهارم  $LPC\_GPIOx \rightarrow FIOPIN$**  : از این رجیستر برای خواندن از ورودی استفاده می شود . البته با این رجیستر می توان پورت ها را مقدار دهی کرد.

---

برنامه ای بنویسید که در صورتیکه کلید متصل به بیت اول پورت ۲ زده شد LED متصل به بیت صفرم پورت ۲ را روشن کند . نکته مهم در برنامه این است که پایه های ورودی میکرو *lpc1768* به طور پیش فرض *pull up* هستند و برای اینکه تنها بیت اول پورت ۲ مد نظر است باید آن را با یک AND کرد.

```
#include <lpc17xx.h>
void delay (uint32_t Time) {
    uint32_t i;
    i = 0;
    while (Time--) {
        for (i = 0 ; i<5000 ; i++) ;
    }
}
int main (void) {
    LPC_GPIO2->FIODIR = 0x1;
    LPC_GPIO2->FIOCLR = 0x1;
    while(1) {
        LPC_GPIO2->FIOCLR = 0x1;
        while((LPC_GPIO2->FIOPIN & 1 << 1) == 0) {
            LPC_GPIO2->FIOPIN = 0x1;
        }
    }
}
```

---

**رجیستر پنجم  $LPC\_GPIOx \rightarrow FIOMASK$**  : این رجیستر ها برای انتخاب پینهایی که می خواهید و یا نمی خواهید از رجیسترها *FIOxPIN* و *FIOxSET* و *FIOxCLR* تأثیر بپذیرند مورد استفاده قرار می گیرند . عملکرد این رجیستر را با ذکر مثالی بیان می کنیم . فرض کنید می خواهیم از پین های شماره *PORT1.16 , PORT1.17 , PORT1.18 , PORT1.19* به عنوان خروجی استفاده کنیم . در صورتی که بقیه بیت ها برای کار دیگری در نظر گرفته شده است، برای اینکه خیالتان از بابت تغییر بقیه بیت ها راحت باشد می توانید با استفاده از رجیستر *FIOxMASK* دسترسی به بیت های مورد نظرتان را فعال و بقیه را غیرفعال کنید . با قرار دادن صفر در هر یک از بیت های رجیستر *FIOxMASK* بیتهای مورد نظرتان فعال (یعنی در دسترس) و با قرار دادن یک غیر فعال می شوند . در مثال زیر پین های *PORT1.16 , PORT1.19 , PORT1.18 , PORT1.17* در ابتدا به صورت خروجی پیکره بندی شده است و فقط

دسترسی به همین ۴ پین فعال شده است. حال اگر پین های دیگر را بخواهیم صفر و یک نماییم بدلیل عدم دسترسی به آنها این کار انجام نخواهد شد.

```
LPC_GPIO1->FIODIR = 0x000F0000;
LPC_GPIO1->FIOMASK = 0xFFFF0FFF;
LPC_GPIO1->FIOSET = 0x548319FC;
LPC_GPIO1->FIOCLR = 0xF2D398FF;
```

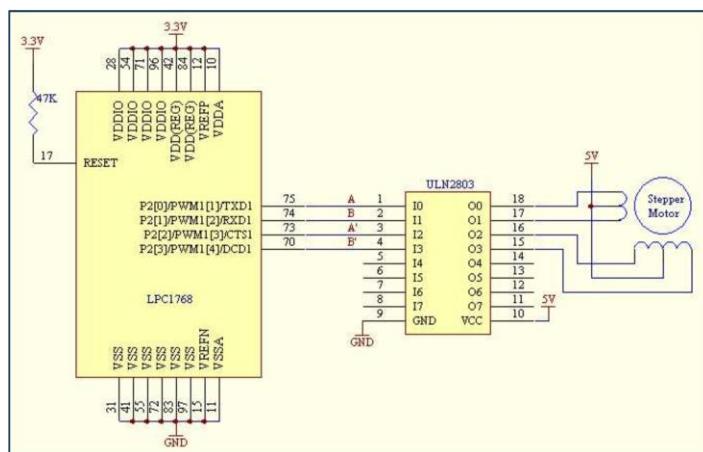
## ❖ راه اندازی موتور پله ای با GPIO

مотор های پله ای از مو تور ها هستند که حرکت شافت رتور آنها به صورت پله ای و تحت زاویه خاص انجام می شود . عموماً این نوع موتورها دارای ۶ سیم می باشند که دو سیم آن به صورت مشترک و ۴ سیم دیگر به سیم پیچ های استاتور موتور متصل است . اگر به صورت متوالی با رعایت فاصله زمانی پالس اعمال نماییم ، موتور در جهت راست و یا چپ طبق جدول زیر می چرخد.

حال اگر جهت پالس ها را عوض کنیم جهت چرخش موتور عوض خواهد شد .

راست گرد	شماره پله	A پین	A* پین	B پین	B* پین	چپ گرد
↓	1	1	0	0	0	↑
	2	0	1	0	0	
	3	0	0	1	0	
	4	0	0	0	1	

با اتصال سیم های مشترک به قطب منفی و اتصال ۴ سیم استاتور به میکرو کنترلر میتوانیم موتور را کنترل نماییم  
نکته مهم : به دلیل عدم جریان دهی کافی پورت های میکرو کنترلر ، باید بین میکرو کنترلر و موتور از یک مدار واسطه به عنوان تامین کننده جریان استفاده نماییم(درایو ULN2803 استفاده شده است).



موتور مورد استفاده در این بورد دارای زاویه چرخش ۱۵ درجه می باشد.

---

برنامه اول : برنامه ای بنویسید که موتور پله در جهت خلاف جهت عقربه ساعت شروع به چرخش نماید.

```
#include <lpc17xx.h>
void delay (uint32_t Time) {
    uint32_t i;
    i = 0;
    while (Time--) {
        for (i = 0 ; i<5000 ; i++);
    }
}
int main(void) {
    LPC_GPIO2->FIODIRO = 0x0F;           //turn off
    while(1) {
        LPC_GPIO2->FIOPIN = 0x01;
        delay(100);
        LPC_GPIO2->FIOPIN = 0x02;
        delay(100);
        LPC_GPIO2->FIOPIN = 0x04;
        delay(100);
        LPC_GPIO2->FIOPIN = 0x08;
        delay(100);
    }
}
```

---

برنامه دوم: برنامه ای بنویسید که موتور پله ای به اندازه ۱۵۰ درجه در جهت ساعتگرد بچرخد.

```
#include <lpc17xx.h>
unsigned k = 0;
void delay (uint32_t Time) {
    uint32_t i;
    i = 0;
    while (Time--) {
        for (i = 0 ; i<5000 ; i++);
    }
}
```

```

int main(void)
{
    LPC_GPIO2->FIODIRO = 0x0F;
    for(k = 0 ; k<3 ; k++)
    {
        LPC_GPIO2->FIOPIN = 0x08;
        delay(10000);
        LPC_GPIO2->FIOPIN = 0x04;
        delay(10000);
        LPC_GPIO2->FIOPIN = 0x02;
        delay(10000);
        LPC_GPIO2->FIOPIN = 0x01;
        delay(10000);
    }
}

```

---

## ❖ دستور کار:

- I. هشت بیت کم ارزش پورت ۲ را به ۸ LED متصل کنید سپس برنامه ای بنویسید که LED های آن یک در میان روشن شود.
- II. برنامه ای بنویسید که LED ها از بیت صفر تا ۷ به ترتیب روشن شود.
- III. هشت بیت اول پورت صفر را به کلید وصل کنید و سپس ۸ بیت دوم پورت صفر را به ۸ LED متصل کنید و برنامه ای بنویسید که متناظر با هر کلید ، LED مربوطه به آن روشن شود.
- IV. برنامه ای بنویسید که موتور پله ای یک دور راستگرد و دو دور جهت چپگرد بچرخد.
- V. برنامه ای بنویسید که موتور پله ای به اندازه ۱۸۰ درجه در جهت ساعتگرد بچرخد.

## آزمایش سوم: آشنایی با نمایشگرهای ماتریسی نقطه ای، ۱۶ قسمتی و ۷ قسمتی

### ❖ اهداف :

- آشنایی و راه اندازی نمایشگر ماتریسی نقطه ای

**(DOTMATRIX)**

- آشنایی و راه اندازی نمایشگر ۱۶ قسمتی

- آشنایی و راه اندازی نمایشگر ۷ قسمتی

### ❖ مقدمه :

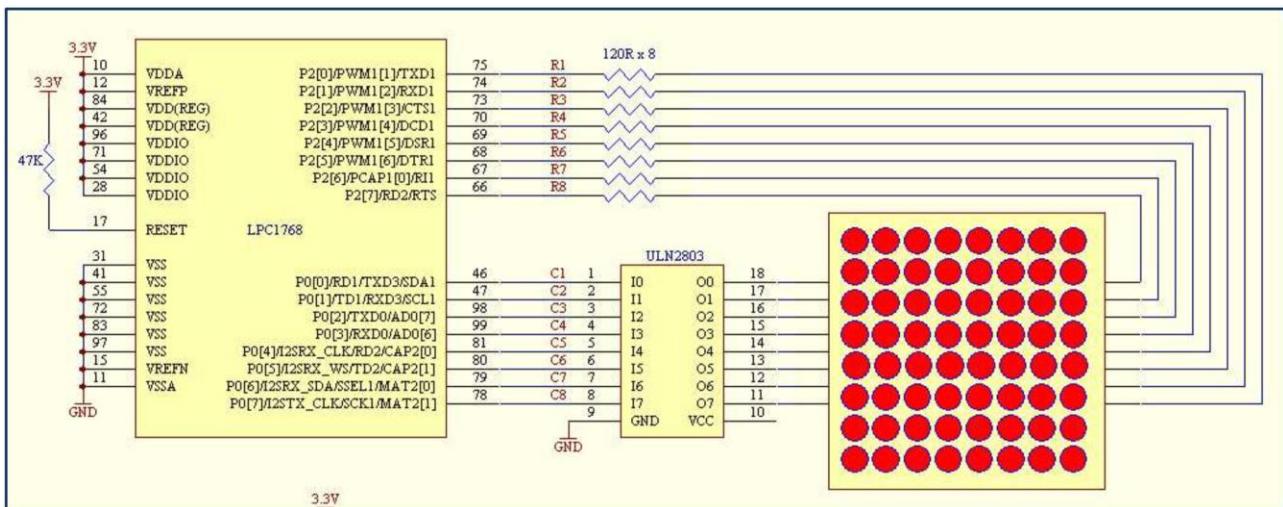
هدف از انجام این آزمایش آشنایی با نحوه کار کردن نمایشگرهای ماتریسی نقطه ای و نمایشگر ۱۶ و ۷ قسمتی به همراه نمایش اعداد و حروف بر روی آنهاست.

### ❖ نمایشگر ماتریسی نقطه ای (DOTMATRIX)

یک عدد نمایشگر DOTMATRIX از نوع ۸\*۸ (۸ ستون و ۸ ردیف) درب لوکی به نام *Display* قرار داده شده است . با اتصال ماتریسی LED ها به میکرو، تعداد پایه های خروجی را به ۱۶ پین کاهش می دهیم . برای کنترل نمایشگر باید از روش جاروب کردن سطر و ستون استفاده کرد، به این صورت که ابتدا سطر اول را یک (مابقی صفر) میکنیم و سپس مقادیر (خاموش یا روشن بودن هر LED در این سطر) مربوط به ستونها را روی ۸ پین ستون قرار می دهیم . حال در مرحله بعدی سطر دوم را یک (مابقی صفر) میکنیم و سپس مقادیر مربوط به ردیف دوم را بر روی ۸ پین ستون قرار می دهیم و همینطور تا سطر هشتم ادامه می دهیم. به یاد داشته باشید که تمامی این مراحل باید با سرعت بالا انجام پذیرد تا چشم انسان قادر به دیدن تاخیری که بین روشن نمودن سطرهای مختلف است نباشد . در این بورد جریان ستون ها توسط آی سی *ULN2803* فراهم شده است و می توانیم سطرهارا مستقیما به GPIO متصل کنیم . در این صورت با جاروب کردن ستون ها می توان یک شکل را در نمایشگر نشان داد.

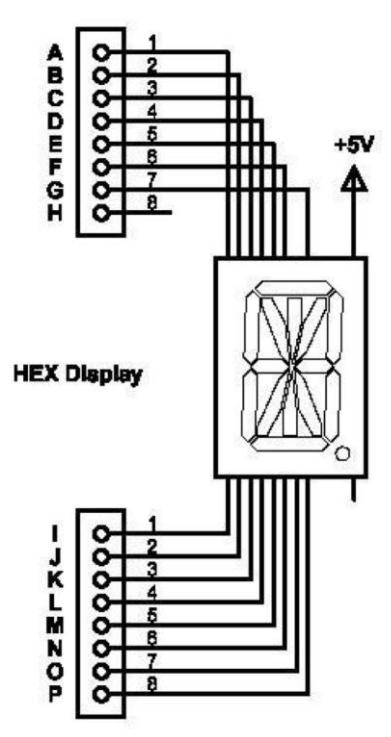
**برنامه نمونه :** در این مثال قصد داریم تا عدد ۲ را بر روی این نمایشگر نمایش دهیم برای این منظور همانطور که در شکل صفحه بعد نمایش داده شده است پین های سطر به هشت پین پایین پورت ۲ و پین های ستون از طریق آی سی به هشت پین پایین پورت صفر متصل شده است .

```
#include <LPC17xx.h>
void delay(int i){
    while(i--);
}
int main(void) {
    LPC_GPIO2->FIODIRO = 0xFF;
    LPC_GPIO0->FIODIRO = 0xFF;
    while(1) {
        LPC_GPIO0->FIOPINO = 0x01;
        LPC_GPIO2->FIOPINO = 0X00;
        delay(5000);
        LPC_GPIO0->FIOPINO = 0x02;
        LPC_GPIO2->FIOPINO = 0XC6;
        delay(5000);
        LPC_GPIO0->FIOPINO = 0x04;
        LPC_GPIO2->FIOPINO = 0Xe3;
        delay(5000);
        LPC_GPIO0->FIOPINO = 0X08;
        LPC_GPIO2->FIOPINO = 0Xb1;
        delay(5000);
        LPC_GPIO0->FIOPINO = 0X10;
        LPC_GPIO2->FIOPINO = 0Xb1;
        delay(5000);
        LPC_GPIO0->FIOPINO = 0X20;
        LPC_GPIO2->FIOPINO = 0X9b;
        delay(5000);
        LPC_GPIO0->FIOPINO = 0X40;
        LPC_GPIO2->FIOPINO = 0X8e;
        delay(5000);
        LPC_GPIO0->FIOPINO = 0X80;
        LPC_GPIO2->FIOPINO = 0X00;
        delay(5000);
    }
}
```



## ❖نمایشگر ۱۶ قسمتی :

این نمایشگر که از ۱۶ قسمت LED کوچک تشکیل شده است قادر به نمایش حروف لاتین و اعداد به صورت مجزا می باشد . نمایشگر مورد استفاده در بورد از نوع کاتد مشترک است .

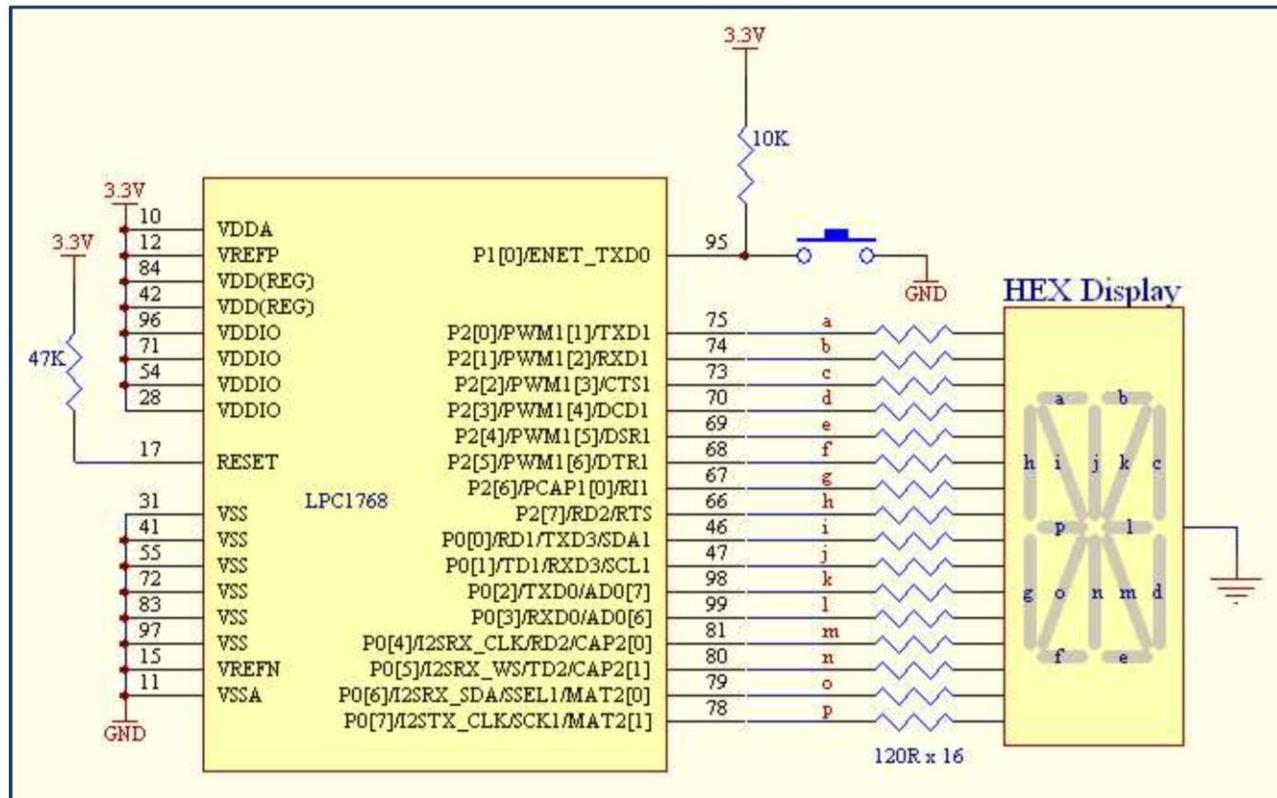


برنامه نمونه : در این برنامه قصد داریم در صورت ابودن حرف A و در صورت صفر بودن عدد ۹ نمایش داده شود PORT1.0 برای نمایش مقدار A باید مقادیر زیر را بر روی PORT1 و PORT2 جایگذاری کنیم .

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
1	1	1	1	0	0	1	1	0	0	0	1	0	0	0	1

همچنین برای نمایش مقدار ۹ باید مقادیر زیر را بر روی PORT1 و PORT2 جایگذاری کنیم .

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p
1	1	1	1	1	1	0	1	0	0	0	1	0	0	0	1



```

#include "LPC17xx.h"

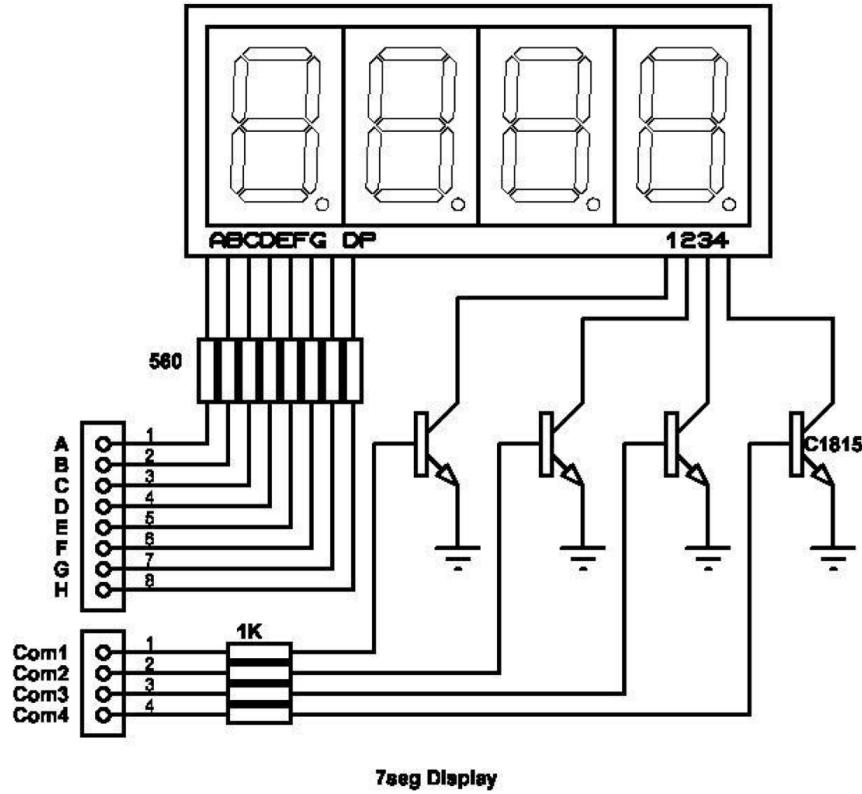
void delay (int i) {
    while(i--);
}

int main(void) {
    LPC_GPIO2->FIODIR = 0xFFFFFFFF;
    LPC_GPIO0->FIODIR = 0xFFFFFFFF;
    LPC_GPIO1->FIODIR = 0X00000000;
    while(1) {
        if((LPC_GPIO1->FIOPIN & (1<<0))==0) {
            LPC_GPIO2->FIOPIN = 0xBF;
            LPC_GPIO0->FIOPIN = 0x88;
            delay(400000);
        }
        else {
            LPC_GPIO2->FIOPIN = 0xCF;
            LPC_GPIO0->FIOPIN = 0x88;
            delay(400000);
        }
    }
}

```

## ❖ نمایشگر 7Segment :

دربورآموزشی که در اختیار داردید ۴ عدد 7segment مالتی پلکس شده (منظور از مالتی پلکس شدن، موازی بودن خطوط DATA میباشد) به رنگ قرمز از نوع کاتندمشترک تحت بلوکی به عنوان 7segment Display قرارداده شده است. همانطور که در شکل مشخص شده است با اعمال یک منطقی به خطوط COM، ترانزیستور متناظر در ناحیه یا شباع قرار گرفته و پایه متناظر 7segment را به زمین سویچ می نماید. به این ترتیب با اعمال کد های مربوط به هر عدد می توان اعداد دلخواه را بر روی هر کدام از 7segment ها نمایش داد.



**برنامه نمونه :** در برنامه زیر قصد داریم تا عدد ۱۵ را با نمایشگر 7Segment نمایش دهیم.

```
#include "lpc17xx.h"
unsigned char digit[10] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0xF};
void Delay (unsigned int Time){
    unsigned int i = 0;
    while (Time--) {
        for (i = 0 ; i<5000 ; i++);
    }
}
int main() {
    LPC_GPIO0->FIODIR = 0xFFFFFFFF;
    LPC_GPIO2->FIODIR = 0xFFFFFFFF;
    while(1) {
        LPC_GPIO0->FIOPIN = 1;
        LPC_GPIO2->FIOPIN = digit[5];
        Delay(10);
    }
}
```

```

LPC_GPIO0->FIOPIN = 2;
LPC_GPIO2->FIOPIN = digit[1];
Delay(10);
}
}

```

---

برنامه نمونه دوم : در برنامه زیر قصد داریم تا بر روی نمایشگر 7segment یک شمارنده ایجاد کنیم که قابلیت شمارش از عدد ۱ تا ۹۹ را داشته باشد .

```

#include "lpc17xx.h"
unsigned char digit[10] =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0xF};
void Delay (unsigned int Time) {
    unsigned int i = 0;
    while (Time--) {
        for (i = 0 ; i<5000 ; i++);
    }
}
int i,j,y;
int main() {
    LPC_GPIO0->FIODIR = 0xFFFFFFFF;
    LPC_GPIO2->FIODIR = 0xFFFFFFFF;
    while(1){
        Delay(250);
        for(i=0;i<=9;i++) {
            for(j=0;j<=9;j++) {
                for(y=0;y<=100;y++) {
                    LPC_GPIO0->FIOPIN = 1;
                    LPC_GPIO2->FIOPIN = digit[j];
                    Delay(10);
                    LPC_GPIO0->FIOPIN = 2;
                    LPC_GPIO2->FIOPIN = digit[i];
                    Delay(10);
                }
            }
        }
    }
}

```

## ❖ دستور کار:

- .I. برنامه ای بنویسید که از ۱ تا ۹۹۹ را به صورت صعودی و نزولی بشمارد.
- .II. برنامه ای بنویسید که با فشار دادن یک کلید به مقدار فعلی 7Segment یک واحد اضافه شود و با فشردن کلید دیگر یک واحد از مقدار کم شود. بازه اعداد از صفر تا ۹۹۹ است.
- .III. حرف اول اسم خود را بر روی DotMatrix نمایش دهید.
- .IV. حروف اسم خود را با تأخیر ۱ ثانیه بر روی نمایشگر ۱۶ قسمتی نمایش دهید.

# آزمایش چهارم: آشنایی با نمایشگرهای ال سی دی کاراکتری

❖ اهداف :

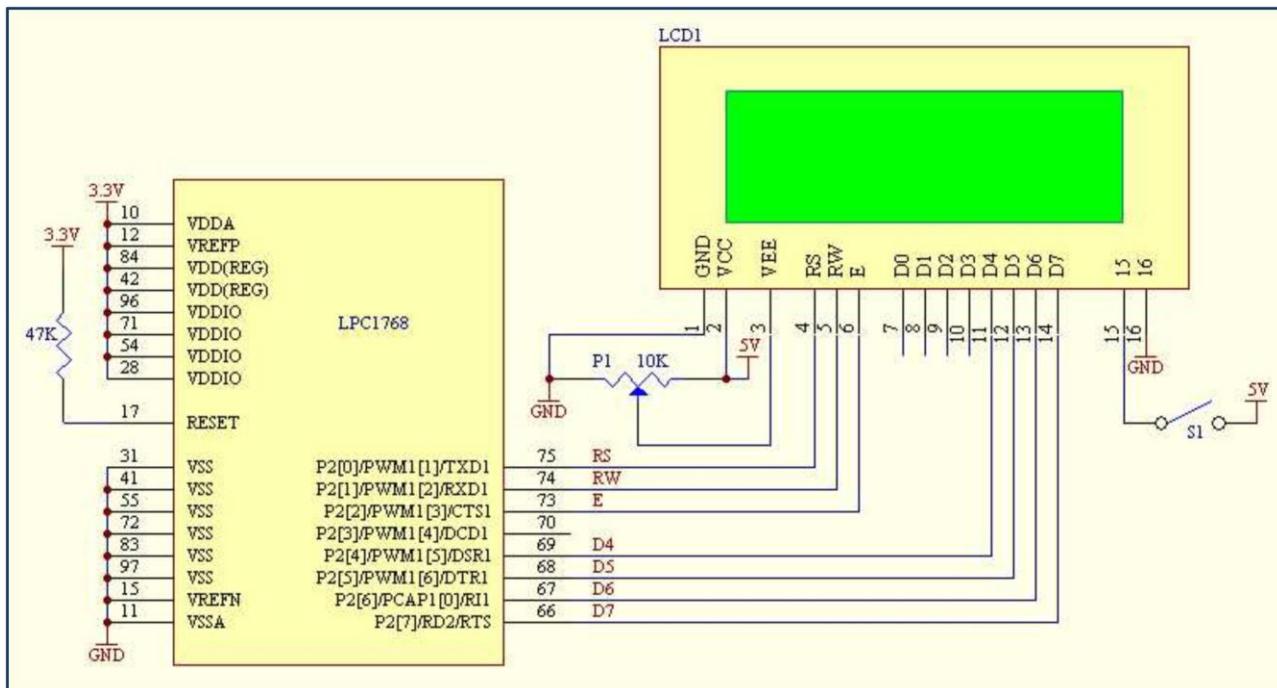
- راه اندازی ال سی دی کاراکتری و کار با آن

❖ مقدمه :

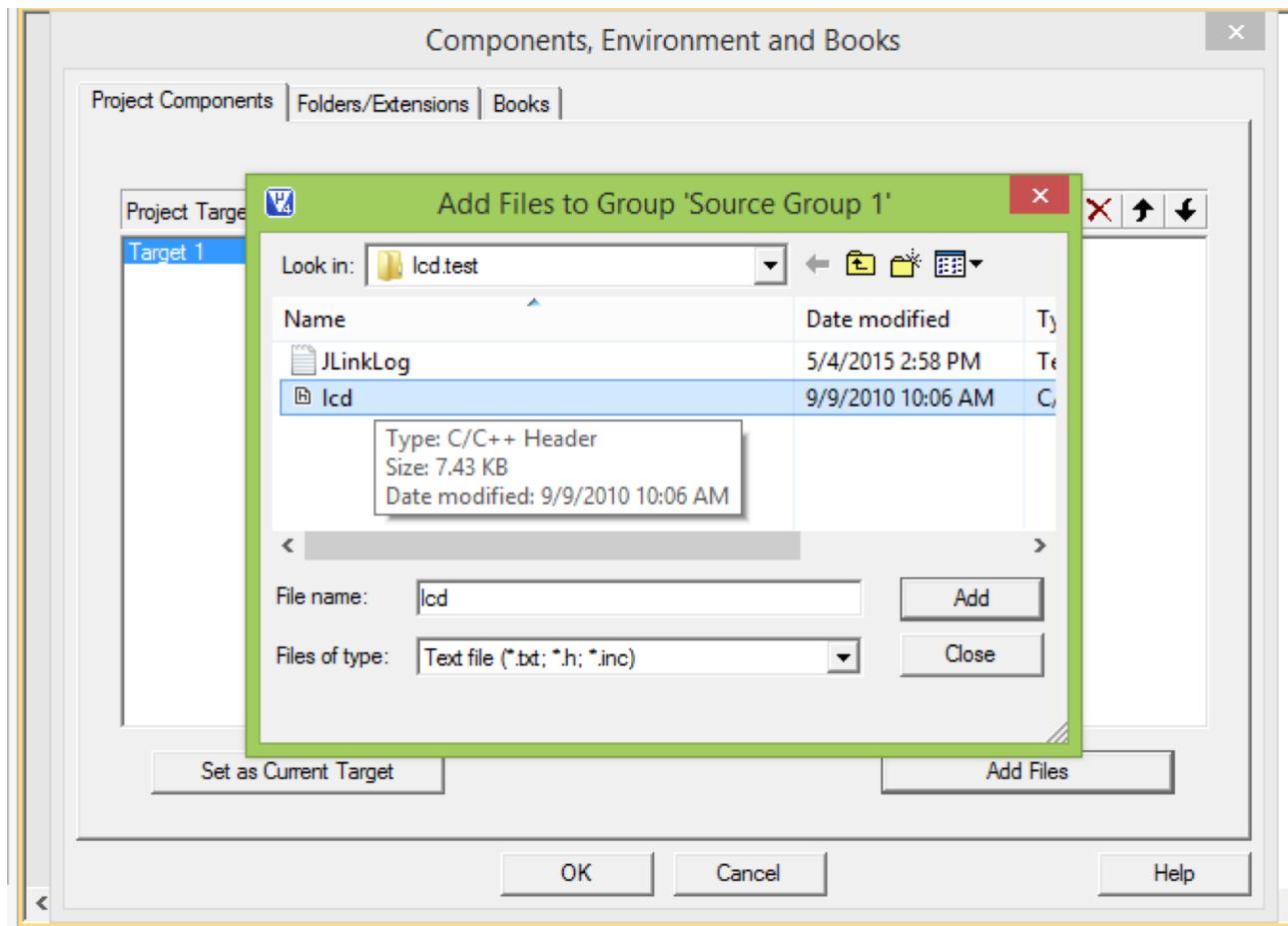
هدف از این آزمایش آشنایی و کار با نمایشگر های ال سی دی کاراکتری است.

## ❖ راه اندازی ال سی دی کاراکتری:

کارکتری یک نمایشگر کریستال مایع است که توسط یک تراشه به نام *HD44780* داریو می شود. این تراشه وظیفه گرفتن اطلاعات از میکروکنترلر و نمایش آن بر روی صفحه کریستال مایع را بر عهده دارد.



به منظور استفاده از *LCD.h* از هدر فایل *LCD.h* که کتابخانه ای شامل توابع کار با *LCD* است استفاده می کنیم. برای استفاده از این کتابخانه باید فایل *LCD.h* در مسیر پروژه خود کپی و سپس این کتابخانه را به پروژه خود اضافه می نماییم.



بعد از آن باید پیکر بندی مربوط به پین ها و پایه های میکرو کنترلر و *LCD* را تعریف کنیم . برای این کار طبق دستور زیر عمل می کنیم :

```
#define LCD_LPC1768
#define LCD_PORT_2
#define LCD_RS    0
#define LCD_RW    1
#define LCD_E     2
#define LCD_DB4   4
#define LCD_DB5   5
#define LCD_DB6   6
#define LCD_DB7   7
```

سپس با استفاده از دستور *include* به کامپایلر اعلام می نماییم قصد استفاده از هدر *LCD* را داریم :

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

## ❖ آشنایی با هدر فایل lcd.h :

- lcd\_init() : این تابع جهت آماده سازی میکروکنترلر به منظور ارتباط با lcd استفاده می شود که می بایست حتماً در ابتدای برنامه اصلی نوشته شود .
- lcd\_clear() : از این تابع جهت پاک کردن صفحه نمایش استفاده می شود .
- lcd\_gotoxy(x,y) : از این تابع جهت آدرس دهی مکان نما استفاده می شود.
- lcd\_putsf() : از این تابع برای چاپ رشته(String) استفاده می شود.
- lcd\_putsf(char flash \*str) : برای چاپ یک کارکتر استفاده می شود.
- lcd\_putchar(char c) : برای چاپ متغیر های عددی استفاده می شود.
- lcd\_puts(char\* str) : برای چاپ عبارت روی صفحه lcd به اندازه عدد داخل آرگومان
- lcd\_shift\_right(int) : برای شیفت راست عبارت روی صفحه lcd به اندازه عدد داخل آرگومان داخل تابع استفاده می شود.
- lcd\_shift\_left(int) : برای شیفت چپ عبارت روی صفحه lcd به اندازه عدد داخل آرگومان داخل تابع استفاده می شود.
- lcd\_command(hex) : این تابع جهت ارسال فرمان های کنترلی lcd استفاده می شود . در جدول زیر کدهای کنترلی مربوط به این lcd آورده شده است.

command	توضیح عملکرد	Command Code
پاک کردن صفحه نمایش		0x01
بازگشت به home		0x02
جا به جایی مکان نما به چپ		0x04
جا به جایی مکان نما به راست		0x06
جا به جایی نمایش به چپ		0x05
جا به جایی نمایش به راست		0x07
نمایش خاموش و مکان نما خاموش		0x08
نمایش خاموش و مکان نما روشن		0x0A
نمایش روشن و مکان نما خاموش		0x0C
نمایش روشن و مکان نما روشن		0x0E
مکان نما به صورت چشمک زن		0x0F

**برنامه نمومه:** برنامه ای بنویسید که عبارت *lcd* را در مکان (۱و۱) *arm* نوشته و عبارت *microcontroller* در سطر دوم بنویسید.

```
#include <LPC17xx.h>
#define LCD_LPC1768
#define LCD_PORT_2
#define LCD_RW      1
#define LCD_E       2
#define LCD_DB4     4
#define LCD_DB5     5
#define LCD_DB6     6
#define LCD_DB7     7
#include "lcd.h"

main() {
    lcd_init();
    lcd_clear();
    lcd_gotoxy(1,1);
    lcd_putsf("ARM");
    lcd_gotoxy(2,1);
    lcd_putsf("microcontroller");
    while (1) { }
}
```

---

## ❖ دستور کار:

- . I. برنامه ای بنویسید که اسم و فامیلتان را بر روی LCD نوشته و نمایش دهد.
- . II. برنامه نوشته شده در قسمت الف را به گونه ای تغییر دهید که متن نوشته شده بر روی LCD به صورت روان از سمت راست LCD خارج شود و پس از خروج کامل آن دوباره از سمت چپ LCD وارد شود.
- . III. برنامه ای بنویسید که با زدن یک سوئیچ از ۱ تا ۹۹ به صورت صعودی بشمارد و با زدن یک سوئیچ دیگر به صورت نزولی بشمارد.
- . IV. برنامه ای یک ولوم را بنویسید که با فشردن هر بار یک سوئیچ به عدد پیشفرض (صفر) یک واحد افزوده شود و با فشردن سوئیچ دیگر یک واحد کاهش دهد. رنج مجاز عدد صفر تا ۱۰۰ می باشد.

## آزمایش پنجم : آشنایی با نمایشگرهای ال سی دی گرافیکی و رنگی

❖ اهداف:

- آشنایی ، راه اندازی و کار با ال سی دی گرافیکی
- راه اندازی و کار با ال سی دی رنگی

❖ مقدمه:

هدف از این آزمایش آشنایی و کار با نمایشگر های گرافیکی نظیر LCD و GLCD رنگی می باشد.

❖ کار با **:GLCD**

LCD مورد استفاده از نوع گرافیکی با رزولوشن  $128 \times 64$  ( دارای ۱۲۸ ستون و ۶۴ ردیف ) است که در آن از تراشه KS0108 استفاده شده است. برای راه اندازی این LCD به یک پتانسیومتر جهت تعیین ولتاژ کنترast نیاز است.

نام پایه	توضیح عملکرد
CD	برای انتخاب دستور یا دیتا
RW	برای انتخاب حالت خواندن یا نوشتن
E	فعال سازی ورودی دیتا
CS1	فعال ساز چیپ ۱
CS2	فعال ساز چیپ ۲
DB0-DB7	پایه های دو جهته برای خواندن یا نوشتن داده در LCD
RST	LCD ریست کرد

❖ آشنایی با هدرفایل **KS0180.h**

این هدرفایل حاوی دستورات آماده برای کار با LCD گرافیکی است که در نرم افزار Keil نوشته شده است .  
برخی از توابع موجود و کاربردی و نحوه استفاده از آن ها در زیر معرفی شده است :

**GLCD\_GOTO(x,y)**: جهت بردن کرسربه یک موقعیت خاص برای نوشتن یک عبارت. موقعیت x و y از بالا سمت چپ شروع می شود و در قسمت پایین سمت راست به آخرین حد خود می رسد این نکته را در مقداردادن به آرگومانها در تابع زیر در نظر بگیرید.

**GLCD\_WriteChar()**: نوشتن یک کاراکتر در موقعیت کرسر.

**GLCD\_WriteString()**: نوشتن یک عبارت در موقعیت فعلی کرسر.

**GLCD\_SetPixel()**: روشن کردن یک پیکسل در موقعیت مربوطه.

```
GLCD_SetPixel(unsigned char x, unsigned char y, ...)
```

**GLCD\_Bitmap()**: نمایش یک تصویر سیاه و سفید با استفاده ازتابع زیراستفاده میشود.

```
GLCD_Bitmap(char * bmp, unsigned char x, unsigned char y, unsigned char dx, unsigned char dy)
```

که ارگومانهای آن شامل موارد زیراست:

\***bmp**: آرایه ای که یک های آن معرف پیکسل های روشن و ترتیب نمایش آن بر روی LCD نیز از سمت بالا به پایین و چپ به راست است.

**x,y**: موقعیت شروع تصویر بر روی LCD را مشخص می کند.

**dx,dy**: طول و عرض تصویر بر روی LCD را مشخص می کند.

**GLCD\_ClearScreen()**: جهت پاک کردن کل صفحه.

**GLCD\_Rectangle()**: نمایش مستطیل توخالی در موقعیت مربوطه با طول و عرض به ترتیب **b** و **a**.

```
GLCD_Rectangle(unsigned char x, unsigned char y, unsigned char b, unsigned char a)
```

**GLCD\_Circle()**: رسم دایره ای به مرکز CX , CY و شعاع radius.

```
GLCD_Circle(unsigned char cx, unsigned char cy, unsigned char radius)
```

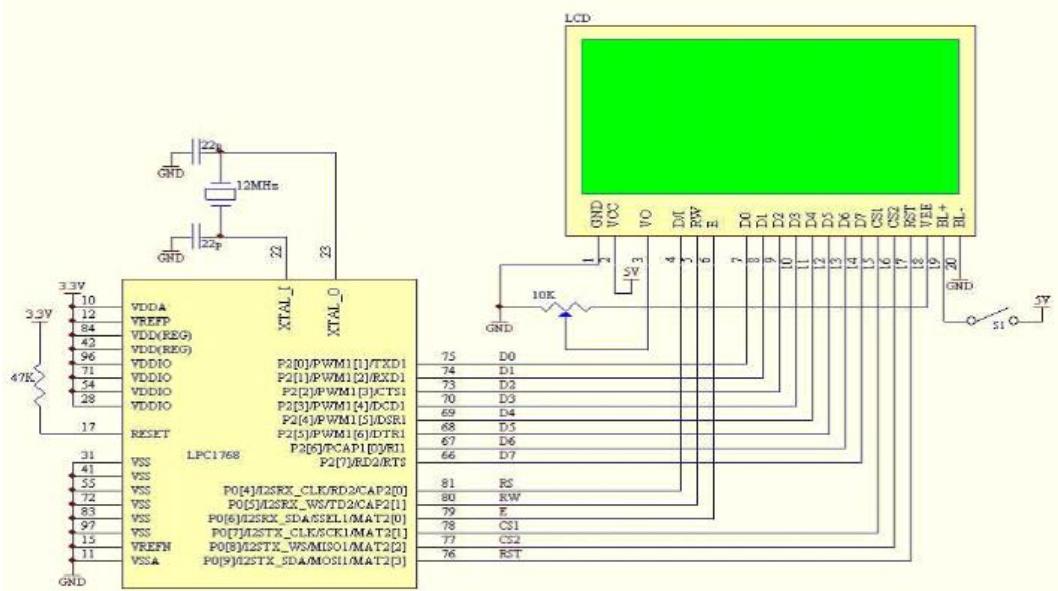
**GLCD\_Line()**: ترسیم خطی از مختصات اول به دوم.

```
GLCD_Line(unsigned int X1,unsigned int Y1,unsigned int X2,unsigned int Y2)
```

نکته: جهت نمایش تصویر Bitmap میتوان به واسطه نرم افزار مربوطه تصویر را کد تبدیل و با اضافه کردن به کد برنامه واستفاده از نام آرایه ای آن تصویر را نمایش داد.

---

برنامه نمونه: در برنامه زیر قصد داریم تا با استفاده ازتابع های کتابخانه LCD گرافیکی برنامه ای بنویسیم تا بر روی صفحه یک مربع ، خط و دایره نمایش دهد و در زیر آن عبارت ARM laboratory را نمایش دهد .



```

#include <LPC17xx.h>
#define GLCD_LPC17xx
#define GLCD_DATAPORT_2
#define GLCD_CONTROLPORT_0
#define GLCD_RS 4
#define GLCD_RW 5
#define GLCD_EN 6
#define GLCD_CS1 7
#define GLCD_CS2 8
#define GLCD_CS3 9
#define GLCD_D0 0
#include "ks0108.h"

int main(){
    SystemInit();
    GLCD_Initialize();
    GLCD_ClearScreen();
    GLCD_Line(50,20,60,30);
    GLCD_Rectangle(10,15,15,15);
    GLCD_Circle(110,25,8);
    GLCD_GoTo(25,5);
    GLCD_WriteString("ARM laboratory");
    while(1);
}

```

## ❖ کار با LCD رنگی :

LCD مورد استفاده در این برد از نوع ۳۲۰ \* ۲۴۰ می باشد. این بدان معناست که این نمایشگر دارای ۳۲۰ پیکسل عمودی و ۲۴۰ پیکسل افقی می باشد. این LCD با نام T28-ILI9325-ILI9328 معرفی شده که در بازار به N96 موبایلی LCD معروف است. چیپ کنترل آن ILI9325-ILI9328 است که نمونه برنامه و هدر فایل های آن برای میکروکنترلرهای مختلف موجود است. برای راه اندازی این LCD از هدر فایل tftlcd\_functions.h استفاده می نماییم که این هدر فایل را باید در برنامه های ای که برای این LCD می نویسیم اضافه کنیم .

نام پایه	توضیح عملکرد
RS	برای انتخاب دستور یا دیتا - ۰ برای دستور و ۱ برای دیتا
WR	برای انتخاب حالت خواندن یا نوشت - ۰ برای نوشت و ۱ برای خواندن
CS	فعال ساز چیپ ("L")
RD	فعال ساز خواندن و نوشت - ۰ برای غیر فعال کردن خواندن و نوشت و ۱ برای فعال کردن خواندن و نوشت
DB0 - DB16	پایه های دو جهته برای خواندن یا نوشت داده در LCD
RST	ریست کرد LCD

## ❖ آشنایی با هدر فایل tftlcd\_functions.h

این هدر فایل حاوی دستورات آماده برای کار با LCD رنگی است که در نرم افزار Keil نوشته شده است .  
برخی از توابع موجود و کاربردی و نحوه استفاده از آن ها در زیر معرفی شده است :

**tftlcd\_init()**: این تابع LCD را برای نوشت اطلاعات آماده می کند که حتماً می بایست در ابتدای برنامه یک بار نوشته شود .

**tftlcd\_clear()** : از این تابع جهت پاک کردن صفحه نمایش استفاده می شود .

**tftlcd\_fill(color)** : از این تابع جهت پر کردن صفحه نمایش با یک رنگ استفاده می شود. برای استفاده از این تابع باید رنگ مورد نظر که قصد داریم صفحه نمایش را به آن رنگ در آوریم به عنوان ورودی به آن بدھیم.

**tftlcd\_write\_pic\_fast(\*pointer)** : از این تابع جهت نمایش عکس استفاده می شود .

برای استفاده از این تابع باید اشاره گر به آرایه ای که عکس مورد نظر را در آن ذخیره کرده ایم را به عنوان ورودی به این تابع بدهیم .

```
void tftlcd_write_pic(const unsigned short *pointer)
```

از این تابع جهت رسم خط استفاده می شود . برای استفاده از این تابع باید مختصات نقطه ابتدا و انتهای خط مورد نظر و رنگی که برای آن انتخاب کرده اید را به ترتیب در ورودی های تابع بالا وارد کنیم .

```
void tftlcd_draw_line(int, int, int, int, int)
```

برای استفاده از این تابع باید مختصات مرکز شعاع و پر یا تو خالی بودن دایره(1) برای توپر بودن و صفر برای توخالی بودن ) و همچنین رنگ دایره به ترتیب به عنوان ورودی به تابع داده شود .

```
void tftlcd_draw_circle(int x0, int y0, int radius, char fill, int color)
```

برای استفاده از این تابع برای رسم مستطیل استفاده می شود . برای استفاده از این تابع باید مختصات دو سر قطر آن توپر یا توخالی بودن(همانند دایره) و رنگ آن را مشخص کنیم .

```
void tftlcd_draw_rectangle(int x0, int y0, int x1, int y1, char fill, int color)
```

**برنامه نمونه:** در نمونه برنامه زیر قصد داریم با استفاده از تابع هایی که در هدر فایل tftlcd\_functions.h موجود است ابتدا کل صفحه را روشن و سپس یک دایره تو پر در وسط LCD به شعاع ۳۰ پیکسل ایجاد کنیم.

```
#include <lpc17xx.h>
#define family_lpc17xx
#define TFTLCD_DATAPORT_1
#define TFTLCD_DATAPORT_OFFSET 15
#define TFTLCD_CONTROLPORT_0
#define TFTLCD_RST 26
#define TFTLCD_RS 23
#define TFTLCD_CS 22
```

```

#define TFTLCD_RD 25
#define TFTLCD_WR 24
#define PORTRAIT
#include "tftlcd_functions.h"
int main() {
    SystemInit();
    tftlcd_init();
    tftlcd_clear();
    tftlcd_fill(0x0000ff);
    wait_ms(4000);
    tftlcd_draw_circle(120,160,30,1,0xdbc311);
    while(1);
}

```

---

## ❖ دستور کار:

- I. برنامه ای بنویسید که در داخل یک مستطیل اسم خود و در داخل یک دایره فامیل خود ، سپس این دو در جهت قریبه های ساعت بچرخانید.
- II. برنامه فوق را برروی ال سی دی رنگی تکرار کنید.

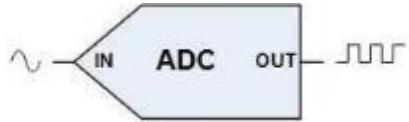
## آزمایش ششم: آشنایی با مبدل قیاسی به رقمی و حسگر دما

### ❖ اهداف

- آشنایی با نحوه تبدیل سیگنال قیاسی به رقمی
- استفاده از مبدل قیاسی به رقمی
- استفاده از حسگر دما LM35

### ❖ مقدمه:

در دنیای خارج از میکرو تمامی مقادیر و سیگنال‌ها و کمیت‌ها پیوسته هستند، اما در دنیای دیجیتال کمیت‌ها و سیگنال‌ها شکل گسسته به خود می‌گیرند. بنابراین برای به کار بردن این مقادیر و کمیت‌ها در سیستم‌های دیجیتال ناگزیر استفاده از مبدلی برای تبدیل سیگنال‌های آنالوگ به دیجیتال می‌باشد. در این آزمایش با نحوه تبدیل سیگنال‌های آنالوگ به دیجیتال آشنا شده و سپس اطلاعات یک حسگر دما به درون میکروکنترلر خوانده می‌شود.



سنسور‌ها مقدار یک کمیت آنالوگ را به ولتاژ یا جریان تبدیل می‌کنند، سپس این ولتاژ آنالوگ به مبدل آنالوگ به دیجیتال میکرو داده می‌شود و مبدل آنالوگ به دیجیتال با استفاده از روش‌هایی مقدار ولتاژ را به کمیت دیجیتال متناظر تبدیل می‌کند، سپس این مقدار دیجیتال با اعمال ریاضی به مقدار عددی متناظر تبدیل می‌شود و روی LCD و 7-segment نمایش داده می‌شود.

### ❖ ویژگی‌های ADC میکروکنترلر : LPC1768

- دقت تبدیل ۱۲ بیت
- ۸ کanal ورودی
- قابلیت کاهش توان
- محدوده اندازه گیری بین VREFP و VREFN
- فرکانس نمونه برداری 200KHz
- دارای وضعیت Burst و نرم افزاری (معمولی)

## ❖ بررسی رجیسترها و نحوه استفاده از ADC :

همانطور که گفته شد از ADC میکرو LPC1768 در دو حالت نرم افزاری و Burst می توان استفاده نمود. در حالت نرم افزاری برای خواندن مقدار مورد نظر هر بار میکرو تنظیماتی را انجام و سپس مقدار خوانده شده را به ما می دهد. ۸ کanal ADC میکروکنترلر LPC1768 به پایه های زیر متصل هستند:

AD0.0 (A/D converter 0 input 0) [P0.23]
AD0.1 (A/D converter 0 input 1) [P0.24]
AD0.2 (A/D converter 0 input 2) [P0.25]
AD0.3 (A/D converter 0 input 3) [P0.26]
AD0.4 (A/D converter 0 input 4) [P1.30]
AD0.5 (A/D converter 0 input 5) [P1.31]
AD0.6 (A/D converter 0 input 6) [P0.3]
AD0.7 (A/D converter 0 input 7) [P0.2]

برای استفاده از ADC ابتدا با ید دو کار انجام داد :

.I فعال کردن رجیستر توان مربوط به ADC

.II تنظیمات رجیستر مربوط به PINSEL را در وضعیت ADC قرار دهیم. به عنوان مثال برای استفاده از

کanal ADC0.0 تنظیمات زیر را انجام می دهیم:

```
LPC_PINCON->PINSEL1 = 1<< 14;           // p0.23 select as ADC0.0  
LPC_SC->PCONP |= (1 << 12);           // Enable power to AD block
```

بررسی رجیستر **ADCR** ( رجیستر کنترل مبدل A/D ) : بیت های صفر تا ۷ این رجیستر جهت انتخاب کanal ADC استفاده می شود و در مد نرم افزاری تنها یکی از این بیت ها می تواند یک باشد. به عنوان مثال جهت انتخاب کanal صفر یعنی AD0.0 از دستور زیر استفاده می نماییم :

```
LPC_ADC->ADCR |= 0x01;           //select AD0.0
```

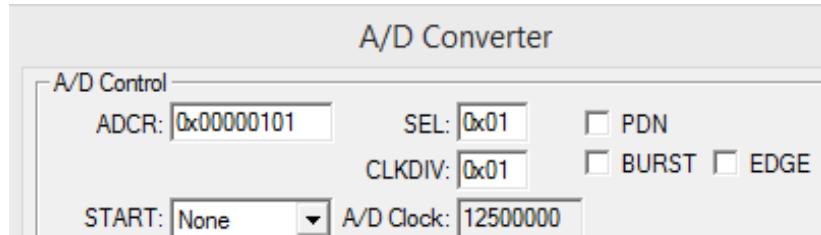
بیت های ۸ تا ۱۵ به نام CLKCIV جهت تنظیم کردن کلک ADC استفاده می شود. براساس مقداری که به آن می دهیم فرکانس کلک ADC تقسیم می شود و حداقل کلک 13MHz ADC است. رابطه این تقسیم در زیر آمده است :

CLKDIV= (PCLK/ADCLK)-1

ADCLK=PCLK / (CLKDIV+1)

به عنوان مثال با PCLK=25MHz جهت داشتن فرکانس کلک ADC، 12.5MHz بیت های ۸ تا ۱۵ به صورت زیر مقدار دهی می شوند:

```
LPC_ADC->ADCR |= 0x0100;           //ADCLK is 12.5 MHz
```



قسمت مهم بعدی این رجیستر بیت شماره ۲۱ یعنی ADC را فعال می نماید:

```
LPC_ADC->ADCR |= 0x200000;
```

بیت های شماره ۲۴ تا ۲۶ این رجیستر نیز جهت START نمونه برداری استفاده می شوند ، وقتی مقدار ۱ را به آن ها دهیم، عملیات تبدیل شروع به اجرا می شود. البته شروع نمونه برداری را توسط قسمت های دیگر مثل پایه P2.10 و یا تایمرها نیز می توان کنترل نمود ، که در تصویرهای زیر لیست قسمت هایی که می توانند عملیات کنترل شروع نمونه برداری را انجام دهند آمده است :

26:24	START	When the BURST bit is 0, these bits control whether and when an A/D conversion is started:	0
000	No start (this value should be used when clearing PDN to 0).		
001	Start conversion now.		
010	Start conversion when the edge selected by bit 27 occurs on the P2.10 / EINT0 / NMI pin.		
011	Start conversion when the edge selected by bit 27 occurs on the P1.27 / CLKOUT / USB_OVRCRn / CAP0.1 pin.		
100	Start conversion when the edge selected by bit 27 occurs on MAT0.1. Note that this does not require that the MAT0.1 function appear on a device pin.		
101	Start conversion when the edge selected by bit 27 occurs on MAT0.3. Note that it is not possible to cause the MAT0.3 function to appear on a device pin.		
110	Start conversion when the edge selected by bit 27 occurs on MAT1.0. Note that this does not require that the MAT1.0 function appear on a device pin.		
111	Start conversion when the edge selected by bit 27 occurs on MAT1.1. Note that this does not require that the MAT1.1 function appear on a device pin.		

```
LPC_ADC->ADCR |= 0x01000000; // Start A/D Conversion
```

**رجیستر ADGDR** : این رجیستر وضعیت مبدل آنالوگ به دیجیتال را مشخص می نماید و یکی از بیتهاي مهم این رجیستر برای ما بیت شماره ۳۱ می باشد که به وسیله این بیت متوجه می شویم که تبدیل انجام شده است یا نه و زمانی که تبدیل کامل شد این بیت یک می شود.

**رجیستر AD0DR8 تا AD0DR0** : در این رجیستر مقدار دیجیتال سیگنال تبدیل شده در بیتهاي ۴ تا ۱۵ قرار می گیرد و برای خواندن ADC از این رجیستر استفاده می نماییم.

```
i = LPC_ADC->ADDR0;
```

**برنامه نمونه اول:** در این برنامه قصد داریم تا کانال AD0 را راه اندازی کرده و سپس مقدار تبدیل شده را برگرداند.

```
unsigned short ADC_Read(void)
{
    unsigned int i;
    LPC_PINCON->PINSEL1 = 0x00004000; // p0.23 select as ADC0.0
    LPC_SC->PCONP |= (1 << 12); // Enable power to AD block
    LPC_ADC->ADCR |= 0x01; //select AD0.0
    LPC_ADC->ADCR |= 0x0100; //ADCLK is 12.5 MHz
    LPC_ADC->ADCR |= 0x200000; //Power up,1 << 21
    LPC_ADC->ADCR |= 0x01000000; // Start A/D Conversion
    while ((LPC_ADC->ADGDR & 0x80000000) == 0); // Wait for end of
    A/D Conversion
    i = LPC_ADC->ADDR0; // Read A/D Data Register
    return (i >> 4) & 0xFFFF; // bit 4:15 is 12 bit AD value
}
```

---

**برنامه نمونه دوم:** برنامه ای بنویسید که به وسیله سنسور دما lm35 دما را خوانده و بر روی LCD نمایش دهد.

```
#include <lpc17xx.h>
#define LCD_LPC1768
#define LCD_PORT_2
#define LCD_RS 0
#define LCD_RW 1
#define LCD_E 2
#define LCD_DB4 4
#define LCD_DB5 5
#define LCD_DB6 6
#define LCD_DB7 7
#include "lcd.h"
void delay(int w) {
    while(w--);
}
int main() {
    int i = 0;
    LPC_GPIO2 -> FIODIR = 0xffffffff;
    LPC_PINCON -> PINSEL1 = 1<<14; // Select P23.0 as ADC
    LPC_SC -> PCONP |= (1<<12); // Enable power to AD block
    LPC_ADC -> ADCR |= 0x01; // Select ADC0.0
    LPC_ADC -> ADCR |= 0x0400; // ADC Clock is 25MHz / 5
    LPC_ADC -> ADCR |= 0x200000; // Power up,1 << 21
```

```

while(1) {
    LPC_ADC -> ADCR |= (0<<24); // Stop ADC
    LPC_ADC -> ADCR |= (1<<24); // Start ADC
    while((LPC_ADC -> ADGDR & 0x80000000) == 0); // Wait for
    end of A/D Conversion
    i = (LPC_ADC -> ADGDR & 0xffff0);
    i = i / 12.41; // (3.3V / 2^12)=0.0806 mV 10mV / 0.0806
    = 12.41 unit
    lcd_clear();
    lcd_gotoxy(1,1);
    lcd_puts(i);
    delay(2000000);
}
}

```

---

## ❖ دستور کار

- I. برنامه ای بنویسید که هر ده ثانیه یکبار اطلاعات دما را خوانده و بروی نمایشگر نشان دهد
- II. برنامه ای بنویسید که در صورت فشردن یک کلید اطلاعات دما بروی نمایشگر بروز رسانی گردد

## آزمایش هفتم: آشنایی با مبدل رقمی به قیاسی

### ❖ اهداف:

- استفاده از مبدل رقمی به قیاسی

- تولید موج سینوسی با استفاده از مبدل رقمی به قیاسی

### ❖ مقدمه:

پس از انجام پردازش های لازم بروی سیگنال های رقمی درون پردازنده ها لازم است که سیگنال های تولیدی مجدداً بصورت یک سیگنال آنالوگ به سیستم مورد نظر اعمال شود. برای تبدیل سیگنال دیجیتال تولیدی بوسیله کامپیوتر به یک سیگنال آنالوگ از IC های DAC استفاده می شود. در این آزمایش با واسط DAC میکروکنترلر آشنا شده و سپس یک موج سینوسی بروی خروجی میکرو ایجاد می کنید.

### ❖ مبدل دیجیتال به آنالوگ (DAC)

یک مبدل دیجیتال به آنالوگ یک کد دیجیتال n بیتی را عنوان ورودی گرفته و یک ولتاژ یا جریان آنالوگ را به عنوان خروجی تحويل می دهد.

یک ADC عموماً شامل قسمت های زیر است:

- یک ولتاژ مرجع دقیق و پایدار
- خود مبدل DAC
- یک تقویت کننده عملیاتی (Op-Amp)

وظیفه اصلی مبدل های دیجیتال تبدیل داده های دیجیتال به داده های آنالوگ یا پیوسته می باشد. به عنوان مثال یک صوت که به صورت دیجیتال نمونه برداری شده است را می توان به یک سیگنال آنالوگ قابل پخش در بلندگو تبدیل نماییم و یا می توان به وسیله آن شکل موج های آنالوگ را تولید نمود. میکرو کنترلر LPC1768 دارای یک کanal مبدل دیجیتال به آنالوگ ۱۰ بیتی است. خروجی این مبدل DAC به پایه P0.26 میکروکنترلر متصل می باشد. اولین کاری که در استفاده از مبدل DAC باید انجام دهیم این است که پایه مورد نظر یعنی P0.26 در وضعیت DAC ، یعنی حالت ۲ قرار دهیم. این کار با رجیستر PINSEL مربوط انجام می دهیم.

LPC\_PINCON->PINSEL1 = 2<<20

رجیستر اصلی مبدل دیجیتال به آنالوگ ، رجیستر DACR می باشد. که مقدار دیجیتالی را که می خواهیم به آنالوگ تبدیل شود در بیت شماره ۶ تا ۱۵ قرار می دهیم و طبق رابطه زیر مقدار ولتاژ خروجی به دست می آید.

$$\text{VALUE} \times ((V_{\text{REFP}} - V_{\text{REFN}})/1024) + V_{\text{REFN}}$$

دو مقدار VREFP و VREFN دو ولتاژ مثبت و منفی مرجع هستند که برای تبدیل بین سطوح صفر تا ۳,۳ ولت آن ها را به ترتیب ۳,۳ و صفر متصل می نماییم. پین های VSSA و VDDA نیز پین های تغذیه بلوک های DAC و ADC می باشند که برای جلوگیری از تبادل نویز ، از تغذیه بخش دیجیتال ایزووله شده اند که این پین ها را نیز برای استفاده در سطح صفر ولت تا ۳,۳ ولت می توان به ترتیب به سطح ولتاژ های ۳,۳ ولت و صفر ولت متصل شوند. به دلیل اینکه بیتهاش شماره ۶ تا ۱۵ مقدار دیجیتال مورد نظر ما را به مقدار آنالوگ تبدیل می نماید ، هنگامی که می خواهیم دیتا خود را قرار دهیم باید آن را به اندازه ۶ واحد به سمت چپ شیفت دهیم تا در بیتهاش مربوطه قرار گیرند.

```
LPC_DAC->DACR= (sin[i]<<6);
```

---

**برنامه نمونه:** برنامه ای بنویسید که یک شکل موج سینوسی در پایه P0.26 میکرو ایجاد نماید.

```
#include "LPC17xx.h"
volatile uint16_t GussInTable[45] =
{
    410, 467, 523, 576, 627, 673, 714, 749, 778,
    799, 813, 819, 817, 807, 789, 764, 732, 694,
    650, 602, 550, 495, 438, 381, 324, 270, 217,
    169, 125, 87, 55, 30, 12, 2, 0, 6,
    20, 41, 70, 105, 146, 193, 243, 297, 353
};

int main(void) {
    uint8_t i;
    LPC_PINCON->PINSEL1 = 2<<20; /* set p0.26 to DAC output */
    while (1) {
        for (i = 0; i < 45; i++) {
            LPC_DAC->DACR = (GussInTable[i] << 6);
        }
    }
}
```

## ❖ دستور کار:

- .I. برنامه ای بنویسید که یک شکل موج دندانه اره ای تولید نماید.
- .II. برنامه ای بنویسید که به وسیله مبدل DAC و تقویت کننده صوتی موجود در بورد یک صدای آذیر تولید نماید.

## آزمایش هشتم: آشنایی با شمارنده و زمانسنج

### ❖ اهداف:

- آشنایی با واحد شمارنده
- معرفی ثبات های شمارنده و زمانسنج و کاربا آنها

### ❖ مقدمه:

در این آزمایش با واحد شمارنده و زمانسنج میکروکنترلر LPC1768 و قابلیتهای آن آشنا می شویم . واحد زمانسنج یک بخش مهم در برنامه ریزی ارتباطات با مدارهای جانبی می باشد.

### ❖ معرفی واحد زمانسنج میکروکنترلر 1768 :

میکروکنترلر 1768 دارای ۴ تایمیر ۳۲ بیتی می باشد که به صورت تایمیر و یا شمارنده مورد استفاده قرار می گیرند. به طور کلی تایمیرها به شمارش کلک می پردازند و براساس این شمارش به زمانی خاصی می توان رسید و سپس در زمان سرریز شدن تایمیر کار مد نظر ما را انجام دهند. در حالت کانتر نیز به شمارش پالس های بیرونی می پردازند.

### ❖ معرفی رجیستر های تایمیر و نحوه پیکر بندی تایمیر:

رجیستر **TC** : ریجستر تایمیر/کانتر. این ریجستر که وظیفه ذخیره شمارنده تایمیر را بر عهده دارد یک ریجستر ۳۲ بیتی است . که قادر به شمارش از صفر تا 0xFFFFFFFF می باشد . این شمارنده هر PR+1 سیکل ، یک واحد اضافه خواهد شد .

رجیستر **PR** : ریجستر Prescaler. هنگامی که مقدار ریجستر (Prescalre Counter) با این ریجستر برابر شود در کلک بعدی یک واحد به این ریجستر TC اضافه خواهد شد .

رجیستر **PC** : این ریجستر با آمدن هر کلک یک واحد اضافه می شود و تا زمانی که این مقدار با مقدار موجود در PR برابر نشود ، افزایش ادامه دارد . پس از برابری PC و PR یک واحد به TC اضافه شده و ریجستر PC صفر می شود .

**رجیستر TCR** : رجیستر کنترلی تایمر است. از این رجیستر جهت فعال و غیر فعال کردن تایمر و کانتر استفاده می شود. در ابتدای کار با تایمر باید ابتدا توسط این رجیستر باید تایمر ریست شود . بیت صفرم این رجیستر جهت فعال کردن تایمر و بیت اول جهت ریست کردن به کار می رود که با نوشتن دستورات زیر می توان این کار انجام داد. به جای X شماره تایмер را قرار می دهیم .

```
LPC_TIMx->TCR = 2; // reset counter
LPC_TIMx->TCR = 1; //enable counter
```

**رجیستر MCR** : زمانی که مقدار تایمر به رجیستر مقایسه می رسد با استفاده از این رجیستر می توان مشخص نمود که چه اتفاقی رخ دهد. در جدول زیر بیت های این رجیستر مشخص شده است . به عنوان مثال در صورتی که از رجیستر مقایسه MR0 استفاده کنیم با سه بیت اول این رجیستر می توان سه حالت راه اندازی وقفه ، ریست کردن شمارنده و متوقف کردن شمارنده را انجام داد . و با دستور روبرو آن را مقدار دهی می کنیم. در صورتی که به خواهیم از وقفه آن استفاده کنیم باید تنظیمات مربوط به وقفه ها نیز انجام شود.

Table 429. Match Control Register (T[0/1/2/3]MCR - addresses 0x4000 4014, 0x4000 8014, 0x4009 0014, 0x4009 4014)  
bit description

Bit	Symbol	Value	Description	Reset Value
0	MR0I	1	Interrupt on MR0: an interrupt is generated when MR0 matches the value in the TC.	0
		0	This interrupt is disabled	
1	MR0R	1	Reset on MR0: the TC will be reset if MR0 matches it.	0
		0	Feature disabled.	
2	MR0S	1	Stop on MR0: the TC and PC will be stopped and TCR[0] will be set to 0 if MR0 matches the TC.	0
		0	Feature disabled.	
3	MR1I	1	Interrupt on MR1: an interrupt is generated when MR1 matches the value in the TC.	0
		0	This interrupt is disabled	
4	MR1R	1	Reset on MR1: the TC will be reset if MR1 matches it.	0
		0	Feature disabled.	
5	MR1S	1	Stop on MR1: the TC and PC will be stopped and TCR[0] will be set to 0 if MR1 matches the TC.	0
		0	Feature disabled.	

فرض کنید بخواهیم در صورتی که مقدار تایمر با رجیستر MR0 برابر شد ، تایمر ریست و وقفه فعال شود با دستورات زیر این کار را انجام می دهیم.

```
LPC_TIM0->MCR = 3;
NVIC_SetPriority(TIMER0_IRQn, 0);
NVIC_EnableIRQ(TIMER0_IRQn);
```

**رجیستر تطابق MRx** : شامل چهار رجیستر به شماره صفر تا ۳ است و مقدار تایمرها با این رجیستر مقایسه شده و براساس رجیستر MCR که گفته شد ، سه حالت مذکور رخ می دهد.

کلک مربوط به بلوک TIMER توسط ریجستر های PCLKSEL0 و PCLKSEL1 به صورت پیش فرض  
برابر با  $\frac{CPU_{clk}}{4}$  تنظیم شده است . به همین صورت کلک متصل به ریجستر TC از رابطه زیر به دست می آید :

$$\frac{\frac{CPU_{clk}}{4}}{\text{prescaler}}$$

همچنین داریم :

$$M_{Rx} = \frac{t * PCLK_{TIMx}}{\text{prescaler}}$$

**برنامه نمونه:** برنامه ای بنویسید که با تایمر صفر LED متصل به پورت 2.1 را دو ثانیه روشن و یک ثانیه خاموش شود.

```
#include <lpc17xx.h>
int i = 0;
int dummy = 0;
void delay() {
    dummy++;
}
void TIMER0_IRQHandler(void) {
    if(i > 0) {
        LPC_GPIO2 -> FIOPIN = 0xf;
        i++;
        if(i == 3)
            i = 1;
    }
    else{
        i++;
        LPC_GPIO2 -> FIOPIN = 0x0;
    }
    LPC_TIM0 -> IR = (1 << 0);           //Clear interrupt flag
    delay();
}
int main() {
    LPC_TIM0 -> CTCR = 0; //Set timer mode and TC incremented when
the PC matches the PR
    LPC_TIM0 -> PR = 0 ;           //Set prescale 0
    LPC_TIM0 -> MRO = 50000000;    //set MRO for 1s
    LPC_TIM0 -> TCR = 2;          //Reset Timer 1
    LPC_TIM0 -> MCR = 3;          //The TC will be reset if MRO
matches it and the TC and PC will be stopped
}
```

```

NVIC_SetPriority(TIMER0_IRQn, 0);
NVIC_EnableIRQ(TIMER0_IRQn);
LPC_GPIO2->FIODIR = 0xf;
LPC_GPIO3->FIODIR = 1<<26;
LPC_TIM0->TCR = 1; //Enable Timer 1
while(1){
}
}

```

نکته مهم در این برنامه پاک کردن پرچم وقفه ها می باشد.

**بررسی رجیستر EMR :** چهار بیت اول آن در صورت تطابق می توانند toggle ، از سطح صفر به یک و بر عکس روند. بیتهاي EMCx مشخص می کند که زمانی که تطابق رخ داد چه حالتی برای پایه MATx رخ دهد. نکته مهم این است که جهت استفاده از این پایه ها است که ابتدا باید آنها را با رجیستر PINSELL در مدار قرار دهیم. به عنوان مثال فرض کنید بخواهیم در صورت تطابق پایه MAT0.0 که مربوط به پورت ۱ و بیت شماره ۲۸ آن است Toggle شود.

```

LPC_TIM0->EMR = 0x30;
LPC_PINCON->PINSEL3 = 3<< 24;

```

---

## ❖ کانتر

همانطور که می دانید تفاوت کانتر با تایмер ادر این است که کانتر کلاک های خارجی را می شمارد در حالی که تایмер کلاک داخلی خود را می شمارد. رجیسترهاي کانتر همه مثل تایмер هستند و تنها با یک رجیستر کنترلی تایмер را در مدار کانتر قرار می دهیم.

**بررسی رجیستر CTCR :** این رجیستر جهت کنترل کانتر می باشد در صورتی که به دو بیت اول آن مقدار ۰۰ را دهیم در مدار تایмер و در صورتی که مطابق جدول زیر مقادیر دیگری دهیم می توان در مدار کانتر استفاده نمود. به عنوان مثال با قرار دادن مقدار ۰۱ به این رجیستر در لبه های بالا رونده را می شمارد.

Table 428. Count Control Register (T[0/1/2/3]CTCR - addresses 0x4000 4070, 0x4000 8070, 0x4009 0070, 0x4009 4070) bit description

Bit	Symbol	Value	Description	Reset Value
3:2	Count Input Select		When bits 1:0 in this register are not 00, these bits select which CAP pin is sampled for clocking.	00
		00	CAPn.0 for TIMERn	
		01	CAPn.1 for TIMERn	
		10	Reserved	
		11	Reserved	
			<b>Note:</b> If Counter mode is selected for a particular CAPn input in the TnCTCR, the 3 bits for that input in the Capture Control Register (TnCCR) must be programmed as 000. However, capture and/or interrupt can be selected for the other 3 CAPn inputs in the same timer.	
31:4	-	-	Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

برنامه نمونه: برنامه ای بنویسید که با فشردن کلید متصل به CAP1.0 را به کانتر یک واحد اضافه شود و نتیجه را بر روی LCD کارکتری نمایش دهد.

```
#include <lpc17xx.h>
#define LCD_LPC1768
#define LCD_PORT_2
#define LCD_RS 0
#define LCD_RW 1
#define LCD_E 2
#define LCD_DB4 4
#define LCD_DB5 5
#define LCD_DB6 6
#define LCD_DB7 7
#include "lcd.h"
void delay(){
    int i =0;
    for(i=0;i<50000 ; i++);
}
int main(){
    LPC_TIM1 -> CTCR = 1; //Set Counter mode and
                           //rising edges on the CAP
    LPC_TIM1 -> PR = 0 ; //Set prescale 0
    LPC_TIM1 -> MRO = 0xFFFF;
    LPC_TIM1 -> TCR = 2; //Reset Counter 1
    LPC_TIM1 -> MCR = 3; //The TC will be reset
    if MRO mach and the TC and PC will be stopped
    LPC_PINCON -> PINSEL3 = 0x00000030; //Set for CAP1.0
    LPC_GPIO2 -> FIODIR = 0xff;
    LPC_TIM1 -> TCR = 1; //Enable Counter 1
```

```
lcd_gotoxy(1,0);
lcd_clear();
while(1){
    delay();
    lcd_puts(LPC_TIM1 -> TC);
}
}
```

---

## ❖ دستور کار

. ا. برنامه ای بنویسید که با تنظیم تایمرها هر دقیقه اطلاعات سنسور دما را خوانده و بروی نمایشگر نشان دهد.

# آزمایش نهم: آشنایی با مدولاسیون عرض پالس و موتورهای DC

❖ اهداف:

- آشنایی با واحد PWM میکروکنترلر LPC1768
- آشنایی با کاربردهای PWM
- راه اندازی موتور DC

❖ مقدمه:

مدولاسیون پهنهای باند جهت تولید موج های مربعی با Duty Cycle قابل تنظیم به کار می رود . با استفاده از این موج های مربعی و تغییر مدت زمان یک بودن آن ها می توان سرعت موتورهای الکتریکی جریان مستقیم را کنترل نمود. کاربردهای دیگر این بخش می توان به تولید موجهای مربعی برای منابع تغذیه سویچینگ ، درایو موتورهای سه فاز، UPS و ... را نام برد. در این آزمایش با واحد PWM و همچنین راه اندازی موتورهای DC آشنا می شویم.

## ❖ آشنایی با واحد PWM میکرو کنترلر LPC1768 :

بلوک PWM میکرو lpc1768 در اصل یک تایمر کانتر ۳۲ بیتی می باشد که دارای خروجی جهت تولید امواج PWM می باشد. در این میکرو ۶ کانال خروجی PWM وجود دارد .

بررسی رجیسترها و نحوه راه اندازی بلوک PWM: در راه اندازی واحد PWM ابتدا باید پایه مد نظرمان را با استفاده از رجیستر PINSEL3 در حالت PWM قرار دهیم. فرض کنید بخواهیم از کانال دوم PWM استفاده کنیم. حال از پایه P1.20 به عنوان PWM استفاده می کنیم و رجیستر PINSEL3 را با دستور زیر در حالت PWM قرار می دهیم.

```
LPC_PINCON -> PINSEL3 = 0x00000200;
```

رجیستر TCR: رجیستر کنترل تایمر می باشد. این رجیستر شبیه رجیستر TCR در تایمر هاست. در ابتدا باید توسط این رجیستر تایمر PWM را ریست کنیم و پس از انجام تنظیمات دیگر رجیسترها و زمانی که خواستیم از PWM استفاده کنیم آن را فعال می کنیم. بیت صفر این رجیستر جهت فعال کردن و بیت یک جهت ریست کردن به کار می رود. با دو دستور زیر می توان تایمر را ریست و فعال نمود.

```
LPC_PWM1->TCR = 0x02;//reset pwm counter  
LPC_PWM1->TCR = 0x01;//enable pwm counter
```

**رجیستر MR0 :** مقدار کانتر PWM با رجیستر MR0 مقایسه می شود و هنگامی که این دو رجیستر برابر شدند بسته به تنظیم رجیستر MCR سه حالت وقفه ، ریست شدن کانتر و توقف کانتر رخ می دهد. در واقع به وسیله این رجیستر می توان فرکانس موج PWM را کنترل نمود . به عنوان مثال اگر فرکانس تایمر روی 25MHz تنظیم شده باشد با قرار دادن ۱۰۰ در MR0 می توان یک موج PWM با فرکانس 250KHz تولید نمود.

```
LPC_PWM1->MR0 = 100;
```

**رجیستر MCR :** رجیستر تطابق تایمر است و بسته به تنظیم آن می توان در صورت تطابق شمارنده کانتر با رجیسترهاي MR6 تا MR1 ، MR0 سه حالت وقفه ، ریست شدن کانتر و توقف رخ دهد. به عنوان مثال با تنظیم این رجیستر به صورت زیر هنگامی که شمارنده تایمر با MR0 برابر شد شمارنده ریست می شود.

```
LPC_PWM1->MCR = 0x02; /* setup match control to reset on MR0 */
```

**رجیستر PCR :** رجیستر کنترل PWM است که توسط آن می توان کanal PWM مد نظر مان را فعال نمود. به عنوان مثال در صورتی که بخواهیم از کanal دوم PWM را فعال نماییم از دستور زیر استفاده می کنیم.

```
LPC_PWM1->PCR = 1 << 10;
```

**رجیسترهاي MR6 تا MR1 :** طریقه کار کرد واحد PWM در میکرو LPC1768 به این صورت است که شمارنده از مقدار صفر شروع به شمارش می کند تا به مقدار رجیسترهاي MR6 تا MR1 برسد و زمانی که به مقدار این رجیسترها رسید خروجی PWM از مقدار یک به صفر می رود شمارنده به شمارش ادامه می دهد تا به مقدار رجیستر MR0 رسد و زمانی که به این مقدار رسید خروجی PWM از صفر به یک می رود و تایمر بسته به تنظیم رجیستر MCR می تواند وقفه ، ریست و متوقف شود.

**رجیستر PR :** این رجیستر فرکانس واحد PWM را تقسیم می کند. این رجیستر معرف PreScale تایمر کانتر مربوطه می باشد که با رسیدن رجیستر PC به آن باعث افزایش یک واحدی تایمر PWM می شود.

**رجیستر TC :** رجیستر ۳۲ بیتی حاوی مقدار فعلی کانتر واحد PWM است.

**رجیستر CTCR :** رجیستر کنترل کانتر می باشد و به وسیله آن مشخص می کنیم که تایمر در حالت عادی باشد و کلاک داخلی را بشمارد و یا می توان آن را در مدد کانتر قرار داد و مقدار پایه های PCAP را شمارش کند.

Table 448. PWM Count control Register (PWM1CTCR - address 0x4001 8070) bit description

Bit	Symbol	Value	Description	Reset Value
1:0	Counter/ Timer Mode	00	Timer Mode: the TC is incremented when the Prescale Counter matches the Prescale Register.	00
		01	Counter Mode: the TC is incremented on rising edges of the PCAP input selected by bits 3:2.	
		10	Counter Mode: the TC is incremented on falling edges of the PCAP input selected by bits 3:2.	
		11	Counter Mode: the TC is incremented on both edges of the PCAP input selected by bits 3:2.	
3:2	Count Input Select		When bits 1:0 of this register are not 00, these bits select which PCAP pin which carries the signal used to increment the TC.	00
		00	PCAP1.0	
		01	PCAP1.1 (Other combinations are reserved)	
31:4	-		Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined.	NA

**رجیستر CCR** : با این رجیستر می توان بارگزاری یکی از ۴ رجیستر کپچر با محتوی رجیستر تایمر کانتر در زمان ایجاد رویداد خارجی لبه ها و همچنین ایجاد وقفه آن را کنترل نموداین حالت میتواند لبه پایین رونده – لبه بالا رونده – ایجاد وقفه باشد. انتخاب همزمان این حالت نیز ممکن است.

**رجیستر IR** : شامل پرچم های وقفه رجیسترها کپچر و تطابق – با نوشتن یک پاک می شوند.

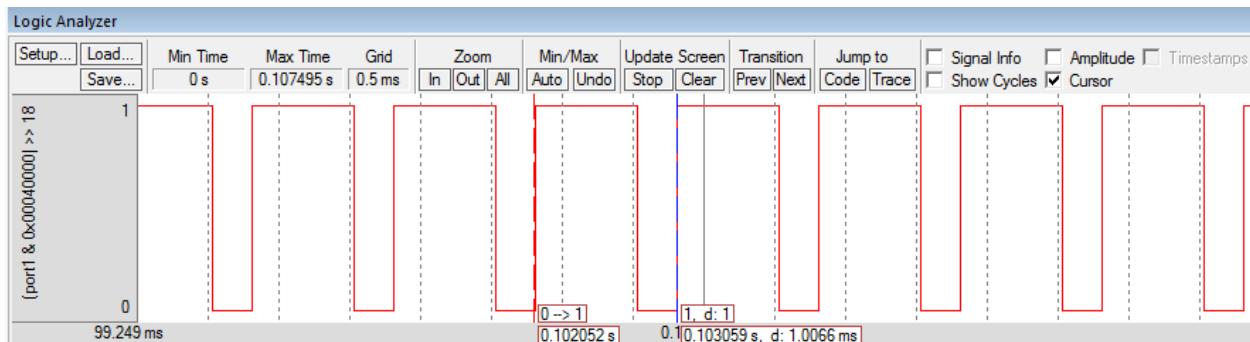
**رجیستر LER** : رجیستر فعال سازی لچ است. این رجیستر با امکان انتخاب حالت لچ برای هر یک از کanal های تطابق پس از ریست شدن PWM باعث لود شدن رجیسترها انتخاب شده تطابق در رجیستر های سایه می شوند.

---

**برنامه نمونه:** برنامه ای بنویسد که با استفاده از PWM یک و پایه P1.18 میکرو با 70 Duty Cycle و فرکانس 1KHz ایجاد کنید و به موتور الکتریکی متصل نمایید.

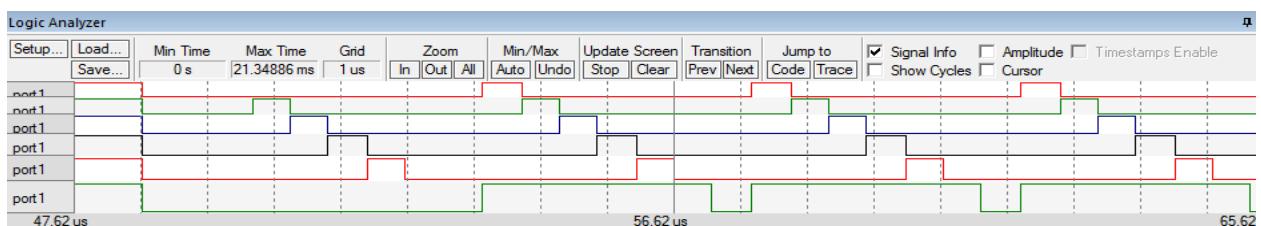
```
#include <lpc17xx.h>

int main() {
    LPC_PINCON -> PINSEL3 = 2 << 4; //Set P1.18 for PWM1.
    LPC_PWM1 -> TCR = 0x02;           //Reset timer counter for PWM.
    LPC_PWM1 -> MCR = 0x02;           //PWMC will be reset if PWMR0
matches it.
    LPC_PWM1 -> PCR = (1<<9);      //PWM1 output enable.
    LPC_PWM1 -> MR0 = 25000;          //Set MR0 for 1KHz
    LPC_PWM1 -> MR1 = 17500;          //For 70% duty cycle
    LPC_PWM1 -> TCR = 0x01;           //Enable timer counter for
PWM.
    while(1){
    }
}
```



نکته مهم دیگر تولید پشت سر هم موجهای PWM است یعنی پس از اینکه یکی از خروجی‌ها از صفر رفت خروجی دیگر از صفر به یک می‌رود واز آن‌ها می‌توان برای راه اندازی موتورهای پله‌ای استفاده نمود. شکل شبیه‌سازی این موج‌ها در زیر آمده است. برای تولید این موجها باید بیت‌های شماره ۲ تا ۶ رجیستر PCR را یک نمود.

```
LPC_PWM1->PCR = (1<<2) | (1<< 3) | (1<< 4) | (1<< 5) | (1<<9) | (1<<10)
| (1<<11) | (1<<12) | (1<<13) | (1<<14);
```



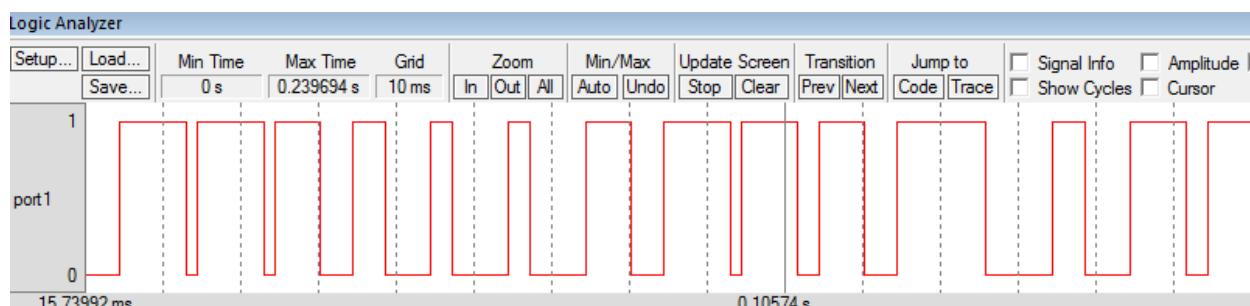
**وقفه PWM**: برای راه اندازی وقفه PWM در حالت وقفه قرار دهیم و هر بار که مقدار MR0 با مقدار TC برابر شد وقفه رخ دهد و وارد زیر روال وقفه شود، البته در صورت تطابق با رجیسترهای MR1 تا MR6 نیز با تنظیم این رجیستر، وقفه می‌تواند رخ دهد. نکته مهم دیگر این است هنگام خروج از وقفه پرچم‌های وقوع وقفه پاک شود. همچنین برای استفاده از وقفه باید بردار وقفه PWM الوبت داده شود وفعال نیز گردد.

```
LPC_PWM1->MCR =0x00000003; //reset and enable interrupt on MR0
NVIC_SetPriority(PWM1_IRQn, 0);
```

---

## ❖ دستور کار:

- I. برنامه‌ای بنویسید که به وسیله دو عدد Push button سرعت موتور DC را کاهش و افزایش دهد و سرعت موتور توسط کانتر بخوانید و بر روی LCD نمایش دهید.
- II. برنامه‌ای بنویسید که به وسیله آن به توان فرکانس PWM و Duty Cycle موج مربعی را با عدد کلید کنترل نمود.
- III. برنامه‌ای بنویسید که به وسیله PWM یک موج مربعی که ابتدا Duty cycle آن زیاد شود و سپس کم شود.
- IV. کاربرد شکل موج زیر چیست؟



- V. برنامه برای استپرموتور را طبق نکته صفحه قبل بنویسید و آن را شبیه سازی کنید

# آزمایش دهم: آشنایی با واحد ساعت بی‌درنگ RTC

❖ اهداف:

- آشنایی با ساعت داخلی میکرو و نحوه استفاده از آن

## ❖ واحد RTC

RTC مخفف کلمات Real Time Counter است که یک واحد ساعت داخلی بسیار دقیق می‌باشد. برای استفاده از این واحد باید از یک کریستال 32786KHz و یک باطری پشتیبان استفاده نمود و به دلیل توان مصرفی کم این واحد به مدت طولانی با باطری پشتیبان به کار خود ادامه می‌دهد. از قابلیت‌های دیگر این واحد امکان تولید وقفه به ازای ثانیه، دقیقه و... است و همچنین امکان ایجاد وقفه در زمان مشخص است.

## ❖ کار با واحد RTC

جهت استفاده از این واحد از هدر فایل RTC استفاده می‌کنیم. در داخل این هدر توابعی جهت راه اندازی و استفاده راحت‌تر از این واحد قرار دارد که در ادامه به بررسی آن‌ها خواهیم پرداخت.

تابع **RTCInit()**: این تابع جهت تنظیمات اولیه رجیسترهاي RTC مثل فعال کردن واحد RTC می‌باشد.  
برنامه این تابع در زیر نوشته شده است.

```
void RTCInit( void )
{
    alarm_on = 0;
    LPC_SC->PCONP |= (1 << 9);           //Enable CLOCK into RTC
    if (LPC_RTC->RTC_AUX & (0x1<<4))   //If RTC is stopped, clear
    {                                         STOP bit.
        LPC_RTC->RTC_AUX |= (0x1<<4);
    }
    /*--- Initialize registers ---*/
    LPC_RTC->AMR = 0;
    LPC_RTC->CIIR = 0;
    LPC_RTC->CCR = 0;
    return;
}
```

**رجیسترهاي تنظيم اوليه زمان:** در ابتدا با استفاده از اين رجيسترها زمان مدنظر خودمان را با آن ها تنظيم می نمایيم و سپس زمانی که در آن هستیم را به تابع RTCSetTime می دهیم. در زیر مثالی از نحوه استفاده از اين تابع آورده ايم.

```
local_time.RTC_Sec = 0;
local_time.RTC_Min = 10;
local_time.RTC_Hour = 0;
local_time.RTC_Mday = 0;
local_time.RTC_Wday = 0;
local_time.RTC_Yday = 0;           // current date 04/12/2012
local_time.RTC_Mon = 12;
local_time.RTC_Year = 2012;
RTCSetTime( local_time );         // Set local time
```

**تابع RTCStart:** وظيفه اين تابع فعال کردن ساعت استفاده می شود که برنامه‌ی اين تابع را در زير مشاهده می نمایيد.

```
void RTCStart( void )
{
    /*--- Start RTC counters ---*/
    LPC_RTC->CCR |= CCR_CLKEN;
    LPC_RTC->ILR = ILR_RTCCIF;
    return;
}
```

**تابع RTCTime RTCGetTime:** اين تابع زمان فعلی را به باز می گرداند. در زير برنامه اين تابع نيز آمده است.

```
RTCTime RTCGetTime( void )
{
    RTCTime LocalTime;
    LocalTime.RTC_Sec = LPC_RTC->SEC;
    LocalTime.RTC_Min = LPC_RTC->MIN;
    LocalTime.RTC_Hour = LPC_RTC->HOUR;
    LocalTime.RTC_Mday = LPC_RTC->DOM;
    LocalTime.RTC_Wday = LPC_RTC->DOW;
    LocalTime.RTC_Yday = LPC_RTC->DOY;
    LocalTime.RTC_Mon = LPC_RTC->MONTH;
    LocalTime.RTC_Year = LPC_RTC->YEAR;
    return ( LocalTime );
}
```

**برنامه نمونه:** برنامه بنویسید که یک زمان مشخص را در داخل برنامه برای RTC تنظیم نمایید و سپس مقدار ساعت و تاریخ را بر روی LCD نمایش دهید.

```
#include <lpc17xx.h>
#include "rtc.h"
#define LCD_LPC1768
#define LCD_PORT_2
#define LCD_RS 0
#define LCD_RW 1
#define LCD_E 2
#define LCD_DB4 4
#define LCD_DB5 5
#define LCD_DB6 6
#define LCD_DB7 7
#include "lcd.h"
#include "delay.h"
RTCTime local_time, current_time;
int main(){
    lcd_init();
    lcd_clear();
    RTCInit();
    local_time.RTC_Sec = 22;      //Second value - [0,59]
    local_time.RTC_Min = 30;      //Minute value - [0,59]
    local_time.RTC_Hour = 9;       //Hour value - [0,23]
    local_time.RTC_Mday = 15;     //Day of the month value - [1,31]
    local_time.RTC_Wday = 0;       //Day of week value - [0,6]
    local_time.RTC_Yday = 0;       //Day of year value - [1,365]
    local_time.RTC_Mon = 8;        //Month value - [1,12]
    local_time.RTC_Year = 2015;   //Year value - [0,4095]
    RTCSetTime(local_time);
    RTCStart();

    while(1){
        current_time = RTCGetTime();
        lcd_gotoxy(1,1);
        lcd_puts(current_time.RTC_Year);
        lcd_putchar('/');
        lcd_puts(current_time.RTC_Mon);
        lcd_putchar('/');
        lcd_puts(current_time.RTC_Mday);
        lcd_gotoxy(2,1);
        lcd_puts(current_time.RTC_Hour);
        lcd_putchar(':');
        lcd_puts(current_time.RTC_Min);
        lcd_putchar(':');
        lcd_puts(current_time.RTC_Sec);
    }
}
```

```
delay_ms(1000);  
lcd_clear();  
}  
}
```

---

## ❖ دستور کار:

- i. برنامه یک ساعت را بنویسید که به وسیله چند کلید فشاری بتوان آن را تنظیم نمود.
- ii. در رابطه با توابع دیگر هدر RTC و کاربردهایشان و همچنین وقفه RTC و نحوه استفاده از آن گزارشی تهیه نمایید و سپس برنامه‌ی یک آلام را بنویسید که به وسیله آن در یک زمان خاص یک LED روشن شود.

# آزمایش یازدهم: آشنایی با ارتباط SPI

## ❖ اهداف:

- آشنایی با پروتکل ارتباطی SPI و نحوه کار با آن
- بررسی رجیسترها و توابع مربوط به SPI

## ❖ مقدمه:

در این آزمایش با استاندارد ارتباطی SPI آشنا می شویم سپس با استفاده از این ارتباط اطلاعات یک مبدل قیاسی به رقمی در خارج میکروکنترلر را دریافت می کنید.

## ❖ بررسی بلوک SPI :

از این پروتکل جهت ارتباط به صورت سریال سنکرون با ادوات جانبی استفاده می شود . جهت استفاده از این پروتکل یکی از وسایل ارتباطی به صورت MASTER و طرف دیگر ارتباط به صورت SLAVE می باشد . وظیفه MASTER کنترل ارتباط ما بین ادوات ارتباطی مختلف می باشد. جهت ارتباط و ارسال اطلاعات از MASTER به سمت SLAVE از ۴ پایه SCK، MOSI، MISO و SS استفاده می شود. وظیفه SCK یا MASTER کلک سنکرون سازی بین MASTER و SLAVE است. MOSI سیم ارسال اطلاعات از MASTER به سمت SLAVE است و MISO به طور عکس اطلاعات را از SLAVE به سمت MASTER می فرستد. از پایه SS زمانی استفاده می شود که چند SLAVE داریم و می خواهیم با آنها ارتباط برقرار کنیم . در SLAVE ها این پایه به صورت ورودی است و زمانی امکان برقراری ارتباط می دهد که مقدار آن صفر باشد و اگر یک باشد SLAVE مدد نظرمان غیرفعال می شود.

## ❖ بررسی رجیسترها SPI و نحوه کار با آن :

رجیستر **SPCR** : رجیستر کنترل SPI است. بیت دوم این رجیستر جهت ارسال بیش از ۸ بیت این بیت را فعال می کنیم. در صورتی که بخواهیم به صورت ۸ بیت انتقال دهیم نیاز به فعال کردن بیت دوم یا Bit Enabe نیست. بیتهاش شماره ۳ و ۴ به نام های CPOL و CPHA جهت تعیین فاز و پلاریته ای اطلاعات مورد استفاده قرار می گیرد. بیت شماره ۵ یا MSTR جهت تعیین MASTER یا SLAVE است و در صورت یک دادن به این بیت میکرو در مدد MASTER قرار می گیرد. بیت شماره ۶ یا LSBF در صورت یک بودن ارسال اطلاعات از بیت کم ارزش صورت می گیرد در صورت صفر بودن از بیت پر ارزش ارسال می شود. بیت شماره ۷ یا SPIE در صورت یک بودن وقفه برای ما فعال می کند. بیتهاش ۸ تا ۱۱ نیز جهت مشخص کردن تعداد بیت ارسالی در

صورت یک بودن بیت دوم این رجیستر است. ساده ترین تنظیم برای این رجیستر بدین صورت است که SPI درمد SLAVE یا قرار دهیم و ارسال داده از بیت کم ارزش LSB وارسال به صورت ۸ بیتی صورت گیرد. به عنوان مثال :

```
LPC_SPI-> SPCR = (1<<5) | (1<<6); // (Master) (LSBF)
```

1:0	-	Reserved, user software should not write ones to reserved bits. NA The value read from a reserved bit is not defined.	
2	BitEnable	0	The SPI controller sends and receives 8 bits of data per transfer.
		1	The SPI controller sends and receives the number of bits selected by bits 11:8.
3	CPHA	Clock phase control determines the relationship between the data and the clock on SPI transfers, and controls when a slave transfer is defined as starting and ending.	0
		0	Data is sampled on the first clock edge of SCK. A transfer starts and ends with activation and deactivation of the SSEL signal.
		1	Data is sampled on the second clock edge of the SCK. A transfer starts with the first clock edge, and ends with the last sampling edge when the SSEL signal is active.
4	CPOL	Clock polarity control.	0
		0	SCK is active high.
		1	SCK is active low.
5	MSTR	Master mode select.	0
		0	The SPI operates in Slave mode.
		1	The SPI operates in Master mode.
6	LSBF	LSB First controls which direction each byte is shifted when transferred.	0
		0	SPI data is transferred MSB (bit 7) first.
		1	SPI data is transferred LSB (bit 0) first.
7	SPIE	Serial peripheral interrupt enable.	0
		0	SPI interrupts are inhibited.
		1	A hardware interrupt is generated each time the SPIF or MODF bits are activated.
11:8	BITS	When bit 2 of this register is 1, this field controls the number of bits per transfer:	0000
		1000	8 bits per transfer
		1001	9 bits per transfer
		1010	10 bits per transfer
		1011	11 bits per transfer
		1100	12 bits per transfer
		1101	13 bits per transfer
		1110	14 bits per transfer
		1111	15 bits per transfer
		0000	16 bits per transfer
31:12 -		Reserved, user software should not write ones to reserved bits.	NA

**رجیستر SPSR** : رجیستر وضعیت SPI است که با استفاده از آن می توان به کامل شدن عملیات ارسال و دریافت پی برد. یکی از بیتها مهم برای ما بیت شماره ۷ یا SPIF است و زمانی که ارسال کامل شد این بیت یک می شود و در صورت فعال بودن بیت SPIE در رجیستر کنترل وقفه رخ می دهد. به عنوان مثال با دستور زیر منتظر کامل شدن ارسال می شویم.

```
while( !(LPC_SPI->SPSR & 0x80) ); //wait for transfer to be completed
```

**رجیستر SPDR** : رجیستر داده است و داده های ارسالی و دریافتی در داخل این رجیستر قرار می گیرد از دو قسمت ۸ بیتی کم ارزش و پر ارزش تشکیل شده است و زمانی که از حالت ۸ بیتی استفاده کنیم تنها قسمت کم ارزش مورد استفاده قرار می گیرد. دستور آن نیز به صورت LPC\_SPI->SPDR است.

**رجیستر SPCCR** : رجیستر تنظیم کلاک واحد SPI است و مقدار این رجیستر تقسیم می شود.

رجیستر **SPINT** : رجیستر پرچم وقفه است.

## ❖ مراحل پیکربندی :

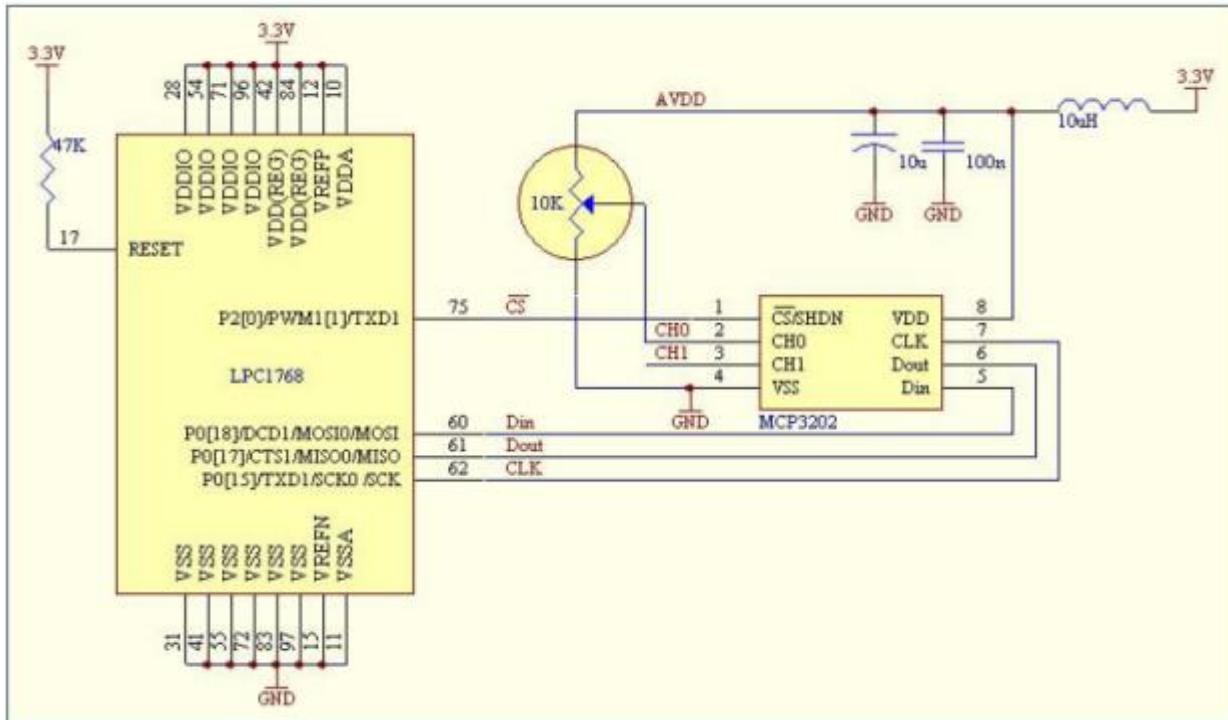
تنظیم پایه ها مربوط به SPI با استفاده از رجیستر PINSELL انجام می گیرد به طوریکه پایه P0.15 برای تنظیم پایه های P0.17 و P0.18 به ترتیب MOSI و MISO قرار گیرد. SCK

```
LPC_PINCON->PINSEL1 |= (3<<2) | (3<<4);  
LPC_PINCON->PINSEL0 |= (3<<30);
```

## ❖ راه اندازی مبدل آنالوگ به دیجیتال خارجی با پروتکل SPI :

بر روی بورد آموزشی یک عدد آی سی مبدل آنالوگ به دیجیتال به شماره MCP3202 از شرکت میکروچیپ قرار داده شده است. این آی سی با استفاده از پروتکل ارتباطی SPI به میکرو متصل می شود. ارتباط سریال این آی سی دوطرف است و اطلاعات دیجیتال را در قالب ۴ کلمه ۸ بیتی می توان خواند.

**برنامه نمونه:** برنامه ای بنویسید که به وسیله آی سی MCP3202 ولتاژ ورودی آی سی به صورت ۱۲ بیتی خوانده شده و نتیجه توسط ۱۲ عدد LED نمایش داده شود. (مطابق شکل زیر)



```
#include <lpc17xx.h>

int main(void) {
    int data = 0;
    LPC_SC -> PCONP |= (1 << 8); //Enable SPI power/clock control
    bit
    LPC_PINCON -> PINSEL0 |= 0xC0000000; //Enable P0.15 for SCK
    LPC_PINCON -> PINSEL1 |= 0x0000003C; //Enable P0.17 for MISO
    and P0.18 for MOSI
    LPC_SPI -> SPCCR = 2; //Divide Pclock/2 for SPI
    LPC_SPI -> SPCR = 0x20; //SPI send & receive 8 bit per transfer
    and SPI work as Master

    LPC_GPIO0 -> FIODIRO0 = 0x01; //Set P0.0 output for CS
    LPC_GPIO2 -> FIODIRL = 0xFFFF; //Set P2.0 - P2.11 output for
    LEDs

    while(1){
        LPC_GPIO0 -> FIOPIN0 = 0x00; //Enable CS

        LPC_SPI -> SPDR = 0x01;
        while(!(LPC_SPI -> SPSR & 0x80)); //Wait for send data

        LPC_SPI -> SPDR = 0xA0;
        while(!(LPC_SPI -> SPSR & 0x80)); //Wait for send data
        data = LPC_SPI -> SPDR;
```

```

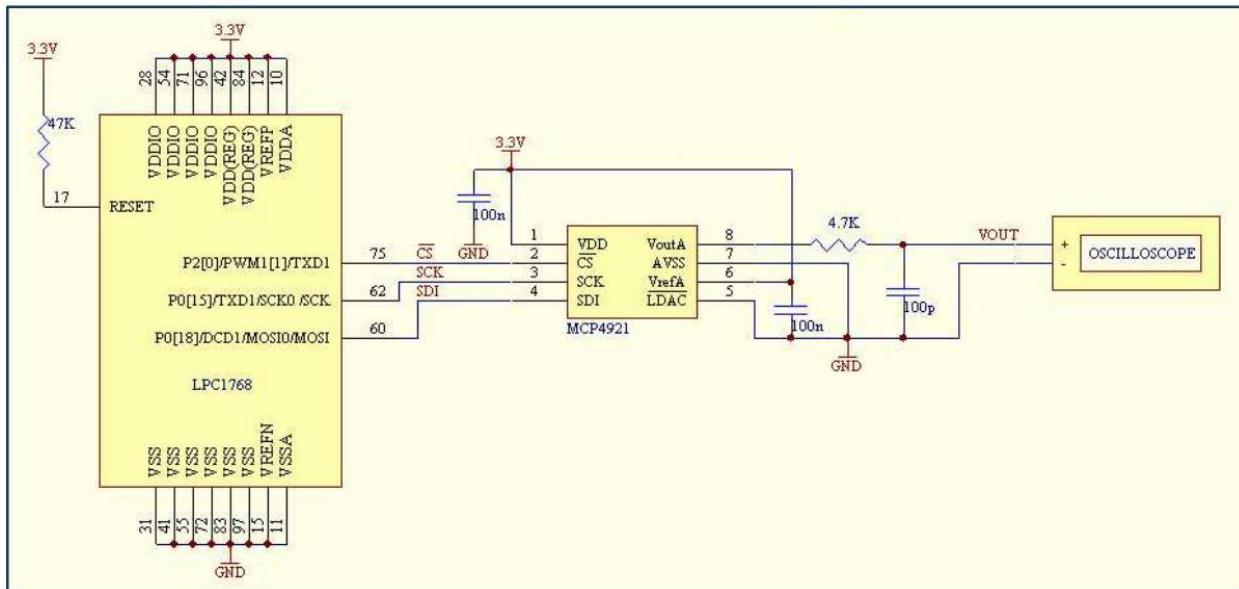
LPC_SPI -> SPDR = 0x00;
while(!(LPC_SPI -> SPSR & 0x80)); //Wait for send data
data = data << 8;
data |= LPC_SPI -> SPDR;

LPC_GPIO0 -> FIOPIN0 = 0x01;//Disable CS
LPC_GPIO2 -> FIOPINL = data;
data--;
data--;
}
}
}

```

## ❖ دستورکار:

- ا. با استفاده از واسط SPI مبدل رقمی به قیاسی MCP4921 که بر روی برد تعییه شده است را راه اندازی نمایید و یک شکل موج سینوسی تولید نمایید و نتیجه را بر روی اسیلوسکوپ مشاهده نمایید.



# آزمایش دوازدهم: آشنایی با پروتکل ارتباطی (UART(RS232)

❖**اهداف:**

- آشنایی با پروتکل ارتباطی **UART** و امکانات آن
- راه اندازی پروتکل **UART** در میکروکنترلر **LPC1768**

❖**مقدمه:**

پروتکل دیگری که در این آزمایشگاه مورد استفاده قرار می گیرد پروتکل ارتباطی **UART** است. ارتباط سریال (**UART**) یکی مهم ترین پروتکل های ارتباطی است که به وسیله‌ی آن می‌توان با تجهیزات جانبی و ماژول های الکترونیکی مختلفی ارتباط برقرار کرد. اکثر ماژول های ارتباطی که نیاز به نرخ ارسال اطلاعات با حجم و سرعت کم می باشند از این پروتکل استفاده می کنند.

## ❖ **LPC1768** در میکروکنترلر **UART**

میکروکنترلر مذکور دارای ۴ پورت سریال مجزا است. شایان ذکر است که **UART1** قابلیت ارتباط به صورت مودم و **UART0,2,3** قابلیت ارتباط مادون قرمز را دارند.

در تصویر زیر خلاصه ای از قابلیت های ارتباط سریال با **UART0,2,3** در این میکرو آمده است :

- Data sizes of 5, 6, 7, and 8 bits.
- Parity generation and checking: odd, even mark, space or none.
- One or two stop bits.
- 16 byte Receive and Transmit FIFOs.
- Built-in baud rate generator, including a fractional rate divider for great versatility.
- Supports DMA for both transmit and receive.
- Auto-baud capability
- Break generation and detection.
- Multiprocessor addressing mode.
- IrDA mode to support infrared communication.
- Support for software flow control.

در این تصویر قابلیت های ارتباطی پورت سریال 1 UART1 آمده است :

- Full modem control handshaking available
  - Data sizes of 5, 6, 7, and 8 bits.
  - Parity generation and checking: odd, even mark, space or none.
  - One or two stop bits.
  - 16 byte Receive and Transmit FIFOs.
  - Built-in baud rate generator, including a fractional rate divider for great versatility.
  - Supports DMA for both transmit and receive.
  - Auto-baud capability
  - Break generation and detection.
  - Multiprocessor addressing mode.
  - RS-485 support.
- 

## ❖ راه اندازی و استفاده از بلوک سریال ❖

جهت راه اندازی و استفاده از UART در میکرو Ipc1768 به وسیله رجیسترها باید ابتدا جهت پایه ها را با رجیستر PINSELL PCLKSELL تنظیم کنند. در مدار سریال قرار داد و سپس کلک این بلوک را با رجیستر LCR نخ ارسال ارتباط سریال نمود و در ادامه با استفاده از رجیسترها FCR , DLM , DLL و آسنکرون مشخص می شود و پس از پیکربندی با رجیستر THR می توان داده ها را ارسال و با استفاده از رجیستر RBR جهت دریافت اطلاعات استفاده کرد. در صفحه بعدی لیست کاملی از رجیستر های UART مشاهده خواهید کرد.

با توجه به تعداد زیاد رجیسترها و مشکلاتی در راه اندازی به صورت مستقیم از کتابخانه UART جهت استفاده از ارتباط سریال استفاده می کنیم.

برای استفاده از این تابع ابتدا نرخ ارسال Baud Rate را برابر با ۹۶۰۰ می گذاریم:

```
#define UART0_BPS 9600
```

Table 271. UART0/2/3 Register Map

Generic Name	Description	Access	Reset value <sup>[1]</sup>	UARTn Register Name & Address
RBR (DLAB =0)	Receiver Buffer Register. Contains the next received character to be read.	RO	NA	U0RBR - 0x4000 C000 U2RBR - 0x4009 8000 U3RBR - 0x4009 C000
THR (DLAB =0)	Transmit Holding Register. The next character to be transmitted is written here.	WO	NA	U0THR - 0x4000 C000 U2THR - 0x4009 8000 U3THR - 0x4009 C000
DLL (DLAB =1)	Divisor Latch LSB. Least significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0x01	U0DLL - 0x4000 C000 U2DLL - 0x4009 8000 U3DLL - 0x4009 C000
DLM (DLAB =1)	Divisor Latch MSB. Most significant byte of the baud rate divisor value. The full divisor is used to generate a baud rate from the fractional rate divider.	R/W	0x00	U0DLM - 0x4000 C004 U2DLM - 0x4009 8004 U3DLM - 0x4009 C004
IER (DLAB =0)	Interrupt Enable Register. Contains individual interrupt enable bits for the 7 potential UART interrupts.	R/W	0x00	U0IER - 0x4000 C004 U2IER - 0x4009 8004 U3IER - 0x4009 C004
IIR	Interrupt ID Register. Identifies which interrupt(s) are pending.	RO	0x01	U0IIR - 0x4000 C008 U2IIR - 0x4009 8008 U3IIR - 0x4009 C008
FCR	FIFO Control Register. Controls UART FIFO usage and modes.	WO	0x00	U0FCR - 0x4000 C008 U2FCR - 0x4009 8008 U3FCR - 0x4009 C008
LCR	Line Control Register. Contains controls for frame formatting and break generation.	R/W	0x00	U0LCR - 0x4000 C00C U2LCR - 0x4009 800C U3LCR - 0x4009 C00C
LSR	Line Status Register. Contains flags for transmit and receive status, including line errors.	RO	0x60	U0LSR - 0x4000 C014 U2LSR - 0x4009 8014 U3LSR - 0x4009 C014
SCR	Scratch Pad Register. 8-bit temporary storage for software.	R/W	0x00	U0SCR - 0x4000 C01C U2SCR - 0x4009 801C U3SCR - 0x4009 C01C
ACR	Auto-baud Control Register. Contains controls for the auto-baud feature.	R/W	0x00	U0ACR - 0x4000 C020 U2ACR - 0x4009 8020 U3ACR - 0x4009 C020
ICR	IrDA Control Register. Enables and configures the IrDA mode.	R/W	0x00	U0ICR - 0x4000 C024 U2ICR - 0x4009 8024 U3ICR - 0x4009 C024
FDR	Fractional Divider Register. Generates a clock input for the baud rate divider.	R/W	0x10	U0FDR - 0x4000 C028 U2FDR - 0x4009 8028 U3FDR - 0x4009 C028
TER	Transmit Enable Register. Turns off UART transmitter for use with software flow control.	R/W	0x80	U0TER - 0x4000 C030 U2TER - 0x4009 8030 U3TER - 0x4009 C030

توابع مهم و کاربردی کتابخانه **UART** عبارتند از:

**UARTx\_Init()** : جهت پیکر بندی پورت سریال x است.

**UARTx\_SendByte()** : جهت ارسال یک بایت اطلاعات استفاده می شود.

**UARTx\_GetChar()** : جهت دریافت یک کاراکتر از ورودی استفاده می شود.

**UARTx\_SendString()** : جهت ارسال یک رشته به خروجی استفاده می شود.

**UARTx\_SendChar()** : جهت ارسال کارکتر به خروجی استفاده می شود.

در ادامه به بررسی بخش اول این کتابخانه وتابع مهم **UART0\_Init** می پردازیم. در ابتدای این کتابخانه با استفاده از عبارت **#define** قسمت های مختلف کلاک را معرفی می نماید:

```
#define FOSC           12000000 //External Clock
#define FCCLK          (FOSC * 8)
#define FCCO           (FCCLK * 3)
#define FPCLK          (FCCLK / 4)
#define UART0_BPS      9600
#define UART2_BPS      9600
```

در صورتی که کلاک اسیلاتور خارجی ما عددی غیر از 12MHz باشد کافی است مقدار FOSC را با مقدار فرکانس اسیلاتور خارجی مقداردهی کنیم. در حالت عادی همین مقدار داده شده درست می باشد.

سپس به بررسی تابع مهم و اصلی **UART0\_Init** می پردازیم. با دو دستور زیر استفاده پورت های P0.2 و P0.3 را به عنوان پورت های دریافت و ارسال سریال مشخص می نماییم:

```
LPC_PINCON->PINSEL0 |= (1 << 4); /* Pin P0.2 used as TXD0 (Com0) */
LPC_PINCON->PINSEL0 |= (1 << 6); /* Pin P0.3 used as RXD0 (Com0) */
```

سپس با دستور زیر تنظیمات مربوط به Stop bit و بیت parity و ... را انجام می دهیم:

```
LPC_UART0->LCR = 0x83; // 1stop bit , 8bit character length , disable parity , Enable access to divisor latches
```

سپس با دستورات زیر به محاسبه و مقدار دهی رجیستر های Divisor latches می پردازیم:

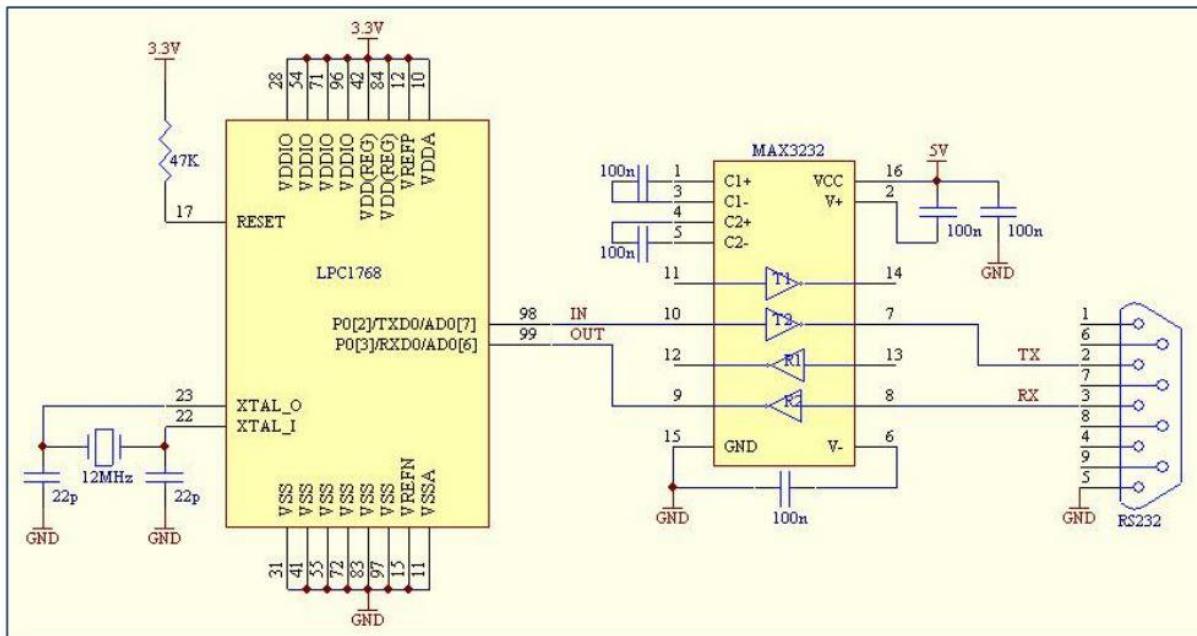
```
uint16_t usFdiv;
usFdiv = (FPCLK / 16) / UART0_BPS;
LPC_UART0->DLM = usFdiv / 256;
LPC_UART0->DLL = usFdiv % 256;
```

سپس با دستورات زیر دسترسی به ریجیستر های Divisor latches را غیرفعال می نماییم و همچنین به ریست FIFO دریافت و ارسال می پردازیم:

```
LPC_UART0->LCR = 0x03;
LPC_UART0->FCR = 0x06;
```

**برنامه نمونه:** برنامه ای بنویسید که اسم و فامیل شما را با پورت سریال یک به کامپیوتر بفرستد و در ادامه به صورت پیوسته یک کارکتر از کامپیوتر بگیرد و دوباره به کامپیوتر بفرستد و در کامپیوتر نتیجه را با یک نرم افزار مانیتور پورت سریال ببیند. برای دیدن نتیجه ارسال و دریافت داده ها از نرم افزار Termite استفاده نمایید و تنظیمات آن را مطابق با تنظیمات کتابخانه استفاده شده قرار دهید.

نکته مهم با توجه به اینکه ارتباط سریال میکروکنترلرها TTL است، جهت تبدیل این منطق به منطق RS232 و ارتباط بین میکرو و کامپیوتر از چیپ MAX3232 استفاده می شود.



```
#include <LPC17xx.h>
#include <stdio.h>
#include "uart.h"
void Delay (uint32_t Time){
    uint32_t i;
    i = 0;
    while (Time--) {
        for (i = 0; i < 5000; i++);
    }
}
```

```

int main(void) {
    int i;
    SystemInit();
    UART0_Init();
    UART0_SendString("\r\nMyName Family\r\n");
    while(1) {
        UART0_SendString("your character= ");
        i=UART0_GetChar();
        UART0_SendByte((char) i);
        UART0_SendString("\r\n");
    }
}

```

---

## ❖ دستور کار:

- I. به وسیله ارتباط سریال دو میکرو را به هم متصل نمایید و میکروی اول پیامی را به میکرو دوم می فرستد و میکرو دوم پیام دریافتی را بروی LCD چاپ و پیام دریافت شده را برای میکرو اول ارسال می کند.
- II. در رابطه با وسائل جانبی که به وسیله ارتباط سریال می توان به میکرو متصل نمود تحقیق کنید و سعی کنید یکی از این مازول ها را با میکرو LPC1768 راه اندازی نمایید. از جمله مازول های SIM800 و WIFI و RF و ...

# آزمایش سیزدهم: آشنایی با پروتکل I2C

❖ اهداف:

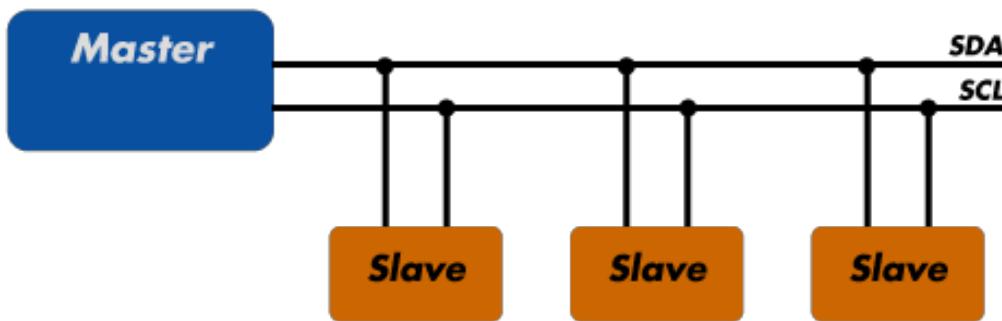
- آشنایی با پروتکل ارتباطی I2C و امکانات آن
- راه اندازی بلوک I2C میکروکنترلر LPC1768

❖ مقدمه:

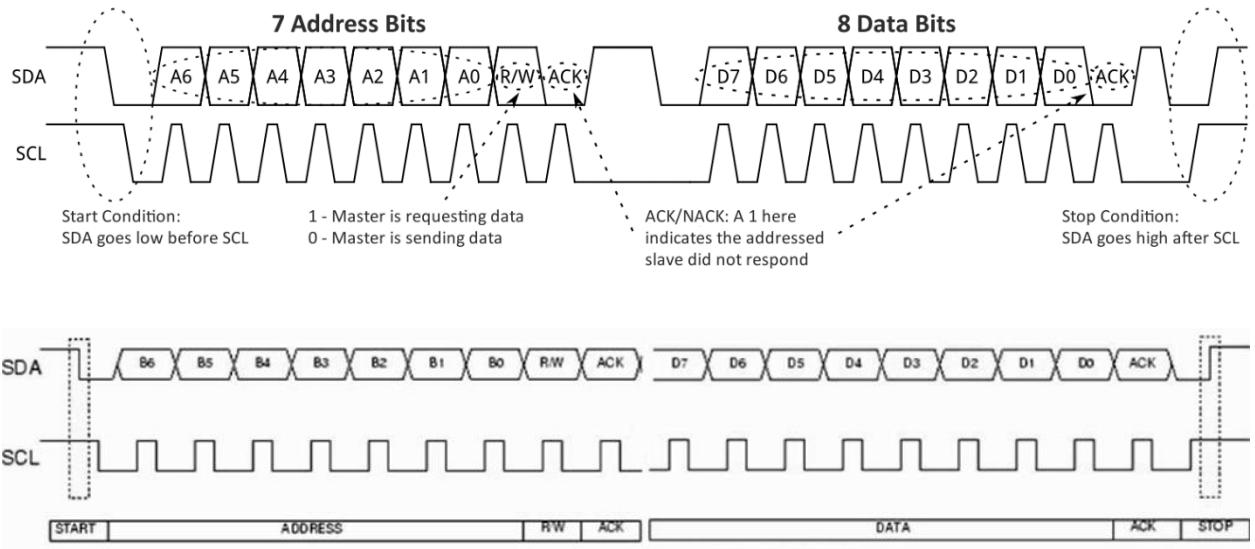
پروتکل ارتباطی I2C که مخفف Inter Integrated Circuit است و نام دیگر آن TWI می باشد ، یک پروتکل ارتباطی جهت ارتباط با وسایل جانبی ، حافظه ها ، سنسورها ... می باشد . این پروتکل توسط شرکت فیلیپس معرفی شده است. از قابلیت های این پروتکل برقراری ارتباط از طریق تنها دو سیم می باشد.

## ❖ واحد I2C میکروکنترلر LPC1768 :

همانطور که در مقدمه ذکر شد از طریق این پروتکل تنها از طریق دو سیم می توان ارتباط سنکرون برقرار نمود که این سیمهای به نامهای (Serial clock) SCL و (Serial Data) SDA می باشد.



با توجه به شکل به وسیله این پروتکل یک MASTER می تواند با چند SLAVE ارتباط برقرار نماید. ارسال اطلاعات در پروتکل I2C در ۵ مرحله صورت می پذیرد که ابتدا ۱- شرایط اولیه شروع ۲- ارسال آدرس ۳- تصدیق صحت آدرس ۴- ارسال دیتا ۵- توقف که در زیر شکل مربوط به این مراحل قرار داده شده است.



نکته مهم در رابطه با این پروتکل آدرس می باشد که توسط بیت پارازش برای مشخص کردن وسیله جانبی و ۳ بیت کم ارزش جهت آدرس دستگاه است . پس به وسیله این پروتکل تا ۱۲۸ دستگاه جانبی مختلف و ۸ وسیله یکسان را به باس متصل نمود.

میکروکنترلر LPC1768 دارای ۳ کanal I2C است که باید توسط رجیستر PINSEL در مد SDA و SCL قرار گیرند.

## ❖:Rجیسترهاي I2C❖

رجیسترهاي I2C در ۵ گروه اصلی کنترلی ، وضعیت ، کلاک ، آدرس و داده تقسیم می شوند که در زیر شمای کلی آن ها را مشاهده می کنید. به دلیل فراوانی و پیچیدگی های رجیستر های I2C از کتابخانه آماده این واسط برای برقراری ارتباط استفاده می نماییم. در ادامه به بررسی توابع مهم این کتابخانه می پردازیم:

**Table 384. I<sup>2</sup>C register map**

Generic Name	Description	Access	Reset value <sup>[1]</sup>	I <sup>2</sup> Cn Name & Address
I2CONSET	<b>I<sup>2</sup>C Control Set Register.</b> When a one is written to a bit of this register, the corresponding bit in the I <sup>2</sup> C control register is set. Writing a zero has no effect on the corresponding bit in the I <sup>2</sup> C control register.	R/W	0x00	I2C0CONSET - 0x4001 C000 I2C1CONSET - 0x4005 C000 I2C2CONSET - 0x400A 0000
I2STAT	<b>I<sup>2</sup>C Status Register.</b> During I <sup>2</sup> C operation, this register provides detailed status codes that allow software to determine the next action needed.	RO	0xF8	I2C0STAT - 0x4001 C004 I2C1STAT - 0x4005 C004 I2C2STAT - 0x400A 0004
I2DAT	<b>I<sup>2</sup>C Data Register.</b> During master or slave transmit mode, data to be transmitted is written to this register. During master or slave receive mode, data that has been received may be read from this register.	R/W	0x00	I2C0DAT - 0x4001 C008 I2C1DAT - 0x4005 C008 I2C2DAT - 0x400A 0008
I2ADR0	<b>I<sup>2</sup>C Slave Address Register 0.</b> Contains the 7-bit slave address for operation of the I <sup>2</sup> C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00	I2C0ADR0 - 0x4001 C00C I2C1ADR0 - 0x4005 C00C I2C2ADR0 - 0x400A 000C
I2SCLH	<b>SCH Duty Cycle Register High Half Word.</b> Determines the high time of the I <sup>2</sup> C clock.	R/W	0x04	I2C0SCLH - 0x4001 C010 I2C1SCLH - 0x4005 C010 I2C2SCLH - 0x400A 0010
I2SCLL	<b>SCL Duty Cycle Register Low Half Word.</b> Determines the low time of the I <sup>2</sup> C clock. I2nSCLL and I2nSCLH together determine the clock frequency generated by an I <sup>2</sup> C master and certain times used in slave mode.	R/W	0x04	I2C0SCLL - 0x4001 C014 I2C1SCLL - 0x4005 C014 I2C2SCLL - 0x400A 0014
I2CONCLR	<b>I<sup>2</sup>C Control Clear Register.</b> When a one is written to a bit of this register, the corresponding bit in the I <sup>2</sup> C control register is cleared. Writing a zero has no effect on the corresponding bit in the I <sup>2</sup> C control register.	WO	NA	I2C0CONCLR - 0x4001 C018 I2C1CONCLR - 0x4005 C018 I2C2CONCLR - 0x400A 0018
MMCTRL	<b>Monitor mode control register.</b>	R/W	0x00	I2C0MMCTRL - 0x4001 C01C I2C1MMCTRL - 0x4005 C01C I2C2MMCTRL - 0x400A 001C
I2ADR1	<b>I<sup>2</sup>C Slave Address Register 1.</b> Contains the 7-bit slave address for operation of the I <sup>2</sup> C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00	I2C0ADR1 - 0x4001 C020 I2C1ADR1 - 0x4005 C020 I2C2ADR1 - 0x400A 0020

Table 384. I<sup>2</sup>C register map

Generic Name	Description	Access	Reset value <sup>[1]</sup>	I <sup>2</sup> Cn Name & Address
I2ADR2	<b>I<sup>2</sup>C Slave Address Register 2.</b> Contains the 7-bit slave address for operation of the I <sup>2</sup> C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00	I2C0ADR2 - 0x4001 C024 I2C1ADR2 - 0x4005 C024 I2C2ADR2 - 0x400A 0024
I2ADR3	<b>I<sup>2</sup>C Slave Address Register 3.</b> Contains the 7-bit slave address for operation of the I <sup>2</sup> C interface in slave mode, and is not used in master mode. The least significant bit determines whether a slave responds to the General Call address.	R/W	0x00	I2C0ADR3 - 0x4001 C028 I2C1ADR3 - 0x4005 C028 I2C2ADR3 - 0x400A 0028
I2DATA_BUFFER	<b>Data buffer register.</b> The contents of the 8 MSBs of the I2DAT shift register will be transferred to the I2DATA_BUFFER automatically after every 9 bits (8 bits of data plus ACK or NACK) has been received on the bus.	RO	0x00	I2C0DATA_BUFFER - 0x4001 C02C I2C1DATA_BUFFER - 0x4005 C02C I2C2DATA_BUFFER - 0x400A 002C
I2MASK0	<b>I<sup>2</sup>C Slave address mask register 0.</b> This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address ('0000000').	R/W	0x00	I2C0MASK0 - 0x4001 C030 I2C1MASK0 - 0x4005 C030 I2C2MASK0 - 0x400A 0030
I2MASK1	<b>I<sup>2</sup>C Slave address mask register 1.</b> This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address ('0000000').	R/W	0x00	I2C0MASK1 - 0x4001 C034 I2C1MASK1 - 0x4005 C034 I2C2MASK1 - 0x400A 0034
I2MASK2	<b>I<sup>2</sup>C Slave address mask register 2.</b> This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address ('0000000').	R/W	0x00	I2C0MASK2 - 0x4001 C038 I2C1MASK2 - 0x4005 C038 I2C2MASK2 - 0x400A 0038
I2MASK3	<b>I<sup>2</sup>C Slave address mask register 3.</b> This mask register is associated with I2ADR0 to determine an address match. The mask register has no effect when comparing to the General Call address ('0000000').	R/W	0x00	I2C0MASK3 - 0x4001 C03C I2C1MASK3 - 0x4005 C03C I2C2MASK3 - 0x400A 003C

## ❖ بررسی توابع مهم کتابخانه I<sup>2</sup>C

بررسی تابع **I<sup>2</sup>C\_Init** : این تابع برای مقدار دهی اولیه رجیسترها واسط I<sup>2</sup>C استفاده می شود و زیرکدهای موجود در این تابع را مشاهده می نمایید:

```

uint32_t I2CInit( uint32_t I2cMode )
{
    LPC_SC->PCONP |= (1 << 19);

    /* set PIO0.27 and PIO0.28 to I2C0 SDA and SCK */
    /* function to 01 on both SDA and SCK. */
    LPC_PINCON->PINSEL1 &= ~0x03C00000;
    LPC_PINCON->PINSEL1 |= 0x01400000;

    /*---- Reset registers ---*/
    LPC_I2C0->I2SCLL = I2SCLL_SCLL;
    LPC_I2C0->I2SCLH = I2SCLH_SCLH;
    if ( I2cMode == I2CSLAVE )
    {
        LPC_I2C0->I2ADRO = 0xA0;           //Set Slave Address
    }

    /* Install interrupt handler */
    NVIC_EnableIRQ(I2C0_IRQn);

    LPC_I2C0->I2CONSET = I2CONSET_I2EN; //Enable I2C
    return( 1 );
}

```

**بررسی تابع I2C\_WriteNByte** : با استفاده از این تابع می توان مقادیر موجود در یک روى واسط I2C انتقال داد. در زیر امضای تابع ذکر شده را مشاهده می نمایید:

```

uint8_t I2C_WriteNByte(uint8_t sla, uint8_t suba_type, uint32_t suba,
                      uint8_t *s, uint32_t num);

```

در پارامتر های دریافتی **sla** همان **slave address** میزان بایت **suba\_type**، **suba** ، **sub address** همان **s**، **subaddress** اشاره گر به بافر و **num** میزان داده برای نوشتن را مشخص نماید.

**بررسی تابع I2C\_ReadNByte** : با استفاده از این تابع می توان مقادیر موجود در یک بافر را از روی واسط I2C خواند . در زیر امضای تابع ذکر شده را مشاهده می نمایید:

```

uint8_t I2C_ReadNByte (uint8_t sla, uint8_t suba_type, uint32_t suba,
                      uint8_t *s, uint32_t num);

```

توضیح پارامتر های دریافتی این تابع مانند تابع قبلی می باشد.

**برنامه نمونه:** در برنامه‌ی زیر نحوه نوشتن خواندن بر روی حافظه EEPROM با استفاده از پروتکل I2C آمده است.

```
#include "lpc17xx.h"
#include "i2c.h"
extern uint8_t buf[32];
void Delay(uint32_t delaydata) {
    uint32_t i,j,k;
    for(i=0;i<delaydata;i++)
        for(j=0;j<1000;j++)
            for(k=0;k<100;k++);
}
int main (void) {
    uint32_t i;
    SystemInit();
    LPC_GPIO2->FIODIR      = 0x000000ff;          //LEDs PORT2 are Output
    LPC_GPIO0->FIODIR      |= 0x03f80000;
    LPC_GPIO0->FIOSET       = 0x03f80000;
    if ( I2CInit( (uint32_t)I2CMASTER ) == 0 ){ //initialize I2c
        while ( 1 );                                //Fatal error
    }
    for ( i = 0; i < 8; i++ ){                      //clear buffer
        buf[i]=i+1;
    }
    I2C_WriteNByte(0xa0, 1, 0x00, buf, 8); //write buf array into
    Delay(50);                                     EEPROM
    for ( i = 0; i < 8; i++ ){                      //clear buffer
        buf[i] =0;
    }
    I2C_ReadNByte (0xa0, 1, 0x00, buf, 8); //read from EEPROM & save
    Delay(50);                                     in buff array
    while(1){
        for(i=0;i<8;i++){
            LPC_GPIO2->FIOPIN = buf[i];
            Delay(300);
        }
    }
}
```

## ❖ دستور کار:

- I. برنامه بالا یه صورت عملی بروی میکرو بیندید و تست کنید.
- II. به وسیله ارتباط I2C دو میکرو را به هم متصل نمایید و میکروی اول پیامی را به میکرو دوم می فرستد و میکرو دوم پیام دریافتی را بروی LCD چاپ و پیام دریافت شده را برای میکرو اول ارسال می کند.