1:

شروع کار با AVR Studio و AVR-GCC

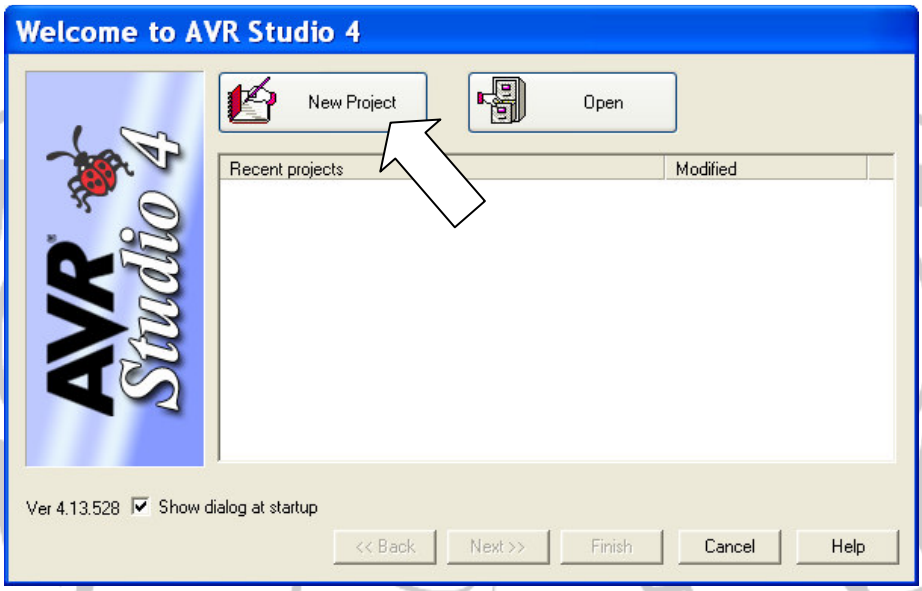
مولف: رضا سپاس‌یار

انجمن تخصصی برق و الکترونیک (www.eca.ir)

2:

۱. پس از نصب WinAVR، برنامه ی AVR Studio را نصب کرده و آن را اجرا کنید.

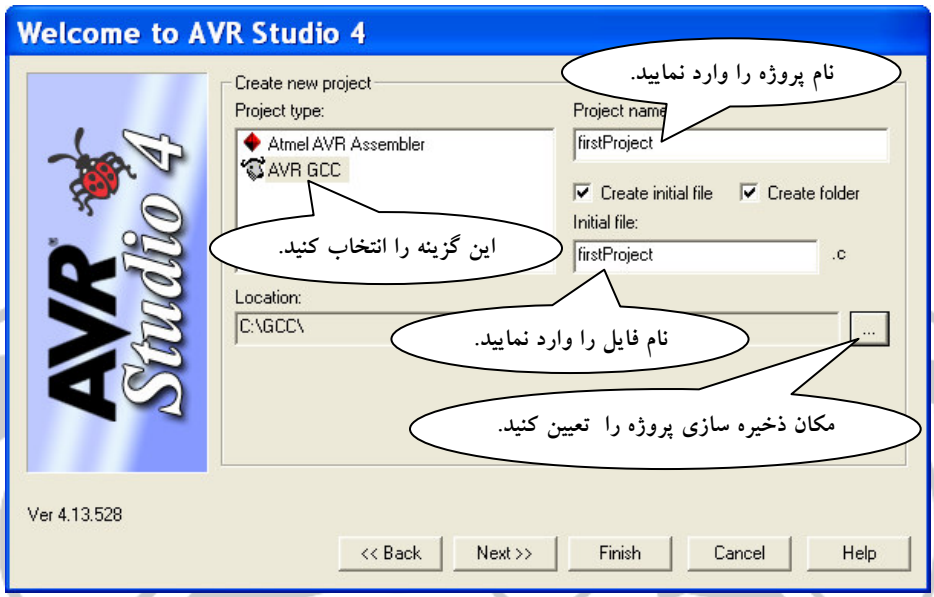
۲. ویزاد اجرا شده و شکل ۱ را مشاهده می کنید. بر روی دکمه ی New Project کلیک کنید.



شکل ۱

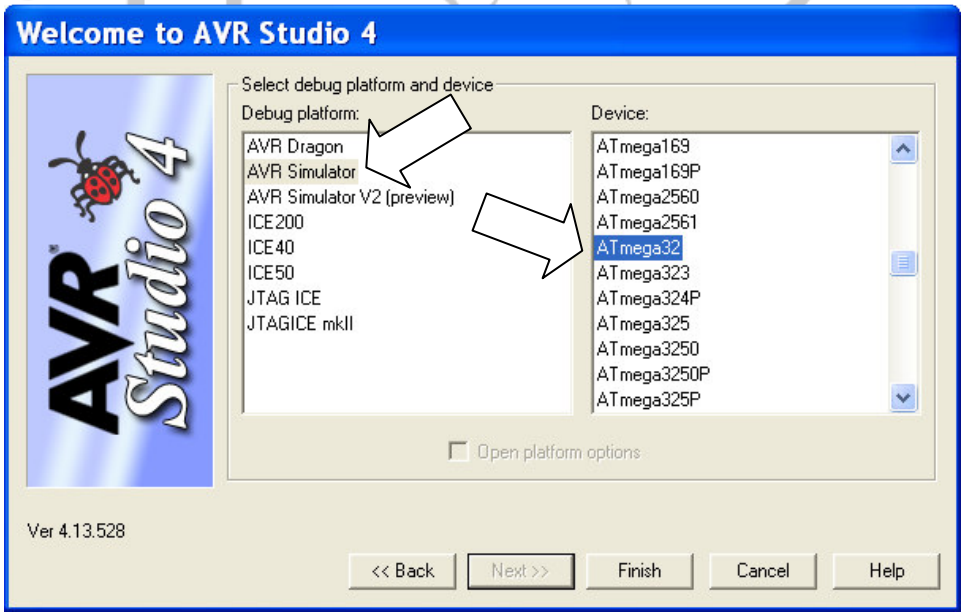
مطابق با شکل ۲ تنظیمات را انجام دهید. در صورتی گزینه ی Create initial file را از حالت انتخاب خارج کنید، فایل اولیه ی پروژه (با پسوند c) ایجاد نخواهد شد و با با انتخاب گزینه ی Create folder به برنامه اجازه می دهید تا پروژه را در یک پوشه ی مستقل ایجاد نماید. پس از انجام تنظیمات بر روی Next کلیک کنید.

3:



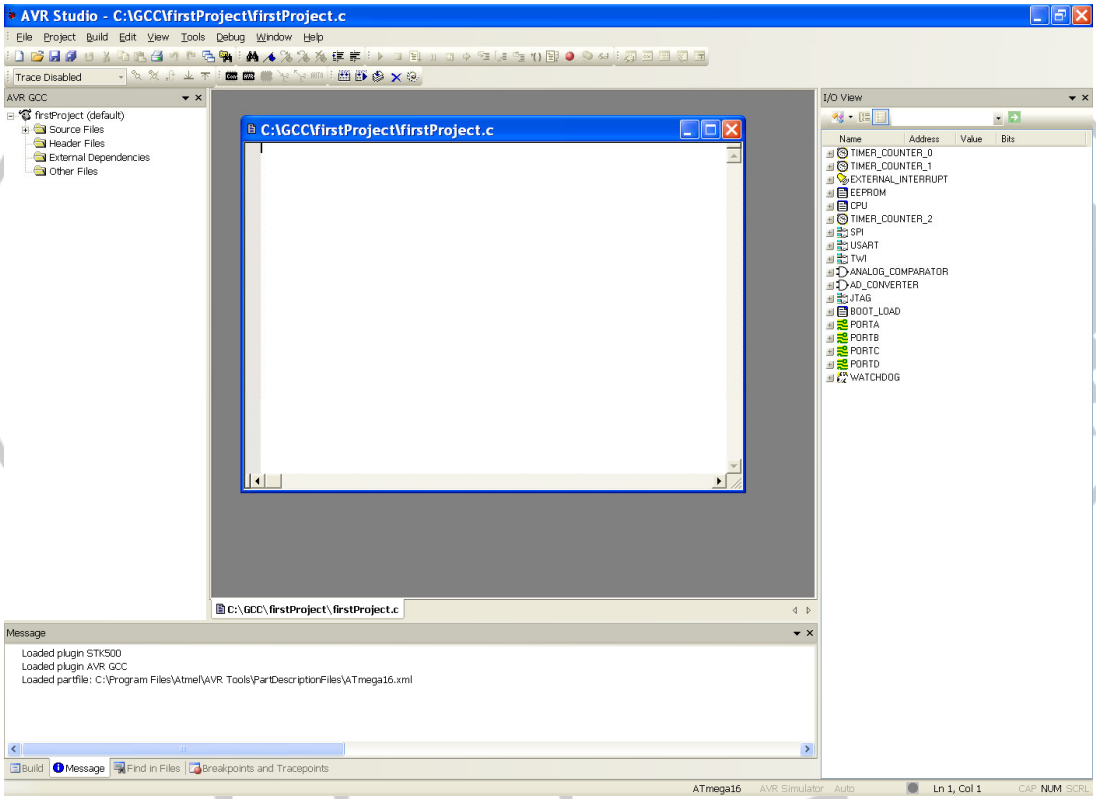
شکل ۲

۳. از لیست Debug platform گزینه ی AVR Simulator و از لیست Device قطعه ی ATmega32 را انتخاب کرده و بر روی Finish کلیک کنید.



شکل ۳

٤. ویزاد تمام می شود و پروژه ای با نام مورد نظر ساخته می شود. (شکل ٤)



شکل ٤

٥. برنامه ی ١ را تایپ کنید. دقت کنید که C یک زبان Case Sensitive بوده و به کوچک و بزرگ بودن حروف حساس می باشد.

5:

```
#include <avr/io.h>

int main (void)

{

	DDRD = 0xFF;
	int i;

	while(1)
	{

		for(i = 1; i <= 128; i = i*2)
		{
			PORTD = i;
			__asm__ volatile ("nop");
		}

		for(i = 64; i > 1; i = i/2)
		{
			PORTD = i;
			__asm__ volatile ("nop");
		}

	}

}
```

برنامه ۱

٦. در نوار ابزار بر روی دکمه ی  کلیک کرده و یا از منوی **Build** گزینه ی **Build and Run** را انتخاب کنید. در صورتی که در نوشتن برنامه خطایی نداشته باشد، برنامه کامپایل شده و پیکان زرد رنگی در ابتدای تابع **()main** قرار می گیرد (شکل ۵).

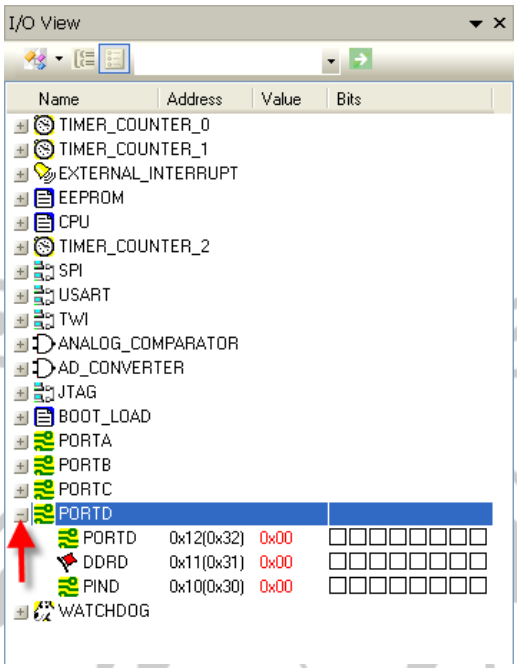
6:

```
#include <avr/io.h>

int main (void)
{

    DDRD = 0xFF;
    int i;

    while(1)
    {
        for(i = 1; i <= 128; i = i*2)
        {
            PORTD = i;
            __asm__ volatile ("nop");
        }

        for(i = 64; i > 1; i = i/2)
        {
            PORTD = i;
            __asm__ volatile ("nop");
        }
    }
}
```

شکل ٥

۷. از پانل I/O View بر روی علامت به علاوه ی کنار PORTD کلیک کنید. (شکل ٦)

7:



شکل ٦

۸. بسیار خوب! برنامه آماده ی شبیه سازی است. از نوار ابزار بر روی  کلیک کرده و یا از منوی **Debug** گزینه ی **Auto Step** را انتخاب کنید. ملاحظه می کنید که خانه ی روشن **PORTD** در پانل **I/O View** دائما از سمت راست به چپ و چپ به راست حرکت می کند. برای متوقف نمودن آن ابتدا بر روی  و سپس  کلیک کنید.

۹. بر روی دکمه ی  کلیک کرده و یا از منوی **Debug** گزینه ی **Start Debugging** را انتخاب کنید. ملاحظه می کنید که مجددا برنامه آماده ی شبیه سازی می باشد.

۱۰. از نوار ابزار بر روی دکمه ی  کلیک کرده و یا از منوی **Debug** گزینه ی **Step Into** را انتخاب نمایید. این بار می توانید برنامه را خط به خط اجرا نموده و تغییرات را در پانل **I/O View** مشاهده نمایید.

بررسی برنامه

- خط اول از رهنمودهای پیش پردازنده بوده و عبارت اجرایی نمی باشد. بدین ترتیب که قبل از کامپایل شدن برنامه، پیش پردازنده محتویات فایل **io.h** را به برنامه می الحاق می کند. توسط این فایل و فایل دیگری که از **io.h** فراخوانی

می شود آدرس رجیسترهای I/O و بیت های آن ها و برخی اطلاعات دیگر قطعه ی ATmega32 تعریف می شود.

- تابع main() نقطه اجرای برنامه بوده و هر برنامه ی C حداقل یک تابع، که تابع main() می باشد را داراست. همانطور که در شبیه سازی مشاهده نمودید اجرای برنامه از این تابع می باشد.

- با اجرای عبارت زیر جهت تمام پین های PORTD خروجی خواهد شد.

```
DDRD = 0xFF;
```

- برای اندیس حلقه ی تکرار نیاز به یک متغیر داریم که مکانی از حافظه از نوع int با نام i به شکل زیر تعریف می شود:

```
int i;
```

- بر خلاف نرم افزارهای کامپیوتری، هر Firmware نوشته شده برای یک Embedded System، نیاز به یک حلقه ی نامتناهی دارد که حلقه ی اصلی برنامه بوده و بجز زمان پاسخ دهی به وقفه، برنامه هیچگاه آن خارج نمی شود.

```
while(1)
{
حلقه ی اصلی برنامه
}
```

علت اینکه در نرم افزارهایی که بر روی کامپیوتر اجرا می شوند این حلقه وجود ندارد این است که در آنجا با اتمام برنامه، سیستم عامل کنترل سخت افزار را به دست می گیرد، در حالیکه در سیستم های Embedded معمولا سیستم عامل خود Firmware بوده و با اتمام آن دستورالعملی برای اجرا وجود ندارد.

همانطور که می دانید، ساختار for نوعی حلقه ی تکرار است که یک بلوک را به تعدادی که توسط شرط پایان حلقه مشخص می شود تکرار می کند. در اولین گام، اندیس حلقه برابر با یک بوده که در شرط پایان حلقه آزمایش شده و چون ۱۲۸=>۱ می باشد امکان اجرا نمودن بلوک می باشد. بعد از اجرای بلوک مجددا اندیس حلقه که این بار برابر با ۲*۱ بوده در شرط آزمایش شده و چون ۱۲۸=>۲ صحیح می باشد، بلوک اجرامی شود. به همین ترتیب تا زمانی که i برابر با ۲۵۶ شده و دیگر شرط حلقه صحیح نبوده و برنامه از حلقه خارج می شود.

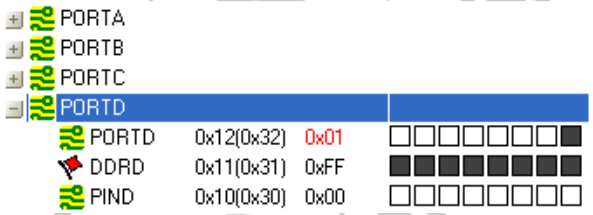
9:

با هر بار اجرای بلوک، مقدار اندیس حلقه در PORTD کپی شده و دستورالعمل اسمبلی nop یا No Operation اجرا می شود. nop به اندازه ی یک سیکل ماشین تاخیر ایجاد کرده و در زمان اجرای آن هیچ عملی انجام نمی شود. در کامپایلر avr-gcc برای نوشتن دستورات اسمبلی در کد زبان C می توان به همین شکل از __asm__ استفاده نمود.

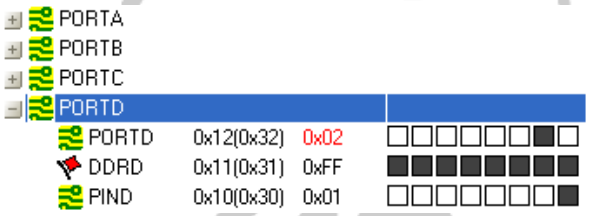
```
for(i = 1; i <= 128; i = i*2)
{
      PORTD = i;
      __asm__ volatile ("nop");
{
```

این حلقه را می توان با AVR Studio آزمایش نمود. نتیجه به صورت زیر می باشد.

اولین اجرای حلقه:



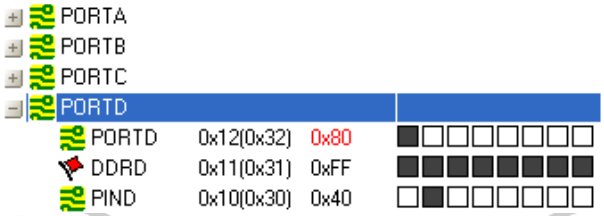
دومین اجرای حلقه:



●
●
●

10:

هشتمین اجرای حلقه:



پس از اتمام حلقه ی for نخست، دومین حلقه اجرا شده که عملکرد آن نیز مشابه این حلقه بوده و تنها اندیس و گام های آن متفاوت می باشد.

موفق باشید (-: