**فصل سوم- توضیح فرآیند کلی کد**

در این فصل به بررسی روال اجرای برنامه میپردازیم.

مطابق همه برنامه های دیگر در زبان C ،تابع main ابتدا شروع به کار میکند.این تابع که در بخش قبل هم توضیح داده شد کدی به صورت زیر دارد:

1 int main (void) {

2 /\* Main Thread of the TcpNet \*/

3

4 init ();

5 LEDrun = \_\_TRUE;

6 dhcp\_tout = DHCP\_TOUT;

7 while (1) {

8 timer\_poll ();

9 main\_TcpNet ();

10 dhcp\_check ();

11 blink\_led ();

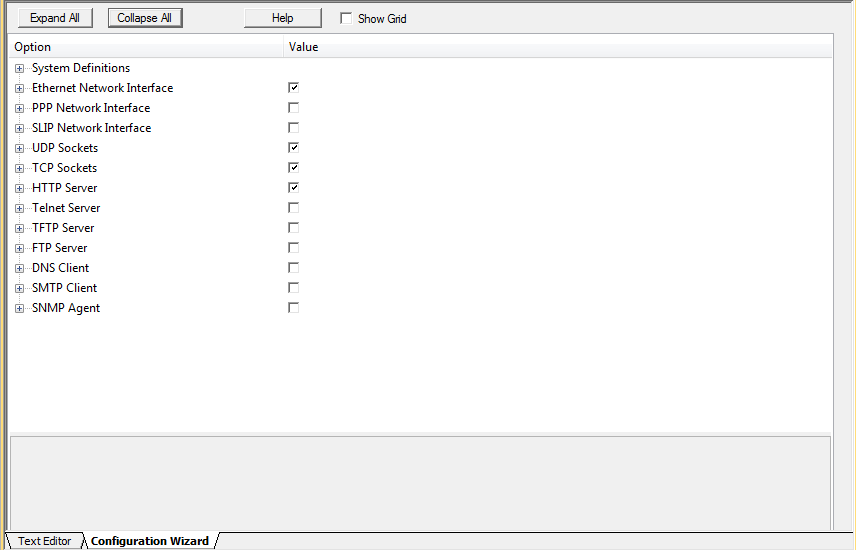
12 }

13 }

پیش از ورود به حلقه تکرار،ابتدا init() را اجرا میکند که در فصل قبل توضیحات مربوط به آن آمده است.انچه در این فصل جالب توجه است تابع init\_TcpNet() است که عملکرد اختصاص IP را مدیریت و ضمن hand shaking های اولیه با روتر شبکه یک ای پی اختصاصی از آن میگیرد در صورتی که DHCP فعال باشد و بنا باشد که IP به صورت دینامیکی به برد اختصاص یابد.در غیر اینصورت هم کانفیگ های اولیه را برقرار میکند.این تنظیمات در فایل net\_config.c انجام میگیرد.

# فایل : Net\_Config.c

این فایل شامل تنظیمات اصلی شبکه است. تنظیمات انواع پروتکل ها و همچنین تنظیمات اصلی شبکه را میشود به صروت گرافیکی با استفاده از رابط گرافیکی آن انجام داد، و به صورت خودکار مقادیر رجیسترهای لازم را تغییر میدهد.گرچه از منوی پایین ان امکان ویرایش مستقیم فایل متنی و کار با رجیستر ها هم وجود دارد.شکل کلی رابط گرافیکی به صورت زیر است:

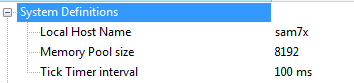


در این بخش به بحث راجع به تنظیماتی میپردازیم که در پروتکل HTTP و کار با این برنامه کارایی داشته اند.

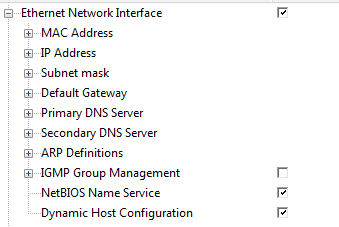
زیر شاخه اول در این فایل تعریف های اصلی سیستم است.اسم هاست محلی(local HOST NAME) همان نامیست که چنانچه برد دارای آی پی استاتیک باشد با این نام میتوان آن را در مرورگر فراخوانی کرد.

مورد بعدی سایز memory pool به بایت است.این حافظه نقش بافر را برای ما ایفا میکند تا اجرا تابع main\_tcpNet در سری بعدی.

Tick timer interval زمان هر تیک در تایمر را مشخص میکند.در واقع در این بخش به شبکه میگوییم که با چه فواصل زمانی آن را صدا خواهیم زد.



گزینه بعدی که با آن ارتباط داشتیم رابط ارتباط اترنت بود:



حال به بررسی تک تک زبانه ها میپردازیم:

MAC Address: یک آدرس منحصر به فرد برای هر سیستم موجود در شبکه است.این آدرس در حافظه ای خاص در روتر به ادرس IP در شبکخ map میشود.جزئیات تئوری را میتوان در فصل اول در توضیحات تئوری پیدا کرد.

IP Address: آدرس منحصر به فردی است که هر وسیله متصل به شبکه به خود اختصاص میدهد.آی پی اختصاص داده شده در اینجا چنانچه که DHCP غیرفعال باشد و یا روتر جواب به موقعی به ما نمیدهد ،بورد از این آی پی استفاده خواهد کرد.

Subnet mask: محدوده قابل رویت برای یک آی پی را مشخص میکند.

Default gateway: یک گره از شبکه است که به گره دیگری از شبکه سرویس ارائه میدهد.

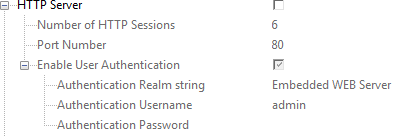
Primary DNS ServerوSecondary DNS Server: آدرسی در شبکه است که آدرس های اینترنتی را به ای پی متناظر آن ها map میکند.

ARP Definition: تنظیمات ARP است که در فصل اول توضیح داده شده اند.

IGMP Group Management: مربوط به لایه های دیگر دیگر شبکه بوده و اطلاعات لازم را در منابع مربوط به شبکه میتوانید پیدا کنید.

NetBIOS Name Service:این بخش مربوط به پروتکلی است که باعث میشود که سیستم های قدیمی بتوانند با پروتکل TCP.IP جدید کار کنند.

بخش بعدی مربوط به HTTP است:



Number of HTTP Sessions: تعداد درخواست های همزمانی که ابورد میتواند به طور همزمان به ان پاسخ دهد.

PORT Number: برای استاندارد بودن پروتکل HTTP این عدد باید 80 باشد.

Enable User Authentication: این گزینه به ما امکان محافظت از اسکریپت را میدهد .که یک یوزرنیم و پسورد از ما میگیرد.

بقیه گزینه های این فایل در اینجا کاری برای ما انجام نداده بنابراین به منظور جلوگیری از سردرگمی خواننده از توضیح انها خودداری میکنیم.

# ادامه مسیر اجرای برنامه

در ادامه برنامه یک متغیر را true کرده ایم:

5 LEDrun = \_\_TRUE;

این متغیر یک متخیر جهانی(global) است که همانطور که در فصل قبل هم توضیح داده شد با \_\_TRUE شدن آن روند چشمک زدن دیود ها آغاز میشود.و نشان از پیکربندی شدن شبکه در init دارد چرا که این متغیر بعد از اجرای تابع اجرا میشود.

نکته دیگری که شاید به درک بهتر این خط کمک کند این است که مقدار ثابت \_\_TRUE و \_\_FALSE در سراسر برنامه به ترتیب 1 و 0 است.

پس از این خط در خط بعدی،تعداد دفعات تلاش برای فعال کردن تخصیص آی پی بصورت دینامیک تعیین میشود.این تعداد درخواست ها ضربدر زمان هر تیک تایمر تیک در فایل netconfig مدت زمان ماکزیممی را که ما منتظر خواهیم بود تا ای پی به بورد اختصاص یابد را به ما میدهد.

پس از این وارد حلقه همواره درست برنامه میشویم.این حلقه به طور مداوم در پردازنده تکرار خواهد شد.ابتدا تابع timer\_poll صدا زده میشود. این تابع هم در فصل قبل به صورت جدا گانه توضیحات مربوط به آن(با ورود به جزئیات کد) داده شده است.کار کلی ان مدیریت tick های مربوط به TCP است که اینکار را از طریق PIT (Periodic Interval Timer) انجام میدهد.

در خط بعد تابع main\_TcpNet اجرا میشود.این تابع همانطور که در فصل قبل هم توضیح داده شد ابتدا بررسی میکند که آیا بسته ای دریافت شده است یا نه.سپس با توجه به متد داده دریافتی کنترل برنامه را به تواع درون فایل HTTP\_CGI میبرد.بهتر است در اینجا برای آشنایی بیشتر به بررسی توابع موجود در فایل HTTP\_CGI بپردازیم:

# فایل:HTTP\_CGI.c

این فایل مفسر اصلی درخواست های HTTP بوده و داده های ورودی و خروجی از فیلتر توابع موجود در آن عبور میکنند.در این به بخش به نویت توابع تعریف شده در آن و هنگامی که صدا زده میشوند را بیان میکنیم:

1)

void cgi\_process\_var (U8 \*qs)

این تابع هنگامی صدا زده میشود که درخواستی به سرور برسد که متد درخواست ارسالی GET باشد.به طور کلی دو فرمت اصلی برای ارسال اطلاعات از سمت کاربر به سمت سرور موجود است که یکی از انها GET و دیگری POST نام دارد.تفاوت این دو در ان است که در متد GET اطلاعات ارسالی از طریق آدرس و پس از یک علامت “?” به صورت “variable=value” به سمت سرور ارسال میشود که مقادیر متغیر های مختلف با علامت “&” از یکدیگر جدا میشوند.حال آنکه در متد POST این اطلاعات در هدر های HTTP جاسازی میشوند.برای اطلاعات بیشتر در این زمینه به فصل اول مراجعه شود.

2)

void cgi\_process\_data (U8 code, U8 \*dat, U16 len)

این تابع توسط وب سرور هنگامی صدا می شود که data از طریق متد Post توسط مرورگر ( سمت client) ارسال می شود. این اطلاعات ارسالی شامل سه بخش می باشد. 1) code 2) dat 3) len

الف) قسمت code یکی از حالت های زیر را داراست :

0 = www-url-encoded form data

1 = filename for file upload (0-terminated string)

2 = file upload raw data

3 = end of file upload (file close requested)

4 = any xml encoded POST data (single or last stream)

5 = the same as 4, but with more xml data to follow

Use http\_get\_content\_type() to check the content type

ب) dat اشاره گر است به داده post دریافتی

ج) len نیز طول داده دریافتی را مشخص می کند

محتوای تابع به صورت زیر است :

void cgi\_process\_data (U8 code, U8 \*dat, U16 len) {

U8 passw[12],retyped[12];

U8 \*var,stpassw;

switch (code) {

case 0:

/\* Url encoded form data received. \*/

break;

default:

/\* Ignore all other codes. \*/

return;

}

P2 = 0;

LEDrun = \_\_TRUE;

if (len == 0) {

/\* No data or all items (radio, checkbox) are off. \*/

LED\_out (P2);

return;

}

stpassw = 0;

var = (U8 \*)alloc\_mem (40);

do {

/\* Parse all returned parameters. \*/

dat = http\_get\_var (dat, var, 40);

if (var[0] != 0) {

/\* Parameter found, returned string is non 0-length. \*/

if (str\_scomp (var, "led0=on") == \_\_TRUE) {

P2 |= 0x01;

}

else if (str\_scomp (var, "led1=on") == \_\_TRUE) {

P2 |= 0x02;

}

else if (str\_scomp (var, "led2=on") == \_\_TRUE) {

P2 |= 0x04;

}

else if (str\_scomp (var, "led3=on") == \_\_TRUE) {

P2 |= 0x08;

}

else if (str\_scomp (var, "led4=on") == \_\_TRUE) {

P2 |= 0x10;

}

else if (str\_scomp (var, "led5=on") == \_\_TRUE) {

P2 |= 0x20;

}

else if (str\_scomp (var, "led6=on") == \_\_TRUE) {

P2 |= 0x40;

}

else if (str\_scomp (var, "led7=on") == \_\_TRUE) {

P2 |= 0x80;

}

else if (str\_scomp (var, "ctrl=Browser") == \_\_TRUE) {

LEDrun = \_\_FALSE;

}

else if (str\_scomp (var, "pw=") == \_\_TRUE) {

/\* Change password. \*/

str\_copy (passw, var+3);

stpassw |= 1;

}

else if (str\_scomp (var, "pw2=") == \_\_TRUE) {

/\* Retyped password. \*/

str\_copy (retyped, var+4);

stpassw |= 2;

}else if (str\_scomp (var, "text=") == \_\_TRUE) {

str\_copy (text, var+5);

LCD(text);

}

}

}while (dat);

free\_mem ((OS\_FRAME \*)var);

LED\_out (P2);

if (stpassw == 0x03) {

len = strlen ((const char \*)passw);

if (mem\_comp (passw, retyped, len) == \_\_TRUE) {

/\* OK, both entered passwords the same, change it. \*/

str\_copy (http\_auth\_passw, passw);

}

}

}

3)

U16 cgi\_func (U8 \*env, U8 \*buf, U16 buflen, U32 \*pcgi)

این تابع توسط وب سرور جهت تولید خروجی با فرمت مناسب استفاده می گردد به صورت زیر

U16 cgi\_func (U8 \*env, U8 \*buf, U16 buflen, U32 \*pcgi) {

TCP\_INFO \*tsoc;

U32 len = 0;

U8 id, \*lang;

int a=4;

float adcVoltage=-1;

//unsigned char adc\_v[16];

switch (env[0]) {

/\* Analyze the environment string. It is the script 'c' line starting \*/

/\* at position 2. What you write to the script file is returned here. \*/

case 'a' :

/\* Network parameters - file 'network.cgi' \*/

switch (env[2]) {

case 'i':

/\* Write the local IP address. The format string is included \*/

/\* in environment string of the script line. \*/

len = sprintf((char \*)buf,(const char \*)&env[4],LocM.IpAdr[0],

LocM.IpAdr[1],LocM.IpAdr[2],LocM.IpAdr[3]);

len = sprintf((char \*)buf,"%d",a);

break;

case 'm':

/\* Write local Net mask. \*/

len = sprintf((char \*)buf,(const char \*)&env[4],LocM.NetMask[0],

LocM.NetMask[1],LocM.NetMask[2],LocM.NetMask[3]);

break;

case 'g':

/\* Write default gateway address. \*/

len = sprintf((char \*)buf,(const char \*)&env[4],LocM.DefGW[0],

LocM.DefGW[1],LocM.DefGW[2],LocM.DefGW[3]);

break;

case 'p':

/\* Write primary DNS server address. \*/

len = sprintf((char \*)buf,(const char \*)&env[4],LocM.PriDNS[0],

LocM.PriDNS[1],LocM.PriDNS[2],LocM.PriDNS[3]);

break;

case 's':

/\* Write secondary DNS server address. \*/

len = sprintf((char \*)buf,(const char \*)&env[4],LocM.SecDNS[0],

LocM.SecDNS[1],LocM.SecDNS[2],LocM.SecDNS[3]);

break;

}

break;

case 'b':

/\* LED control - file 'led.cgi' \*/

if (env[2] == 'c') {

/\* Select Control \*/

len = sprintf((char \*)buf,(const char \*)&env[4],LEDrun ? "" : "selected",

LEDrun ? "selected" : "");

break;

}

/\* LED CheckBoxes \*/

id = env[2] - '0';

if (id > 7) {

id = 0;

}

id = 1 << id;

len = sprintf((char \*)buf,(const char \*)&env[4],(P2 & id) ? "checked" : "");

break;

case 'c':

/\* TCP status - file 'tcp.cgi' \*/

while ((len + 150) < buflen) {

tsoc = &tcp\_socket[MYBUF(pcgi)->xcnt];

MYBUF(pcgi)->xcnt++;

/\* 'sprintf' format string is defined here. \*/

len += sprintf((char \*)(buf+len),"<tr align=\"center\">");

if (tsoc->State <= TCP\_STATE\_CLOSED) {

len += sprintf ((char \*)(buf+len),

"<td>%d</td><td>%s</td><td>-</td><td>-</td>"

"<td>-</td><td>-</td></tr>\r\n",

MYBUF(pcgi)->xcnt,state[tsoc->State]);

}

else if (tsoc->State == TCP\_STATE\_LISTEN) {

len += sprintf ((char \*)(buf+len),

"<td>%d</td><td>%s</td><td>-</td><td>-</td>"

"<td>%d</td><td>-</td></tr>\r\n",

MYBUF(pcgi)->xcnt,state[tsoc->State],tsoc->LocPort);

}

else {

len += sprintf ((char \*)(buf+len),

"<td>%d</td><td>%s</td><td>%d.%d.%d.%d</td>"

"<td>%d</td><td>%d</td><td>%d</td></tr>\r\n",

MYBUF(pcgi)->xcnt,state[tsoc->State],

tsoc->RemIpAdr[0],tsoc->RemIpAdr[1],

tsoc->RemIpAdr[2],tsoc->RemIpAdr[3],

tsoc->RemPort,tsoc->LocPort,tsoc->AliveTimer);

}

/\* Repeat for all TCP Sockets. \*/

if (MYBUF(pcgi)->xcnt == tcp\_NumSocks) {

break;

}

}

if (MYBUF(pcgi)->xcnt < tcp\_NumSocks) {

/\* Hi bit is a repeat flag. \*/

len |= 0x8000;

}

break;

case 'd':

/\* System password - file 'system.cgi' \*/

switch (env[2]) {

case '1':

len = sprintf((char \*)buf,(const char \*)&env[4],

http\_EnAuth ? "Enabled" : "Disabled");

break;

case '2':

len = sprintf((char \*)buf,(const char \*)&env[4],http\_auth\_passw);

break;

}

break;

case 'e':

/\* Browser Language - file 'language.cgi' \*/

lang = http\_get\_lang();

if (strcmp ((const char \*)lang, "en") == 0) {

lang = "English";

}

else if (strcmp ((const char \*)lang, "en-us") == 0) {

lang = "English USA";

}

else if (strcmp ((const char \*)lang, "en-gb") == 0) {

lang = "English GB";

}

else if (strcmp ((const char \*)lang, "de") == 0) {

lang = "German";

}

else if (strcmp ((const char \*)lang, "de-ch") == 0) {

lang = "German CH";

}

else if (strcmp ((const char \*)lang, "de-at") == 0) {

lang = "German AT";

}

else if (strcmp ((const char \*)lang, "fr") == 0) {

lang = "French";

}

else if (strcmp ((const char \*)lang, "sl") == 0) {

lang = "Slovene";

}

else {

lang = "Unknown";

}

len = sprintf((char \*)buf,(const char \*)&env[2],lang,http\_get\_lang());

break;

case 'y':

/\* Button state - xml file 'button.cgx' \*/

len = sprintf((char \*)buf,"<checkbox><id>button%c</id><on>%s</on></checkbox>",

env[1],(get\_button () & (1 << (env[1]-'0'))) ? "true" : "false");

break;

case 'k':

len = sprintf((char \*)buf,(const char \*)&env[2],a);

case 'l' :

len = sprintf((char \*)buf,"%s",text);

break;

case 'n' :

adcVoltage=getADC();

len = sprintf((char \*)buf,"%f",adcVoltage);

break;

}

return ((U16)len);

}

# فایل :EMAC\_SAM7X.c

این فایل لایه فیزیکی را پیکربندی میکند.و پینهای میکرو را در اختیار واحد اترنت قرار میدهدو رجیستر ها را مقدار دهی میکند.در عمل نیازی به ویرایش این فایل نخواهد بود چرا که در هر پروژه مربوط به اترنت تنها کافیست آنرا include کنید.