**توابع و کتابخانه ها**

* **توابع استفاده شده در HTTP Sample code :**
  + **توابع موجود در Http\_demo :**

نکته ای که در مورد توابع Http\_demo وجود دارد این است که توابع موجود در قسمت #ifndef مربوط به قبل از برقراری ارتباط با computer  
(user) است و توابع موجود در قسمت #else مربوط به بعد از برقراری ارتباط با computer (user) است.

**تابع init\_io :**

static void init\_io () {

/\* I/O Ports configured as Output. \*/

\*AT91C\_PMC\_PCER = (1 << AT91C\_ID\_PIOB) | /\* Enable Clock for PIO (LEDs) \*/

(1 << AT91C\_ID\_PIOA); /\* (Joystick) \*/

\*AT91C\_PIOA\_PER = 0xf<<19; /\* Enable PIO for LED1..4 \*/

\*AT91C\_PIOA\_OER = 0xf<<19; /\* LED1..4 are Outputs \*/

\*AT91C\_PIOA\_SODR = 0xf<<19; /\* Turn on LED's \*/

\*AT91C\_PIOB\_PER = 0xf<<19; /\* Enable PIO for LED5..8 \*/

\*AT91C\_PIOB\_OER = 0xf<<19; /\* LED5..8 are Outputs \*/

\*AT91C\_PIOB\_SODR = 0xf<<19; /\* Turn off LED's \*/

}

توضیحات :

AT91C\_ID\_PIOB و AT91C\_ID\_PIOA مقادیر ثابتی هستند که در کتابخانه AT91Sam7X256 تعریف (define) شده اند :

#define AT91C\_ID\_PIOA ( 2) // Parallel IO Controller A

#define AT91C\_ID\_PIOB ( 3) // Parallel IO Controller B

7 کلید بر روی بورد وجود دارد که 2 تای آنها مربوط به interrupt ها هستند و 5 تای دیگر برای ورودی دادن به صورت دستی می باشد. کلیدهای S6 و S7 مربوط به interrupt ها هستند و به پورت PA14 و PA30 متصل می باشند. جهت کار با کلید های S1-S5 لازم است که پایه کلاک اصلی سیستم به این پورت متصل شود تا با هر بار خوردن کلاک وضعیت آنها در رجیستر PIOB\_PDSR ذخیره شود.جهت دادن کلاک به پورت های B از دستور : PMC\_PCER = 1<<3  
 (PMC\_PCER = 1 << AT91C\_ID\_PIOB) استفاده می کنیم. چنانچه بخواهیم interrupt ها فعال شود با دستور (PMC\_PCER = 1 << AT91C\_ID\_PIOA) استفاده می کنیم.

\*AT91C\_PMC\_PCER = (1 << AT91C\_ID\_PIOB) | (1 << AT91C\_ID\_PIOA);

در بورد ما تعداد 8 LED تعبیه شده است که 4 تا از آنها توسط پورت های A(19-22) و چهار تای دیگر توسط پورت های B(19-22) قابل کنترل است. برای روشن کردن یک LED ابتدا به کمک رجیستر های PIOx\_PER ( صفحه 240 از Datasheet) پورت مورد نظر را فعال می کنیم که x نوع پورت را مشخص می کند (A یا B). سپس توسط رجیستر PIOx\_OER (صفحه 241 از Datasheet) پورت مورد نظر را به عنوان خروجی پیکر بندی می کنیم. هنگامی که پورتی به عنوان خروجی معرفی میشود به صورت عادی LED مربوط به آن خاموش می گردد و هنگامی که خروجی تعریف نشده باشد یا توسط رجیستر PIOx\_ODR از خروجی بودن ساقط می شود LED مربوطه روشن میگردد. توسط رجیستر های PIOx\_SODR و PIOx\_CODR (صفحه 244 و 245 از Datasheet) می توان پورت هایی که به عنوان خروجی فعال شده اند را set و clear کرد. (LED ها را روشن و خاموش می کنیم)

\*AT91C\_PIOA\_PER = 0xf<<19; /\* Enable PIO for LED1..4 \*/

\*AT91C\_PIOA\_OER = 0xf<<19; /\* LED1..4 are Outputs \*/

\*AT91C\_PIOA\_SODR = 0xf<<19; /\* Turn on LED's \*/

\*AT91C\_PIOB\_PER = 0xf<<19; /\* Enable PIO for LED5..8 \*/

\*AT91C\_PIOB\_OER = 0xf<<19; /\* LED5..8 are Outputs \*/

\*AT91C\_PIOB\_SODR = 0xf<<19; /\* Turn off LED's \*/

**تابع init :**

#ifndef RTX\_KERNEL

ابتدا لازم است ساختار #ifndef (#ifdef) توضیح داده شود:

#ifdef <token>

/\* code \*/

#else

/\* code to include if the token is not defined \*/

#endif

#ifdef چک می کند که آیا عبارت مقابل آن (token) در داخل فایل جاری یا سایر فایل های include شده تعریف شده است. در صورتی که تعریف شده باشد code داخل برنامه قرار می گیرد. در غیر اینصورت code to include if the token is not defined داخل برنامه قرار خواهد گرفت. همین توضیحات برای #ifndef نیز با کمی تصحیح قابل تعمیم می باشد.

static void init () {

لازم به ذکر است که توابع static بدین معنی است که تابع تعریف شده فقط در همان فایلی که تعریف شده (در اینجا منظورمان Http\_demo می باشد.)

/\* Add System initialisation code here \*/

تابع init مربوط به تنظیمات اولیه سیستم جهت برقراری ارتباط از طریق پروتکل tcp-ip می باشد.

init\_io ();

توضیح این تابع قبلا داده شده است.

init\_TcpNet ();

تعریف (prototype) این تابع در کتابخانه rtl.h تعریف شده است و وظیفه این تابع تظیمات لازم از جهت resource ها، پروتکل ها و application های مورد استفاده در پروتکل tcp-ip در بورد ما می باشد.

/\* PITC Timer interval timer, reload to 100ms \*/

\*AT91C\_PITC\_PIMR = (TCNT - 1) | AT91C\_PITC\_PITEN;

AT91C\_PITC\_PITEN به صورت زیر در کتابخانه AT91SAM7X256.H تعریف شده است :

#define AT91C\_PITC\_PITEN (0x1 << 24)

PITEN بیت شماره 5 در رجیستر PITC\_PIMR که وظیفه فعال کردن تایمر PITC را بر عهده دارد. دستور بالا نیز از طریق عبارت (TCNT - 1) | AT91C\_PITC\_PITEN هم تایمر را فعال می کند هم مقدار تایمر را در 100ms تنظیم می کند. (اطلاعات تکمیلی پیرامون تایمرها در صفحه 79 از datasheet)

توجه شود که TCNT در ابتدای برنامه در فولدر Http\_demo تعریف شده است :

#define MCK 47923200 /\* Master Clock \*/   
#define TCNT (MCK/16)/10 /\* Timer Counts for 100ms \*/

که با توجه به کلاک پردازنده مقدار TCNT بالا معادل 100ms است و عبارت TCNT-1 برای این است که مقدار اولیه شمارش تایمر از صفر است.

}

#else

\_\_task void init (void) {

/\* Add System initialisation code here \*/

init\_io ();

init\_TcpNet ();

}

//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

#else

\_\_task void init (void) {

/\* Add System initialisation code here \*/

init\_io ();

init\_TcpNet ();

در این قسمت با استفاده از توابع os\_tsk\_create و os\_tsk\_create\_user یک سری task که بعدا قرار است عملکردشان تعریف شود ، تعریف اولیه می شود.

/\* Initialize Tasks \*/

دستور os\_tsk\_prio\_self این گونه است که اولویت task جاری (در اینجا init) را مشخص می کند که در اینجا اولیت 100 به init داده شده است (اولویت از 0 تا 255 است که 0و 255 اولویت های از پیش رزرو شده هستند. اولویت 255 بالاترین اولویت و 0 پایین ترین اولویت است).

os\_tsk\_prio\_self (100);

دستور os\_tsk\_create در اینجا یک task به نام blink\_led ایجاد کرده و به آن اولویت 20 را اختصاص می دهد. نکته ای که در مورد os\_tsk\_create وجود دارد این است مقدار فضای stack ای که به آن اختصاص داده می شود، مقدار پیش فرض که در rtx\_config.c تعریف می شود، در اینجا 256 bytes می باشد.

os\_tsk\_create (blink\_led, 20);

os\_tsk\_create (timer\_task, 30);

دستور os\_tsk\_create\_user مشابه دستور os\_tsk\_create است با این تفاوت که مقدار حافظه stack اختصاصی آن قابل تعیین است.

os\_tsk\_create\_user (tcp\_task, 0, &tcp\_stack, sizeof(tcp\_stack));

دستور os\_tsk\_delete\_self برای این است که task جاری که init می باشد را از بین برده و برنامه را با اجرای task ای که در اولویت بالا بعد از init است ، در اینجا timer\_task، را اجرا می کند.

os\_tsk\_delete\_self();

}

#endif

**تابع blink\_led :**

لازم به توضیح است که \_\_TRUE و \_\_FALSE به صورت زیر تعریف شده است:  
 #define \_\_TRUE 1  
#define \_\_FALSE 0   
این تعریف در کتابخانه rtl.h صورت گرفته است.

#ifndef RTX\_KERNEL

static void blink\_led () {

/\* Blink the LEDs on AT91SAM7X board \*/

وظیفه این تابع چشمک زدن led های روی بورد است.

const U8 led\_val[16] = { 0x01,0x03,0x07,0x0F,0x1F,0x3F,0x7F,0xFF,0xFE,0xFC,0xF8,0xF0,0xE0,0xC0,0x80,0x00};

واضح است که led\_val یک آرایه 16 تایی از متغیر هایی با تایپ 8 بیتی از نوع unsigned است. (U8)

static U32 cnt;

متغیری که به صورت static تعریف می شود در صورت داشتن مقدار اولیه فقط در اولین باری که تابع صدا می شود اعمال می شود ولی در دفعات بعدی که تابع صدا می شود (تابعی که متغیر در آن تعریف شده است) آخرین مقداری که در متغیر قرار گرفته است مورد استفاده قرار می گیرد.

if (tick == \_\_TRUE) {

tick یک متغیر global است از نوع Bool.

/\* Every 100 ms \*/

tick = \_\_FALSE;

if (LEDrun == \_\_TRUE) {

LEDrun یک متغیر global است از نوع Bool.

LED\_out (led\_val[cnt]);

تابع LED\_out که در ادامه تعریف شده است با دریافت یک مقدار 8 بیتی، یکی از 8 بیت موجود را با توجه به ترتیب مقادیر led\_val مشخص می کند.

if (++cnt >= sizeof(led\_val)) {

cnt = 0;

} } }}

#else

\_\_task void blink\_led () {

/\* Blink the LEDs on AT91SAM7X board \*/

const U8 led\_val[16] = { 0x01,0x03,0x07,0x0F,0x1F,0x3F,0x7F,0xFF,0xFE,0xFC,0xF8,0xF0,0xE0,0xC0,0x80,0x00};

U32 cnt = 0;

LEDrun = \_\_TRUE;

while(1) {

/\* Every 100 ms \*/

if (LEDrun == \_\_TRUE) {

LED\_out (led\_val[cnt]);

if (++cnt >= sizeof(led\_val)) {

cnt = 0;

}}

os\_dly\_wait(10);

تابع os\_dly\_wait از جمله توابعی است که برای ایجاد pause در برنامه استفاده می شود و ورودی آن یک عدد unsigned 16 بیتی است که مقدار delay را بر اساس مدت زمان system\_tick که اینجا 100ms تنظیم شده است مشخص می کند. بدین معنی که عبارت بالا تاخیری معادل 10 برابر system\_tick ایجاد می کند.

}}

#endif

**تابع timer\_task :**

در اینجا task ای ایجاد می کنیم که برای ما tick تولید می کند با فرکانس دلخواه و برای این موضوع از تایمر داخلی پردازنده استفاده می کنیم.

\_\_task void timer\_task (void) {

/\* System tick timer task \*/

os\_itv\_set (10);

وظیفه تابع os\_itv\_set این است که task یا function ای که در آن فراخوانی شده است (در اینجا مقصود همان timer\_task می باشد ) را به صورت periodic صدا می زند. لازم به ذکر است که این تابع به همراه تابع os\_itv\_wait استفاده می شود. تابع os\_itv\_wait سبب می شود در هر جای task هستیم متوقف شده تا تابع os\_itv\_set صدا شود.

while (1) {

timer\_tick ();

tick = \_\_TRUE;

os\_itv\_wait ();

}}

نکته ای که در مورد این تابع وجود دارد این است که هر موقع صدا شود بدون توجه به وضعیت شمارش PIT(Periodic Interval Timer) که آیا PIVR به ماکسیمم خود رسیده یا خیر تولید tick جانبی را آغاز می کند. (برخلاف تابع timer\_poll)

**تابع timer\_poll :**

تفاوت timer\_poll با تابع timer\_task این است که هر بار که صدا زده می شود فقط یک tick ایجاد می کند.

static void timer\_poll () {

/\* System tick timer running in poll mode \*/

volatile U32 val;

if (\*AT91C\_PITC\_PISR & 1) {

AT91C\_PITC\_PISR که Periodic Interval Status Register می باشد دارای مقدار 0 یا 1 است که این مقدار در آدرس 0xFFFFFD34 حافظه ذخیره می شود. اگر مقدار PIVR به مقدار Max تعریف شده رسیده باشد مقدار آن 1 است در غیر اینصورت مقدار آن 0 می باشد.

val = \*AT91C\_PITC\_PIVR;

هدف از ریختن مقدار جاری شمارش counter (AT91C\_PITC\_PIVR;) صفر کردن counter است . (شمارشگر تایمر صفر می شود)

/\* Timer tick every 100 ms \*/

timer\_tick ();

tick = \_\_TRUE;

}}

**تابع tcp\_task :**

\_\_task void tcp\_task (void) {

/\* Main Thread of the TcpNet. This task should have \*/

/\* the lowest priority because it is always READY. \*/

dhcp\_tout = DHCP\_TOUT;

while (1) {

main\_TcpNet();

dhcp\_check ();

os\_tsk\_pass();

}}

**تابع LED\_out :**

این تابع با توجه به ورودی 32 بیتی که دریافت می کند یک LED را روشن می کند.

void LED\_out (U32 val) {

\*AT91C\_PIOA\_CODR = 0xf<<19;

\*AT91C\_PIOB\_CODR = 0xf<<19;

این دو خط تمام LED ها را در لحظه اول اجرای تابع، clear (خاموش) می کند.

if (val & 1 ) {

\*AT91C\_PIOA\_SODR = 1<<19; /\* Turn LED1 On \*/

}

if (val & 2) {

\*AT91C\_PIOA\_SODR = 1<<20; /\* Turn LED2 On \*/

}

if (val & 4) {

\*AT91C\_PIOA\_SODR = 1<<21; /\* Turn LED3 On \*/

}

if (val & 8) {

\*AT91C\_PIOA\_SODR = 1<<22; /\* Turn LED4 On \*/

}

if (val & 0x10 ) {

\*AT91C\_PIOB\_SODR = 1<<19; /\* Turn LED5 On \*/

}

if (val & 0x20) {

\*AT91C\_PIOB\_SODR = 1<<20; /\* Turn LED6 On \*/

}

if (val & 0x40) {

\*AT91C\_PIOB\_SODR = 1<<21; /\* Turn LED7 On \*/

}

if (val & 0x80) {

\*AT91C\_PIOB\_SODR = 1<<22; /\* Turn LED8 On \*/

}}

**تابع get\_button :**

این تابع مشخص می کند کدام کلید (Button) فشرده شده است و خروجی ای 8 بیتی تولید می کند. این خروجی در قسمت HTTP\_CGI.c مورد استفاده می شود.

U8 get\_button (void) {

/\* Read ARM Digital Input \*/

U32 keyA,keyB,val = 0;

رجیستر های AT91C\_PIOB\_PDSR و AT91C\_PIOA\_PDSR وضعیت Button ها را مشخص می کند. لازم به ذکر است که clock لازم برای PIO بودن Button ها قبلا در تابع init\_io داده شده است. نکته ای که در مورد این clk وجود دارد این است که با هر بار خوردن clk رجیستر های AT91C\_PIOB\_PDSR و AT91C\_PIOA\_PDSR به روز می شوند.

keyB = \*AT91C\_PIOB\_PDSR;

keyA = \*AT91C\_PIOA\_PDSR;

AT91B\_SW1-7 در کتابخانه at91sam7x-ek تعریف شده است.

#define AT91B\_SW1 (1<<23) // PA21 Up Button AT91C\_PA21\_TF AT91C\_PA21\_NPCS10

#define AT91B\_SW2 (1<<24) // PA22 Down Button AT91C\_PA22\_TK AT91C\_PA22\_SPCK1

#define AT91B\_SW3 (1<<25) // PA23 Left Button AT91C\_PA23\_TD AT91C\_PA23\_MOSI1

#define AT91B\_SW4 (1<<26) // PA24 Right Button AT91C\_PA24\_RD AT91C\_PA24\_MISO1

#define AT91B\_SW5 (1<<27) // PA25 Push Button AT91C\_PA25\_RK AT91C\_PA25\_NPCS11

#define AT91B\_SW6 (1<<14) // PA24 Right Button AT91C\_PA24\_RD AT91C\_PA24\_MISO1

#define AT91B\_SW7 (1<<30)

if ((keyB & AT91B\_SW1) == 0) {

/\* Check if SW1 is pressed \*/

val |= 0x01;

}

if ((keyB & AT91B\_SW2) == 0) {

/\* Check if SW2 is pressed \*/

val |= 0x02;

}

if ((keyB & AT91B\_SW3) == 0) {

/\* Check if SW3 is pressed \*/

val |= 0x04;

}

if ((keyB & AT91B\_SW4) == 0) {

/\* Check if SW4 is pressed \*/

val |= 0x08;

}

if ((keyB & AT91B\_SW5) == 0) {

/\* Check if SW5 is pressed \*/

val |= 0x10;

}

if ((keyA & AT91B\_SW6) == 0) {

/\* Check if SW4 is pressed \*/

val |= 0x20;

}

if ((keyA & AT91B\_SW7) == 0) {

/\* Check if SW5 is pressed \*/

val |= 0x40;

}

return (val);

}

**تابع dhcp\_check :**

static void dhcp\_check () {

/\* Monitor DHCP IP address assignment. \*/

if (tick == \_\_FALSE || dhcp\_tout == 0) {

return;

}

#ifdef RTX\_KERNEL

tick = \_\_FALSE;

#endif

if (mem\_test (&MY\_IP, 0, IP\_ADRLEN) == \_\_FALSE) {

/\* Success, DHCP has already got the IP address. \*/

dhcp\_tout = 0;

return;

}

if (--dhcp\_tout == 0) {

/\* A timeout, disable DHCP and use static IP address. \*/

dhcp\_disable ();

}

}

**تابع main :**

int main (void) {

/\* Main Thread of the TcpNet \*/

init ();

LEDrun = \_\_TRUE;

dhcp\_tout = DHCP\_TOUT;

while (1) {

timer\_poll ();

main\_TcpNet ();

dhcp\_check ();

blink\_led ();

}}