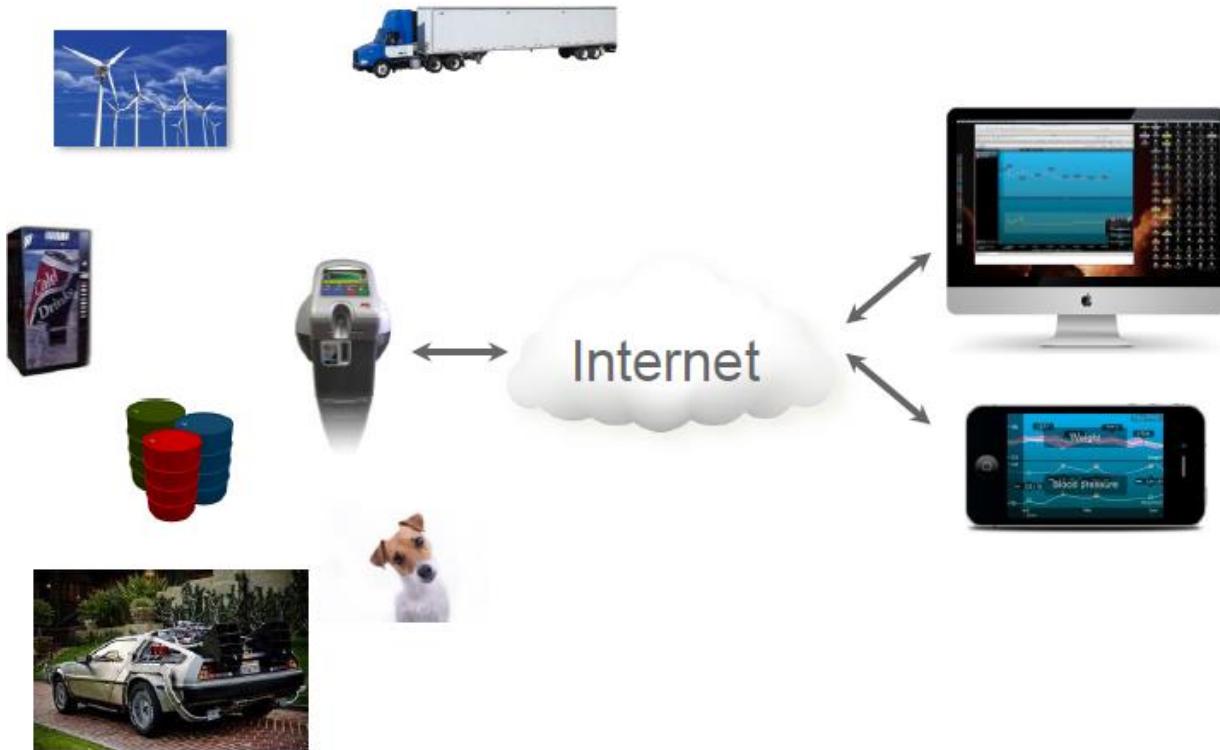


بسم الله الرحمن الرحيم

اینترنت اشیا (IoT) چیست؟



به گزارش اکو ویژن ، طبق تعاریف ارائه شده در ویکی پدیا عبارت اینترنت اشیاء، برای نخستین بار در سال ۱۹۹۹ توسط کوین اشتون مورد استفاده قرار گرفت و جهانی را توصیف کرد که در آن هر چیزی، از جمله اشیا بی جان، برای خود هویت دیجیتال داشته باشد و به کامپیوترها اجازه دهد آن ها را سازماندهی و مدیریت کند. اینترنت در حال حاضر همه مردم را به هم متصل می کند ولی با اینترنت اشیاء تمام اشیاء به هم متصل می شوند . اینترنت اشیاء (Internet of Things) مفهومی جدید در دنیا (IoT) فناوری و ارتباطات بوده و به طور خلاصه فناوری مدرنی است که در آن برای هر موجودی (انسان، حیوان و یا اشیاء) قابلیت ارسال داده از طریق شبکه های ارتباطی، اعم از اینترنت یا اینترانت، فراهم می شود



بستر اینترنت اشیا بر امواج رادیویی بی‌سیمی قرارداده شده که به دستگاههای مختلف این امکان را می‌دهند تا از طریق اینترنت با یکدیگر به برقراری ارتباط پیردازند. این بستر شامل استانداردهایی مانند وای‌فای، بلوتوث کمصرف، NFC، RFID و غیره است که شاید تاکنون اسم آن‌ها را هم نشنیده باشید.

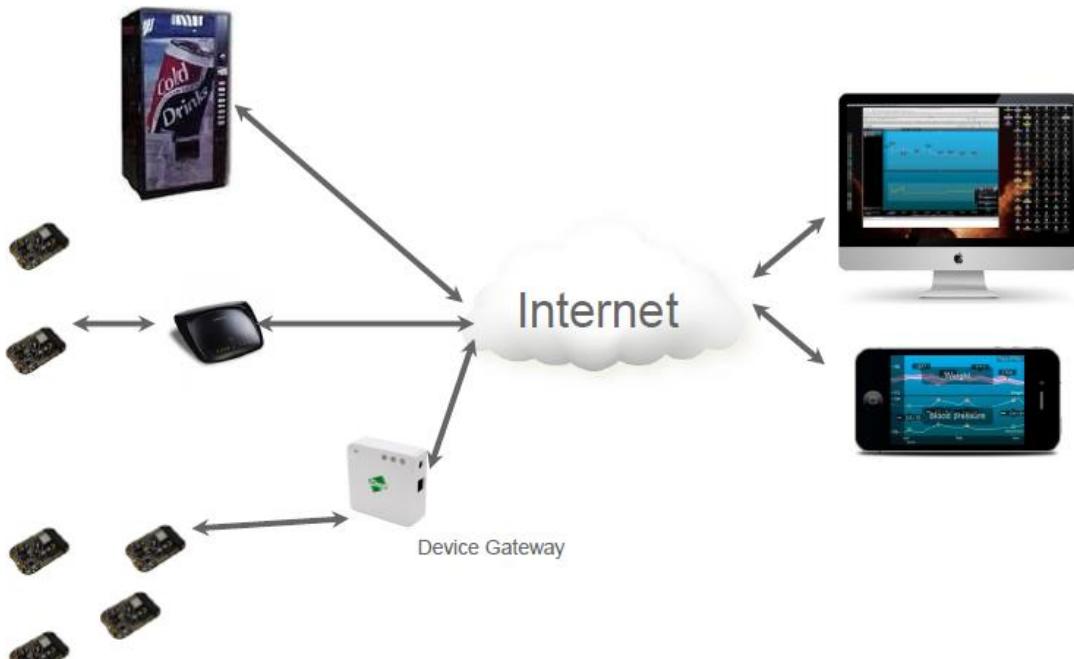
قفل‌های هوشمند، ترمومترات‌های هوشمند، خودروهای هوشمند، مطمئناً این‌ها واژه‌هایی هستند که بارها و بارها شنیده‌اید و البته در سال‌های آتی بیشتر خواهد شد.

همه دستگاههای پیشنهاده در دسته‌ای به نام اینترنت اشیاء یا به طور مخفف IoT قرار می‌گیرند. در سطح پایه‌ای، اینترنت اشیا در واقع به ارتباط اشیای مختلف از طریق اینترنت و برقراری ارتباط با یکدیگر می‌پردازد تا هدف آن یعنی فراهم‌کردن تجربه کارآفر و هوشمندتر محقق شود. همانند دیگر تکنولوژی‌های جدید، IoT نیز می‌تواند در ابتدا مفهومی سردرگم‌کننده به نظر برسد. همچنین این واژه به‌ویژه هنگامی که صحبت از استانداردهای مختلف و همچنین این‌منی و امنیت آن می‌شود می‌تواند مفاهیم جدید و ویژه‌ای پیداکند. به عبارت دیگر ایده طراحی دستگاههای مختلف با امکان برقراری ارتباط بی‌سیم به منظور رهگیری و کنترل از طریق اینترنت و حتی یک برنامه ساده مخصوص گوشی‌های هوشمند، اصطلاح اینترنت اشیاء را توصیف می‌کند.

برای این کار از برنامه‌ی **ESPlorer** استفاده می‌کنیم این برنامه برای ایجاد و ذخیره فایل‌های LUA در ماژول ESP استفاده می‌شود. نکته‌ی قابل توجه این است که این برنامه به زبان جاوا است و فایل اجرایی آن jar است که برای استفاده باید **java** در سیستم نصب شده باشد.

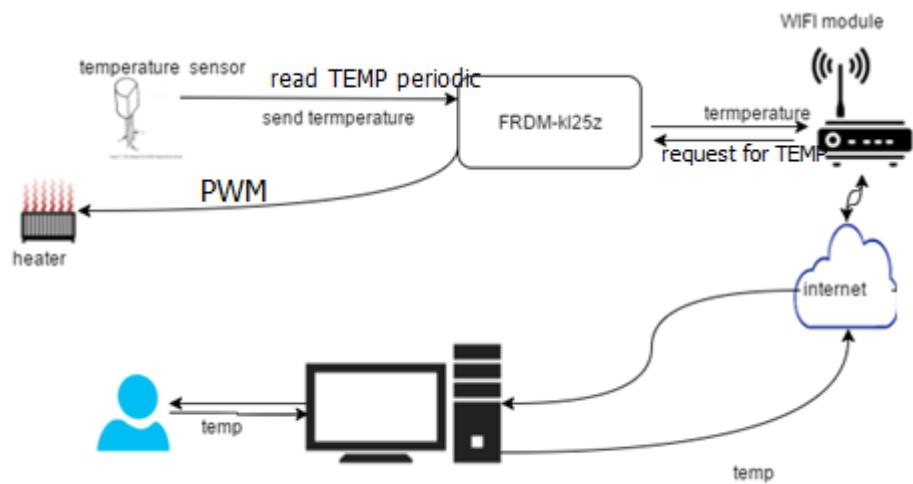
پیش‌بینی‌ها :

بررسی‌های موسسه تحقیقاتی گارتنر نشان می‌دهد که تا سال ۲۰۲۰ بیش از ۲۵ میلیارد وسیله مختلف در جهان از طریق خدمات مبتنی بر اینترنت اشیاء به اینترنت یا دیگر شبکه‌های اطلاع رسانی متصل خواهند شد. بررسی‌های شرکت سیسکو نیز حاکیست که تا غلبه یافتن پدیده اینترنت اشیاء تنها سه سال زمان باقیست. بنابراین در سال ۲۰۱۸ ماشین‌های و سیستم‌های الکترونیکی می‌توانند از طریق اینترنت بدون نیاز به انسان‌ها و حتی بیشتر از آنها با یکدیگر در ارتباط باشند.



سناریوی کلی پروژه :

بورد FRDM-KL25Z به صورت مداوم و بازه های زمانی یک ثانیه دمای محیط را از سنسور LM35 می خواند و در متغیر TEMP ذخیره می کند و با استفاده از UART به سمت ماژول فای ارسال می کند . سپس هر بار که کاربر با کلیک بر روی TEMP دما را تقاضا کند دما در صفحه ی جاری نمایش داده می شود.



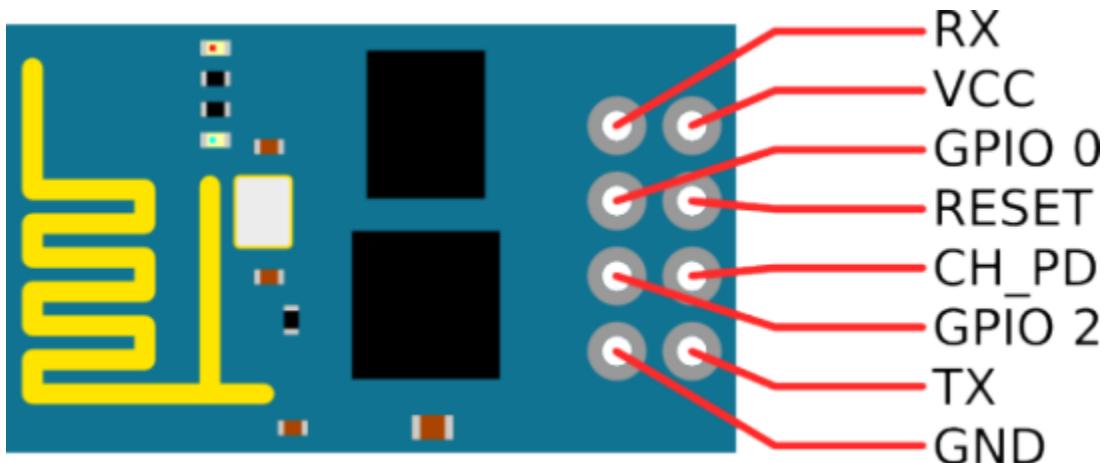
پیاده سازی ماژول وای فای:

ماژول وای فای مورد استفاده در این پروژه ESP8266 می باشد .

ماژول وای فای : ESP8266



شماییک مازول وای فای به صورت زیر است :



و ارتباط پایه ها به صورت زیر است :

RX -> TX

TX -> RX

CH_PD -> 3.3V

GPIO 0 -> GND

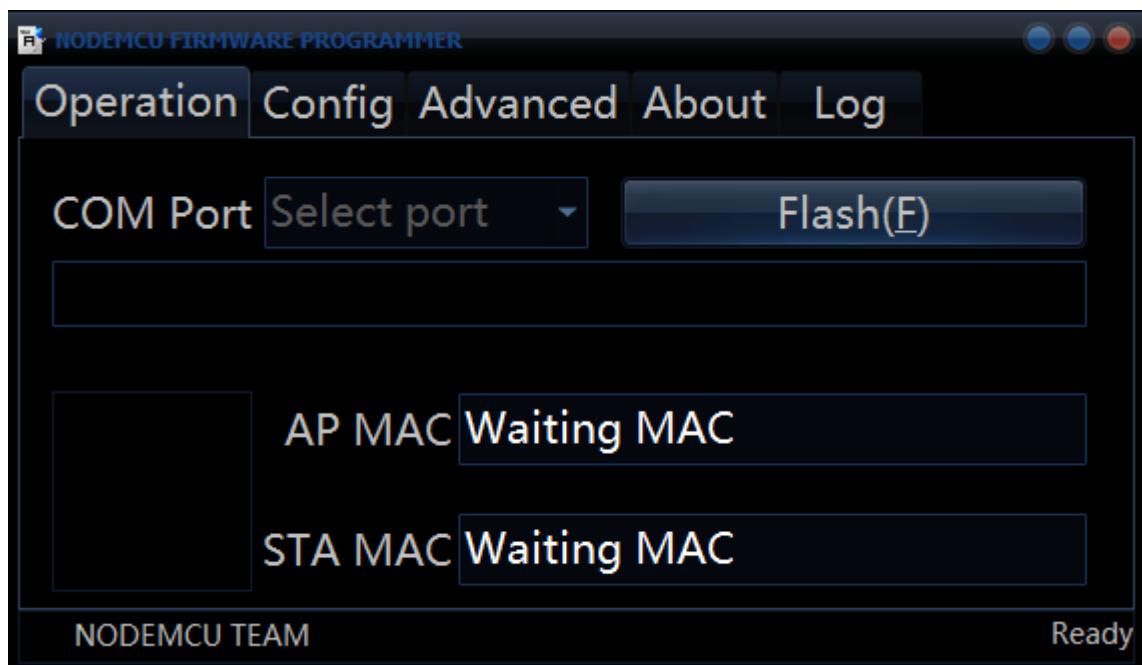
VCC -> 3.3V

GND -> GND

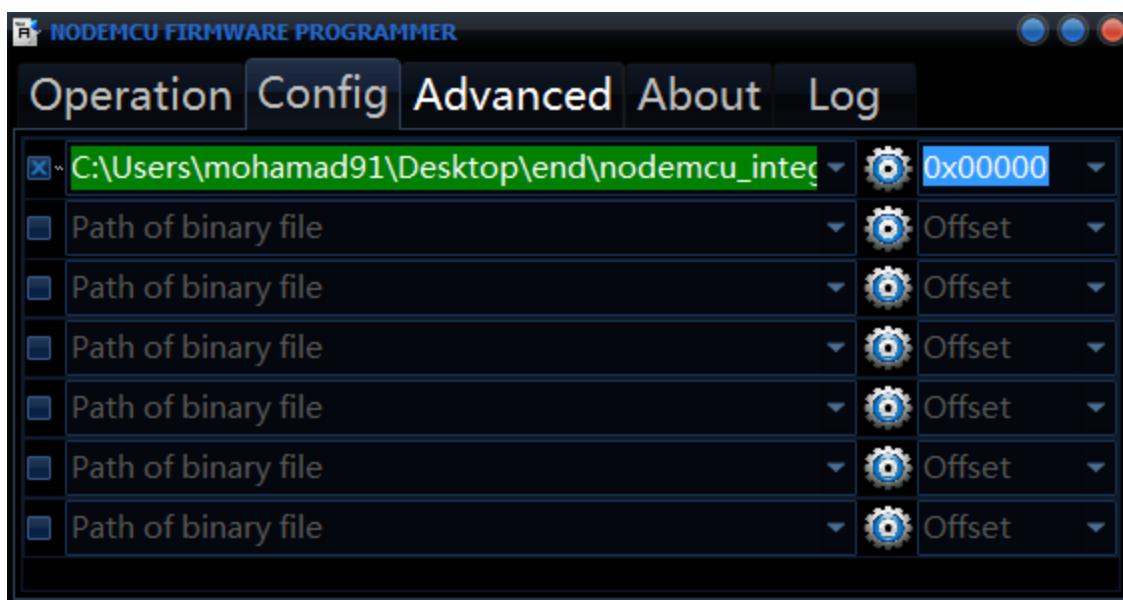
صورت و پایه ی ریست active low است و میتوان به VCC وصل کرد یا اینکه not connected رها کرد .

فلاش کردن :ESP8266

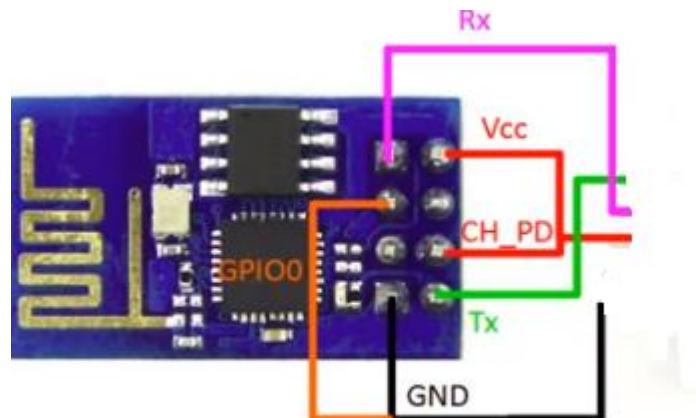
با استفاده از برنامه NodeMCU flasher مازول وای فای را با فریم ویر nodeMCU فلاش میکنیم .نمای کلی nodeMCU



در قسمت config مکان فایل باینری فریم ویر را که دانلود کرده ایم میدهیم :

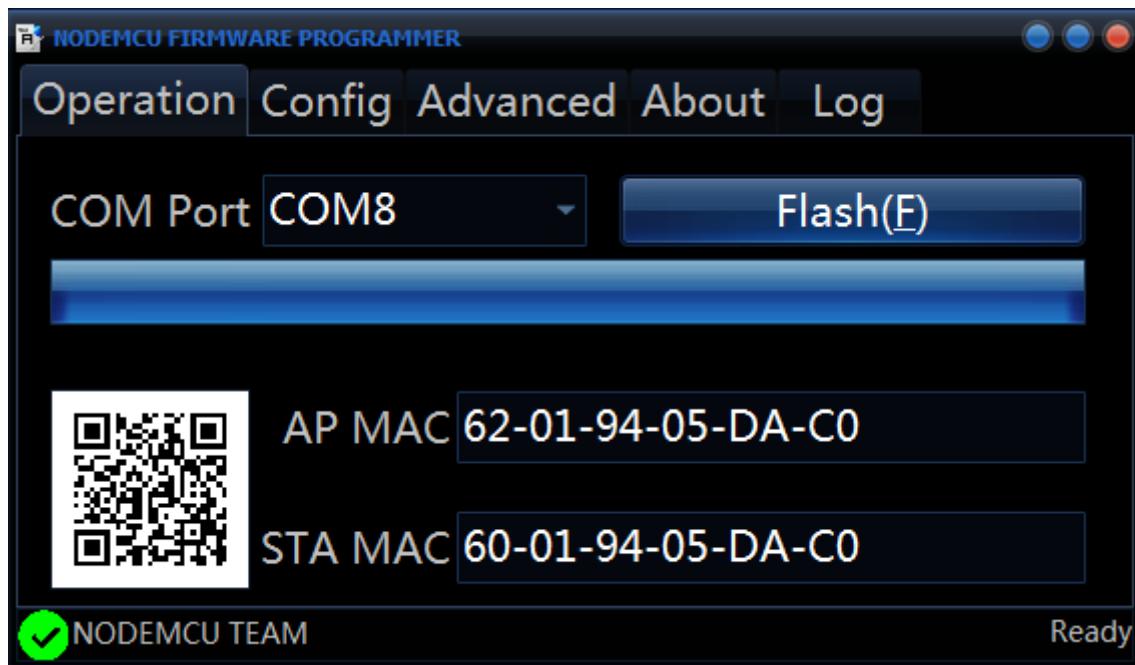


دقت کنید که برای فلش کردن این ماثول باید پایه ی GPIO0 به GND متصل باشد مطابق شکل:



Connect GPIO pin to GND only to refash firmware

روش فلش کردن به این صورت است که ابتدا بر روی فلش کلیک کرده و سپس سیستم به مژول متصل شده حال باید به صورت دستی پاور مژول را قطع کرد و دوباره وصل کرد و سپس بر روی فلش کلیک کنید در این صورت مطابق زیر فلش به درستی بارگذاری می‌شود و باید علامت سبز رنگ در قسمت سمت چپ و پایین به نمایش درآید :

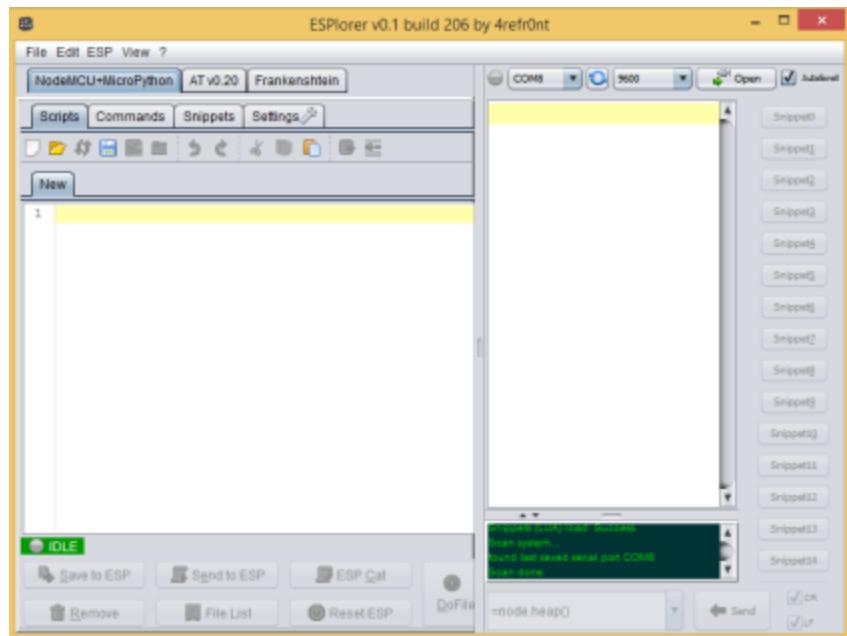


و در قسمت Log نیز مراحل پیشرفت flash نشان داده می‌شود .

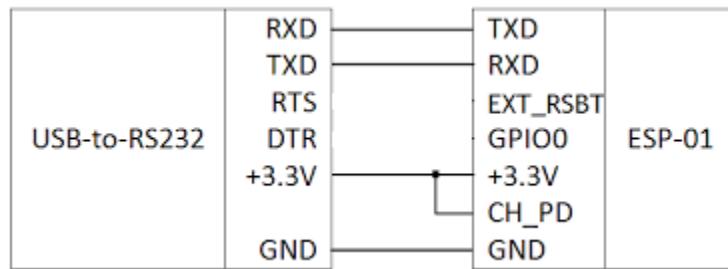
بارگذاری کد در مژول :

برای این کار از برنامه‌ی **ESPlorer** استفاده میکنیم این برنامه برای ایجاد و ذخیره فایل‌های LUA در ماژول ESP استفاده میشود. نکته‌ی قابل توجه این است که این برنامه به زبان جاوا است و فایل اجرایی آن jar است که برای استفاده باید **java** در سیستم نصب شده باشد.

نمای کلی برنامه **ESPlorer**

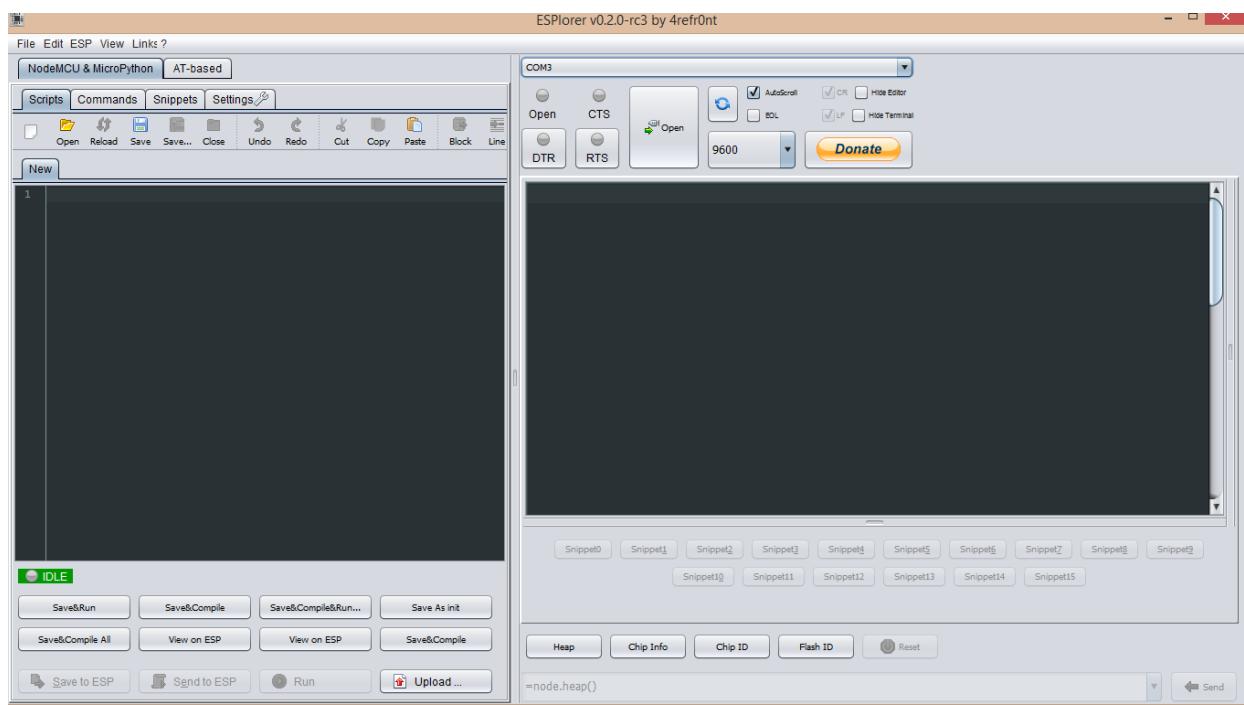


لازم به ذکر است که این ماژول با استفاده از **UART** برنامه ریزی میشود. که برای استفاده از این برنامه و کار با ماژول وای فای نیاز به یک ماژول **RS232** و خروجی **TTL** که به ماژول وای فای متصل میشود نیاز داریم. شماتیک اتصال به صورت زیر است.

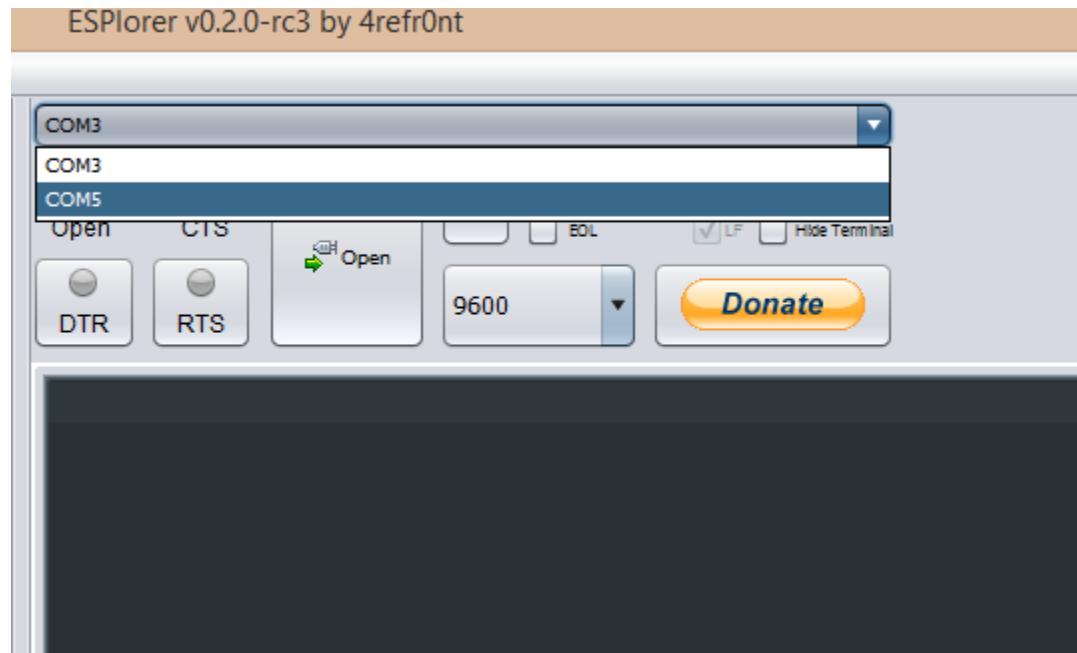


برای برنامه ریزی و پروگرم کردن این مازول از UART و rs232 ttl استفاده میکنیم . برنامه های که برای این منظور و کار کردن با UART برای پروگرم کردن مازول وای فای استفاده میشود esplorer و lualoader میباشد که ما از esplorer استفاده میکنیم .

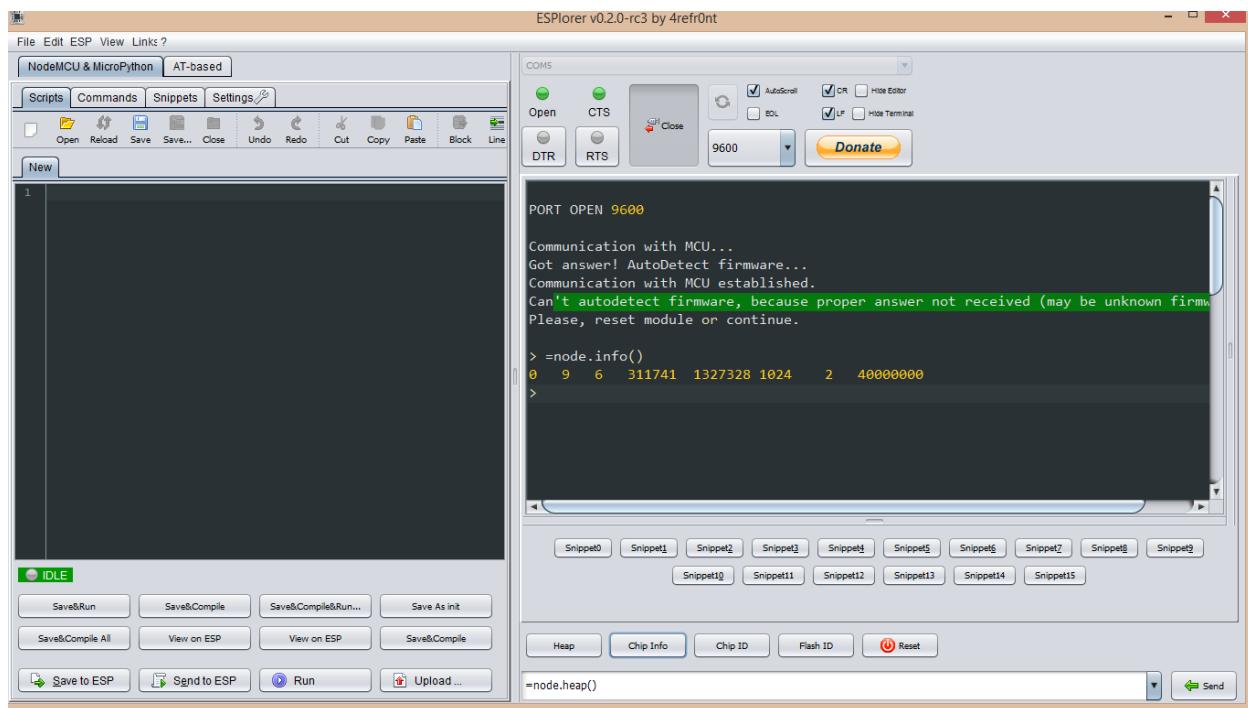
نمای کلی esplorer به صورت زیر میباشد :



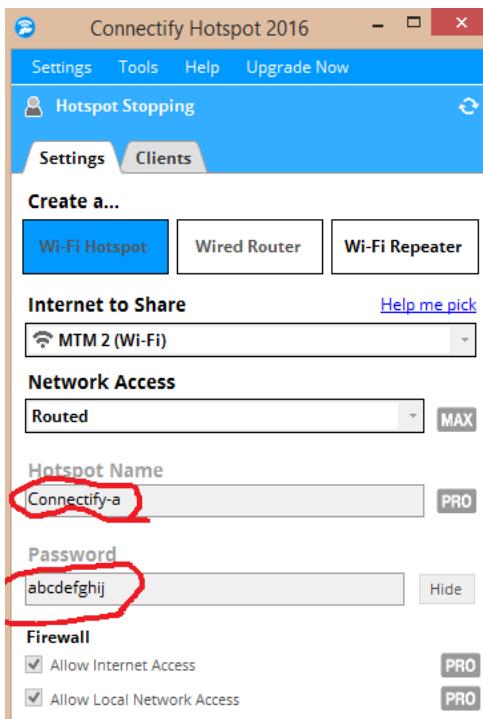
همانطور که در شکل زیر مشخص است ابتدا پورت مربوط به مازول وای فای را انتخاب میکنیم :



با فشردن **open** این برنامه به بورد متصل شده باید کامند زیر را در خروجی ببینید در این شکل chip info نیز فشرده شده است و اطلاعات مازول را نمایش میدهد.



حال با استفاده از این مازول یک وب سرور محلی ایجاد میکنیم . کد مربوطه که به زبان LUA نوشته شده است به صورت زیر میباشد . برای این کار یک شبکه ای محلی با استفاده از connectify hotspot ایجاد کرده که بتوانیم به راحتی به مازول هایی که به آن متصل میشوند دسترسی داشته باشیم . نام شبکه و پسورد آن را در شکل زیر مشاهده میکنید.



حال به تحلیل کد سرور که در مازول وای فای قرار میگیرد میپردازیم :

```
wifi.setmode(wifi.STATION)
wifi.sta.config("shahideharam","abc123456")
print(wifi.sta.getip())

--uart.write(0, ip) or print(ip)
led1 = 3 -- pin number
led2 = 4 -- pin number
waitForTemp=0
gpio.mode(led1, gpio.OUTPUT)
gpio.mode(led2, gpio.OUTPUT)
srv=net.createServer(net.TCP)--net.createServer(net.TCP,1000)
1000s time out
temp=20
-- when 4 chars is received this function call .
uart.on("data","\^",
function(data)

if data=="quit^" then
    uart.on("data") -- unregister callback function
end
end)
```

```

end
waitForTemp=1
dataLen=string.len(data)
print("dataLen with ^",dataLen)
dataLen=dataLen-1 -- without EOD
temp=string.sub(data,1,dataLen)
print("temp is",temp)

if data=="" then
    uart.on("data") -- unregister callback function
end
end, 0)--0:input fromUART will not go to Lua interpreter, can
accept bin data
srv:listen(80,function(conn)

conn:on("receive", function(client,request)
    local buf = "";
    local _, _, method, path, vars = string.find(request,
"([A-Z]+) (.+)?(.+) HTTP");
    if(method == nil)then
        _, _, method, path = string.find(request, "([A-Z]+)
(.+) HTTP");
    end
    local _GET = {}
    if (vars ~= nil)then
        for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
            _GET[k] = v
        end
    end
    local _on,_off = "", ""
    if(_GET.pin == "TempShow")then--send temp req to board
and wait for get temp
        --gpio.write(led1, gpio.HIGH);
        --uart.write(0, 'S0DtempEOD');-- send temp and now
must wait for receive
        --wait until get tmep
        -- while(waitForTemp==0)
        -- do

        -- end
        -- if can delay here . all is ok

```

```

-- sleep(3)
waitForTemp=0
--tmr.delay(3000);-- delay for receive temp in
uart.on

end
if(_GET.setTemp ~= nil)then
    local command = "^temp";
    command=command.._GET.setTemp;
    command=command.."^";

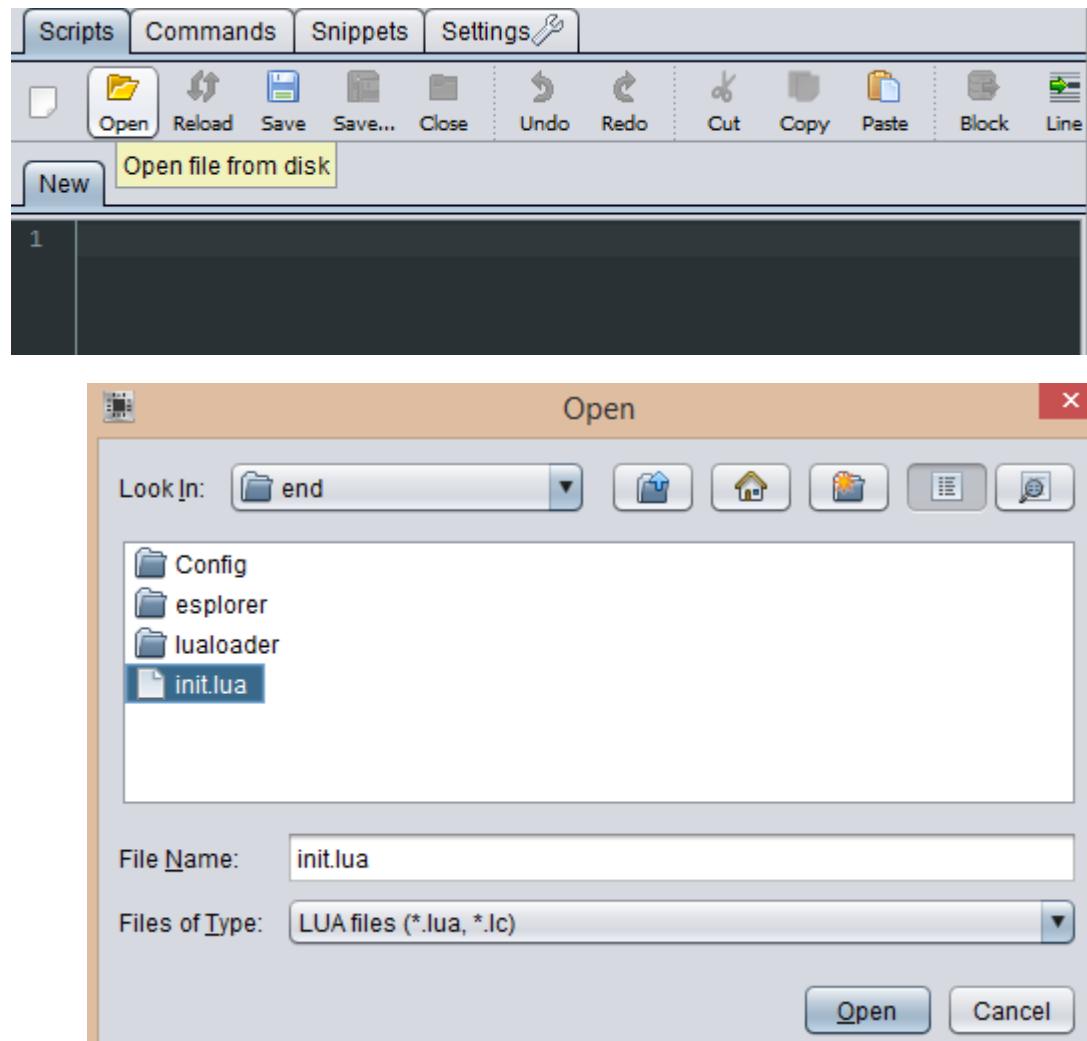
    uart.write(0,command);
end
buf = buf..'<h1 style="background-
color:powderblue;color:yellow;"> Server for Read and change
Temperature </h1>';
    buf = buf.."<p>Temperature Display ..temp.."<a
href=\"?pin=TempShow\"><button>TEMP</button></a>&nbsp;</p>";
    buf = buf..'<form action="/" method="get"><p>insert
your Optimum temperature :</p> <input type="text"
name="setTemp"><input type="submit" value="Send">&nbsp;</form>';
    -- buf = buf.."<p>GPIO2 <a
href=\"?pin=ON2\"><button>ON</button></a>&nbsp;<a
href=\"?pin=OFF2\"><button>OFF</button></a></p>";

    -- elseif(_GET.pin == "OFF1")then
        --     gpio.write(led1, gpio.LOW);
    --elseif(_GET.pin == "ON2")then
        --     gpio.write(led2, gpio.HIGH);
    --elseif(_GET.pin == "OFF2")then
        --     gpio.write(led2, gpio.LOW);

    -- end
    client:send(buf);
    client:close();
    collectgarbage();
end)-- end of conn on receive
end)--end of listen

```

همانطور که در کد میبینید این مازول به شبکه وای فای با نام connectify-a و پسورد abcdefghij متصل میشود که شبکه‌ی محلی است که خودمان ایجاد کردیم حال این کد را در قالب فایل با پسوند lua ذخیره میکنیم و سپس با استفاده از که در esplorer وجود دارد به مازول وای منتقل میکنیم :



File Edit ESP View Links ?

NodeMCU & MicroPython AT-based

Scripts Commands Snippets Settings

Open Reload Save Save... Close Undo Redo Cut Copy Paste Block Line

init.lua

```

1 wifi.setmode(wifi.STATION)
2 wifi.sta.config("Connectify-a","abcdefgij")
3 print(wifi.sta.getip())
4 led1 = 3
5 led2 = 4
6 gpio.mode(led1, gpio.OUTPUT)
7 gpio.mode(led2, gpio.OUTPUT)
8 srv=net.createServer(net.TCP)
9 srv:listen(80,function(conn)
10     conn:on("receive", function(client,request)
11         local buf = ""
12         local _, _, method, path, vars = string.find(request, "([A-Z]")
13         if(method == nil)then
14             _, _, method, path = string.find(request, "([A-Z]+) (.+)")
15         end
16         local _GET = {}
17         if (vars ~= nil)then
18             for k, v in string.gmatch(vars, "(%w+)=(%w+)&*") do
19                 _GET[k] = v
20             end
21         end
22         buf = buf.."<h1> ESP8266 Web Server</h1>";
23         buf = buf.."<p>GPIO0 <a href=\"?pin=ON1\"><button>ON</button>
24         buf = buf.."<p>GPIO2 <a href=\"?pin=ON2\"><button>ON</button>
25         local on_off = "" "

```

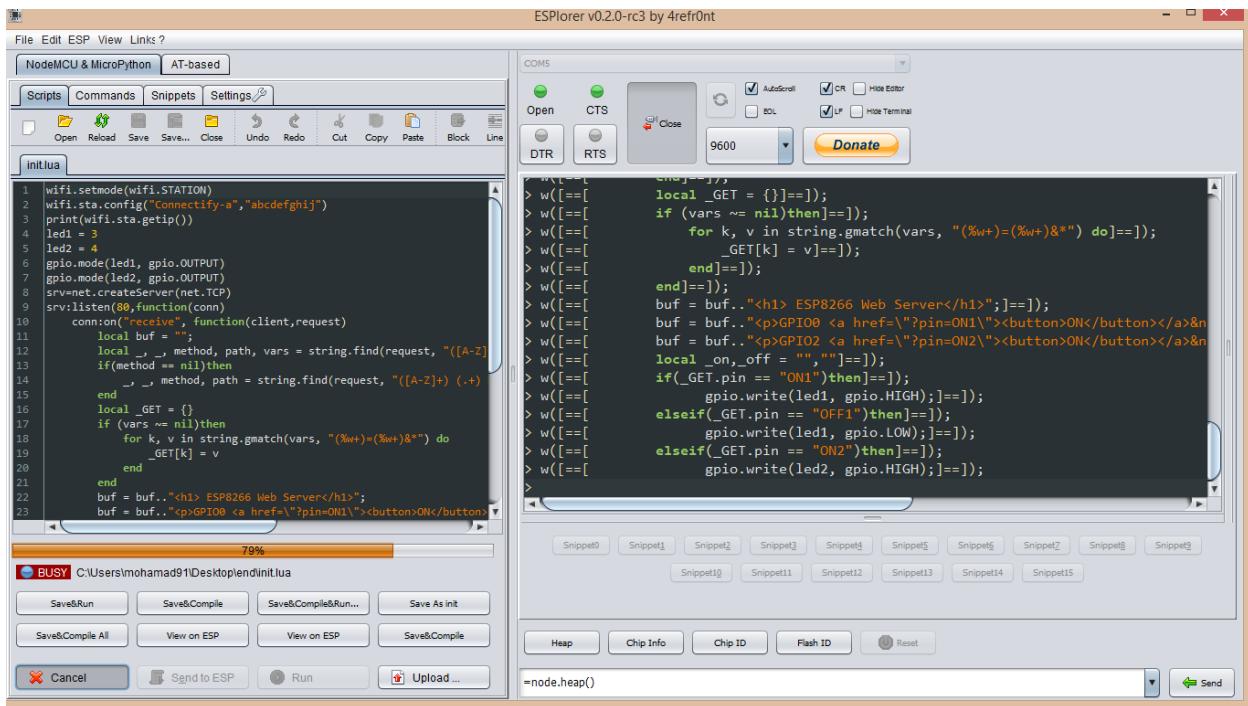
IDLE C:\Users\mohamad91\Desktop\lend\init.lua

Save&Run Save&Compile Save&Compile&Run... Save As init

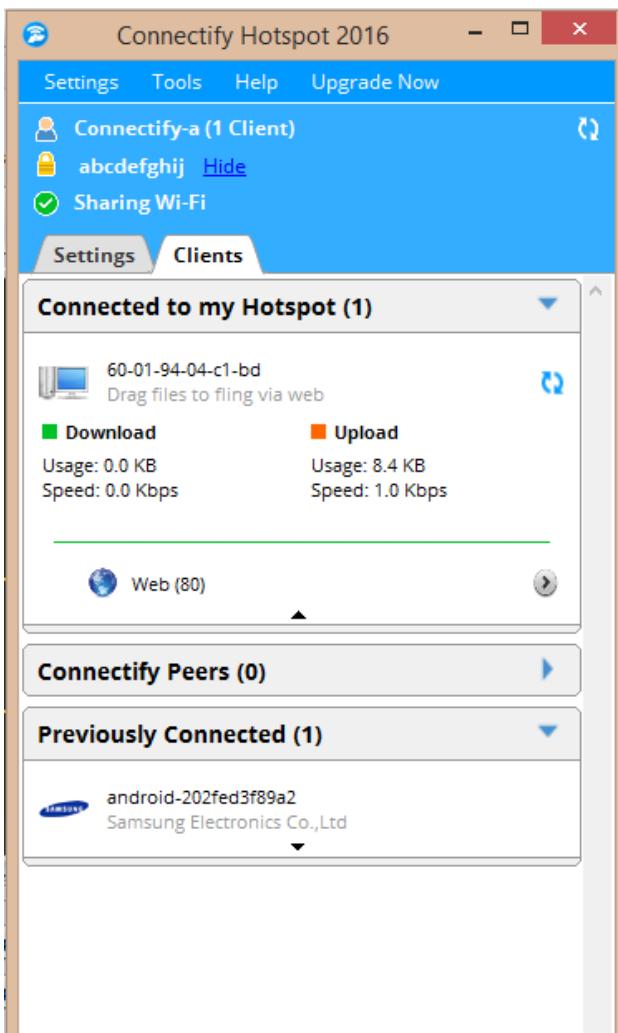
Save&Compile All View on ESP View on ESP Save&Compile

Save to ESP Send to ESP Run Upload ...

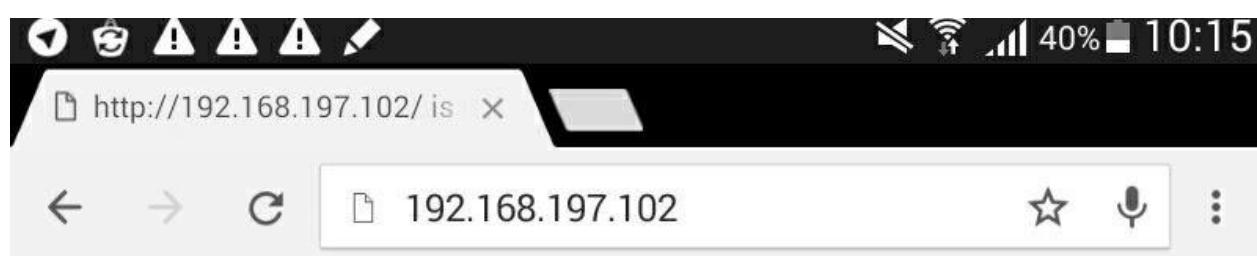
حال روی **save to ESP** کلیک میکنیم.



با تمام شدن انتقال و ریست کردن مازول به شبکه‌ی محلی متصل می‌شود و **IP** اختصاص یافته به مازول در شکل زیر مشخص است. حال برای دسترسی به سرور ایجاد شده با مازول باید یک دستگاه دیگر به شبکه‌ی محلی متصل شده تا بتواند به سرور دسترسی داشته باشد. با ورود به **IP** این مازول صفحه‌ی زیر را مشاهده می‌کنید:



و با وارد کردن آی پی در شبکه‌ی محلی به سرور متصل می‌شویم :



ESP8266 Web Server

GPIO0

GPIO2

حال به بررسی کد برنامه و تغییر برای ارتباط با بورد **FRDM KL25Z** میپردازیم :

منبع :

<https://nodemcu.readthedocs.io/en/master/en/modules/net/#netsocket>

در خط (**wifi.sta.config("Connectify-a","abcdefgij")**) نام شبکه‌ی محلی و پسورد آن داده میشود و این بورد به این شبکه متصل میشود .

به عنوان ورودی **gpio.mode**: تنظیم اولیه‌ی پین به عنوان ورودی یا خروجی و تنظیم پول آپ اختیاری که ایندکس پین و مود پین را به عنوان ورودی گرفته و تنظیم میکند.

ایجاد یک سرور بر مبنای **TCP** و به عنوان خروجی مازول : **net.createServer(net.TCP)** را بر میگرداند .

که یک پارامتر به عنوان پورت و یک تابع در ورودی میگیرد و روی این پورت گوش میکند و در صورت اتصال تابع را صدا میزنند و اجرا میکند . در واقع این متد به صورت :

function net.server.listen(port,[ip],function(net.socket)) می باشد که تابع **net.server.listen** می‌گامی صدا زده میشود که اتصال به درستی برقرار شده باشد و به عنوان ورودی سوکت اتصال ارسال می شود .

این تابع در مازول سوکت برای برخی **event** ها انجام میشود. در کد رخداد این تابع دریافت ("Receive") است . و هنگام دریافت انجام میشود .

به کار گیری **UART** در مازول با استفاده از **LUA** :

<https://nodemcu.readthedocs.io/en/master/en/modules/net/#netsocket>

<https://nodemcu.readthedocs.io>

میخواهیم با استفاده از **UART** با سیستم ارتباط برقرار کنیم :

.....

یکی از مشکلات این بود که به تابع دریافت یوآرت رو عوض کردم و این تابع را لود کردم:

```
uart.on("data",
        function(data)
        print("rrrr", data)
```

```

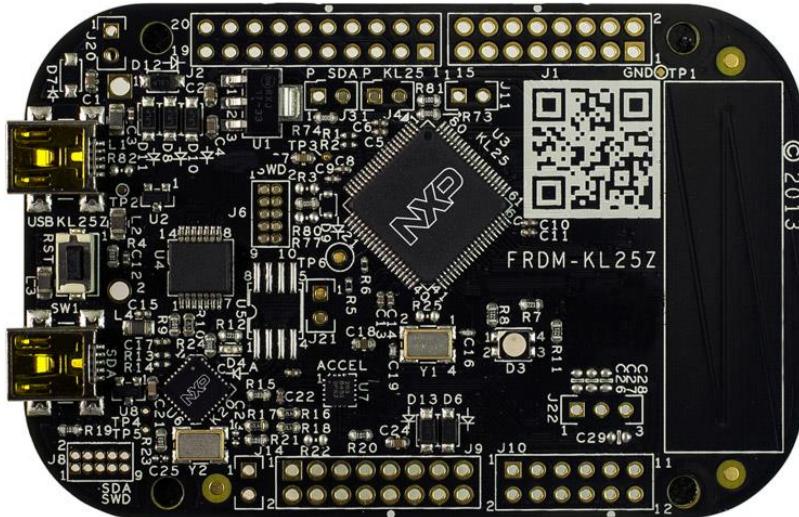
if data=="quit" then
    uart.on("data") -- unregister callback function
end
end, 0

```

که بعد از این دیگر نمیتوان هیچ ارتباطی با مژول داشت زیرا دیگر هر پیامی نیز رد و بدل شود به این صورت تبدیل میشود حتی هیچ هم نمیتوان به دست آورد.

borad مورد استفاده می یک بورد ساخته شرکت **freescale** با نام **FRDM-KL25Z** میباشد.

مشخصات بورد:



این بورد یک پلت فرم توسعه یافته فوق العاده کم هزینه برای میکروهای **Kinetis (MCU)** سری L ساخته شده بر مبنای ARM® Cortex®-M0+ processor میباشد.

از جمله ویژگی های آن میتوان به دسترسی آسان به پایه های **I/O**، عملگر های کم توان ، رابط کاربری خطایابی درونی برای برنامه ریزی فلاش و کنترل اجرا نام برد .

میتوان از **mbed.org** برای دسترسی آسان و بدون هزینه به ابزار ها ، کد های آماده و ارتباط با متخصصان استفاده کرد.

hdk

ویژگی ها:

- MKL25Z128VLK4 MCU – 48 MHz, 128 KB flash, 16 KB SRAM, USB OTG (FS), 80LQFP
- Capacitive touch “slider,” MMA8451Q accelerometer, tri-color LED
- Easy access to MCU I/O
- Sophisticated OpenSDA debug interface
- Mass storage device flash programming interface (default) – no tool installation required to evaluate demo apps
- P&E Multilink interface provides run-control debugging and compatibility with IDE tools
- Open-source data logging application provides an example for customer, partner and enthusiast development on the OpenSDA circuit

Clocks

Clock generation module with FLL and PLL for system and CPU clock generation

4 MHz and 32 kHz internal reference clock

System oscillator supporting external crystal or resonator

Low-power 1kHz RC oscillator for RTC and COP watchdog

Analog peripherals

16-bit SAR ADC w/ DMA support

12-bit DAC w/ DMA support

High speed comparator

Communication peripherals

Two 8-bit Serial Peripheral Interfaces (SPI)

USB dual-role controller with built-in FS/LS transceiver

USB voltage regulator

Two I₂C modules

One low-power UART and two standard UART modules

Timers

One 6-channel Timer/PWM module

Two 2-channel Timer/PWM modules

2-channel Periodic Interrupt Timer (PIT)

Real time clock (RTC)

Low-power Timer (LPT)

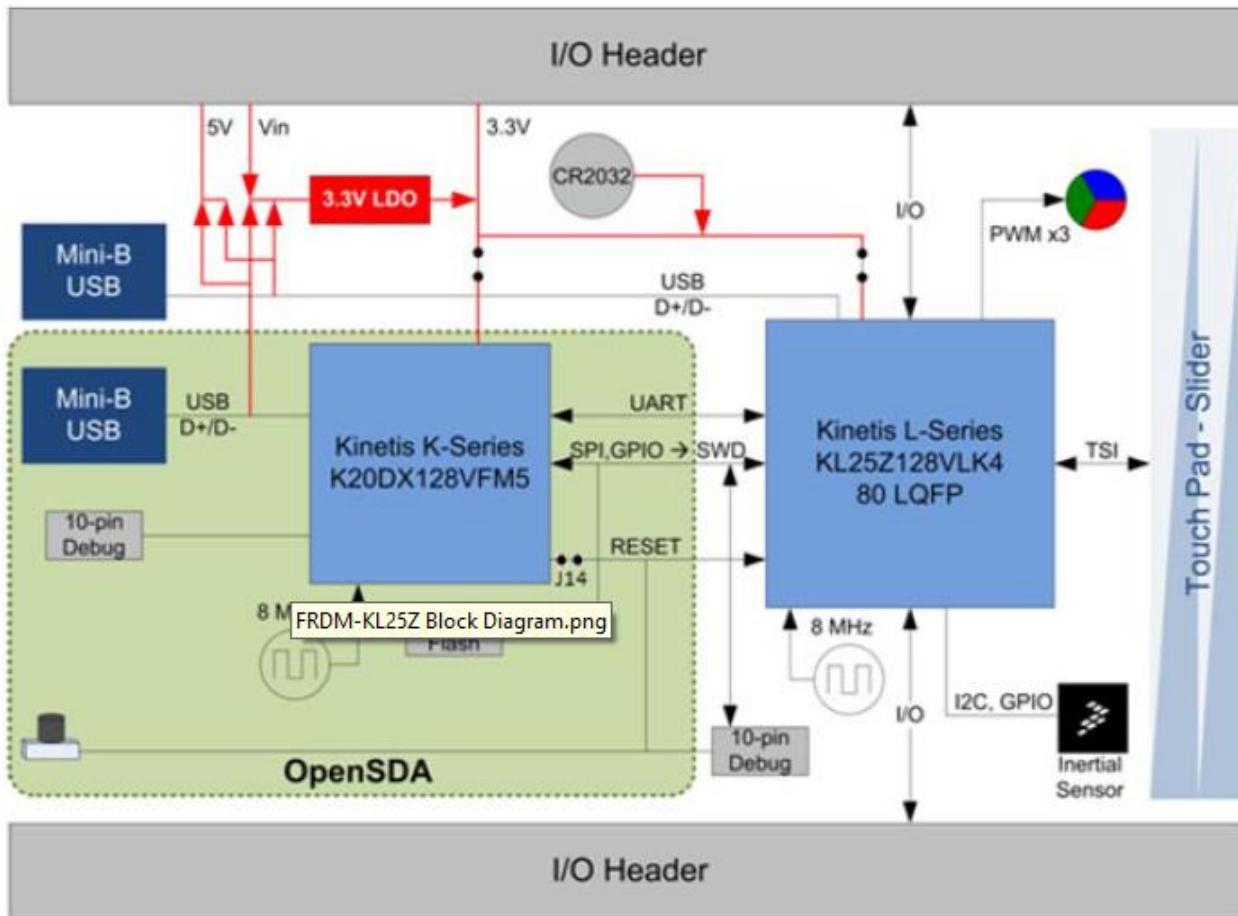
System tick timer

Human-Machine Interfaces (HMI)

General purpose input/output controller

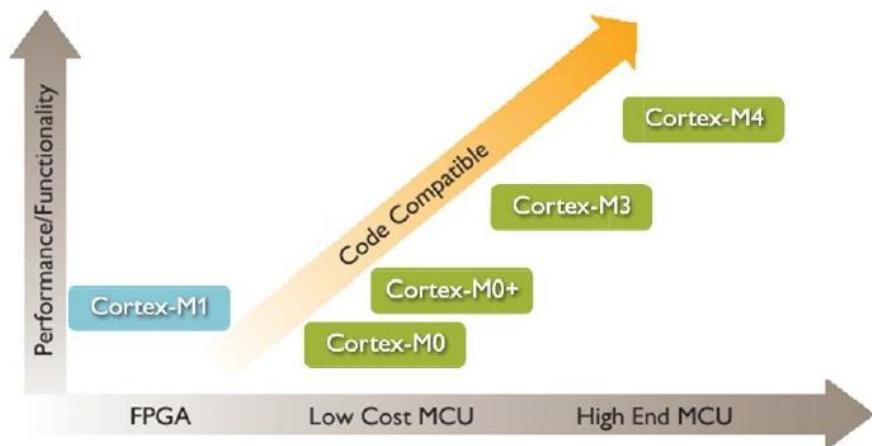
Capacitive touch sense input interface hardware module

شکل زیر بلاک دیاگرام بورد را نمایش میدهد . اجزای اصلی و مکان سخت افزار آنها در شکل مشخص است .



حالا به صورت خلاصه این میکروکنترلر را بررسی کرده و با خانواده های دیگر مقایسه میکنیم . همانطور که میدانیم میکرو استفاده شده در این بورد یک میکرو از نسل M0 کورتکس میباشد که به صورت زیر با نسل های M1,M2,M3,M4 قابل مقایسه است :

ARM® Cortex™-M Processors Series



در واقع این میکرو ساده ترین و راحت‌ترین میکرو در میکرو های سری M میباشد که برای شروع و یادگیری بسیار مناسب است. در واقع هسته های ۳۲ بیتی توان بسیار بیشتری مصرف میکنند و همچنین حافظه‌ی بیشتری را ساپورت میکنند و کارآیی نیز به مراتب بیشتر است ولی در M0 به دلیل هسته‌ی ۱۶ بیتی توان مصرفی پایین‌تر و هزینه‌ی آن نیز کمتر است لذا با توجه به نیاز پروژه میتوان میکروی با کارآیی بالا را انتخاب کرد.

منابع کلاک:

میکرو های k12 سری kinetis on-chip اسیلاتور های سازگار با سه رنج از ورودی های کریستال یا رزوناتور هستند:
32-40 kHz (low freq. mode),
3-8 MHz (high freq. mode, low range) ,
and 8-32 MHz (high freq. mode, high range).

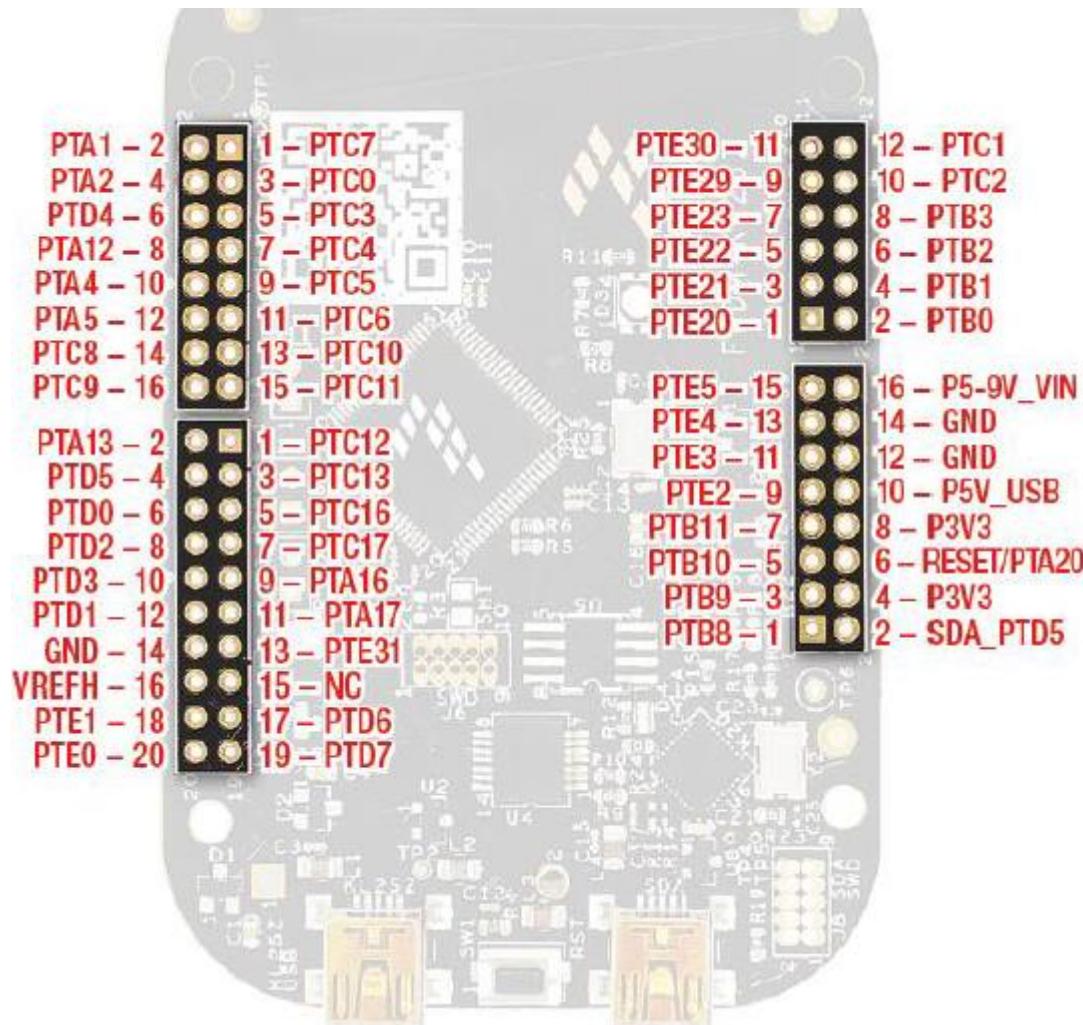
که میکرو KL25Z128 که بر بورد FRDM-KL25Z قرار دارد از کریستال 8MHz استفاده میکند.

پورت سریال :

اولین رابط سریال این بورد به پایه های PTA1,PTA2 متصل شده است. این 2 پایه علاوه بر UART به J1 و J2 I/O connector نیز متصل است.

پایه های خروجی از بورد:

نمایش پایه های خروجی بورد به صورت زیر است :



ای دی های RGB

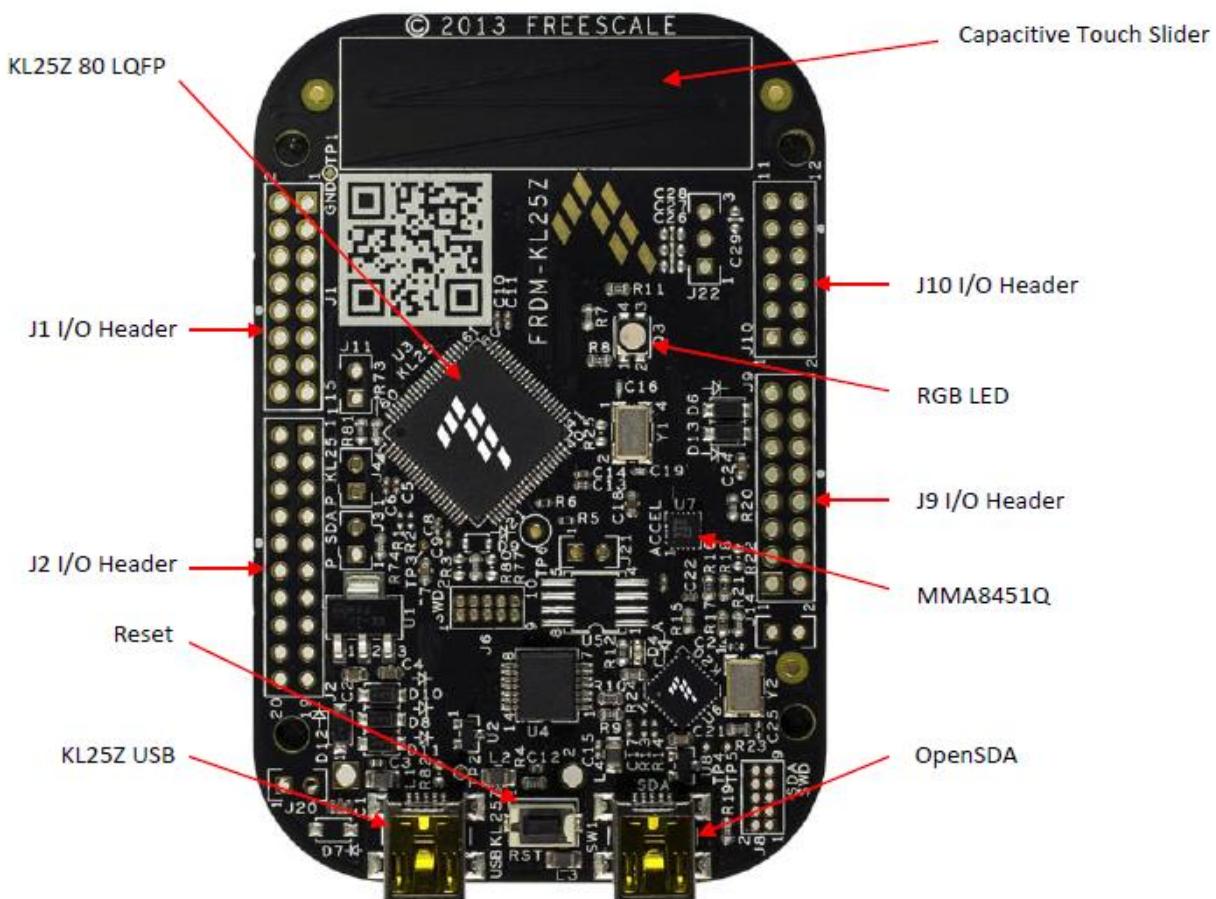
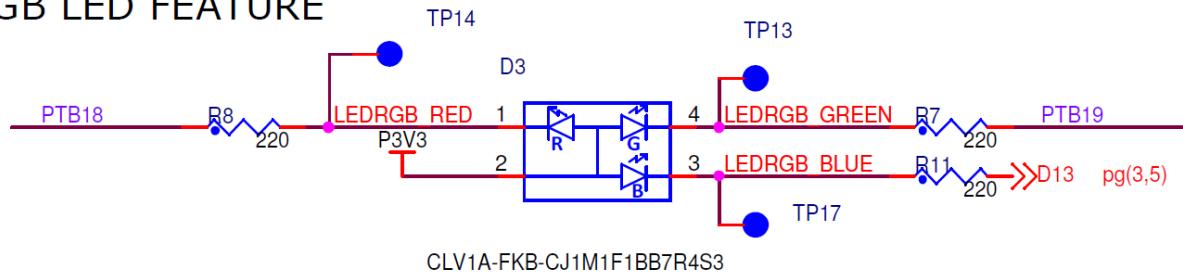
سه سیگنال PWM به سه پایه ای دی های قرمز و سبز و زرد متصل میباشند.

RGB LED Signal Connections

RGB LED	KL25Z128
Red Cathode	PTB18
Green Cathode	PTB19
Blue Cathode	PTD1 ¹

و شماتیک پایه ها نیز به صورت زیر میباشد که برای راه اندازی و کار با LED ها به آن نیاز داریم.

RGB LED FEATURE



<http://www.codeproject.com/Articles/750784/Writing-your-first-program-for-the-Freescale-Freed>

این بورد حاوی ۲ میکروکنترلر هست که یکی برای رابط کاربری پروگرم OPEN SDA و دیگری میکروکنترلری است که برای برنامه های کاربردی مورد استفاده قرار میگرید . میکرو ای که برای برنامه های کاربردی مورد استفاده قرار گرفته است یک میکرو از خانواده سری Kinetis L به نام MKL25Z128 میباشد .

در این بورد ۲ عدد رابط usb mini وجود دارد که یکی برای برنامه ریزی و خطایابی با Open SDA میباشد و دیگری با نام k125z که به عنوان user usb در برنامه های کاربردی مورد استفاده قرار میگیرد .

پروگرم کردن و خطایابی بر عهده Open SDA میباشد که به عنوان یک دستگاه ذخیره سازی (mass storage device) نیز مورد استفاده قرار میگردد و در هنگام متصل کردن این قسمت به کامپیوتر یک درایور دیسک مشاهده می شود که میتوان با کپی فایل باینزی برنامه با فرمت srec بورد را برنامه ریزی کرد .

اجرا شده در میکروکنترل open SDA یک برنامه به نام **OpenSDA application** است . انواع برنامه Firmware وجود دارد از جمله برنامه های فلش کردن خطایابی و برنامه ریزی کد در فلش و دیگری برای MSD که همه های این برنامه های کاربردی در یک برنامه open SDA ذخیره شده است . اگر میخواهید برنامه های Open SDA را به روز رسانی کنید یا تغییر بدهید باید به مود bootloader بروید: کلید ریست را هنگام اتصال به کامپیوتر پایین نگه دارید . یک درایور bootloader در سیستم ایجاد خواهد شد که برنامه های open SDA را در آن کپی کنید . به این ترتیب firmware میکرو را میتوان به روز رسانی یا تغییر داد .

Bootloader چیست ؟

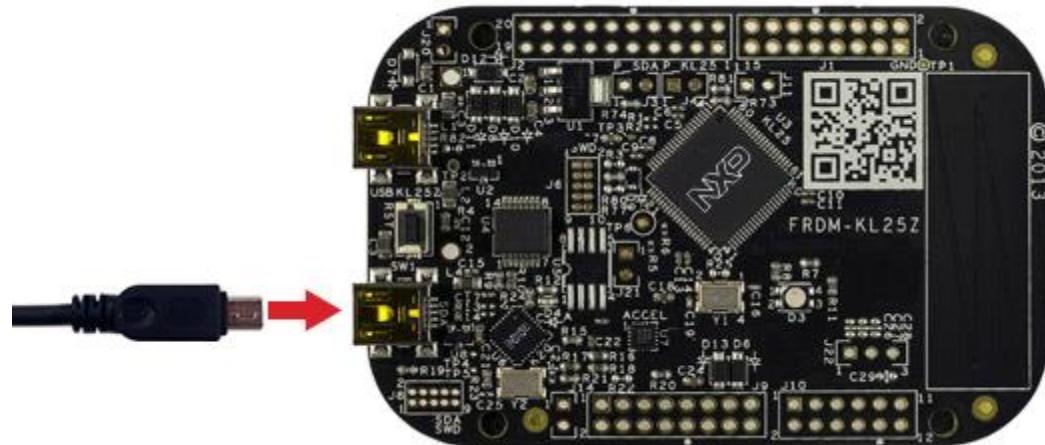
بوت لودر روشی برای پروگرام کردن تراشه های میکروکنترلر است که توسط خود میکروکنترلر اقدام به برنامه ریزی خودش می شود . این روش از پروگرام کردن تراشه مخصوص آن دسته از میکروکنترلرها می باشد که قابلیت نوشتن در حافظه برنامه خود را دارند . در روش بوت لودر ، در ابتدا یک برنامه با حجم پائین در میکروکنترلر پروگرام می شود . وظیفه این برنامه برقراری ارتباط با کامپیوتر و اجرای دستورات دریافتی از آن (خواندن/نوشتن/پاک کردن و ..) می باشد . پس از اینکه برنامه بوت لودر را بر روی میکروکنترلر پروگرام کردید ، پس از هر بار ریست شدن میکروکنترلر و یا قطع و وصل تغذیه ، ابتدا به برنامه بوت لودر پرش می شود تا اگر ارتباط با کامپیوتر برقرار بود ، به اجرای دستورات دریافتی پرداخته شود و اگر ارتباط برقرار نبود ، به برنامه اصلی میکروکنترلر پرش می شود (البته اگر قبل از برنامه اصلی هم پروگرام شده باشد) .

<http://www.ir-micro.com/modules.php?name=Forums&file=printview&t=353&start=0>

شروع :

<http://www.codeproject.com/Articles/750784/Writing-your-first-program-for-the-Freescale-Freed>

ابتدا usb mini را مطابق شکل متصل میکنیم .



نرم افزار های مورد استفاده:

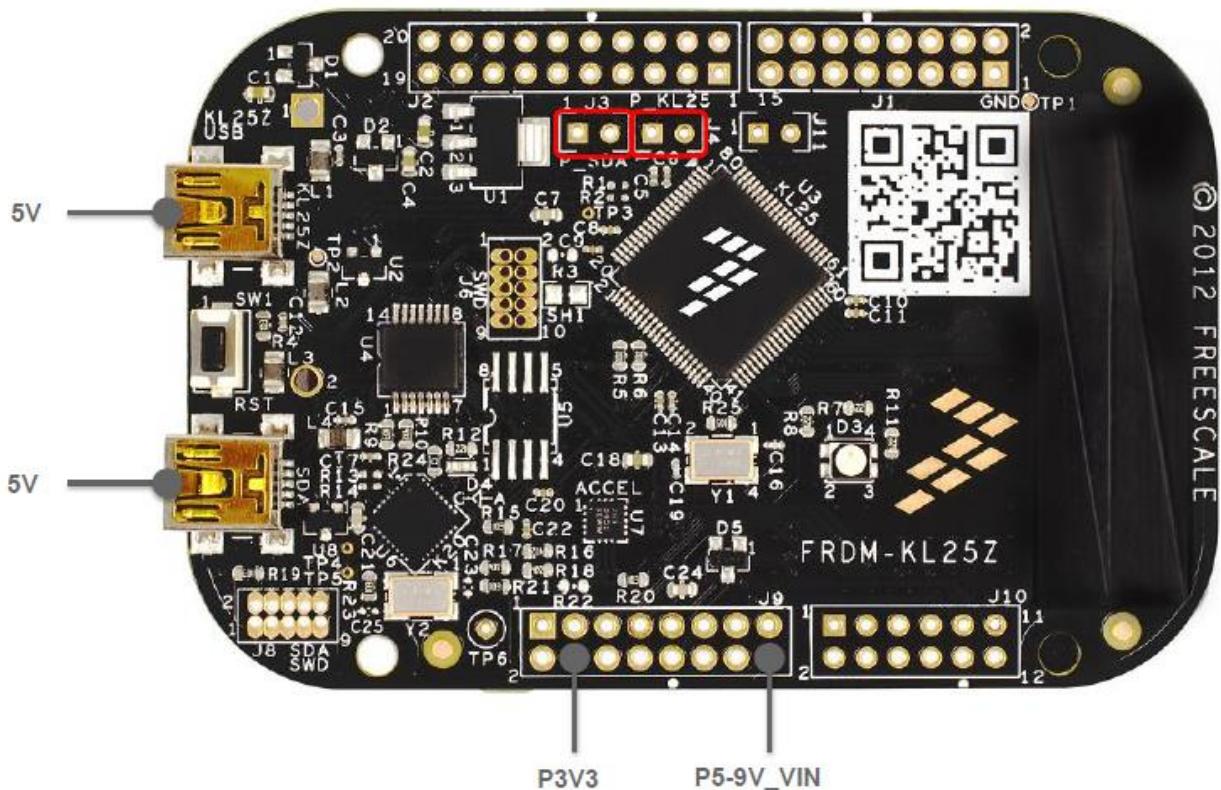
میتوان از [mbed](#) or [coocox](#) **استفاده کرد** ولی `cw` چیز دیگری است.

ولتاژ تغذیه:

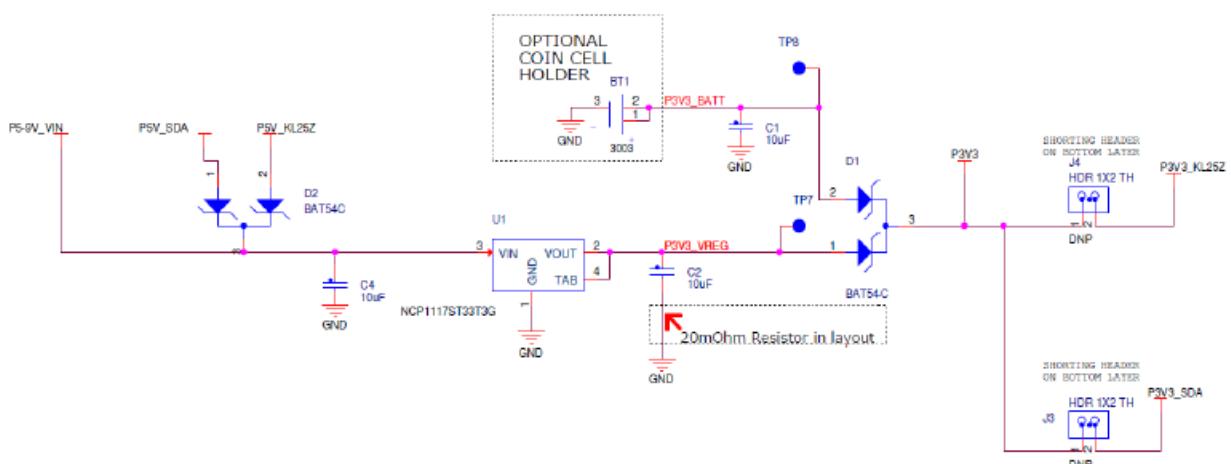
راهای مختلفی برای تامین ولتاژ تغذیه در این بورد وجود دارد که جدول زیر آنها را نشان میدهد :

Supply Source	Valid Range	OpenSDA Operational?	Regulated on-board?
OpenSDA USB (J7)	5V	Yes	Yes
KL25Z USB (J5)	5V	No	Yes
V _{IN} Pin	4.3-9V	No	Yes
3.3V Pin	1.71-3.6V	No	No
Coin Cell Battery	1.71-3.6V	No	No

منبع های **USB**, **Vin** در روی بورد بر سر راهشان یک رگولاتور خطی **3.3v** استفاده میکنند و دو منبع دیگر مستقیم به بورد وارد میشوند. در شکل راهای مختلف ورودی منبع تغذیه مشخص شده است :

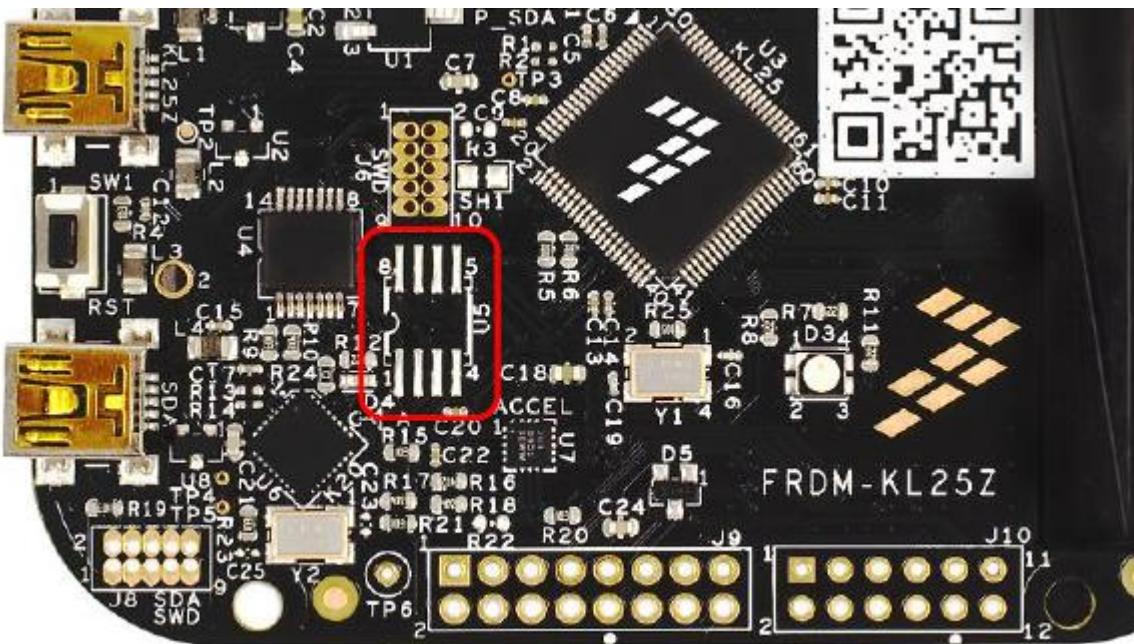


شکل زیر نمایش دهندهٔ شماتیک منابع ولتاژ ورودی است (رگولاتور در شکل مشخص میباشدند):

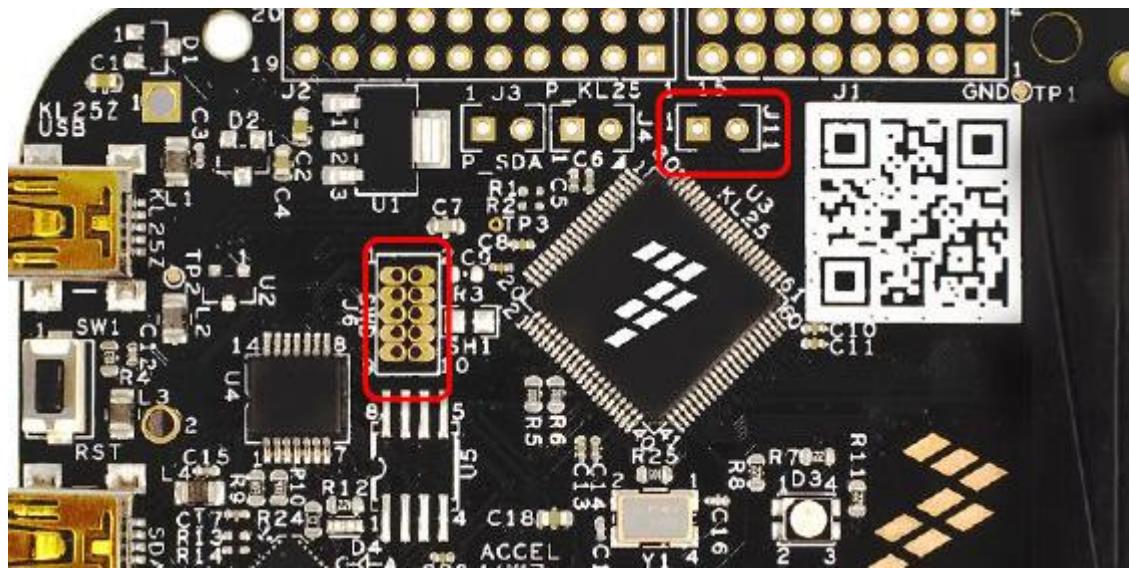


: **SPI**

در صورت استفاده از SPI flash memory این حافظه در قسمت زیر قرار میگیرد :



همچنین ازین بورد میتوان به عنوان **debugger** استفاده کرد که از پایه های زیر باید استفاده شود :

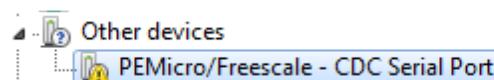


نصب درایور سریال :

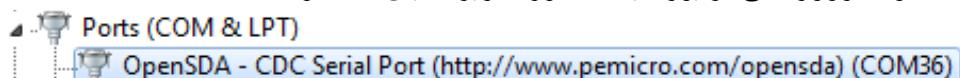
کابل USB را به ورودی **USB mini** **SDA** با برچسب متصل کنید



در تنظیمات سیستم به قسمت **Device Manager** بروید و پورت را در قسمت **other devices** پیدا کنید :



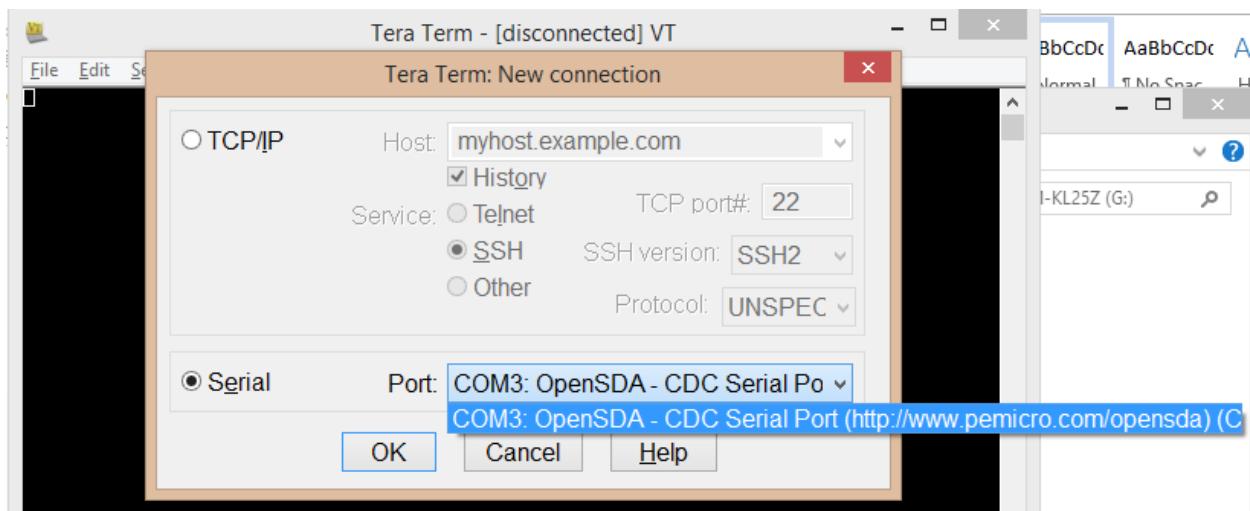
بر روی "PEMicro/Freescale – CDC Serial Port" کلیک راست کرده و نرم افزار درایور را به روزرسانی کنید. بعد از نصب و به روزرسانی درایور باید به صورت زیر نمایش داده شود :



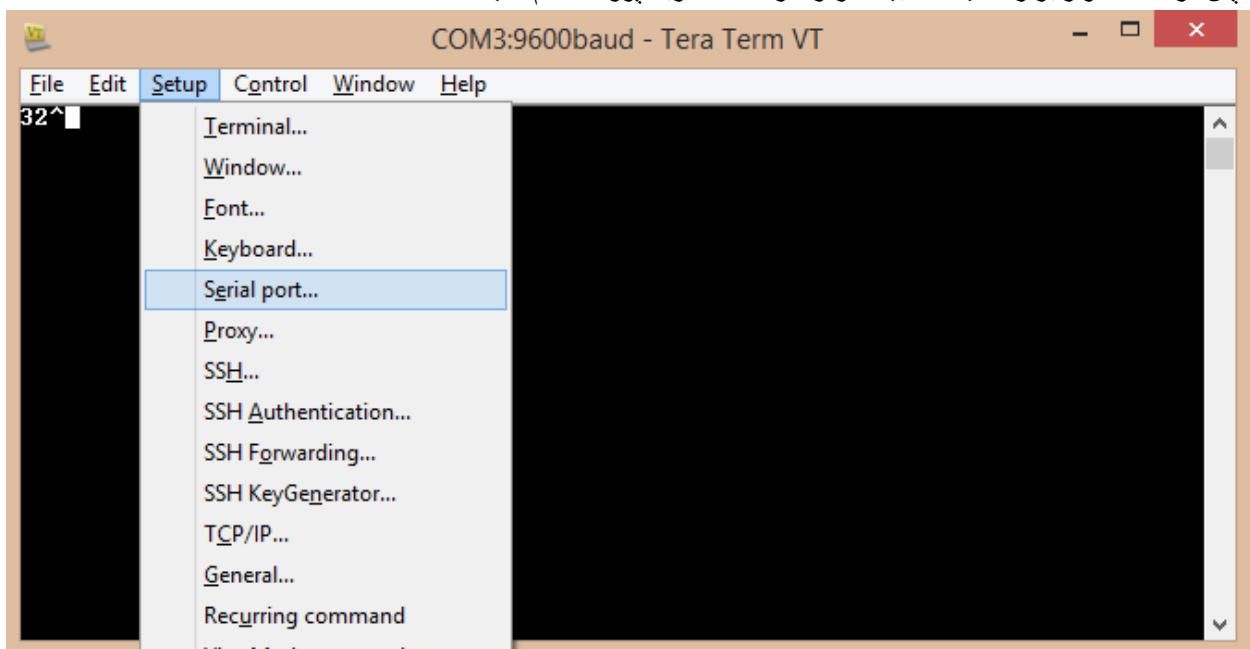
تست اتصال پورت سریال :

ابتدا باید یک شبیه ساز پورت سریال نصب کرده و با استفاده از آن با پورت سریال ارتباط برقرار کنید. چند شبیه ساز وجود دارد :

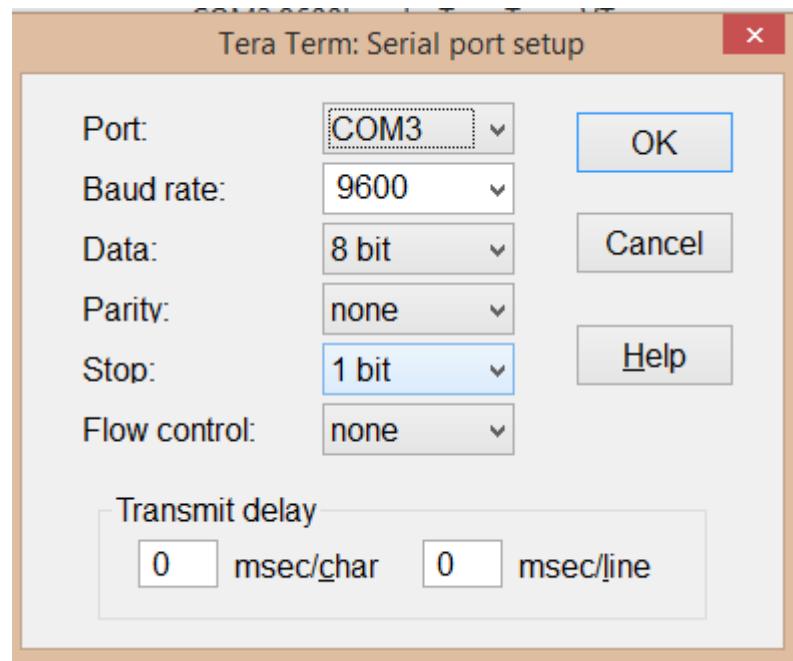
uCon - bit.ly/SvrmM3 و Terra Term - bit.ly/PGUjp2 و PuTTY - bit.ly/Scq1Gm که در این پروژه ما از tera term استفاده میکنیم بعد از باز کردن برنامه باید پورت سریال ورودی را انتخاب کنید :



سپس در قسمت نوار ابزار تنظیمات شبیه ساز را در قسمت سریال پورت انجام دهید:



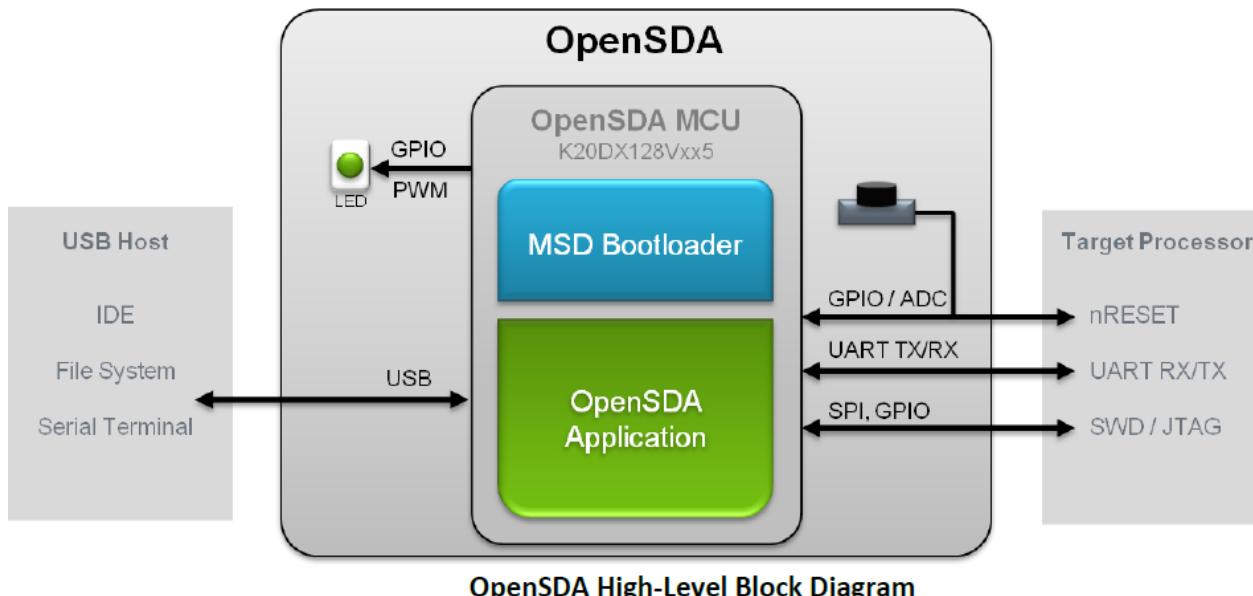
با توجه به ویژگی های پورت سریال تنظیمات را انجام دهید:



نکته‌ی قابل نکر در اینجا این است که در صورت استفاده از مازول‌های تبدیل USB به UART (TTL) ، در صورت نصب نشدن و یا عدم شناسایی مازول توسط سیستم احتمال اینکه این مازول نقابی باشد(در بازار) وجود دارد و شما باید از درایور ورژن ۲۰۰۸ استفاده کنید .

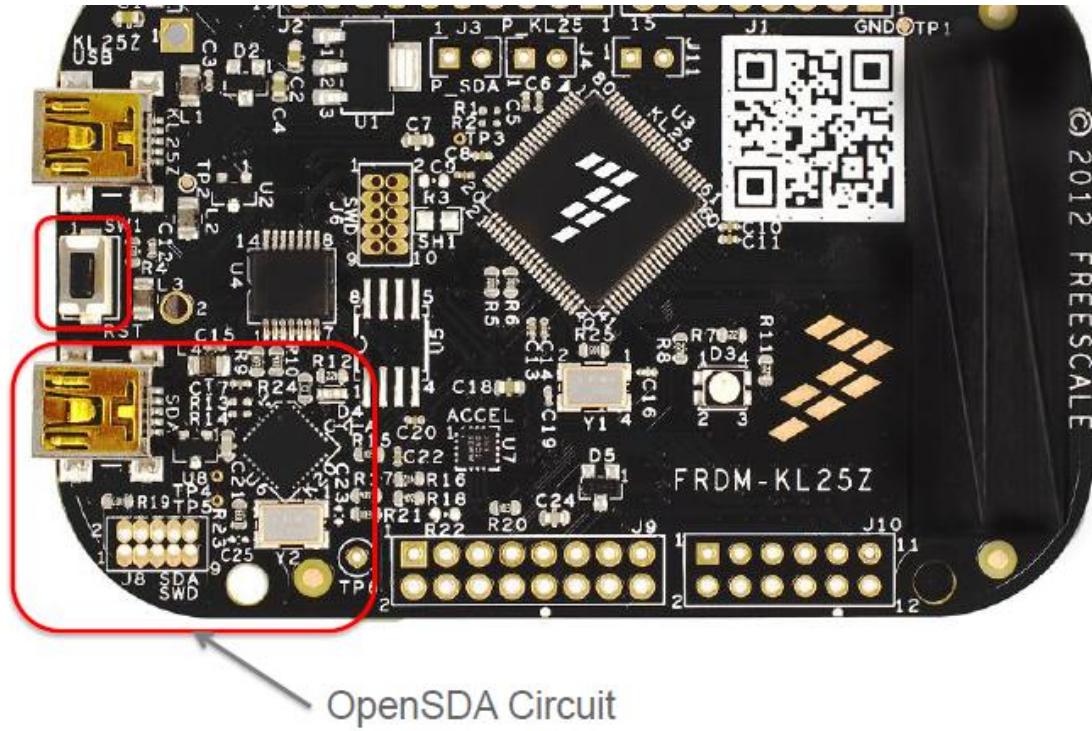
: Open SDA

Open SDA یک آدپتور(وقدنه) سریال و خطیابی است. این آدپتور به عنوان یک پل ارتباط سریال و اشکال یابی را بین یک هاست و یک پردازشگر جاسازشده (embedded processor) برقرار میکند. این مازول دارای یک LED سبز رنگ و یک کلید ریست در بورد میباشد که هنگام فعال شدن استفاده از مازول این LED روشن و از دکمه‌ی ریست برای ریست مدار استفاده میشود. بلوك دیاگرام openSDA به صورت زیر میباشد:



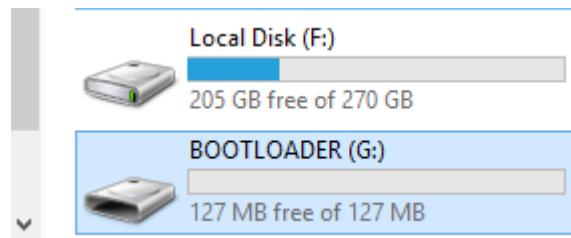
OpenSDA High-Level Block Diagram

محل قرار گیری در بورد :

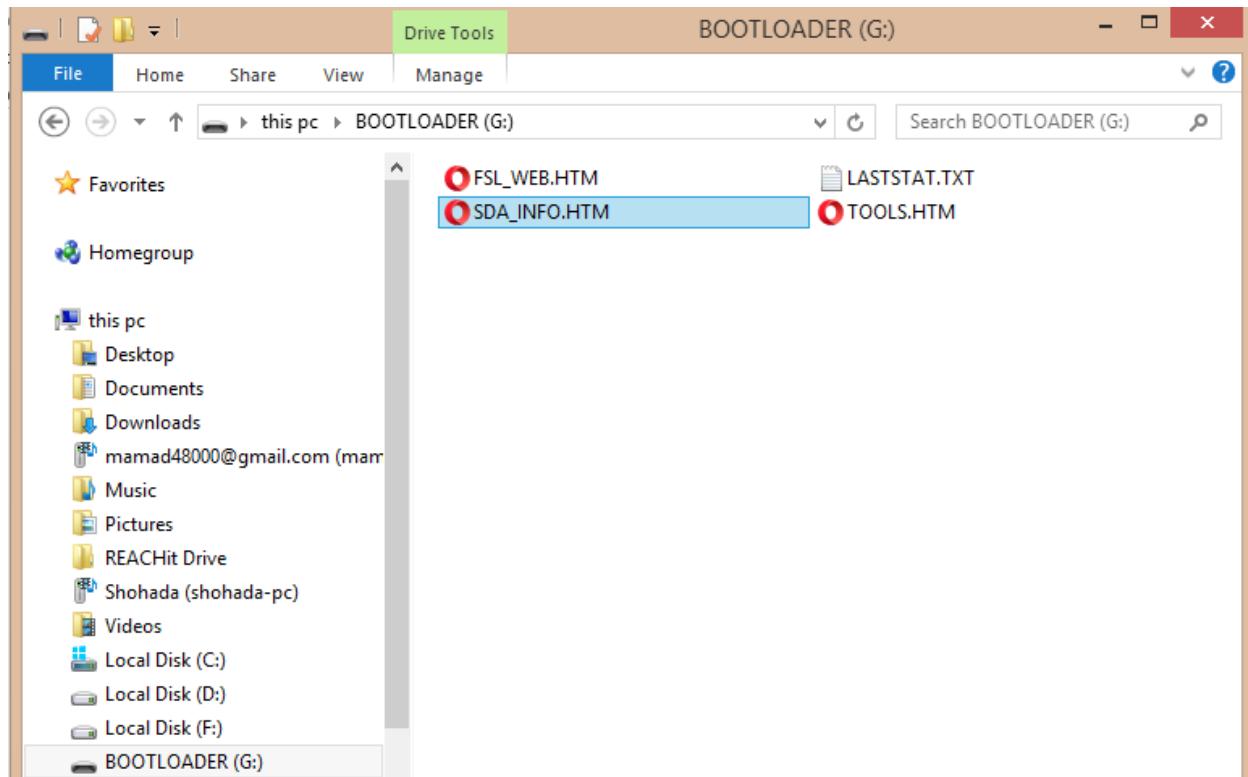


از کلید ریست برای تغییر مود openSDA به مد بوت لودر هم استفاده میشود در مود بوت لودر میتوان نرم افزار openSDA را به روز رسانی کرد .

به این صورت که قبل از اتصال USB به سیستم کلید ریست را نگهدارید و بعد از اتصال USB آنرا رها کنید در این صورت LED مربوط به openSDA به صورت چشمک زن میباشد و درایوری با نام BOOTLOADER ایجاد میشود که به صورت زیر است .

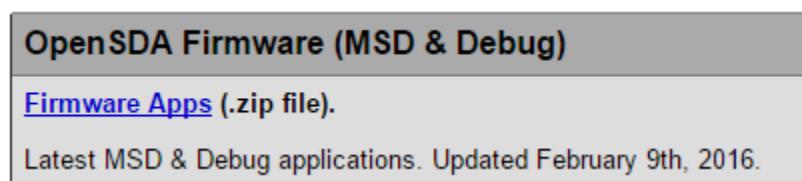


و محتوای داخل آن به صورت زیر است .

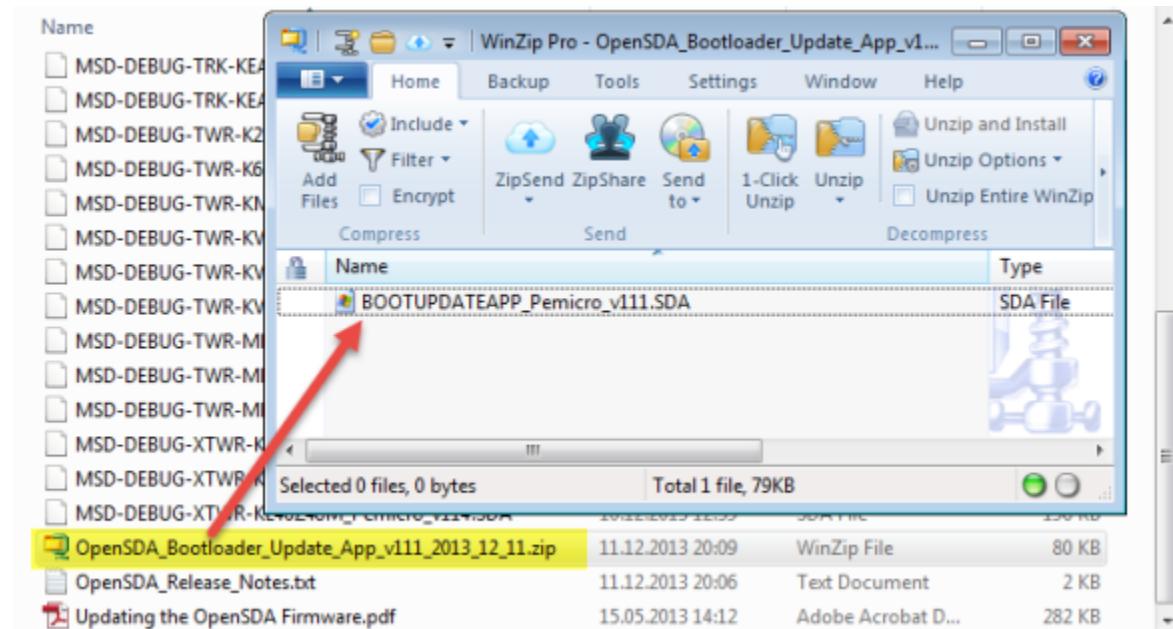


یتوانید فایل های نرم افزار openSDA را برای به روزرسانی از آدرس <https://www.pemicro.com/opensda/> دانلود کنید .

نرم افزاری که در این پروژه دانلود و استفاده شد میباشد . **OpenSDA_Bootloader_Update_App_v111_2013_12_11.zip**



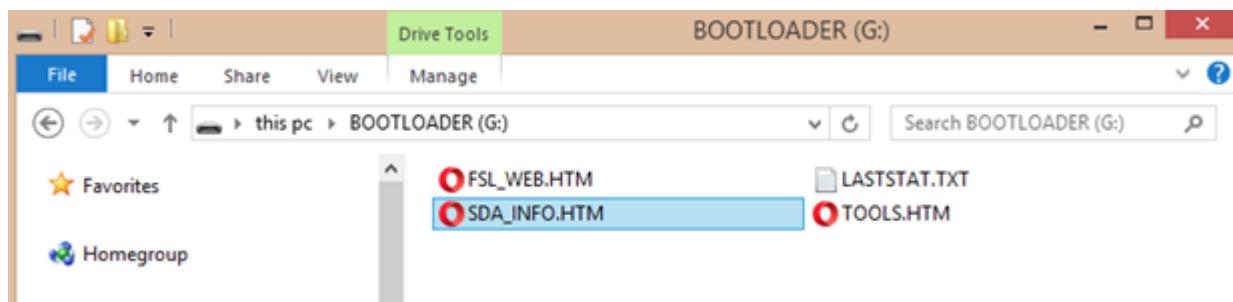
بعد از دانلود و باز کردن فایل زیپ فایل زیر را مشاهده میکنید .



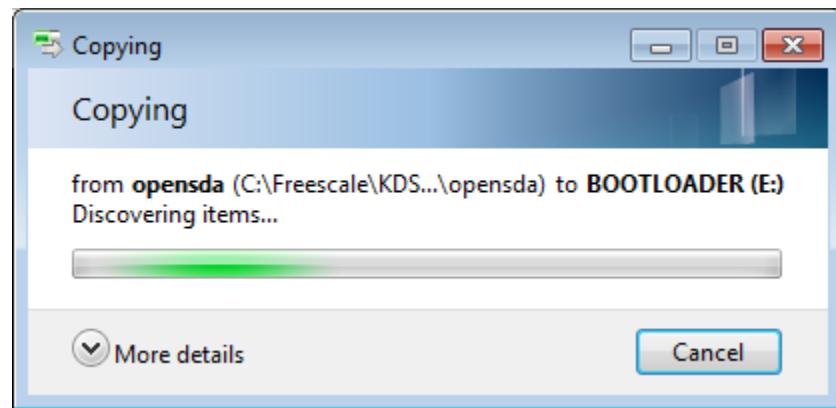
به روزرسانی : BOOTLOADER

قبل از اتصال USB کلید ریست را نگهداشته و در این هنگام USB را متصل کنید (بعد از اتصال LED شروع به چشمک زدن میکند)

درایور bootloader به صورت زیر نشان داده میشود :



فایل **BOOTUPDATEAPP_Pemicro_v111.SDA** که اనلود کرده بودید را به این درایور کپی کنید



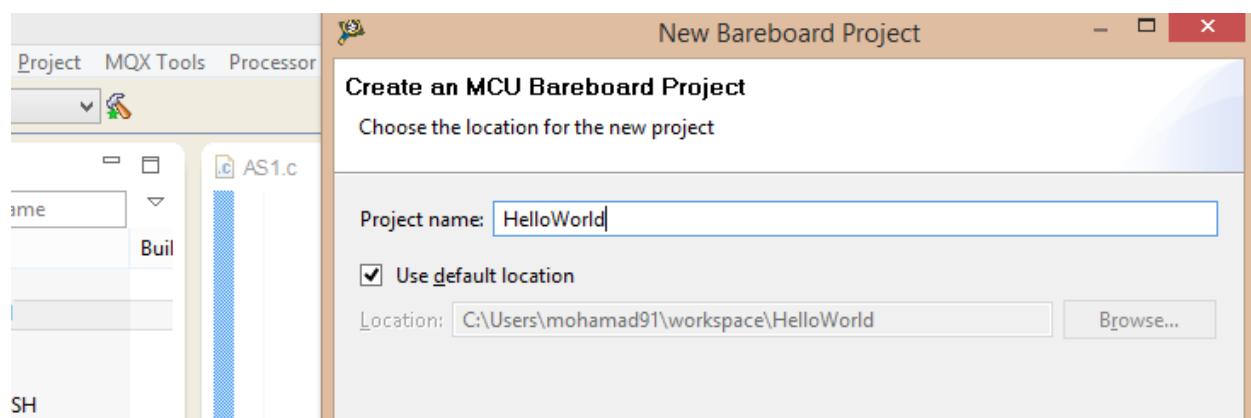
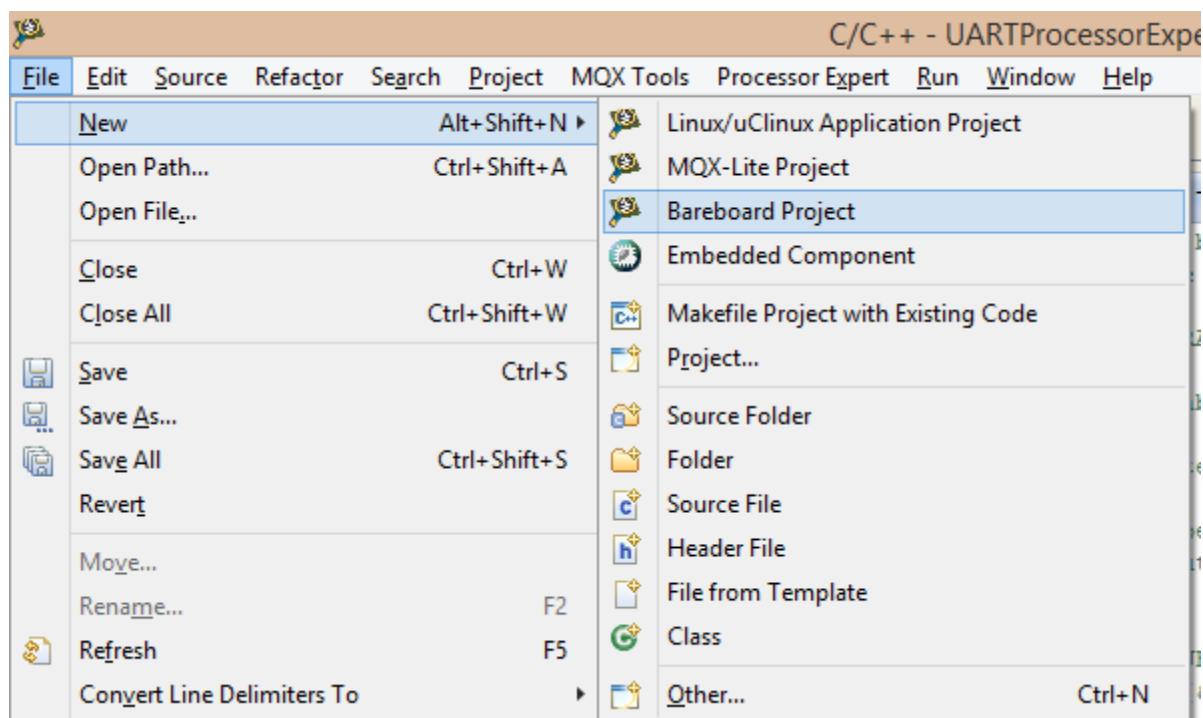
منبع ولتاژ یا همان کابل USB را قطع کنید و سپس متصل کنید bootloader به روزرسانی شد.

نکته‌ی قابل ذکر دیگر این است که openSDA firmware را نیز میتوان به روزرسانی کرد که در این سند به آن اشاره نشده است.

: Hello World

در این قسمت فعال کردن کلاک و کار با ال ای دی را نشان میدهیم . این پروژه یک پروژه‌ی بسیار خوب برای شروع به کار با بورد FRDM-KL25Z میباشد .

ابتدا یک پروژه‌ی bareboard Project بسازید (به انتخاب بورد دقت کنید):



Devices

Select the derivative or board you would like to use

Device or board to be used:

type filter text

- ▷ 56800/E (DSC)
- ▷ ColdFire V1
- ▷ ColdFire V2
- ▷ ColdFire V3
- ▷ ColdFire V4
- ▷ ColdFire V4e
- ▷ ColdFire+
- ▷ Kinetis E Series
- ▷ Kinetis K Series
- ▲ Kinetis L Series
 - ▷ KL0x Family
 - ▷ KL1x Family
 - ▲ KL2x Family
 - ▷ KL24Z (48 MHz) Family
 - ▲ KL25Z (48 MHz) Family
 - MKL25Z32
 - MKL25Z64
 - MKL25Z128**
 - ▷ KL26Z (48 MHz) Family

Project Type / Output:

Application

Library

Creates project for MKL25Z128 (48 Mhz) derivative

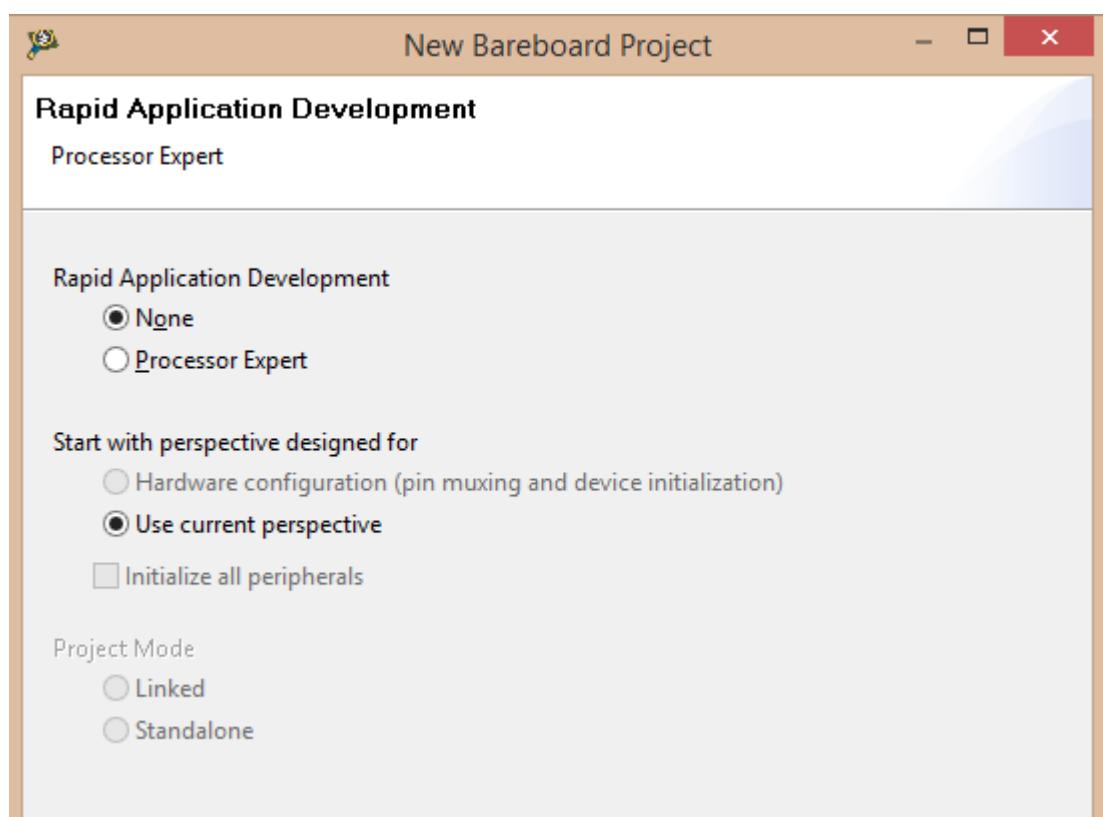
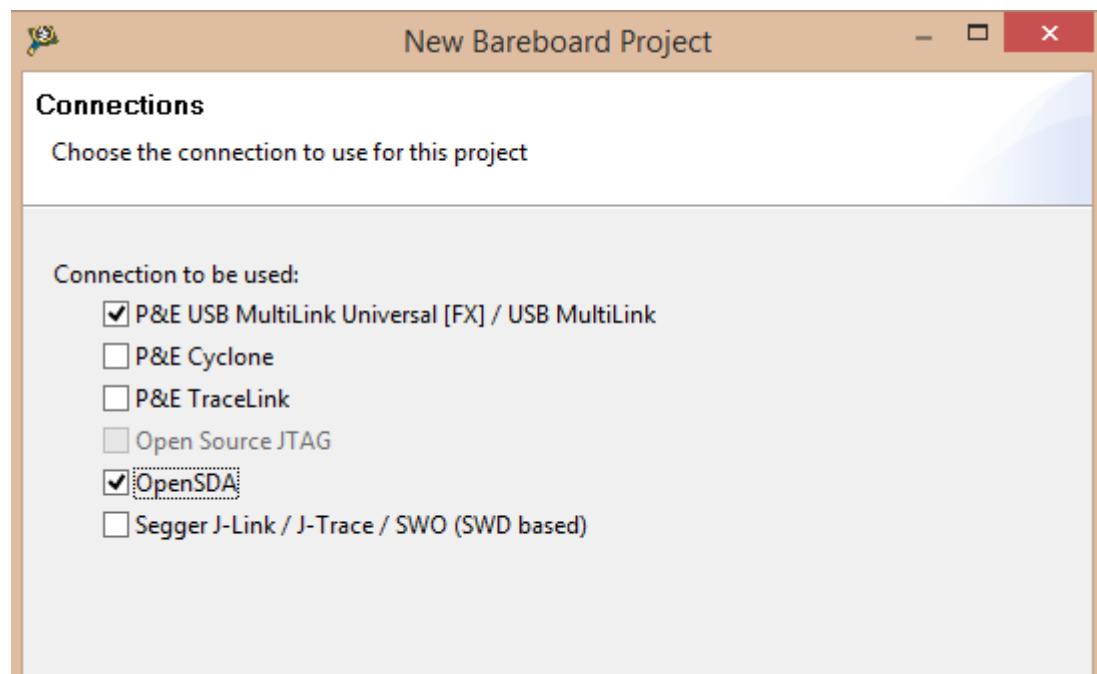


< Back

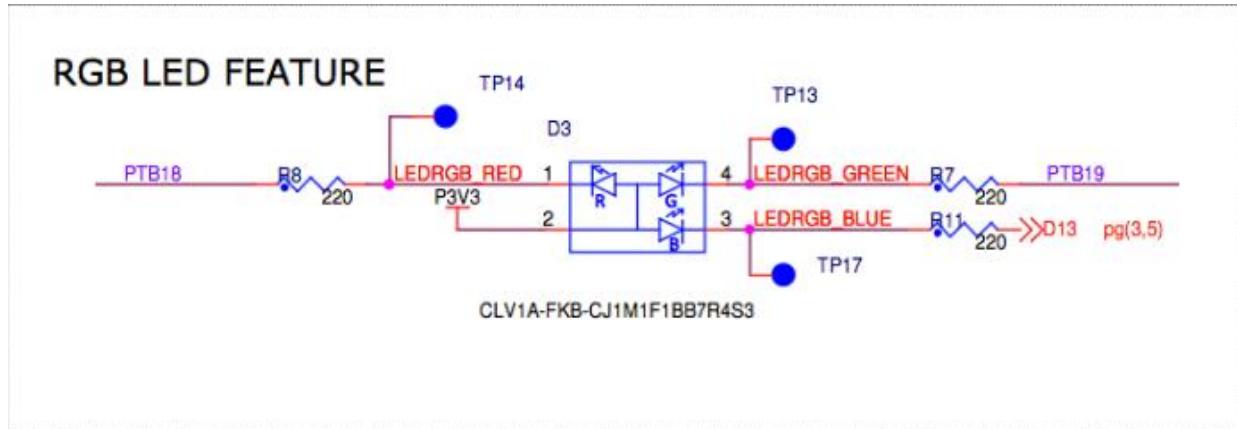
Next >

Finish

Cancel



حال باید با استفاده از دینا شیت ال ای دی های این بورد را پیدا کنیم : به این صورت که ابتدا به شماتیک به منظور فهمیدن روش اتصالات توجه میکنیم :



این شماتیک به ما نشان میدهد که ال ای دی قرمز رنگ به PTB18 متصل است پس ما برای روشن کردن باید با پایه ی PTB18 کار کنیم :

ابتدا باید کلک سیستم را فعال کنیم . در ابتدا تمام سوییچ های کلک بسته میباشد و کلکی به سیستم وارد نمیشود .

`;SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK .`

بوردهای KL25Z دارای مازولی به نام SIM یا System Integration Module میباشند که کارهای مختلفی انجام میدهد که یکی از کارهایش وصل کردن clock به پورت هاست . در ابتدا سوییچ ها بسته میباشند که از SIM برای اتصال کلک به پورت ها استفاده میشود .

حال باید فهمید چگونه با رجیستر های این مازول میتوان این مازول را راه اندازی کرد :

System Clock Gating Control Register 5 (SIM_SCGC5)

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
R	0	0	PORTE	PORTD	PORTC	PORTB	PORTA	1	0	0	TSI	0	0	0	0	LPTMR
W			0	0	0	0	0	1	1	0	0	0	0	0	0	0
Reset	0	0														

همانطور که از توضیحات جدول زیر قابل برداشت است با بیت مشخص شده در بالا میتوان کلک پورت B را کنترل کرد . پس قبل از روشن و خاموش کردن **LED** باید کلک پورت B را فعال کرد .

18-14 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
13 PORTE	Port E Clock Gate Control This bit controls the clock gate to the Port E module. 0 Clock disabled 1 Clock enabled
12 PORTD	Port D Clock Gate Control This bit controls the clock gate to the Port D module. 0 Clock disabled 1 Clock enabled
11 PORTC	Port C Clock Gate Control This bit controls the clock gate to the Port C module. 0 Clock disabled 1 Clock enabled
10 PORTB	Port B Clock Gate Control This bit controls the clock gate to the Port B module.

در سری MKL25Z4.h برای راحتی کار ایجاد شده است که در زیر مشاهده میکنید و با استفاده از این کار با این رجیستر راحت شده است. پس با قطعه کد `SIM_SCGC5 |= SIM_SCGC5_PORTB_MASK;` میتوان کلک را به پورت B فعال کرد. توجه شود که `=` به معنای OR بیتی است.

```

C/C++ - helloworld/Project_Headers/MKL25Z4.h - CodeWarrior
File MQX Tools Processor Expert Run Window Help
main.c MKL25Z4.h
#define SIM_SCGC4_SPI1_MASK 0x80000000
#define SIM_SCGC4_SPI1_SHIFT 23
/* SCGC5 Bit Fields */
#define SIM_SCGC5_LPTMR_MASK 0x1u
#define SIM_SCGC5_LPTMR_SHIFT 0
#define SIM_SCGC5_TSI_MASK 0x20u
#define SIM_SCGC5_TSI_SHIFT 5
#define SIM_SCGC5_PORTA_MASK 0x200u
#define SIM_SCGC5_PORTA_SHIFT 9
#define SIM_SCGC5_PORTB_MASK 0x400u
#define SIM_SCGC5_PORTB_SHIFT 10
#define SIM_SCGC5_PORTC_MASK 0x800u
#define SIM_SCGC5_PORTC_SHIFT 11
#define SIM_SCGC5_PORTD_MASK 0x1000u
#define SIM_SCGC5_PORTD_SHIFT 12
#define SIM_SCGC5 PORTE_MASK 0x2000u

```

حال بعد از فعال کردن پورت **B** با کلاک باید تنظیمات ورودی خروجی پایه‌ی مورد نظر را انجام دهیم. پایه‌ی ای که در پورت **B** برای کار با **LED** قرمز استفاده می‌شود پایه‌ی **PTB18** می‌باشد که باید به صورت **GPIO** تعریف شود. هر پایه‌ی در این بورد کاربردهای زیادی می‌تواند داشته باشد و باید قبل از شروع به کار تنظیم کنیم کدام عمل را می‌خواهیم انجام دهد که به این کار **multiplexing** می‌گویند. در اینجا نیز باید تنظیم کنیم که این پایه‌ی به صورت یک پایه‌ی ورودی خروجی معمولی باشد که **LED** را روشن خاموش می‌کند.

10-8 MUX	Pin Mux Control Not all pins support all pin muxing slots. Unimplemented pin muxing slots are reserved and may result in configuring the pin for a different pin muxing slot. The corresponding pin is configured in the following pin muxing slot as follows: 000 Pin disabled (analog). 001 Alternative 1 (GPIO). 010 Alternative 2 (chip-specific). 011 Alternative 3 (chip-specific). 100 Alternative 4 (chip-specific). 101 Alternative 5 (chip-specific). 110 Alternative 6 (chip-specific). 111 Alternative 7 (chip-specific).
-----------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

حال باشد که می‌خواهیم این پورت ورودی باشد یا خروجی؟ و برای روشن و خاموش کردن **LED** این پایه‌ی باید خروجی باشد. همانطور که در شکل از دیتاشیت مشخص است در ابتدا همه‌ی پایه‌ها به عنوان ورودی انتخاب شده‌اند. با خط کد **PORTB_PDDR18=(1<<18)** می‌توان پایه‌ی ۱۸ پورت **B** را خروجی تعریف کرد.

41.2.6 Port Data Direction Register (GPIOx_PDDR)

The PDDR configures the individual port pins for input or output.

Address: Base address + 14h offset

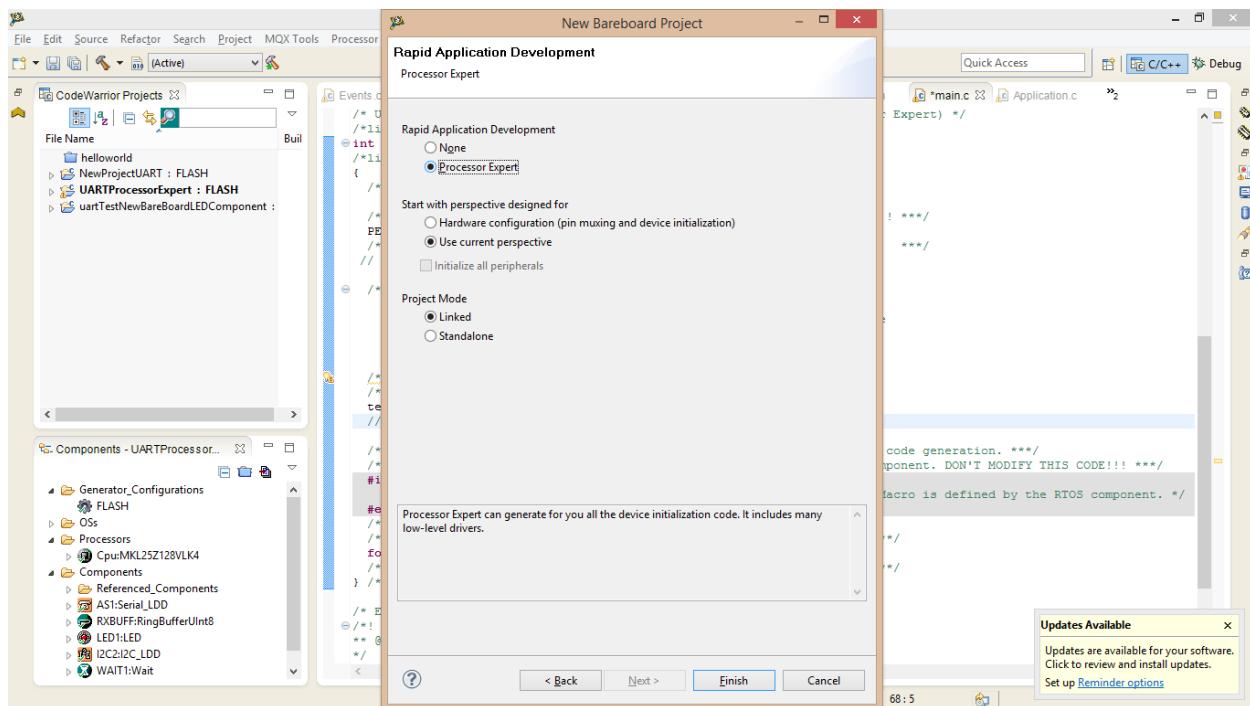
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
R																	PDD																
W																																	

GPIOx_PDDR field descriptions

Field	Description
31-0 PDD	Port Data Direction Configures individual port pins for input or output. 0 Pin is configured as general-purpose input, for the GPIO function. 1 Pin is configured as general-purpose output, for the GPIO function.

حال چگونه این پایه را روشن خاموش کنیم: GPIOB_PTOR=(1<<18)

نرم افزار **CodeWarrior** برای راحتی کار با این بورد روش دیگری برای راه اندازی و پیاده سازی ماثول های مختلف بورد قرار داده است که به آن **Processor Expert** گفته میشود . برای استفاده از این روش باید هنگام ایجاد پروژه تعیین کنید که میخواهید از این روش استفاده کنید همانطور که در شکل زیر نشان داده شده است داریم :



<https://mcuoneclipse.com/2012/09/07/tutorial-enlighting-the-freedom-kl25z-board>

: UART

ابتدا توضیحی درباره ی پورت سریال و ارتباط سریال داده میشود :

http://www.aftabir.com/articles/view/computer_internet_infortmation_tec hnology/internet_network/c14c1221470854_port_p1.php%D8%AA%D8%B4%D8%B1%D8%8C%D8%AD-%DA%A9%D8%A7%D9%85%D9%84-%D9%BE%D9%88%D8%B1%D8%AA%D9%87%D8%A7%D8%8C-%D8%B3%D8%B1%DB%8C%D8%A7%D9%84-%D9%88-%D9%85%D9%88%D8%A7%D8%B2%DB%8C

● پورت سریال

مبادله‌ی بیت به بیت اطلاعات تنها از طریق یک کانال (البته بجز زمین) را ارتباط سریال می‌گویند، که پورتی به همین نام برای اتصال وسایل مانند: مودم، ماؤس، دسته‌ی بازی به کار می‌رود. پورت سریال یکی از متداول‌ترین روش‌های موجود جهت اتصال یک دستگاه به کامپیوتر است. با اینکه سیستمهای جدیتر سعی در استفاده محدود از پورت سریال را داشته و پورت **USB** را مورد توجه بیشتر قرار می‌دهند ولی همچنان دستگاه‌های متعددی نظیر مودم از پورت سریال استفاده می‌نمایند.

پورت‌های سریال **Communication (COM) port** نیز نامیده شده و بصورت دوطرفه می‌باشد. ویژگی فوق این امکان را برای هر دستگاه فراهم کرده تا قادر به ارسال و دریافت اطلاعات باشند. دستگاه‌های سریال از پین‌های متقاولت برای ارسال و دریافت داده استفاده می‌نمایند. استفاده از پین‌های یکسان باعث ارتباطات از نوع **half-dublex** خواهد شد و این بدان معنی است که اطلاعات قادر به حرکت صرفاً در یک جهت می‌باشند. با استفاده از پین‌های متقاولت امکان ارتباطات-**Full-duplex** فراهم شده و امکان حرکت اطلاعات در دو جهت فراهم خواهد گردید.

عملکرد صحیح پورت‌های سریال وابسته به یک کنترل کننده خاص با نام **Universal Asynchronous Receiver/Transmitter (UART)** است. تراشه فوق خروجی موازی گذرگاه سیستم کامپیوتر را اخذ و آن را بصورت سریال از طریق پورت سریال انقال خواهد داد. بمنظور افزایش سرعت، اغلب تراشه‌های **UART** دارای یک بافر با ظرفیت شانزده تا شصت و چهار کیلو بایت می‌باشند. بافر فوق امکان **Cache** نمودن داده‌های و اصله از گذرگاه سیستم را زمانیکه تراشه مشغول پردازش داده‌ها و ارسال آنها برای پورت سریال است را فراهم می‌نماید.

در صورت عدم آشنایی با **UART** میتوانید به صفحات وب زیر مراجعه کنید:

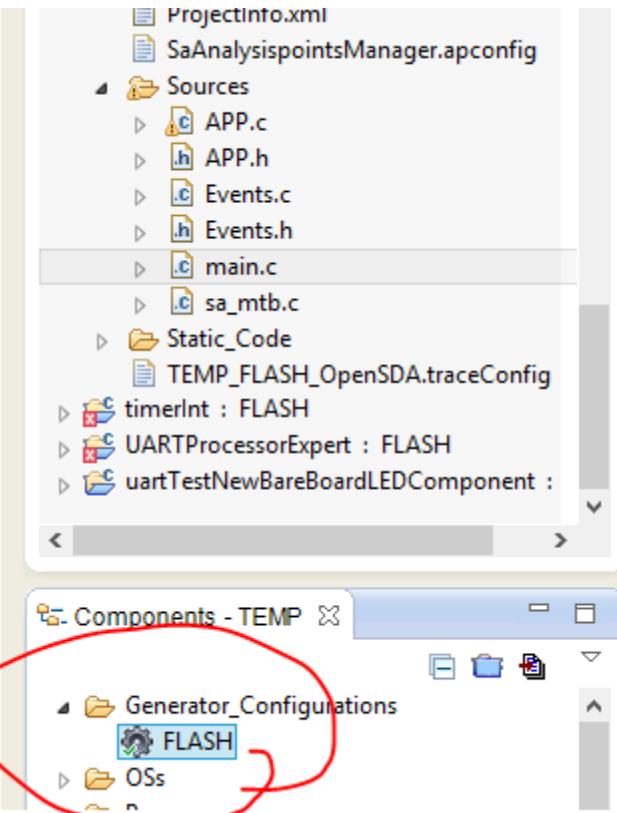
<https://learn.sparkfun.com/tutorials/serial-communication/uarts>

<http://electrovolt.ir/%D8%A8%D8%AE%D8%B4-%D9%86%D9%87%D9%85-%D8%A2%D9%85%D9%88%D8%B2%D8%B4-avr-%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D8%B1%D8%A7%D9%87-%D8%A7%D9%86%D8%AF%D8%A7%D8%B2%DB%8C-%D9%88%D8%A7%D8%AD%D8%AF%D9%87%D8%A7>

به ادامه‌ی مطالب میردادیم: هنگام استفاده از **openSDA** نمیتوان به راحتی از **uart0** استفاده کرد و دلیل آن هم این است که این پروتکل برای ارتباط سریال با بورد از پایه‌هایی استفاده می‌کند که **UART0** به عنوان پایه‌های ارسال و دریافت استفاده می‌کند و در نتیجه **send** از طرف ما به برد درست جواب نمیدهد ولی دریافت از پایه‌ی ارسال بورد به درستی کار می‌کند و برای استفاده از **UART** باید منبع ولتاژ را عوض کرده و با استفاده از پایه‌ی **Vin** که یکی از منبع‌های ولتاژ است تغذیه‌ی بورد را تامین کرد که پایه‌ی **Vin** همان پایه‌ی ۱۶ از سری **J9** می‌باشد.

: **UART** پیاده سازی

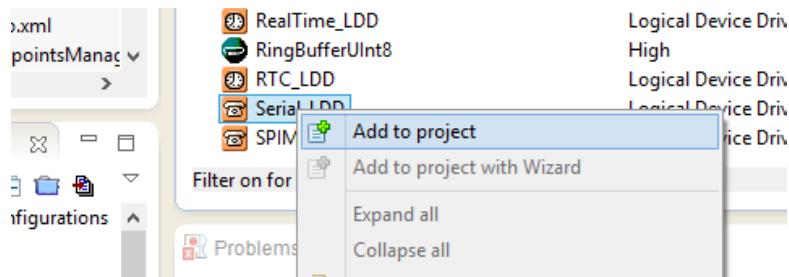
برای پیاده سازی ابتدا شما نیاز به اضافه کردن ۲ مولفه به مولفه‌های **Processor Expert** دارید. در ابتدا بر روی قسمت زیر که در شکل نشان داده شده است کلیک کنید.



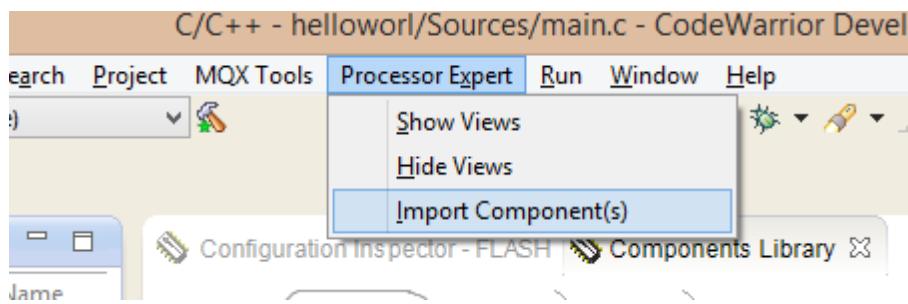
با کلیک بر روی این قسمت **processor expert** باز میشود که اجزای مختلفی که با **component lib** میتوان اجرا کرد را نشان میدهد.

Component	Component Level
ADC	High
ADC_LDD	Logical Device Driver
AnalogComp_LDD	Logical Device Driver
AsynchroSerial	High
BasicProperties	High
BitIO	High
BitIO_LDD	Logical Device Driver

در اینجا انواع مازول هارا از جمله **LED, I2C, ADC, ..** را میتوان مشاهده کرد . برای مازول ارتباط سریال یا همان **UART** باید مازول **Serial_LDD** را از این کتاب خانه اضافه کرد(مطابق شکل زیر داریم)



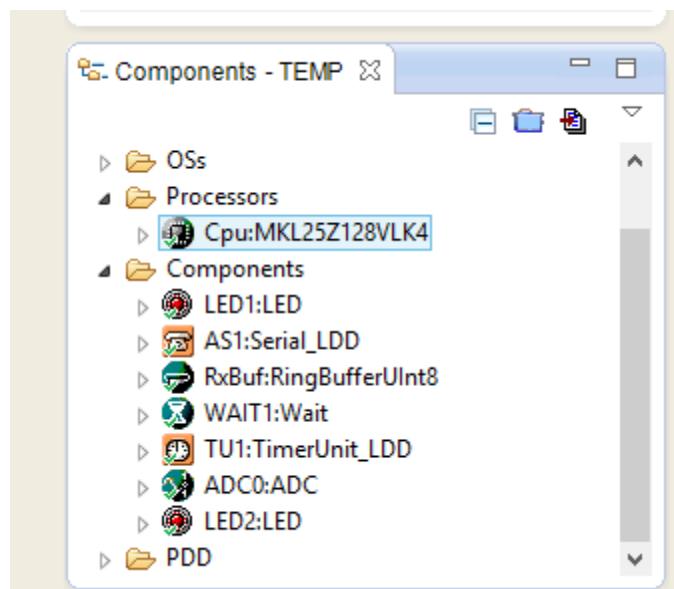
حال برای بافر کردن ورودی ها کامپوننت **RingBufferUInt8** را نیز باید به پروژه اضافه کرد که در کتابخانه وجود ندارد و باید آن را به کتاب خانه اضافه کنیم . در زیر اجزایی که در این پرژه مورد نیاز است و در کتاب خانه وجود ندارد را دانلود کرده و به کتابخانه **processor expert** اضافه میکنیم :



در پنجره **Processor Expert** باید فایل های مربوط به **component** هایی که میخواهید به پروژه اضافه کنید را انتخاب کنید . این فایل ها دارای فرمت **.Peupd*** . میباشد که باید آنها را از آدرس های زیر دانلود به پروژه اضافه کنید . تمام فایل هایی که در این پروژه به آنها نیاز داریم در زیر آورده شده است و شما باید ابتدا آین فایل ها را دانلود کنید (دقت کنید که در قسمت dependencies نیز هرچه هست باید دانلود و به پروژه اضافه کنید).

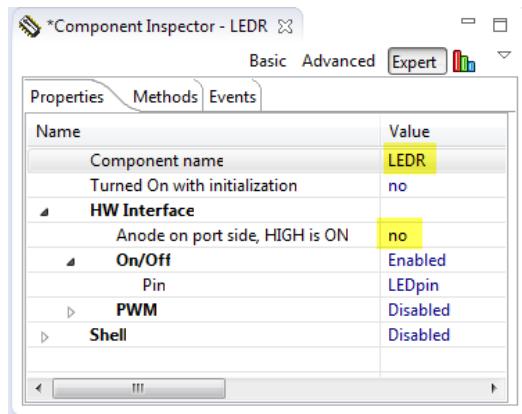
۱. ای دی [/ http://www.steinerberg.com/EmbeddedComponents/LED](http://www.steinerberg.com/EmbeddedComponents/LED):
۲. ورودی خروجی <http://www.steinerberg.com/EmbeddedComponents/GenericBitIO/home.htm>: / <http://www.steinerberg.com/EmbeddedComponents/Wait>: Wait.
۳. حافظه [ی بافر: http://www.steinerberg.com/EmbeddedComponents/RingBufferUInt8](http://www.steinerberg.com/EmbeddedComponents/RingBufferUInt8):

حال باز اضافه کردن به کتابخانه و مشاهده کامپوننت ها میتوان آنها را به پروژه اضافه کرد . بعد از اضافه کردن قسمت processor expert مطابق شکل زیر میشود :

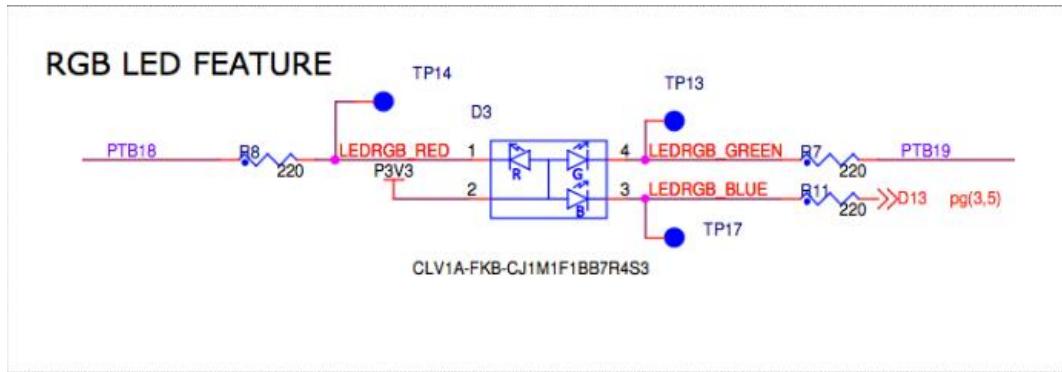


البته دقت کنید مازول هایی که باید در این مرحله اضافه کنید LED, Serial_LDD, RXBUF, WAIT, LED میباشد .
که ما قدم به قدم روش پیکربندی و تنظیمات هر کدام را نشان میدهیم .

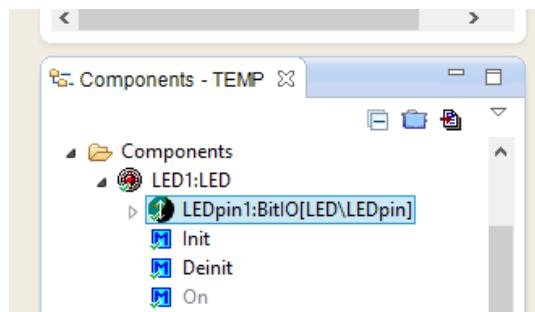
ابتدا تنظیمات ال ای دی را نشان میدهیم :
با دابل کلیک روی مازول اضافه شده در تصویر بالا پنجره‌ی تنظیمات هر کدام از مازول‌ها نمایش داده میشود که برای ال ای دی داریم :



برای تنظیمات همانطور که در شکل مشاهده میکنید ابتدا باید یک نام به مازول اختصاص بدھیم. در خط بعد میگوییم در ابتدا خاموش باشد و در خط سوم باید بگوییم وقتی پایه مقدار منطقی یک میشود روشن نشود و در واقع با صفر روشن بشود. در واقع با توجه به شماتیک بورد متوجه میشویم که کاتد به پایه ی میکروکنترلر (همان پورت PTB18) وصل است که برای روشن کردن ال ای دی باید پایه ی کاتد به GND وصل بشود. پس در این خط میگوییم مقدار یک باعث روشن شدن ال ای دی نشود که در واقع به معنی این است که سمت کاتد که به پایه ی پورت وصل است با مقدار GND روشن شود .



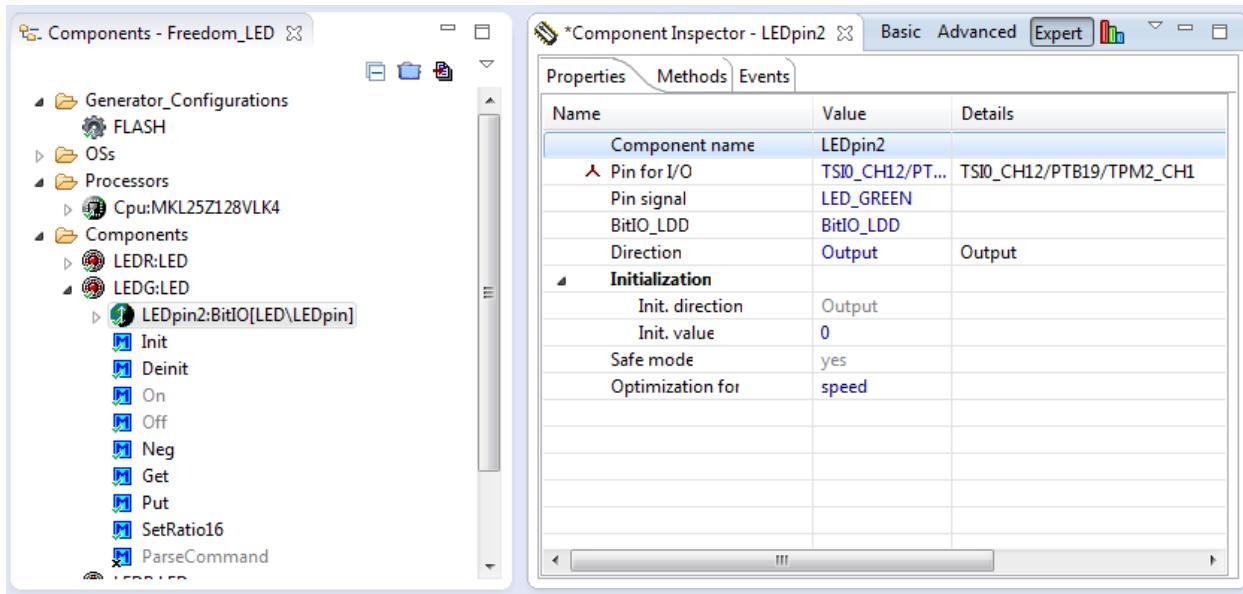
حال یک سری تنظیمات دیگر نیز دارد که با کلیک بر روی قسمت زیر باید صفحه‌ی تنظیمات را ایجاد کنیم :



Name	Value	Details
Component name	LEDpin1	
Pin for I/O	TSI0_CH11/PTB18/TPM2_CH0	
Pin signal	RED_LED	
BitIO_LDD	BitIO_LDD	
Direction	Output	Output
Initialization		
Init. direction	Output	
Init. value	0	
Safe mode	yes	
Optimization for	speed	

در خط اول تنظیمات باید بگوییم کدام پایه را میخوای به عنوان پایه ورودی خروجی استفاده کنیم که با توجه اینکه پایه‌ی PTB18 برای ال ای دی قرمز استفاده شده است آن را انتخاب میکنیم. و خط بعد که باید اسم این پایه را مشخص کنیم و خط بعد که به عنوان خروجی استفاده شده است .

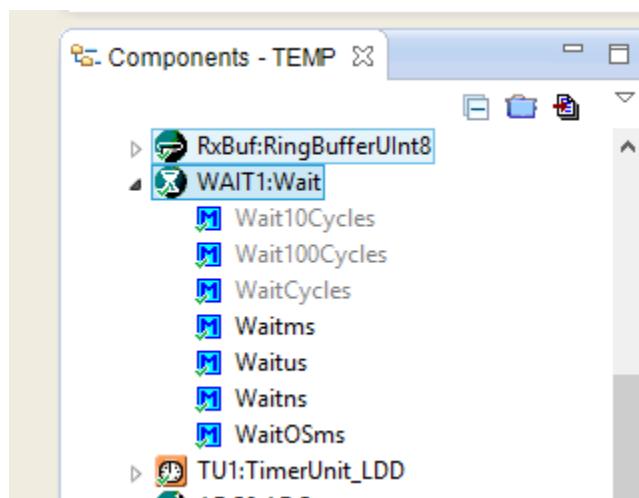
حال در صورت نیاز به ال ای دی های دیگر همین کار ها را دوباره انجام دهد و با توجه به ال ای که میخواهید فعال کنید پایه متصل به آن را انتخاب میکنید :



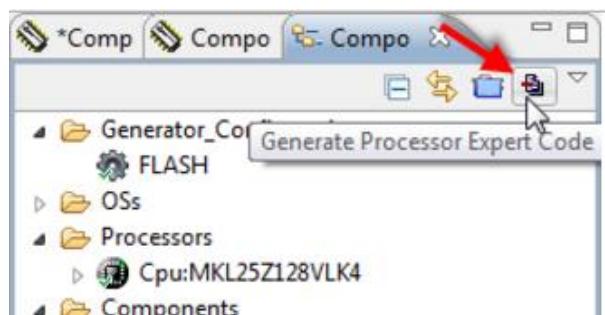
همانطور که در سمت چپ نشان داده شده است در توابعی که میتوان استفاده کرد نشان داده شده است . و ما از تابع Neg استفاده زیاد میکنیم که با هر بار صدا کردن این تابع ال ای دی تغییر حالت میدهد .

:Wait Component

این کامپوننت برای ایجاد وقفه استفاده میشود و تنظیمات خاصی ندارد و برای استفاده از توابع آن باید ایجاد شود(توابع در شکل زیر نشان داده شده است)



بعد از انجام تغییرات و تنظیمات باید به مژول processor expert بگوییم که این تنظیمات را در پروژه اعمال کند که با کلیک روی آیکن ایجاد کد انجام میشود :



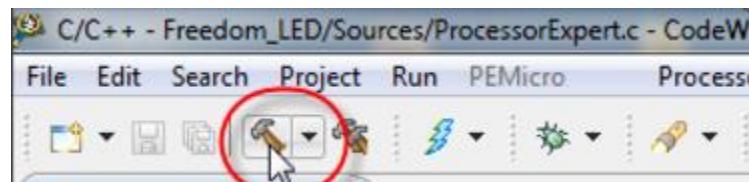
با قرار دادن تکه کد زیر در تابع main() میتوانید ال ای دی چشمک زن ایجاد کنید :

```

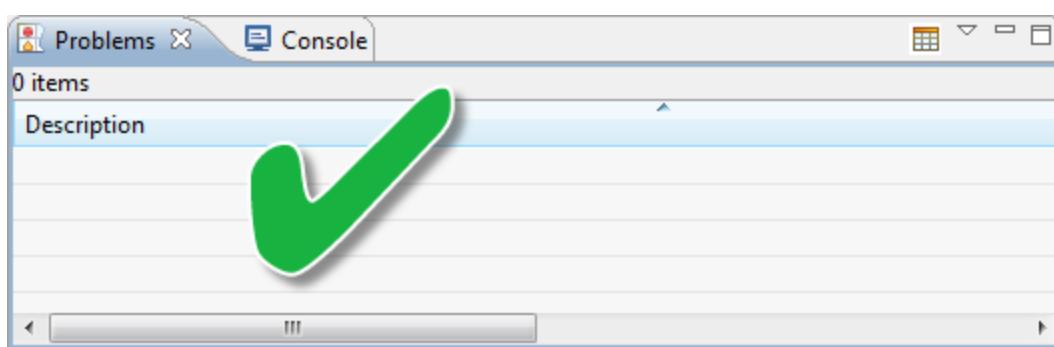
35
36 int main(void)
37 {
38     /* Write your local variable definition here */
39
40     /** Processor Expert internal initialisation. DON'T REMOVE THIS CODE!!! ***/
41     PE_low_level_init();
42     /** End of Processor Expert internal initialisation. ***/
43
44     /* Write your code here */
45     for(;;) {
46         WAIT1_Waitms(100);
47         LEDR_Neg();
48         WAIT1_Waitms(200);
49         LED2_Neg();
50         WAIT1_Waitms(400);
51         LED3_Neg();
52     }
53     /* For example: for(;;) { } */

```

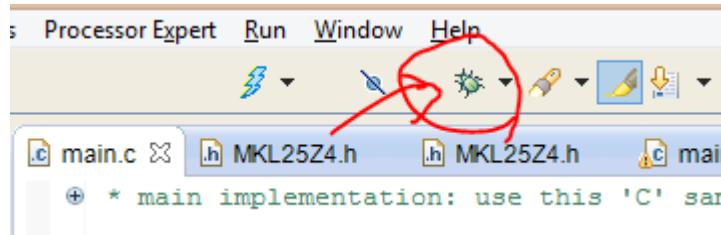
با کلیک روی آیکن زیر پروژه را بسازید :



و باید نتیجه‌ی زیر حاصل شود :



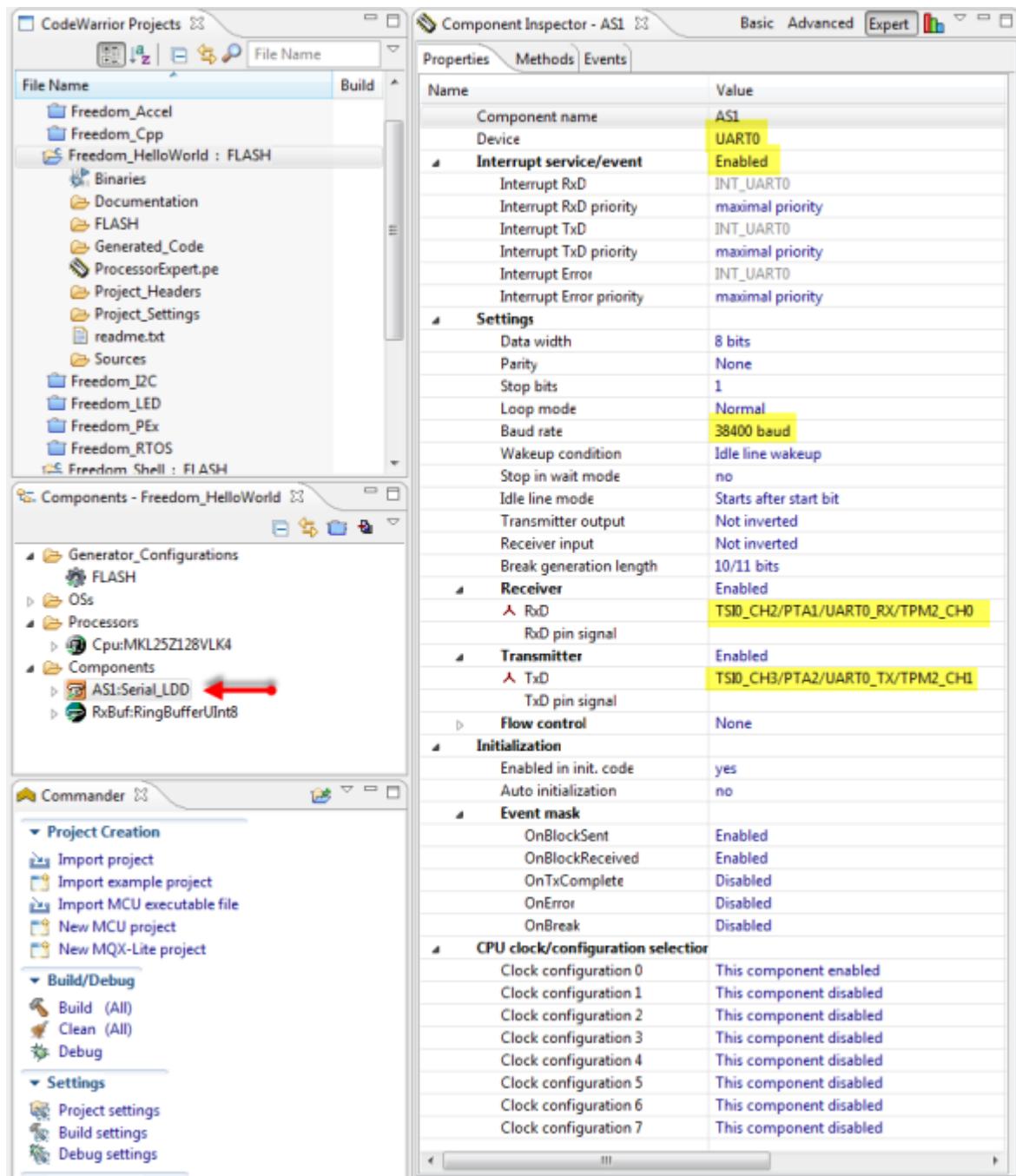
حال از متصل بودن کابل USB از سیستم به بورد اطمینان حاصل نمایید و با کلیک روی آیکن زیر میکرو را برنامه ریزی کنید



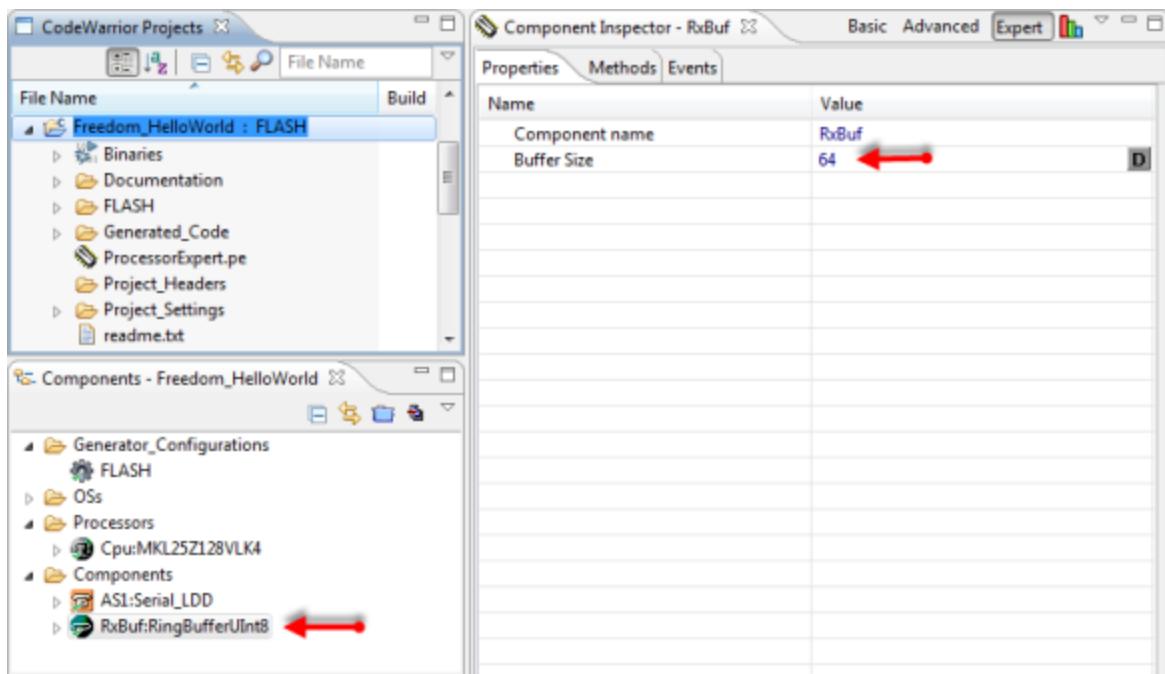
پیاده سازی UART :

حال باید مژول UART را به پروژه اضافه کرده و تنظیمات آن را انجام بدیم. در ابتدای این سند چند نکته درباره UART این بورد و رابطه‌ی آن با OpenSDA آورده شده است که در صورت نخواندن آن قسمت ابتدا آن قسمت را مطالعه کنید. به دلیل اینکه پایه‌های UART0 و openSDA مشترک هستند ما برای تبادل اطلاعات و تست مژول از openSDA استفاده می‌کنیم:

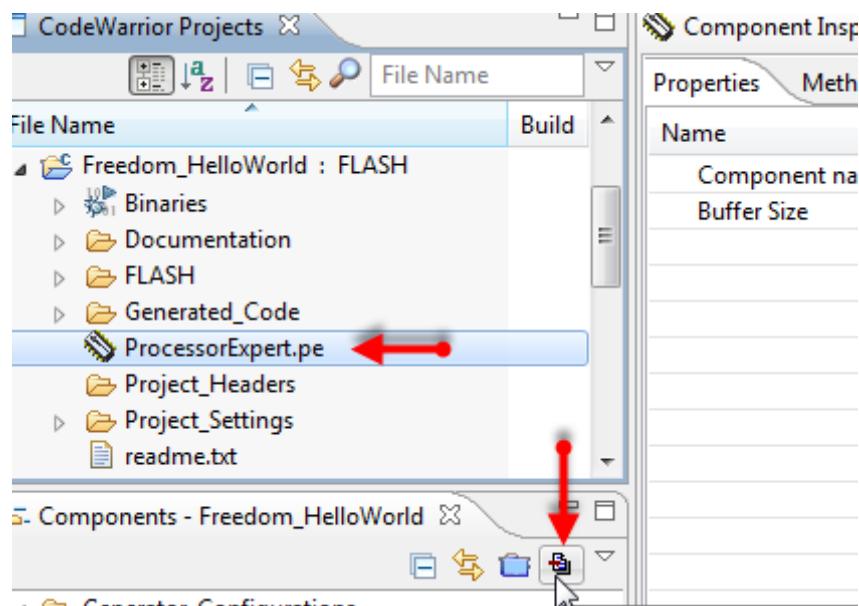
ابتدا مژول UART را به این پروژه اضافه کنید و تنظیمات را مطابق شکل زیر قرار دهید:



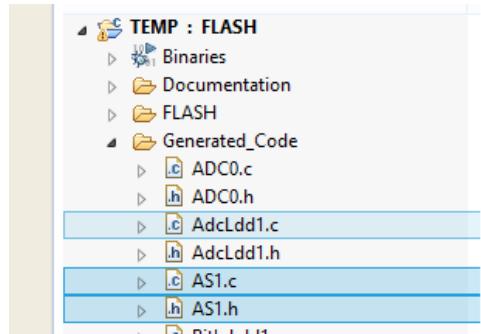
برای ارتباط UART باید یک مازول دیگر نیز بیاده کنید که آن حافظه بافر برای دریافت و ارسال داده هاست . همانطور که در اضافه کردن مازول به component lib انجام دادیم یکی از مازول ها بافر دریافت و ارسال اطلاعات بود که محل استفاده ای آن در UART است ،پس مطابق شکل یک بافر ایجاد کرده و قرار میدهیم :



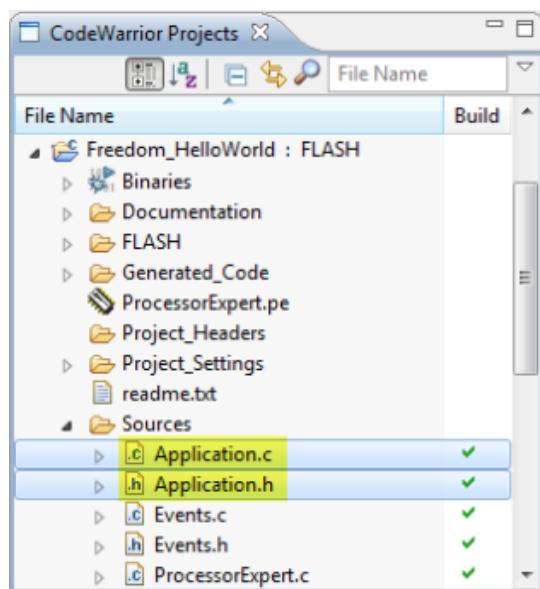
در انتها باید کد را ایجاد کنیم :



حال ۲ فایل به پروژه اضافه میشود که در قسمت زیر مشخص است و میتوانید آنها را باز کرده و محتویات و متد های داخلشان را مشاهده کنید: (در واقع برای هر مازول که در قسمت processor expert ایجاد میکنید بعد از فشردن ایجاد کد ۲ فایل مربوط به مازول در این قسمت ایجاد میشود که آن مازول را میتوان از این طریق کنترل کرد).



برای پیاده سازی نیاز به دو فایل Application.h, Application.c در آن قرار میگیرد :



در این قسمت باید ساختار داده ای که میخواهیم با UART پیاده کنیم را مشخص کنیم که در Application.h این کار را انجام میدهیم :

```

typedef struct {
    LDD_TDeviceData *handle; /* LDD device handle */
    volatile uint8_t isSent; /* this will be set to 1 once the block has been
sent */
    uint8_t rxChar; /* single character buffer for receiving chars */
    uint8_t (*rxPutFct)(uint8_t); /* callback to put received character into
buffer */
} UART_Desc;

void APP_Run(void);

```

در این قطعه کد یک فلگ issent داریم که هنگام ارسال یک میشود و یک اشاره گر به هندر مژول داریم و کاراکتر که داده در آن قرار میگیرد و یک اشاره گر هم برای استفاده از بافر.

تابع (APP_Run) از داخل main صدا زده میشود که تابع اصلی برنامه است و دریافت داده ها در این تابع انجام میشود .

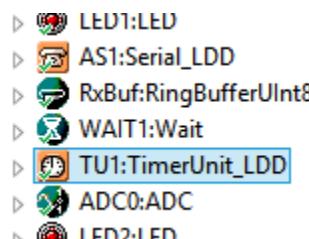
محتوای تابع (APP_Run) به این صورت است که در یک حلقه منتظر دریافت پیام مژول UART میباشد و با دریافت پیام باید دستور رسیده را انجام بدهد . در واقع درین قسمت چک میکند که اگر دستور دریافت شده به صورت $^{temp10^8}$ باشد یعنی دمای محیط باید به 10° درجه برسد . پس ابتدا ابتدای دستور را چک میکند سپس بقیه کاراکتر ها . نکته ای قابل توجه این است که به متدهای هر فایل و در نهایت به توابع AS1.c,AS1.h که درایور UART میباشد دقت کنید . توجه شود که توابع APP.c تعریف شده و از SendChar و SendString استفاده میکند .

سناریو:

میخواهیم با استفاده از این بورد به مژول وای فای متصل شویم و هر یک ثانیه یک بار دمای محیط را برای آن از طریق مژول UART ارسال کنیم در این صورت با استفاده از سرور ایجاد شده دما را به کاربر نشان دهیم . و یک ورودی از کاربر بگیرد که بتواند دمای محیط را با استفاده سیستم های گرمایشی (با سیگال های PWM) تنظیم کند . حال به این مرحله میرسیم که هر یک ثانیه داده را ارسال کند : برای این کار از وقفه ای تامیر استفاده میکنیم . پس یک مژول تایمر باید ایجاد کنیم که هر یک ثانیه وقفه را ایجاد کند :

تایمر :

ابتدا مژول نشان داده شده را به کتابخانه اضافه میکنیم :



صفحه ای تنظیمات زیر را ایجاد میکنیم :

Name	Value	Details
Component name	TU1	
Module name	LPTMR0	LPTMR0
Counter	LPTMR0_CNR	LPTMR0_CNR
Counter direction	Up	
Counter width	16 bits	
Value type	Optimal	uint32_t
Input clock source	Internal	
Counter frequency	128 Hz	128 Hz
Counter restart	On-match	
Period device	LPTMR0_CMR	LPTMR0_CMR
Period	1 sec	1 sec
Interrupt	Enabled	
Interrupt	INT_LPTimer	INT_LPTimer
Interrupt priority	medium priority	2
Channel list	0	1 available channels
Initialization		
Enabled in init. code	yes	
Auto initialization	yes	

به شمارشگر و دوره و فعل بودن وقfe توجه کنید . بعد از ایجاد کد به Event.c مراجع کنید و توابع زیر را بینند :

```
void TU1_OnCounterRestart(LDD_TUserData *UserDataPtr)
{
    /* Write your code here ... */
}
```

در صورت ایجاد وقfe این تابع اجرا میشود . حال در اینجا باید بگوییم که دما را به مازول ارسال کن :

```
// read from analog input temperature

//*****calibrate this value *****
static uint16_t value;

(void)ADC0_Measure(TRUE); /* do conversion and wait for the result */
(void)ADC0_GetValue16(&value); /* get the result into value variable */

value = (((value*2.56)*100)/65535);

char num ;

while(value!=0){
    num=value%10 ;
    num = num+ '0';// create number char from int
    value=value/10;
```

```

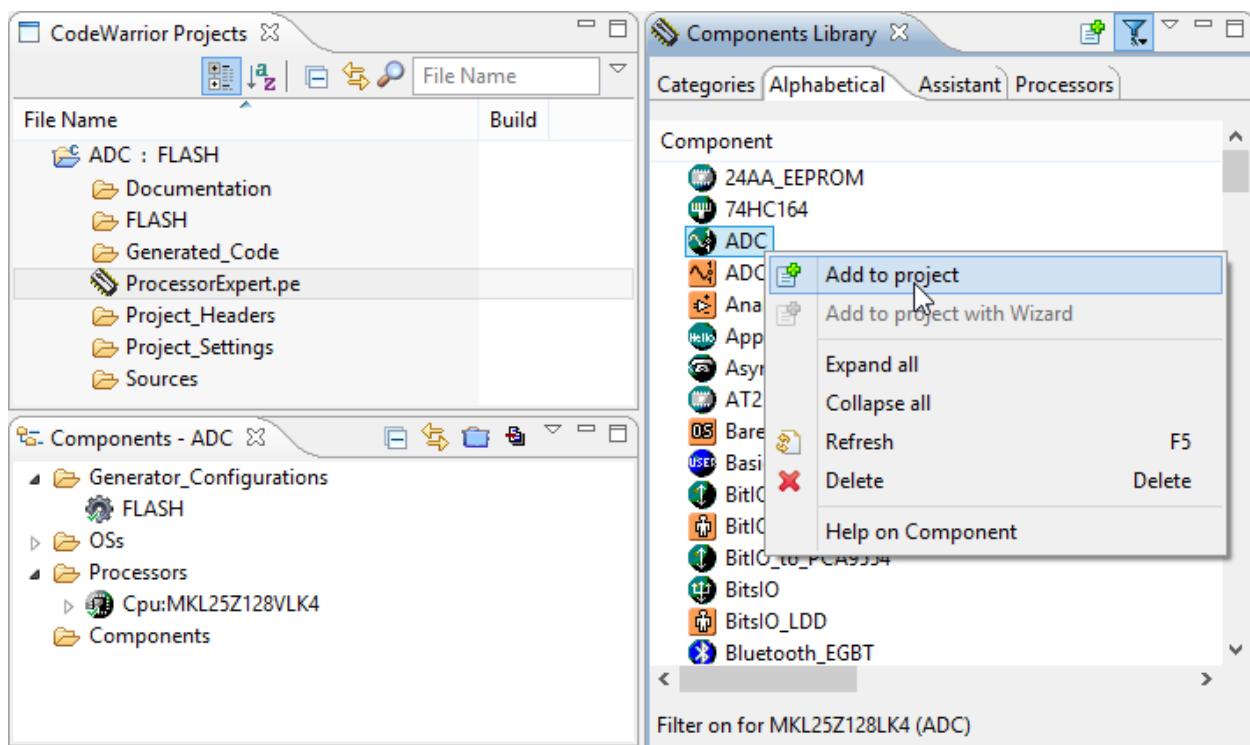
        SendCh(num); //this is temp reverse
    }
    SendCh('^\');
    LED1_Neg();

```

حال در این قسمت باید دمای مازول را به فایل ارسال کنیم . ولی قبل از این باید دمای مازول ADC دریافت کنیم که نیاز به مازول ADC داریم که به صورت زیر پیاده سازی میشود :

پیاده سازی ADC

برای پیاده سازی ADC ابتدا مازول ADC را از قسمت کتابخانه ای اجزا به پروژه اضافه میکنیم . این کتابخانه در قسمت menu Processor Expert > Show Views قابل مشاهده است .



پیکربندی اجزا:

هنگام اضافه شدن مازول باید قسمت های مختلف که در شکل زیر قابل مشاهده است مقدار دهی شود :

Name	Value	Details
A/D converter	ADC0	ADC0
Sharing	Disabled	
ADC_LDD	ADC_LDD	Error in the inherited component s...
Interrupt service/event	Enabled	
A/D interrupt priority	medium priority	2
A/D channels	1	
Channel0		
A/D channel (pin)	ADC0_DM0/ADC0_SE4a/PTE21/TP...	ADC0_DM0/ADC0_SE4a/PTE21/TP...
Mode select	Single Ended	
A/D resolution	Autoselect	16 bits
Conversion time		Unassigned timing
Low-power mode	Disabled	
High-speed conversion mode	Disabled	
Asynchro clock output	Disabled	
Sample time	0 = short	
Initialization		
Enabled in init. code	yes	

ابتدا پایه‌ی مورد استفاده را مشخص می‌کنیم. با استفاده از datasheet و شماتیک بورد مشخص می‌شود که پایه‌ی A0 به ADC0_SE8/PTB0 متصل است پس به صورت زیر انتخاب می‌کنیم. انتخاب PTB0 :

حال مدت زمان تبدیل را تنظیم می‌کنیم :

*Component Inspector - AD1

Basic Advanced Expert

Properties Methods Events

Name	Value	Details
A/D converter	ADC0	ADC0
Sharing	Disabled	
! ADC_LDD	ADC_LDD	Error in the inherited component s...
Interrupt service/event	Enabled	
A/D interrupt priority	medium priority	2
A/D channels	1	
Channel0		
A/D channel (pin)	ADC0_SE8/TSI0_CH0/PTB0/LLWU_...	ADC0_SE8/TSI0_CH0/PTB0/LLWU_...
Mode select	Single Ended	
A/D resolution	Autoselect	
! Conversion time		
Low-power mode	Disabled	
High-speed conversion mode	Disabled	
Asynchro clock output	Disabled	
Sample time	0 = short	
Initialization		
Enabled in init. code	yes	

با این کار یک پنجره‌ی دیگر باز می‌شود که این پنجره یک سری زمان تبدیل قابل قبول نمایش میدهد که با توجه به نیاز یکی از این زمان‌ها را انتخاب می‌کنیم:

Timing dialog - AD1/Conversion time

<< Advanced

Prescalers\Req.value	Speed mode 0	Adjusted values
Clock source:	Auto select	Speed mode 0: ADC0Bu...

Runtime settings type: fixed value

Value type	Value	Unit
Init. value:	9.615385	μs

Possible settings Clock path

Selected speed mode All

Value
1.192093 μs
2.384186 μs
4.768372 μs
4.807692 μs
9.536743 μs
9.615385 μs
19.073486 μs
19.230769 μs

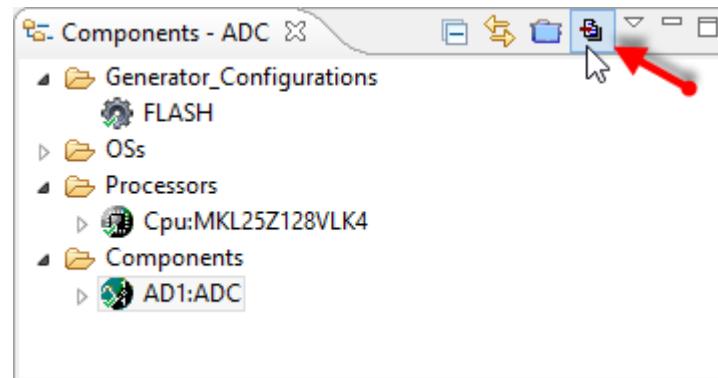
Allowed error: Unit: Minimal timer tick:

5 % 0

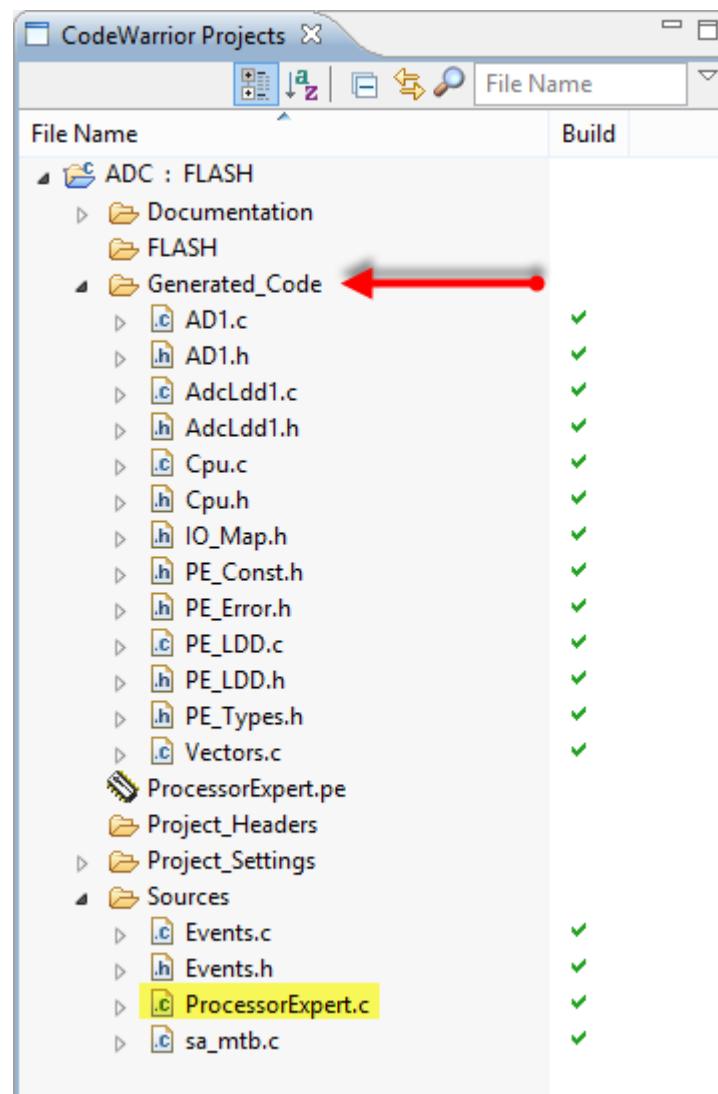
OK Cancel

مدت زمان تبدیل(conversion time) مدت زمانی است که مازول ADC میتواند داده‌ی آنالوگ را به دیجیتال تبدیل کند که این زمان مسلم‌ا بر کیفیت و تبدیل تاثیر میگذارد.

: ایجاد کد



حال فایل‌ها و متدهای تولید شده در قسمت کدهای ایجاد شده مطابق زیر قرار دارند:



حال در قسمت تایمر هر یک ثانیه از ورودی آنالوگ دما را میخوانیم :

```
(void)ADC0_Measure(TRUE); /* do conversion and wait for the result */  
(void)ADC0_GetValue16(&value); /* get the result into value variable */  
این دو متد در قسمت وقفه‌ی تایمر نوشته میشود که هر یک ثانیه انجام میشود و در قسمت قبل نشان داده شد. البته باید کتابخانه Event.c را به فایل ADC0.h اضافه کرد و سپس در قسمت وقفه‌ی تایمر مطابق زیر وارد میکنیم :
```

```
C/C++ - TEMP/Sources/Events.c - CodeWarrior Development Studio  
Processor Expert Run Window Help  
main.c Events.h *Events.c AdcLdd1.h ADC0.c  
Component : TU1 [TimerUnit_LDD]  
/*!  
 * @brief  
 * Called if counter overflow/underflow or counter is  
 * reinitialized by modulo or compare register matching.  
 * OnCounterRestart event and Timer unit must be enabled. See  
 * [SetEventMask] and [GetEventMask] methods. This event is  
 * available only if a [Interrupt] is enabled.  
 * @param  
 * UserDataPtr - Pointer to the user or  
 * RTOS specific data. The pointer passed as  
 * the parameter of Init method.  
 */  
=====  
static uint16_t value;  
void TU1_OnCounterRestart(LDD_TUserData *UserDataPtr)  
{  
    /* Write your code here ... */  
    // read from analog input temperature  
    (void)ADC0_Measure(TRUE); /* do conversion and wait for the result */  
    (void)ADC0_GetValue16(&value); /* get the result into value variable */  
    LED1_Neg();  
}
```

تابع ADC0_Measure(true)

اندازه گیری از ورودی آنالوگ هنگامی که صدا زده میشود و اگر در قسمت آرگومان مقدار true نوشته شود به معنای این است که کد منتظر باشد تا تبدیل انجام شود و سپس ادامه‌ی برنامه اجرا شود. و خروجی میتواند مقادیر زیر را بازگرداند :

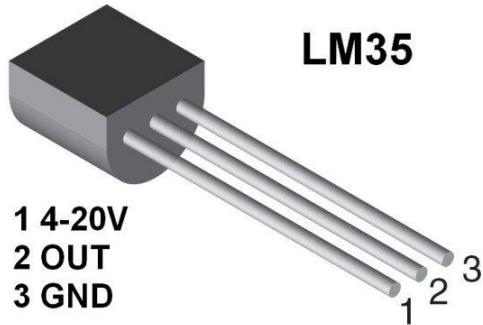
```
- Error code, possible codes:  
** ERR_OK - OK  
** ERR_SPEED - This device does not work in  
the active speed mode  
** ERR_DISABLED - Device is disabled  
** ERR_BUSY - A conversion is already running  
*/
```

تابع ADC0_getValue(value)

برای دریافت آخرین مقدار تبدیل یافته از این تابع استفاده میکنیم که مقدار را به value منتقل میکند .

این دو تابع در قسمت وقفه‌ی تایمر صدا زده میشود و بعد از آینکه صدا زده و مقدار دما در value ذخیره شد با استفاده از UART به مازول وای فای فرستاده میشود . دقت شود که در آخر کاراکتر ۸ که نشان دهنده‌ی اتمام ارسال است فرستاده میشود . این کاراکتر به دلیل کدی که در وای فای نوشته شده است مورد نیاز است و مازول وای فای با استفاده از این کد متوجه اتمام دریافت میشود .

سنسور دمایی مورد استفاده سنسور LM35 میباشد .



<http://noise.blog.ir/post/%DA%A9%D8%A7%D8%B1%DA%AF%D8%A7%D9%87-%D8%B9%D9%85%D9%84%DB%8C-%D8%B4%D9%85%D8%A7%D8%B1%D9%87-2-%DA%A9%D8%A7%D8%B1-%D8%A8%D8%A7-%D9%88%D8%A7%D8%AD%D8%AF-LM35-ADC-%D8%B3%D9%88%D9%86-%D8%B3%DA%AF%D9%85%D9%86%D8%AA-18>

خروجی این سنسور از نوع آنالوگ می باشد و به ازای افزایش هر درجه سانتی گراد، خروجی را ۱۰ میلی ولت افزایش می دهد. برای سنجش مقدار دما، باید خروجی این سنسور را بر حسب میلی ولت اندازه گیری کرده و این مقدار را بر ۱۰ تقسیم نماییم .

این سنسور می تواند دمایی حدود ۴۵-۱۳۰ تا +۰۰۵ سانتی گراد ، با رزولیشن بالای ۰.۰۰۵ سانتی گراد را اندازه گیری نماید.

در مورد تغذیه این ای سی می توان از ولتاژ ۴.۷۵ ولت تا ۷.۲ ولت را به ان وصل کرد که بهترین ولتاژ ۵ ولت می باشد و جریان مصرفی این ای سی حدود ۱۶۰ تا ۲۰۰ میکرو آمپر می باشد.

$$370mV=37^{\circ}C$$

$$20.9mV=20.9^{\circ}C$$

ماژول ADC میکروکنترلر قادر به اندازه گیری ولتاژ آنالوگ از ۰ ولت تا ۵+ ولت می باشد. نمونه گیری از ولتاژ ورودی با دقت ۱۲ بیت انجام می شود. و این یعنی اینکه مقدار اندازه گیری شده عددی بین ۰ تا 65535 خواهد بود. از آنجایی که عدد خوانده شده نسبت به ولتاژ ورودی خطی است پس می توان با یک نسبت تناسب ساده از عدد خوانده شده، مقدار ولتاژ ورودی را بدست آورد:

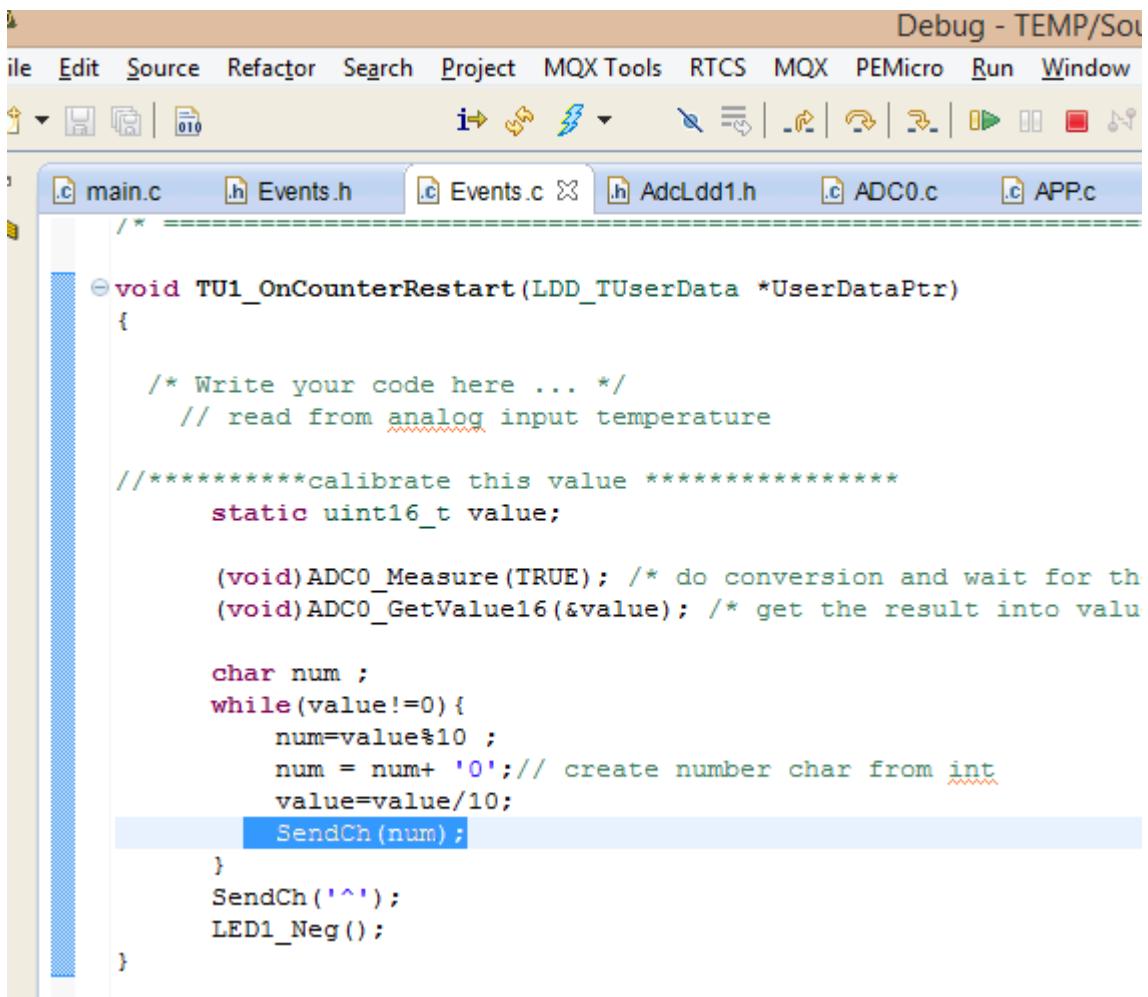
$$\text{value} = (((\text{value} * 2.56) * 100) / 65535);$$

دلیل اضافه شدن عبارت $(* 100)$ این است که سنسور مقدار رو به میلی ولت به ما تحويل میدهد.

در استفاده از این ماژول به دلیل خواندن ADC در وقهی تایмер خواندن درست کار نمیکرد و فلگ اندازه گیری سمت نمیشد بعد از خواندن datasheet در قسمت وقهه ها ، به این موضوع پی برده شد که هنگامی که از ورودی آنالوگ خوانده میشود به دلیل رخداد وقه در قسمت ADC و اینکه این وقه هم سطح با وقهی تایمر است در نتیجه بلاک میشود

تا زمانی که تایمر کار خودش را انجام دهد و چون تایمر منتظر این وقته‌ی ADC است در نتیجه یک deadlock رخ میدهد . که با تغییر سطح اهمیت وقته‌ی سنسور از متوسط به بالا این مشکل حل می‌شود .

مشکل یاد شده هنگام نوشتن کد ارسال دما از بورد به UART نیز رخداد که در این حالت بعد از خواندن دما از ADC و برای ارسال توسط پوآرت نیز ایجاد می‌شود :



The screenshot shows a software development environment with a menu bar including File, Edit, Source, Refactor, Search, Project, MQX Tools, RTCS, MQX, PEMicro, Run, and Window. Below the menu is a toolbar with various icons. The main window displays several tabs: main.c, Events.h, Events.c (which is the active tab), AdcLdd1.h, ADC0.c, and APP.c. The code in main.c is as follows:

```
/* ===== */
void TU1_OnCounterRestart(LDD_TUserData *UserDataPtr)
{
    /* Write your code here ... */
    // read from analog input temperature

    //*****calibrate this value *****
    static uint16_t value;

    (void)ADC0_Measure(TRUE); /* do conversion and wait for th
    (void)ADC0_GetValue16(&value); /* get the result into valu

    char num ;
    while(value!=0){
        num=value%10 ;
        num = num+ '0';// create number char from int
        value=value/10;
        SendCh(num);
    }
    SendCh('`');
    LED1_Neg();
}
```

همانطور که در کد مشخص شده است C در خط زیر به یک حلقه‌ی صیر بی نهایت در قسمت زیر می‌شد :

Debug - TEMP/Sources/APP.c - CodeWarrior D

Edit Source Refactor Search Project MQX Tools RTCS MQX PEMicro Run Window Help

main.c Events.h Events.c AdcLdd1.h ADC0.c APP.c APP.h ADC0.h

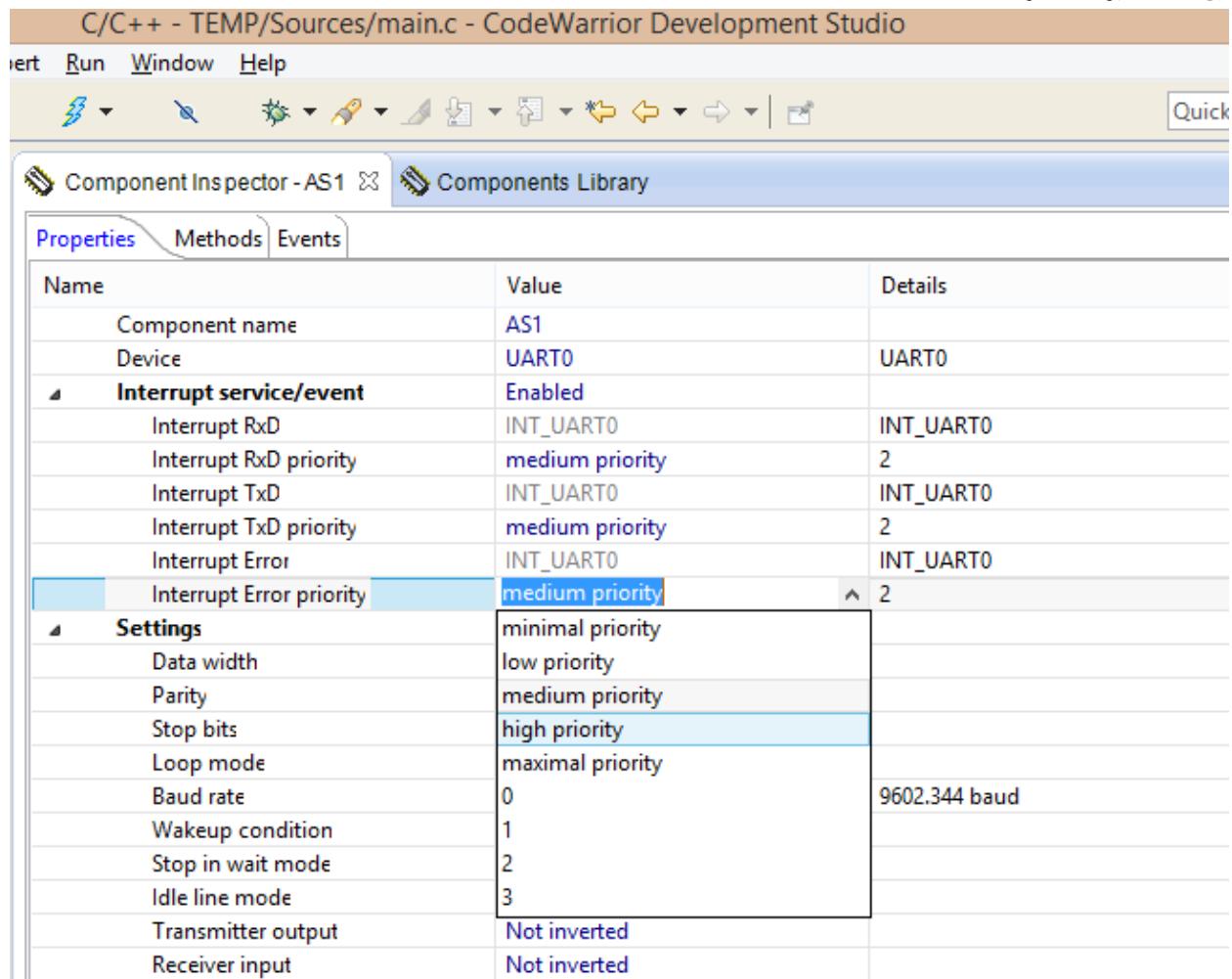
```
#include "App.h"
#include "RxBuf.h"
#include "AS1.h"

static UART_Desc deviceData;

static void SendChar(unsigned char ch, UART_Desc *desc) {
    desc->isSent = FALSE; /* this will be set to 1 once the block has been sent */
    while(AS1_SendBlock(desc->handle, (LDD_TData*)&ch, 1)!=ERR_OK) {} /* Send char */
    while(!desc->isSent) {} /* wait until we get the green flag from the TX interrupt */
}
```

در واقع در این سمت بورد منتظر isSent میباشد که این فلگ در وقفه‌ی UART یک میشود و چون این قسمت حاصل پرش از وقفه‌ی تایمر (با سطح متوجه) میشود و نیز منتظر یک وقفه از طرف UART (که این نیز سطح اهمیت متوجه دارد) میباشد در نتیجه اجازه‌ی وقفه به UART داده نمیشود پس یک deadlock نیز اینجا رخ میدهد پس با افزایش سطح اهمیت وقفه‌ی UART میتوان بر

این مشکل نیز غلبه کرد.



دقت شود که هر سه وقفه **i** RX,ERR,TX را به سطح اولویت بالا قرار دهید.

فریم اطلاعات رد و بدل شده بین مازول وای فای و بورد به صورت زیر و خیلی ساده فقط برای مشخص شدن ابتدا و انتهای command است و به این صورت است که از سمت بورد به مازول وای فای دما فقط به صورت عدد و کاراکتر '۸' ارسال میشود که نشان‌هندگی انتهای فریم است.

به عنوان مثال اگر **32۸** از سمت بورد ارسال شود دمای ۳۲ درجه را از بورد به سمت وای فای ارسال میکند.

یکی از اهدافاین پروژه این است که کاربر دمای محیط را دلخواه تنظیم کند که در سمت سرور یک ورودی از نوع تکست برای دریافت دما از کاربر و تنظیم دمای محیط با استفاده از سیستم های حرارتی ایجاد شده است و فریم ارسال اطلاعات از سمت مازول وای فای به بورد برای مثال به صورت **temp:32۸** میباشد. دلیل استفاده از این فریم و استفاده نکردن روش ساده تر و ارسال دما به صورت قبل این است که در صورت امکان بتوان کار های دیگری مثل وضعیت های اضطراری و خاص را ارسال کرد.

Bit Field: http://en.wikipedia.org/wiki/Bit_field

Bitwise Operation: http://en.wikipedia.org/wiki/Bitwise_operation

Mask: [http://en.wikipedia.org/wiki/Mask_\(computing\)](http://en.wikipedia.org/wiki/Mask_(computing))

Chart: <http://www.mathworks.com/help/fuzzy-foundations-of-fuzzy-logic.html>

<http://www.learning.asarayan.com/%D8%A2%D9%85%D9%88%D8%B2%D8%B4/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%D9%88%DB%8C%D9%86%D8%AF%D9%88%D8%B2/%D8%AF%D8%A7%D9%86%D8%B4%D9%86%D8%A7%D9%85%D9%87/3674-%D8%A7%DB%8C%D9%86%D8%AA%D8%B1%D9%86%D8%AA-%D8%A7%D8%B4%DB%8C%D8%A7%D8%A1-%DB%8C%D8%A7-iot-%DA%86%DB%8C%D8%B3%D8%AA-%D9%88-%DA%86%D9%87-%DA%A9%D8%A7%D8%B1%D8%A8%D8%B1%D8%AF%DB%8C-%D8%AF%D8%A7%D8%B1%D8%AF-%D8%9F.html>