TCP/IP para sistemas empotrados

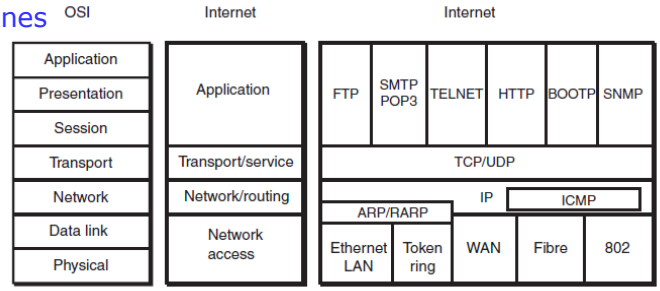
Source: http://contenidosdigitales.ulp.edu.ar/exe/computacion/mundo_tcp_ip.png

Modelo de red

ISO 7-Layer Model	Internet 4-Layer Model	"Real World" Embedded System
Application	Application	Application
Presentation		TCP/IP Stack
Session		
Transport	Transport or Service	
Network	Network or Routing	
Data Link	Network Access or Similar	Ethernet Controller
Physical		Hardware

Modelo de red

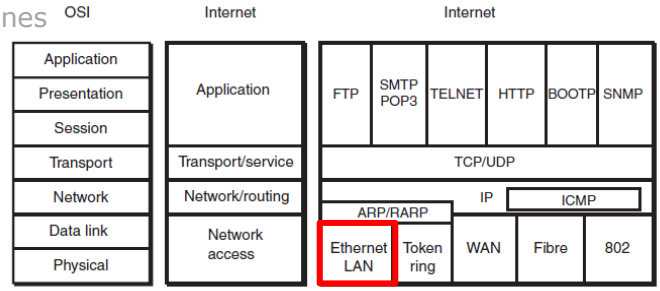
- La implementación básica de TCPIP requiere:
  - Nivel físico (un chip específico)
  - Ethernet II (módulo del micro)
  - IP
  - ARP (Address Resolution Protocol)
  - ICMP (Internet Control Message Protocol)
  - TCP/UDP
  - Aplicaciones



Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Modelo de red

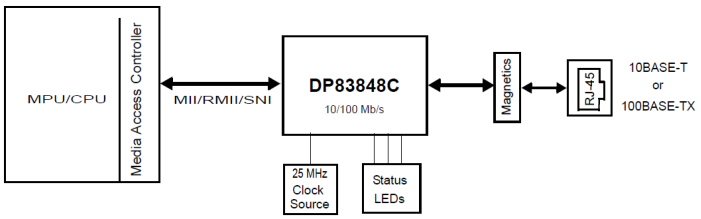
- La implementación básica de TCPIP requiere:
  - Nivel físico (un chip específico)
  - Ethernet II (módulo del micro)
  - IP
  - ARP
  - ICMP
  - TCP/UDP
  - Aplicaciones



Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Interfaz física

- Entre el microcontrolador y el conector 10Base T se sitúa el interfaz PHY.
  - Se comunican mediante un protocolo:
    - MII: **M**edia **I**ndependient **I**nterface (16 señales)
    - RMII: reduced MII (8 señales)
  - Cada tipo de circuito PHY tiene su propio identificador



Typical Application

Nivel físico

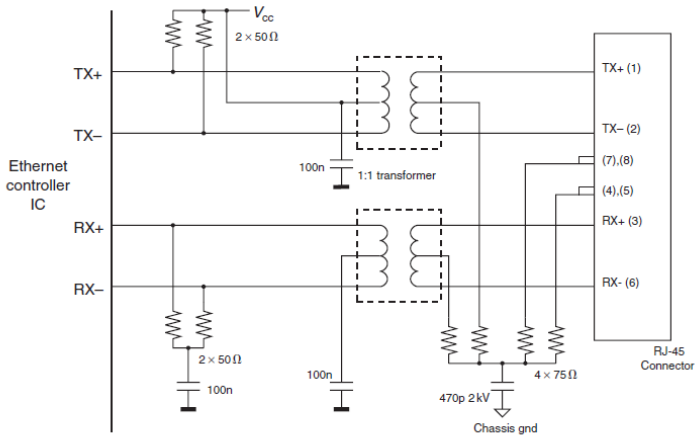


Figure 5-3: Line interface circuit

Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Nivel físico

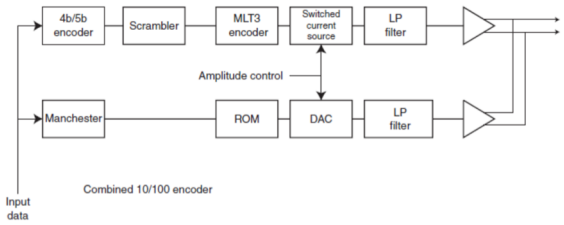


Figure 5-4: Combined 10/100 transmitter section

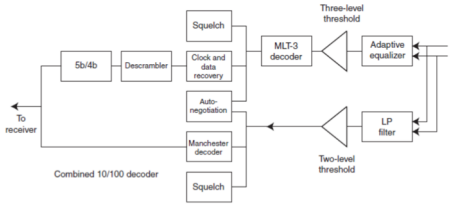
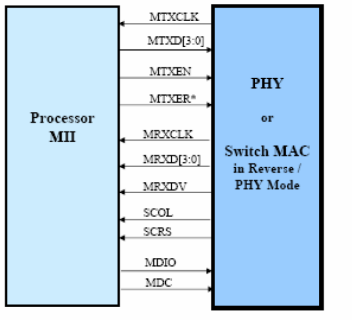


Figure 5-5: Combined 10/100 receiver

Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Interfaz física

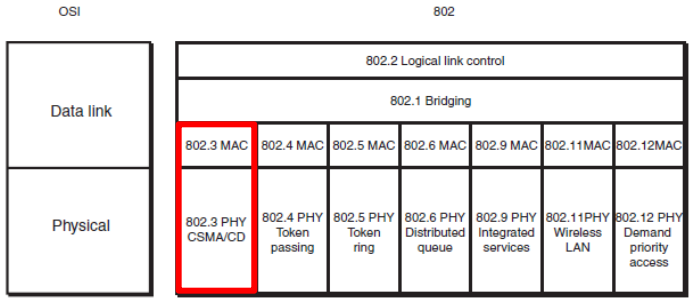


* Tie TXER input pin to ground if MAC does not support transmit error

Manufacturer	Part Number(s)
Broadcom	BCM5221
ICS	ICS1893
Intel	LXT971A
LSI Logic	L80223, L80225, L80227
Micrel	KS8721
National	DP83847, DP83846, DP83843
SMSC	LAN83C185

Nivel II

- Se emplea Ethernet II (en algunos casos IEEE 802.3)
  - Se diferencian en algunos campos de la trama de datos

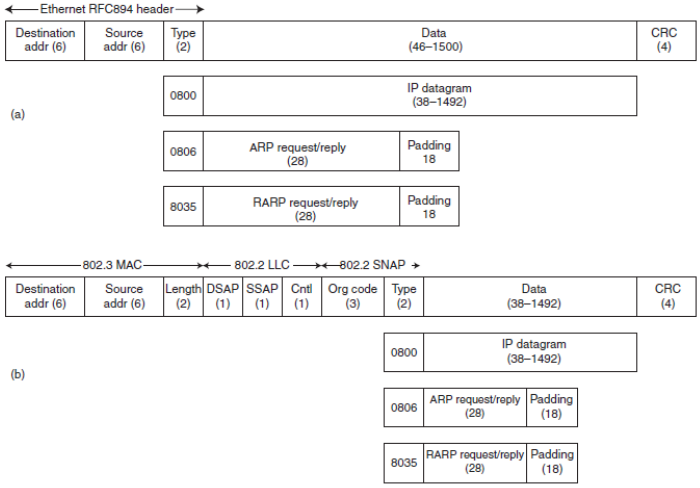


Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Nivel II: Tramas Ethernet y 802.3

- Aspectos básicos:
  - El cableado y conector básico es el 10baseT, especificado en el estándar 802.3.
  - Los sistemas 10Mbits, utilizan codificación Manchester
  - Las tramas Ethernet ocupan un mínimo de 64 bytes y un máximo de 1518bytes.
  - La información útil de una trama se denomina 'data payload'.
  - Cada dispositivo tiene una dirección MAC única de 6 bytes
    - Esta dirección puede venir grabada en flash o ser configurable.
    - Puede solicitarse en http://standards.ieee.org/regauth/oui/forms/index.html.
    - Las direcciones que comienzan en 1 son tramas multicast

Nivel II: Tramas Ethernet y 802.3



Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Nivel II: Módulo Ethernet LPC17xx

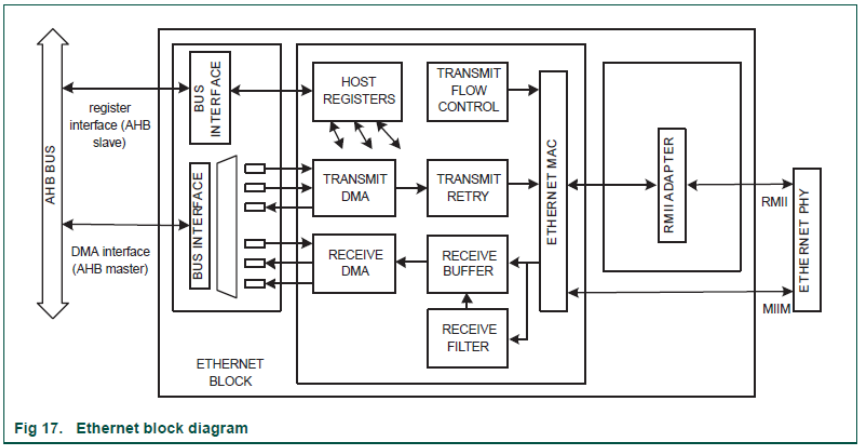


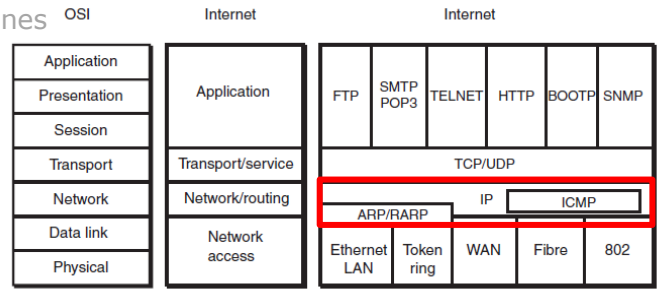
Fig 17. Ethernet block diagram

Nivel II: Módulo Ethernet LPC17xx

- Puede trabajar a 10/100 Mbps.
- Recibe y envía las tramas Ethernet
- Gestiona al dispositivo PHY a través de protocolo RMII
- Realiza la gestión de errores
- Emplea DMA para agilizar el intercambio de datos
- Gestionable por interrupción.
- Suele disponerse de una librería de gestión
  - Keil proporciona EMAC.c

Modelo de red

- La implementación básica de TCPIP requiere:
  - Nivel físico (un chip específico)
  - Ethernet II (módulo del micro)
  - IP
  - ARP
  - ICMP
  - TCP/UDP
  - Aplicaciones



Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Nivel III: IP

- Cada equipo tiene una dirección IP:
  - 4 bytes IPv4 o 16 bytes IPv6.
  - Puede ser estática o dinámica (en este caso proporcionada por un servicio DHCP)
- Para determinar si un equipo está conectado en la misma LAN o no, se dispone de una máscara de 4 bytes.
- Las tramas entre equipos de la misma LAN se intercambian directamente.
- Las tramas para equipos de distinta LAN, se envían al Gateway.

Nivel III: trama IP

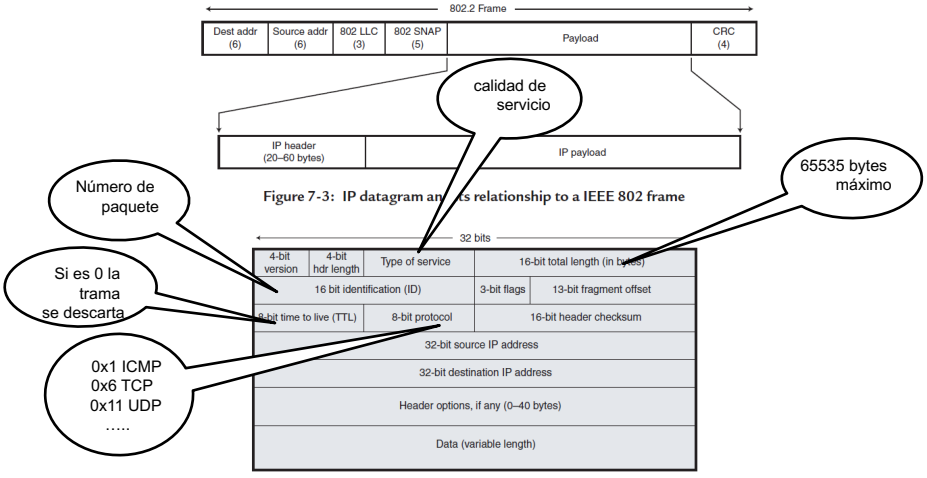
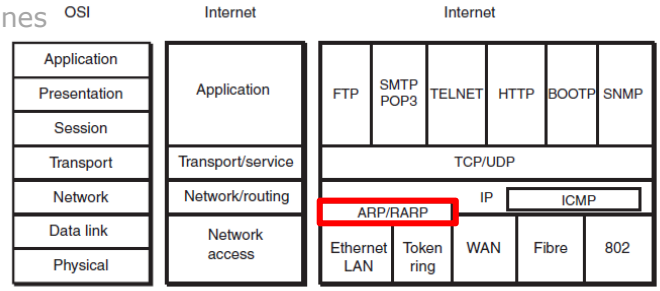


Figure 7-4: IP header format

Modelo de red

- La implementación básica de TCPIP requiere:
  - Nivel físico (un chip específico)
  - Ethernet II (módulo del micro)
  - IP
  - ARP
  - ICMP
  - TCP/UDP
  - Aplicaciones



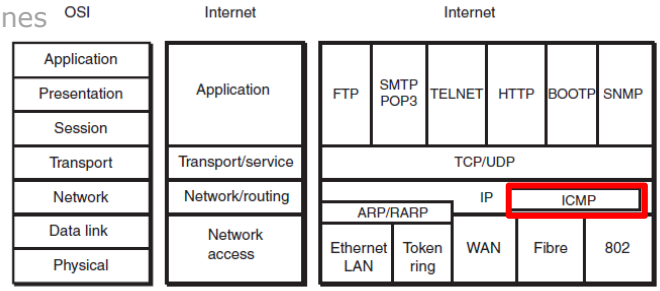
Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

ARP: address resolution protocol

- Permite conocer las direcciones Ethernet de los equipos conectados en la misma LAN, a partir de su dirección IP.
- Cada equipo almacena una tabla con la lista de equipos locales (MAC-IP).
  - Cuando hay que enviar información a un equipo local cuya MAC no está en la tabla, se envía un ARP a la dirección de broadcast (0xFFFFFFFFFFFF).
  - El equipo cuya IP coincida con la contenida en la trama, devuelve una trama ARP con su MAC.

Modelo de red

- La implementación básica de TCPIP requiere:
  - Nivel físico (un chip específico)
  - Ethernet II (módulo del micro)
  - IP
  - ARP
  - ICMP
  - TCP/UDP
  - Aplicaciones



Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Nivel III: ICMP

- Internet Control Message Protocol
- Servicio de mantenimiento
- Ejemplo de uso: ping

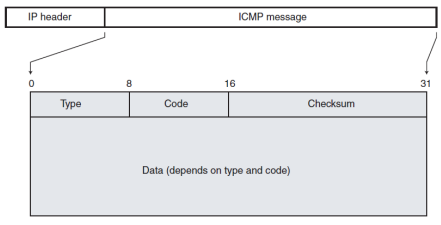


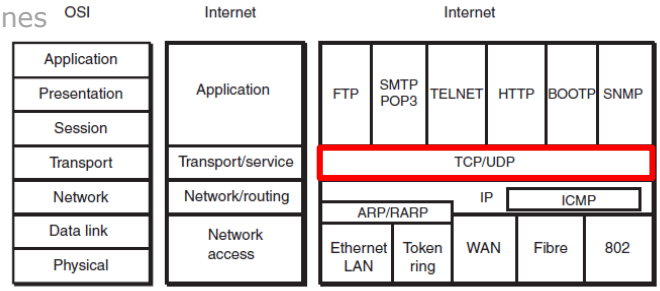
Figure 7-9: ICMP message format

Table 7-2: ICMP type = 3 (destination unreachable) codes

Code	Description
0	Network unreachable
1	Host unreachable
2	Protocol unreachable
3	Port unreachable
4	Fragmentation needed
5	Source route failed
6	Destination network unknown
7	Destination host unknown
8	Source host isolated
9	Destination network prohibited
10	Destination host prohibited

Modelo de red

- La implementación básica de TCPIP requiere:
  - Nivel físico (un chip específico)
  - Ethernet II (módulo del micro)
  - IP
  - ARP
  - ICMP
  - TCP/UDP
  - Aplicaciones



Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Transporte: UDP

- UDP: User Datagram Protocol.
- Protocolo de transporte no orientado a conexión.
- Se manda la trama y si llega bien y si no…
- Dentro de un mismo nodo, se emplean distintos puertos (equivalente a las extensiones telefónicas)
- La combinación IP-puerto establece una comunicación unidireccional y se denomina socket
- Ideal para intercambiar datos entre aplicaciones.
  - El payload contendrá los datos a intercambiar.
  - La interpretación de los mismos deberán establecerse al diseñar las aplicaciones.

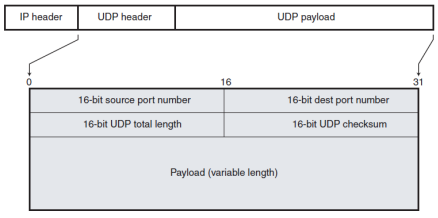


Figure 7-10: UDP datagram format

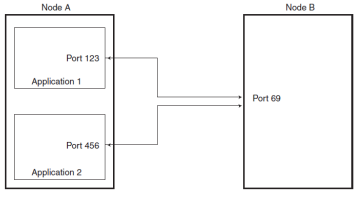


Figure 7-11: UDP ports

Transporte: TCP

- Permite establecer conexiones bidireccionales punto a punto.
- Conexión fiable
- Orientada a conexión.
  - Abrir sesión
  - Intercambio de datos
  - Control del flujo y de errores
  - Cierre de sesión.

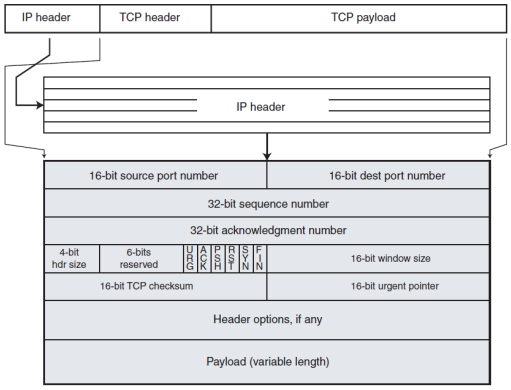


Figure 7-13: TCP Header

Transporte: TCP

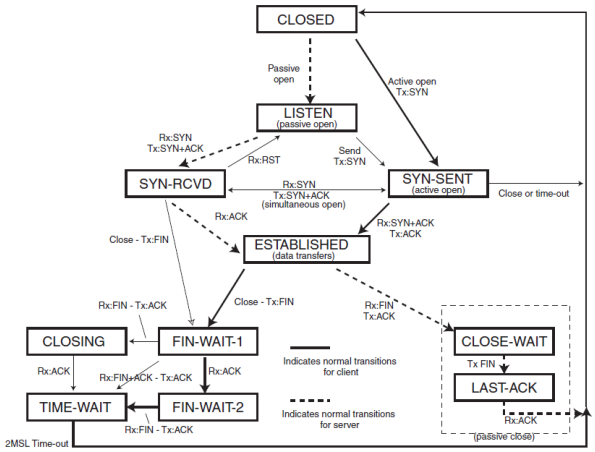
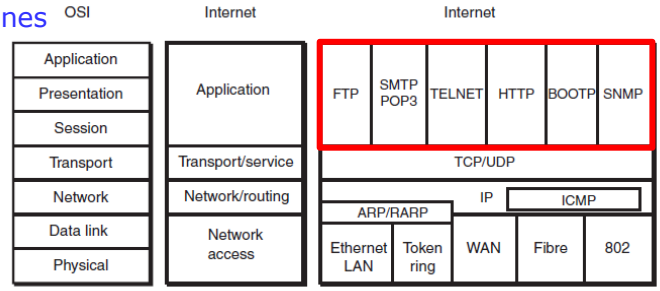


Figure 7-15: TCP state table

Modelo de red

- La implementación básica de TCPIP requiere:
  - Nivel físico (un chip específico)
  - Ethernet II (módulo del micro)
  - IP
  - ARP
  - ICMP
  - TCP/UDP
  - Aplicaciones



Source: TCP/IP Embedded Internet Applications. Embedded Technology. Edward Insam. Elsevier

Nivel aplicación

- Sobre TCP:
  - Telnet (puerto 23)
  - SMTP (puerto 25)
  - POP3 (puerto 110)
  - HTTP (puerto 80)
  - FTP (puerto 21)
- Sobre UDP
  - TFTP (trivial FTP; puerto 69)
  - DHCP (broadcast al puerto 67)
  - RTP (real time protocol; puerto negociable)
  - NTP y SNTP (puerto 123)

Port Number	Protocol
20	FTP Data
21	FTP Control
23	Telnet
25	SMTP
80	HTTP
110	POP3

Implementación básica de un stack TCP/IP

- Módulo de gestión del dispositivo PHY
- Módulo de gestión Ethernet
  - Recibir y enviar tramas Ethernet
  - Gestionar ARP
- Módulo IP
  - Recibir y enviar tramas IP
  - Gestionar ICMP
- Módulo transporte
  - Gestión UPD
  - Gestión TCP
- Soporte a las aplicaciones que se requieran
  - HTTP, etc.

TCPnet de Keil

- Componentes básicos:
  - Net_Config.c
    - Fichero que permite configurar la biblioteca de funciones de de comunicación.
      - Dirección MAC, IP, protocolos activos, reserva de memoria, ...
  - EMAC.C
    - Fichero que implementa las funciones que comunican las librerías de comunicaciones con el módulo Ethernet del microcontrolador y el adaptador externo.
  - TCP_CM3.lib
    - Biblioteca de funciones de comunicación
- Para depurar las comunicaciones por el puerto serie
  - TCPD_CM3.lib
    - En lugar de TCP_CM3.lib
  - Net_Debug.c
    - Para configurar el nivel de depuración

TCPnet de Keil

- Recursos necesarios:

Demo Example	ROM Size (KB)	RAM Size (KB)
HTTP Server (without RTX Kernel)	25.6	20.0
Telnet Server	20.4	20.0
TFTP Server	20.6	24.7
SMTP Server	16.7	19.5
DNS Resolver	12.7	19.6

TCPnet

- Código básico sin RTX

```
void timer_poll ()  {

  if (100mstimeout)  {
    timer_tick ();
    tick = __TRUE;
  }
}

int main (void)  {

  timer_init ();
  init_TcpNet ();
  while (1)  {
    timer_poll ();
    main_TcpNet ();
  }
}
```

Esta función debe llamarse cada 100ms

Cualquier timer Para contar 100 ms

Esta función debe llamarse frecuentemente

init_TcpNet()
- Inicializa la pila de comunicaciones

timer_tick()
- Debe de ser llamado cada 100ms (configurable en Net_Congig.c)
- Activa un flag que sirve para implementar temporizadores SW

main_TcpNet()
- Debe ser llamada con cierta frecuencia.
- Gestiona los timeouts
- Mantiene la tabla de ARP
- Comprueba si hay nuevos datos recibidos.

TCPnet

Código básico con RTX

```
U64 tcp_stack [800/8];

void init (void) __task  {
   init_TcpNet ();
   os_tsk_create (timer_task, 30);
   os_tsk_create_user (tcp_task, 0, &tcp_stack, sizeof (tcp_stack));
   os_tsk_delete_self ();
}

__task void timer_task (void)  {
   os_itv_set (10);
   while (1)  {
      timer_tick ();
      os_itv_wait ();
   }
}

__task void tcp_task (void)  {
   while (1)  {
      main_TcpNet ();
      os_tsk_pass ();
   }
}
```

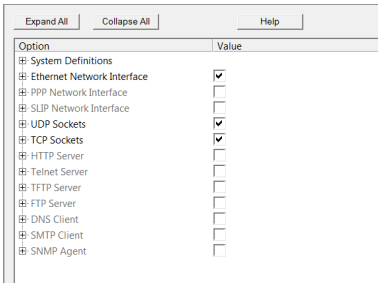
Tarea de menor prioridad

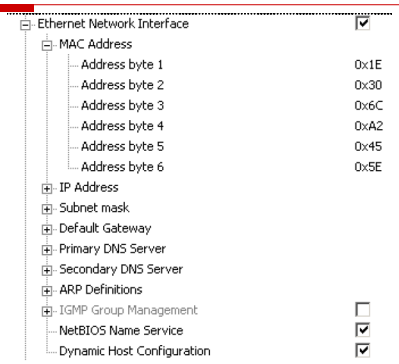
Esta función debe llamarse cada 100ms

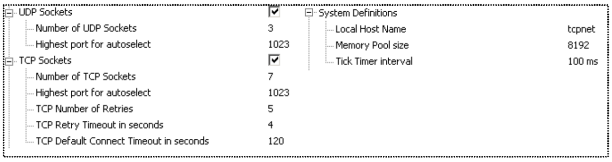
Esta función debe llamarse frecuentemente

TCPnet

Configuración







TCPnet

- Adaptar EMAC al dispositivo PHY utilizado:
  - Modificar EMAC_LPC17xx.c
  - Modificar emac_lpc17xx.h

```
/* PHY Device Address is in bits 12:8 and this value is OR'ed with addresses */
/* Most PHY Devices have pull-up resistors on the PHYAD0 pin to force        */
/* address 1 on RESET as address 0 is reserved */
/* DP83848 PHY Device Address = [00001]00000000 = 0x0100*/
/* LAN8720 PHY Device Address = [00001]00000000 = 0x0100*/
#define DP83848C_DEF_ADR    0x0100      /* DP83848 PHY device address        */
#define LAN8720_DEF_ADR     0x0100      /* LAN8720 PHY device address        */
#define PHY_DEF_ADR         LAN8720_DEF_ADR /* PHY device address                */

/* DP83848 PHY Identifier = 00100000000000000101110010010000 = 0x20005C90    */
/* LAN8720 PHY Identifier = 00000000000011111000000011110000 = 0x0007C0F0    */
/* Final 4 bits are silicon revision number and are AND'ed away when checking*/
#define DP83848C_ID         0x20005C90      /* DP83848 PHY Identifier            */
#define LAN8720_ID          0x0007C0F0      /* LAN8720 PHY Identifier            */
#define PHY_DEVICE_ID       LAN8720_ID      /* PHY Identifier    */
```

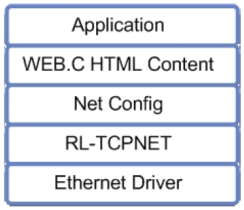
Usar nombre genérico

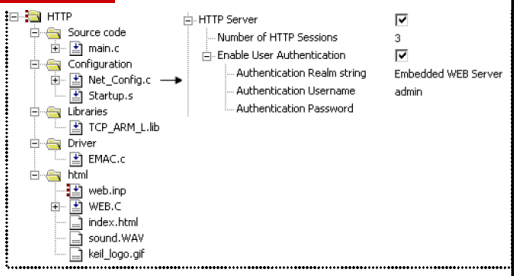
Usar nombre genérico

Se obtiene del manual del cto PHY

TCPnet: HTTP server

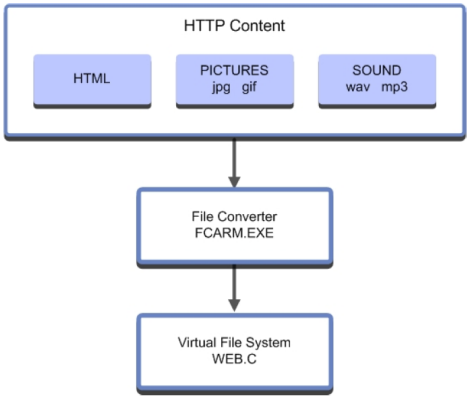
- Para páginas estáticas:





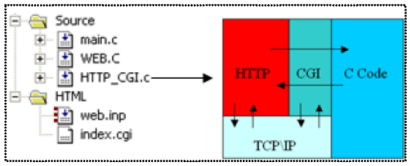
- El contenido de las páginas se guarda en flash
  - Usando la librería RL-Flash se puede actualizar el contenido de la web remotamente.
- El contenido de WEB.c se genera con la utilidad fcarm.exe.
  - El fichero web.inp contiene el listado de los archivos que formarán la web

TCPnet: HTTP server



TCPnet: HTTP server

- Common Gateway Interface:
  - Permite crear contenido dinámico
  - En el fichero HTTP_CGI.c se debe incluir la función *cgi_func()* donde el usuario debe codificar la generación del contenido dinámico.
  - Los ficheros con contenido dinámico tienen que tener extensión .cgi y cada línea del mismo debe comenzar con un comando

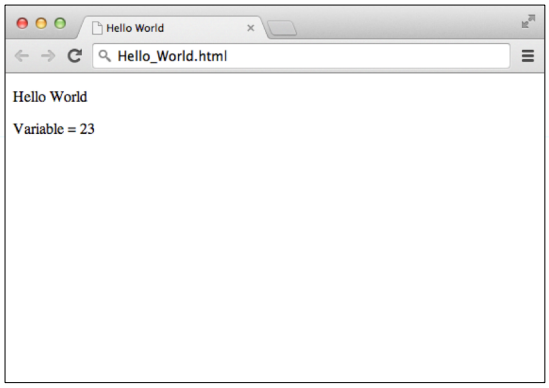


Command	Description
i	Commands the script interpreter to **include a file** from the virtual file system and to output the content on the web browser.
t	Commands that the **line of text** that follows is to be output to the browser.
c	Calls a C function from the **HTTP_CGI.c** file. The function may be followed by the line of text which is passed to **cgi_func()** as a pointer to an environment variable.
#	This is a comment line and is ignored by the interpreter.
.	Denotes the last script line.

Ejemplo "Hello World"

- Ejemplo Hello World

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta content="text/html; charset=windows-1252" http-equiv="content-type">
5      <title></title>
6    </head>
7    <body>
8      <p>Hello World</p>
9      <p>Variable = 23</p>
10     <p><br>
11     </p>
12   </body>
13 </html>
```



Ejemplo "Hello World"

- Ejemplo Hello World

header.inc

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta content="text/html; charset=windows-1252" http-equiv="content-type">
5      <title></title>
6    </head>
7    <body>
8      <p>Hello World</p>
9      <p>Variable = 23</p>
10     <p><br>
11     </p>
12   </body>
13 </html>
```

Footer.inc

```
#Fichero Hello_World.cgi
i header.inc
t <p>Variable = 23</p>
i footer.inc
.
```

Hello_World.cgi

Ejemplo "Hello World"

```
/*--------------------------- cgi_func ---------------------------------------*/

U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
  /* Parameters:                                                              */
  /*   env    - environment variable string                                   */
  /*   buf    - HTTP transmit buffer                                          */
  /*   buflen - length of this buffer (500-1400 bytes - depends on MSS)       */
  /*   pcgi   - pointer to session local buffer used for repeated loops       */
  /*            This is a U32 variable - size is 4 bytes. Value is:           */
  /*            - on 1st call = 0                                             */
  /*            - 2nd call    = as set by this function on first call         */
  U32 len = 0;

  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 'd' :
      switch (env[2]) {
        case '1':
          len = sprintf((char *)buf,"<p>Variable = %d</p>", variable);
          break;
      } //  switch (env[2])
      break;
  } // switch (env[0])
  return ((U16)len);
}
```

```
#Fichero Hello_World.cgi
i header.inc
c d 1
i footer.inc
.
```

Ejemplo "Hello World"

```
/*--------------------------- cgi_func ---------------------------------------*/

U16 cgi_func (U8 *env, U8 *bu
  /* Parameters:
  /*   env    - environment v
  /*   buf    - HTTP transmit
  /*   buflen - length of thi
  /*   pcgi   - pointer to se
  /*            This is a U32
  /*            - on 1st call
  /*            - 2nd call
  U32 len = 0;

  switch (env[0]) {
    /* Analyze the environmen
    /* at position 2. What yo
    case 'd' :
      switch (env[2]) {
        case '1':
          len = sprintf((char *)buf,"<p>Variable = %d</p>", variable);
          break;
      } //  switch (env[2])
      break;
  } // switch (env[0])
  return ((U16)len);
}
```

env = &env[0]

c d 1

env[0] = 'd' env[2] = '1'

buf → ** Variable = 18 **

```
#Fichero Hello_World.cgi
i header.inc
c d 1
i footer.inc
.
```

Ejemplo "Hello World"

```
/*--------------------------- cgi_func ---------------------------------------*/

U16 cgi_func (U8 *env, U8 *buf, U16 buflen, U32 *pcgi) {
  /* Parameters:                                                              */
  /*   env    - environment variable string                                   */
  /*   buf    - HTTP transmit buffer                                          */
  /*   buflen - length of this buffer (500-1400 bytes - depends on MSS)       */
  /*   pcgi   - pointer to session local buffer used for repeated loops       */
  /*            This is a U32 variable - size is 4 bytes. Value is:           */
  /*            - on 1st call = 0                                             */
  /*            - 2nd call    = as set by this function on first call         */
  U32 len = 0;

  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 'd' :
      switch (env[2]) {
        case '1':
          len = sprintf((char *)buf,(const char *)&env[4],variable);
          break;
        } //  switch (env[2])
        break;
  } // switch (env[0])
  return ((U16)len);
}
```

```
#Fichero Hello_World.cgi
i header.inc
c d 1 <p> Variable = %d</p>
i footer.inc
.
```

Ejemplo "Hello World"

```
/*--------------------------- cgi_func ---------------------------------------*/

U16 cgi_func (U8 *env, U8 *b
  /* Parameters:
  /*   env    - environment
  /*   buf    - HTTP transmi
  /*   buflen - length of th
  /*   pcgi   - pointer to s
  /*            This is a U3
  /*            - on 1st cal
  /*            - 2nd call
  U32 len = 0;

  switch (env[0]) {
    /* Analyze the environme
    /* at position 2. What y
    case 'd' :
      switch (env[2]) {
        case '1':
          len = sprintf((char *)buf,(const char *)&env[4],variable);
          break;
        } //  switch (env[2])
        break;
  } // switch (env[0])
  return ((U16)len);
}
```

env = &env[0]
&env[4]

**c d 1 Variable = %d **

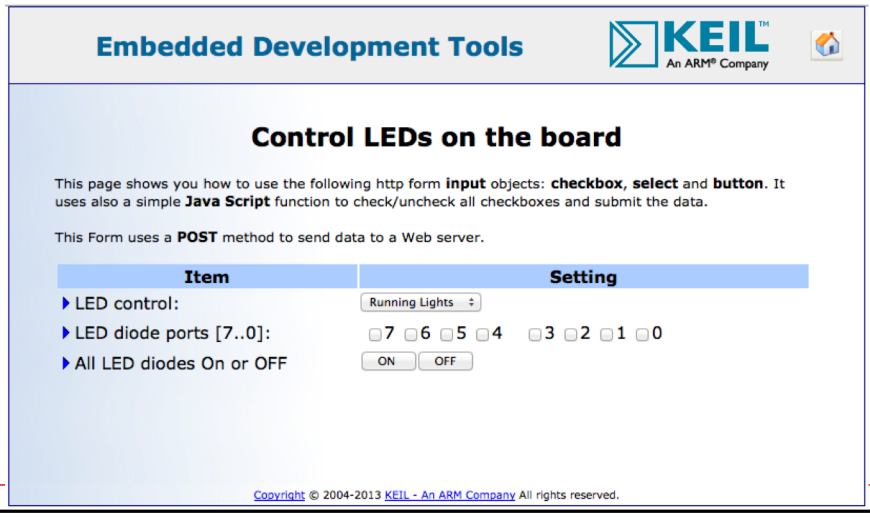
env[0] = 'd' env[2] = '1'

buf → ** Variable = 18 **

```
#Fichero Hello_World.cgi
i header.inc
c d 1 <p> Variable = %d</p>
i footer.inc
.
```

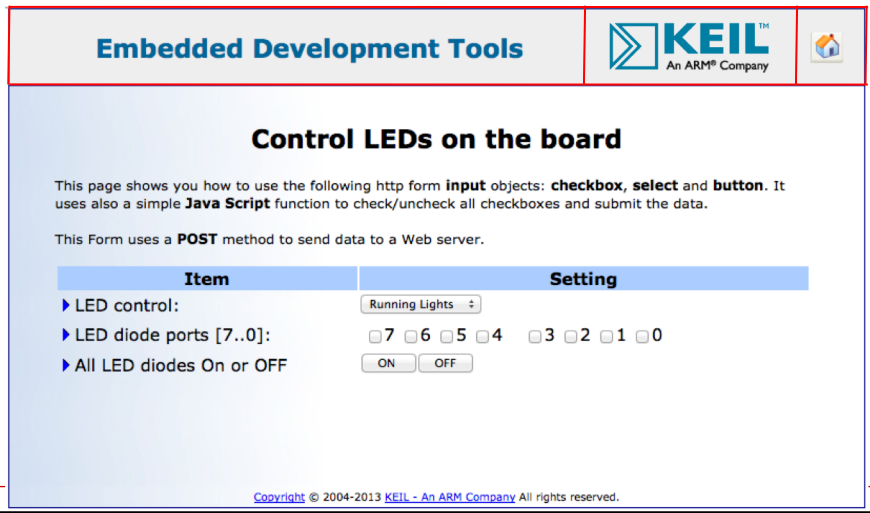
Ejemplo http-demo

- Acceso a http://192.168.5.5/leds.cgi



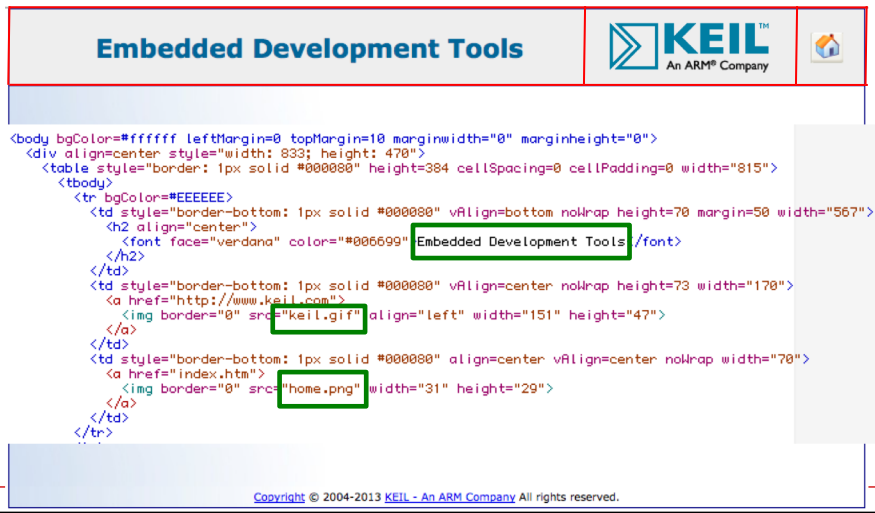
Ejemplo http-demo

- Acceso a http://192.168.5.5/leds.cgi



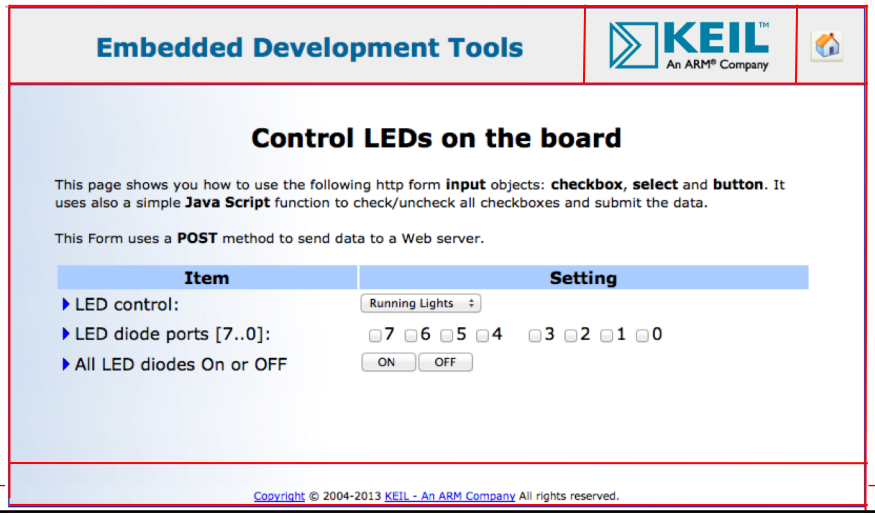
Ejemplo http-demo

- Acceso a http://192.168.5.5/leds.cgi



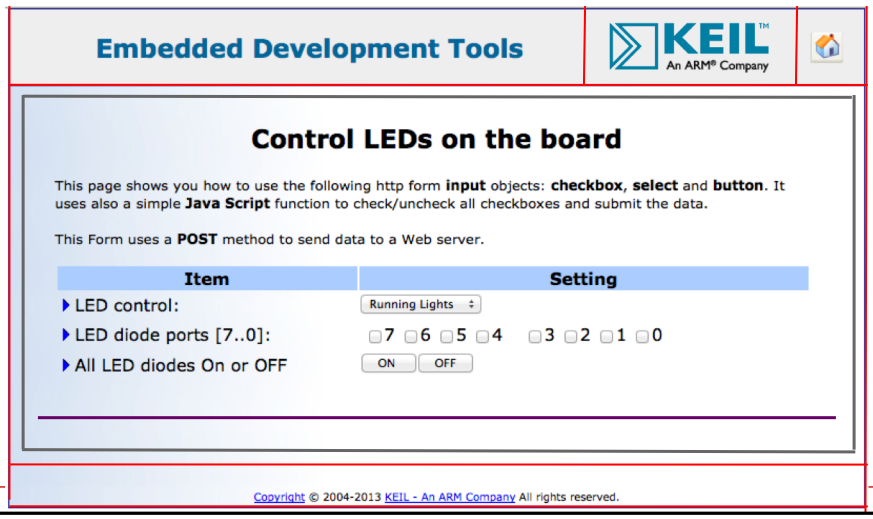
Ejemplo http-demo

- Acceso a http://192.168.5.5/leds.cgi



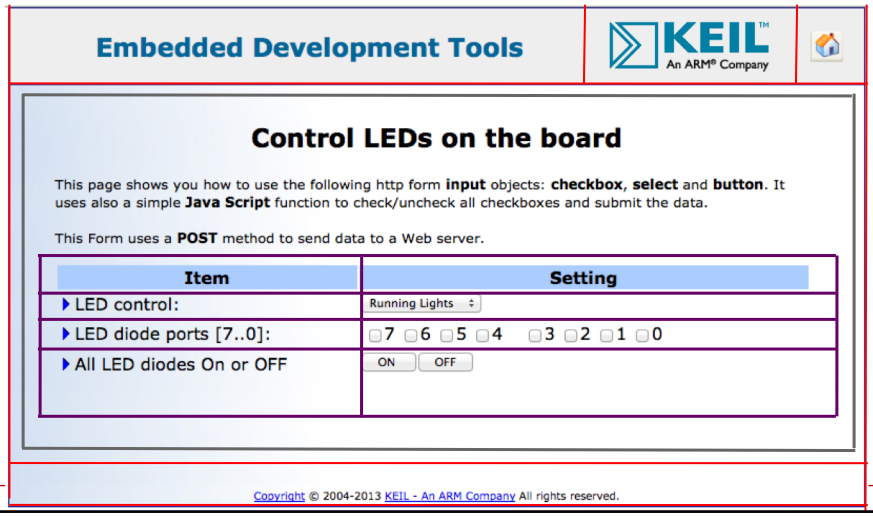
Ejemplo http-demo

- Acceso a http://192.168.5.5/leds.cgi



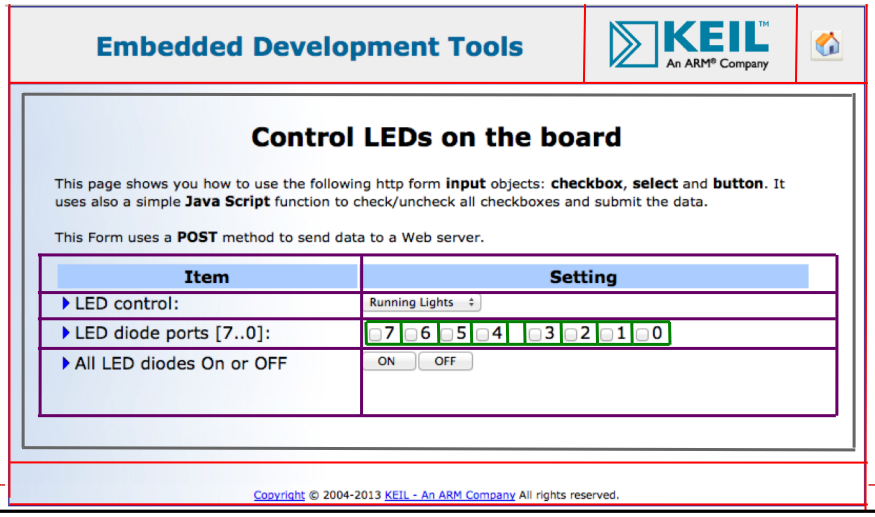
Ejemplo http-demo

- Acceso a http://192.168.5.5/leds.cgi



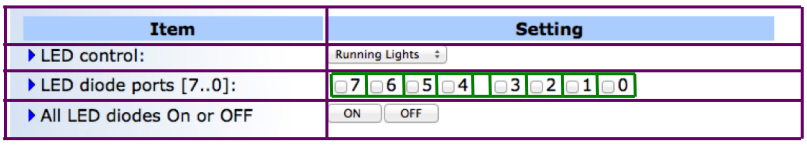
Ejemplo http-demo

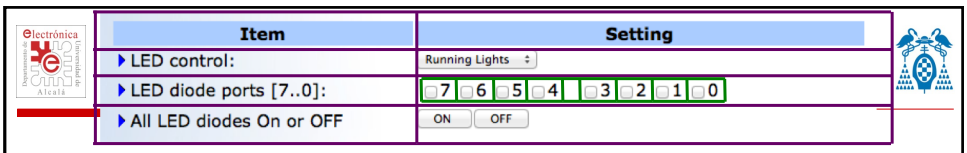
- Acceso a http://192.168.5.5/leds.cgi



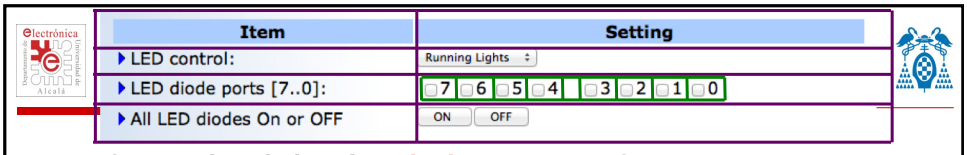
TCPnet: HTTP server

- Ejemplo de envío de datos al servidor utilizando POST
  - En el fichero leds.cgi de HTTP_Demo.






```
<form action=leds.cgi method=post name=form1>
  <input type=hidden value="led" name=pg>
   ...
  <select name="ctrl" onchange="submit();">
          <option >Browser</option>
          <option selected>Running Lights</option>
        </select>
 <td><input type=checkbox name=led7 OnClick="submit();" >7</td>
 <td><input type=checkbox name=led6 OnClick="submit();" >6</td>
 <td><input type=checkbox name=led5 OnClick="submit();" >5</td>
 <td><input type=checkbox name=led4 OnClick="submit();" >4</td>
 <td><input type=checkbox name=led3 OnClick="submit();" >3</td>
 <td><input type=checkbox name=led2 OnClick="submit();" >2</td>
 <td><input type=checkbox name=led1 OnClick="submit();" >1</td>
 <td><input type=checkbox name=led0 OnClick="submit();" >0</td>
 ...
 <input type=button value="ON" onclick="AllSW(true)">
 <input type=button value="OFF" onclick="AllSW(false)">
...
</form>
```




```
<form action=leds.cgi method=post name=form1>
  <input type=hidden value="led" name=pg>
   ...
  <select name="ctrl" onchange="submit();">
          <option >Browser</option>
          <option selected>Running Lights</option>
        </select>
 <td><input type=checkbox name=led7 OnClick="submit();" >7</td>
 <td><input type=checkbox name=led6 OnClick="submit();" >6</td>
 <td><input type=checkbox name=led5 OnClick="submit();" >5</td>
 <td><input type=checkbox name=led4 OnClick="submit();" >4</td>
 <td><input type=checkbox name=led3 OnClick="submit();" >3</td>
 <td><input type=checkbox name=led2 OnClick="submit();" >2</td>
 <td><input type=checkbox name=led1 OnClick="submit();" >1</td>
 <td><input type=checkbox name=led0 OnClick="submit();" >0</td>
 ...
 <input type=button value="ON" onclick="AllSW(true)">
 <input type=button value="OFF" onclick="AllSW(false)">
...
</form>
```

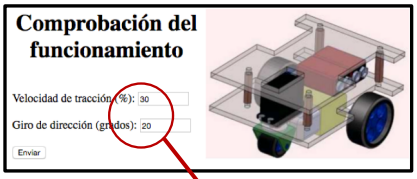
```
function AllSW(st) {
  for(i=0;i<document.form1.length;i++) {
    if(document.form1.elements[i].type=="checkbox") {
      document.form1.elements[i].checked=st;
    }
  }
  document.form1.submit();
}
```

TCPnet: HTTP server

- Paso de información: GET y POST
  - GET: para enviar información que no cambia el contenido del servidor web (ej: petición de consultas)
    - los datos enviados por GET, se pasan a la función ***cgi_process_var()*** que debe estar convenientemente codificada en el fichero *HTTP_CGI.c*
  - POST: para enviar información que cambia alguna variable o contenido de la aplicación empotrada.
    - los datos enviados por POST, se pasan a la función ***cgi_process_data()*** que debe estar convenientemente codificada en el fichero *HTTP_CGI.c*
  - Se analiza a continuación el caso POST. Para GET el procedimiento es similar. Ver documentación de Keil

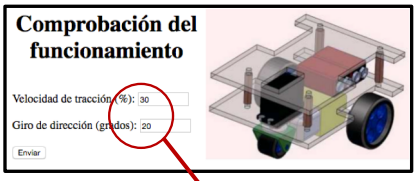
Ejemplo de envío de datos

```
<!DOCTYPE html>
<html>
  <head>
    <meta content="text/html;
          charset=windows-1252"
          http-equiv="content-type">
    <title>Comprobacion funcionamiento robot</title>
  </head>
  <body>
    <table style="width: 100%" border="0">
      <tbody>
        <tr>
          <td>
            <h1 style="text-align: center;">Comprobaci&#243;n del funcionamiento</h1>
            <form method="GET" action="robot.cgi">
              <br> Velocidad de tracci&#243;n (%):
              <input size="10" value="30" name="traccion" type="text">
              <br> Giro de direcci&#243;n (grados):
              <input size="10" value="20" name="direccion" type="text">
              <br> <input value="Enviar" type="submit">
            </form>
          </td>
          <td>
            <img style="width: 421px; height: 311px;" alt="Robot" src="./robot.jpg">
          </td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```



Ejemplo de envío de datos

```
<!DOCTYPE html>
<html>
  <head>
    <meta content="text/html;
          charset=windows-1252"
          http-equiv="content-type">
    <title>Comprobacion funcionamiento robot</title>
  </head>
  <body>
    <table style="width: 100%" border="0">
      <tbody>
```




```
<form method="GET" action="robot.cgi">
  <br> Velocidad de tracci&#243;n (%):
    <input size="10" value="30" name="traccion" type="text">
  <br> Giro de direcci&#243;n (grados):
    <input size="10" value="20" name="direccion" type="text">
  <br> <input value="Enviar" type="submit">
</form>
```

```
        <td>
          <img style="width: 421px; height: 311px;" alt="Robot" src="./robot.jpg">
        </td>
      </tr>
```

http://www.seda. es/robot.cgi?traccion=30&direccion=20

```
  </body>
</html>
```

Ejemplo de envío de datos

```
void cgi_process_var (U8 *qs) {
  U8 *var;
  int velocidad, grados;
  uint8_t flag_traccion = 0;
  uint8_t flag_direccion = 0;

  var = (U8 *)alloc_mem (40);
  do {
    qs = http_get_var (qs, var, 40);
    if (var[0] != 0) {
      if (str_scomp (var, "traccion=") == __TRUE) {
        sscanf ((const char *)&var[9], "%d", &velocidad);
        flag_traccion = 1;
      }
      else if (str_scomp (var, "direccion=") == __TRUE) {
        sscanf ((const char *)&var[9], "%d", &grados);
        flag_direccion = 1
      }
    }
  } while (qs);
  if (flag_traccion && flag_direccion)
    set_movimiento(velocidad, grados);
  free_mem ((OS_FRAME *)var);
}
```

robot.cgi?traccion=30&direccion=20

qs

Ejemplo de envío de datos

```
void cgi_process_var (U8 *qs) {
   U8 *var;
   int velocidad, grados;
   uint8_t flag_traccion = 0;
   uint8_t flag_direccion = 0;

   var = (U8 *)alloc_mem (40);
   do {
      qs = http_get_var (qs, var, 40);
      if (var[0] != 0) {
         if (str_scomp (var, "traccion=") == __TRUE) {
            sscanf ((const char *)&var[9], "%d", &velocidad);
            flag_traccion = 1;
         }
         else if (str_scomp (var, "direccion=") == __TRUE) {
            sscanf ((const char *)&var[9], "%d", &grados);
            flag_direccion = 1
         }
      }
   } while (qs);

   if (flag_traccion && flag_direccion)
      set_movimiento(velocidad, grados);
   free_mem ((OS_FRAME *)var);
}
```

robot.cgi?traccion=30&direccion=20

qs

var → "traccion=30"

&var[9]

velocidad=30"

Ejemplo de envío de datos

```
void cgi_process_var (U8 *qs) {
   U8 *var;
   int velocidad, grados;
   uint8_t flag_traccion = 0;
   uint8_t flag_direccion = 0;

   var = (U8 *)alloc_mem (40);
   do {
      qs = http_get_var (qs, var, 40);
      if (var[0] != 0) {
         if (str_scomp (var, "traccion=") == __TRUE) {
            sscanf ((const char *)&var[9], "%d", &velocidad);
            flag_traccion = 1;
         }
         else if (str_scomp (var, "direccion=") == __TRUE) {
            sscanf ((const char *)&var[7], "%d", &grados);
            flag_direccion = 1
         }
      }
   } while (qs);

   if (flag_traccion && flag_direccion)
      set_movimiento(velocidad, grados);
   free_mem ((OS_FRAME *)var);
}
```

robot.cgi?traccion=30&direccion=20

qs

var → "grados=20"

&var[7]

grados=20"

Ejemplo de envío de datos

```
void cgi_process_var (U8 *qs) {
  U8 *var;
  int velocidad, grados;
  uint8_t flag_traccion = 0;
  uint8_t flag_direccion = 0;

  var = (U8 *)alloc_mem (40);
  do {
    qs = http_get_var (qs, var, 40);
```

http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.kui0062a/rlarm_cgi_process_var.htm

http_get_var

Summary

```
#include <net_config.h>

U8* http_get_var (
    U8*   env,        /* Pointer to a string of environment variables */
    void* ansi,       /* Buffer to store the environment variable value */
    U16   maxlen );   /* Maximum length of environment variable value */
```

Description

The **http_get_var** function processes the string *env*, which contains the environment variables, and identifies where the first variable ends. The function obtains and stores the first variable and its value into the buffer pointed by *ansi*, in ansi format.

The *maxlen* specifies the maximum length that can be stored in the *ansi* buffer. If the decoded environment variable value is longer than this limit, the function truncates it to *maxlen* to fit it into the buffer.

The **http_get_var** function is a system function that is in the RL-TCPnet library. The prototype is defined in net_config.h.

printf, sprintf, sscanf, ...: Cadenas de formato

Fuente Wikipedia: https://es.wikipedia.org/wiki/Printf

Formateador	Salida
%d ó %i	entero en base 10 con signo (int)
%u	entero en base 10 sin signo (int)
%o	entero en base 8 sin signo (int)
%x	entero en base 16, letras en minúscula (int)
%X	entero en base 16, letras en mayúscula (int)
%f	Coma flotante decimal de precisión simple (float)
%lf	Coma flotante decimal de precisión doble (double)
%e	La notación científica (mantisa / exponente), minúsculas (decimal precisión simple ó doble)
%E	La notación científica (mantisa / exponente), mayúsculas (decimal precisión simple ó doble)
%c	caracter (char)
%s	cadena de caracteres (string)

printf, sprintf, sscanf, ...: Cadenas de formato

Fuente Wikipedia: https://es.wikipedia.org/wiki/Printf

Formateador	Salida
%07i	justificado a la derecha, 7 dígitos de largo, sin relleno
%.7i	largo mínimo de 7 dígitos, justificado a la derecha, rellena con ceros
%8.2f	tamaño total de 8 dígitos, con dos decimales
%.*f",x,d)	tamaño predeterminado,**x** numeros decimales
%*.*f",x,y,d)	tamaño igual a **x**, **y** numeros decimales
%s	cadena terminada en null
%5s	primeros cinco caracteres o delimitador
%.5s	primeros cinco caracteres, sin tener en cuenta el delimitador
%20.5s	primeros cinco caracteres, justificados a la derecha, con 20 caracteres de largo
%-20.5s	primeros cinco caracteres, justificados a la izquierda, con 20 caracteres de largo

TCPnet: HTTP server

- POST:

```
#include <net_config.h>

void cgi_process_data (
    U8  code,      /* Type of data in received data buffer. */
    U8* dat,       /* Pointer to the data string from the POST method. */
    U16 len );     /* Number of bytes in the data string. */
```

Code	Data Type	Meaning of *dat pointer*	Meaning of *len*
0	Form data	Pointer to data string returned from the POST method.	Length of data string.
1	Filename	Pointer to a Filename for the http file upload. Filename is a 0-terminated string.	Length of a filename.
2	File data	Pointer to data packet received from the host.	Length of data packet.
3	End of file	*NULL*	Don't care.
4	XML data	Pointer to data string returned from the XML-POST method. A single packet or last packet in xml data stream.	Length of data string.
5	XML data	Pointer to data string returned from the XML-POST method. The same as under 4, but with more xml data to follow.	Length of data string.

TCPnet: HTTP server

- POST:
  - El fichero HTML debe contener:

```
<FORM ACTION=index.htm METHOD=POST NAME=CGI>
 ..
</FORM>
```

  - Para procesar la cadena enviada al servidor empotrado, se usa la función

```
#include <net_config.h>

U8* http_get_var (
    U8*   env,          /* Pointer to a string of environment variables */
    void* ansi,         /* Buffer to store the environment variable value */
    U16   maxlen );     /* Maximum length of environment variable value */
```

  - Hay que llamar a esta función tantas veces como variables de retorno se reciban. Por cada llamada se lee una variable.

HTML

- HTML:
  - Se pueden consultar los elementos del lenguaje HTML en:
    - https://www.w3schools.com/html/default.asp
  - Introducción al HTML (Tutorial)
    - https://www.w3schools.com/html/html_intro.asp
  - Información sobre los formularios y la forma de enviar información de un Navegador al Servidor
    - https://www.w3schools.com/html/html_forms.asp
  - Editor on-line de páginas HTML
    - https://html-online.com/editor/
    - http://www.cubicfactory.com/jseditor/
    - https://html5-editor.net/
    - https://www.w3schools.com/tryit/default.asp

TCPnet: HTTP server

- Ejemplo HTTP_demo (cgi_process_data)

```
/*---------------------------- cgi_process_data -----------------------------*/
void cgi_process_data (U8 code, U8 *dat, U16 len) {
  /* This function is called by HTTP server to process the returned Data    */
  /* for the CGI Form POST method. It is called on SUBMIT from the browser. */
  /* Parameters:                                                            */
  /*   code  - callback context code                                        */
  /*           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    */
  /*   dat   - pointer to POST received data                                */
  /*   len   - received data length                                         */
  U8 passw[12],retyped[12];
  U8 *var,stpassw;

  switch (code) {
    case 0:   /* Url encoded form data received. */
      break;
    default:  /* Ignore all other codes. */
      return;
  }
```

TCPnet: HTTP server

- Ejemplo HTTP_demo (

```
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;
    }
  }
}while (dat);
free_mem ((OS_FRAME *)var);
LED_out (P2);
```

TCPnet: HTTP server

- Además se puede incluir código en Javascript para mejorar la funcionalidad.
- También se puede combinar Javascript con XML (AJAX) para generar páginas fácilmente actualizables.
- Ver ejemplo http_demo en carpeta
  - …keil\ARM\Boards\keil\MCB1700\RL\TCPnet

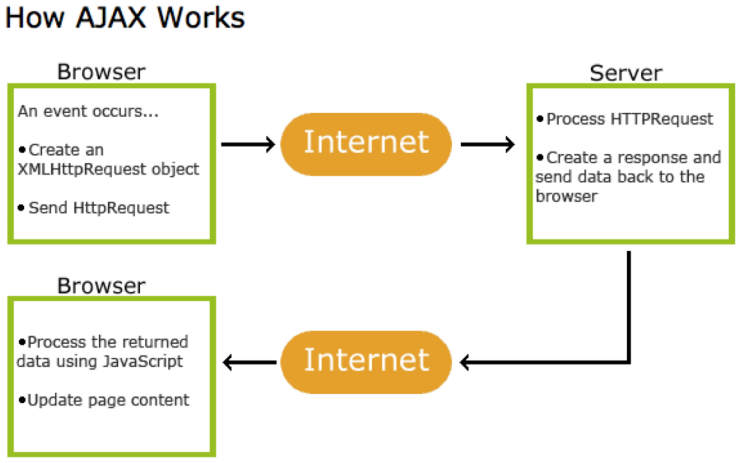
TCPnet: HTTP server

- AJAX: Asynchronous JavaScript and XML
  - No es un lenguaje de programación.
  - Es una forma de intercambiar datos con un servidor actualizando información de una página web sin recargar toda la página.
  - Se utiliza JavaScript en el navegador para el intercambio de datos.
  - Los datos se representan en XML (e**X**tensible **M**arkup **L**anguage)
    - Es un lenguaje de descripción de datos parecido a HTML
    - Diseñado para intercambiar y almacenar datos
    - No está orientado a la visualización de los datos (HTML sí)
    - Las etiquetas no están predefinidas

Fuente: http://www.w3schools.com/ajax/ajax_intro.asp

TCPnet: HTTP server

- Proceso de intercambio de datos con AJAX



Fuente: http://www.w3schools.com/ajax/ajax_intro.asp

TCPnet: HTTP server

- HTTP-DEMO – Lectura de entradas analógicas
  - Organización del código HTML de la página (ad.cgi)

Código Javascript incluído en la página

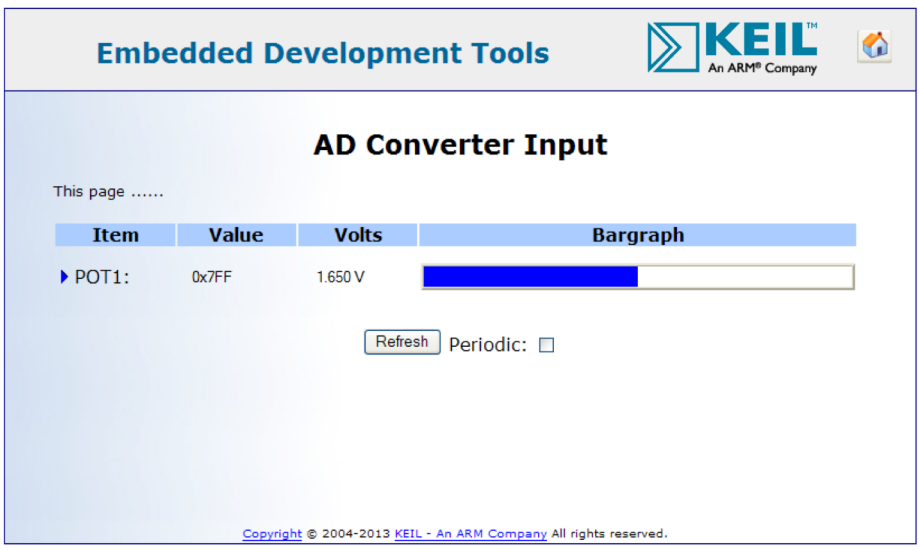
Librerías Javascript no incluidas en la página "xml_http.js"

Cabecera de la página HTML

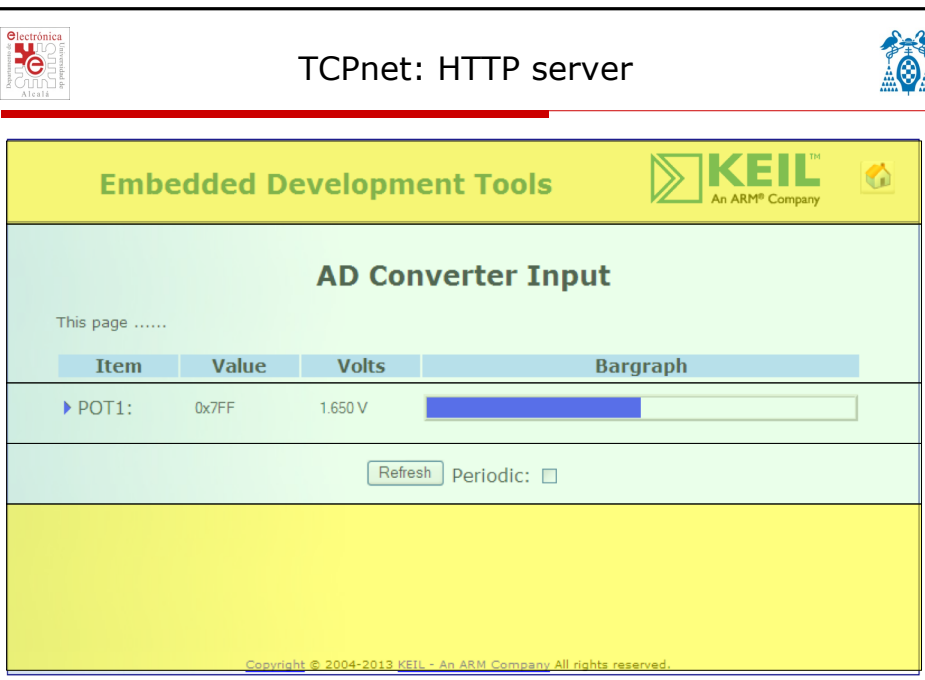
Contenido de la página

Pie de la página

TCPnet: HTTP server



TCPnet: HTTP server



TCPnet: HTTP server

```
<h2 align="center"><br>AD Converter Input</h2>
<p><font size="2">This page ......</font></p>
<form action="ad.cgi" method="post" name="ad">
  <input type="hidden" value="ad" name="pg">
  <table border=0 width=99%><font size="3">
    <tr style="background-color: #aaccff">
      <th width=15%>Item</th>
      <th width=15%>Value</th>
      <th width=15%>Volts</th>
      <th width=55%>Bargraph</th>
    </tr>
      .
      .
      .
    </font>
  </table>
  <p align=center>
  <input type=button value="Refresh" onclick="updateMultiple(formUpdate,plotADGraph)">
  Periodic:
  <input type="checkbox" id="adChkBox" onclick="periodicUpdateAd()">
  </p>
</form>
```

Refresh Periodic: ☐

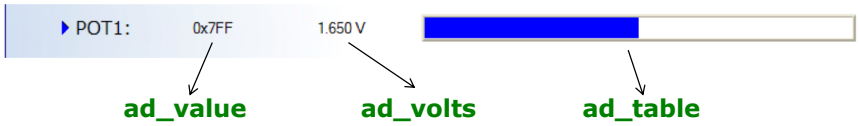
TCPnet: HTTP server

Refresh Periodic: ☐

```
<input type=button value="Refresh" onclick="updateMultiple(formUpdate,plotADGraph)">
Periodic:
<input type="checkbox" id="adChkBox" onclick="periodicUpdateAd()">
```

```
var formUpdate = new periodicObj("ad.cgx", 500);
function plotADGraph() {
 adVal = document.getElementById("ad_value").value;
 numVal = parseInt(adVal, 16);
 voltsVal = (3.3*numVal)/1024;
 tableSize = (numVal*100/1024);
 document.getElementById("ad_table").style.width = (tableSize + '%');
 document.getElementById("ad_volts").value = (voltsVal.toFixed(3) + ' V');
}
function periodicUpdateAd() {
 if(document.getElementById("adChkBox").checked == true) {
  updateMultiple(formUpdate,plotADGraph);
  ad_elTime = setTimeout(periodicUpdateAd, formUpdate.period);
 }
 else
  clearTimeout(ad_elTime);
}
```

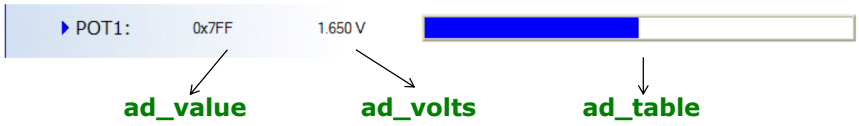
TCPnet: HTTP server



ad_value **ad_volts** **ad_table**

```
<td align="center">
  <input type="text" readonly style="background-color: transparent; border: 0px"
         size="10" id="ad_value" value=" 0x7FF ">
</td>
<td align="center"><input type="text" readonly style="background-color: transparent; border: 0px"
    size="10" id="ad_volts" value=" 1.650 V ">
</td>
<td height=50>
  <table bgcolor="#FFFFFF" border="2" cellpadding="0" cellspacing="0" width="100%">
    <tr>
      <td>
        <table id="ad_table" style="width: 50%" border="0" cellpadding="0" cellspacing="0">
          <tr>
            <td bgcolor="#0000FF"> </td>
          </tr>
        </table>
      </td>
    </tr>
  </table>
</td>
```

TCPnet: HTTP server



ad_value **ad_volts** **ad_table**

```
function plotADGraph() {
 adVal = document.getElementById("ad_value").value;
 numVal = parseInt(adVal, 16);
 voltsVal = (3.3*numVal)/1024;
 tableSize = (numVal*100/1024);
 document.getElementById("ad_table").style.width = (tableSize + '%');
 document.getElementById("ad_volts").value = (voltsVal.toFixed(3) + ' V');
}
```

Cambia los voltios y el tamaño de la barra azul en función del valor leído en hexadecimal

TCPnet: HTTP server

- Estructura del envío de datos
  - Se crea un objeto XMLHttpRequest
  - Se define el puntero a la función que será llamada cuando se reciban los datos
  - Se abre la comunicación con GET o POST
  - Se envían los datos (o como en este caso sólo la solicitud).

```
function updateMultiple(formUpd, callBack, userName, userPassword) {
 xmlHttp = GetXmlHttpObject();
 if(xmlHttp == null) {
  alert("XmlHttp not initialized!");
  return 0;
 }
 xmlHttp.onreadystatechange = responseHandler;
 xmlHttp.open("GET", formUpd.url, true, userName, userPassword);
 xmlHttp.send(null);
```

TCPnet: HTTP server

- Cuando se recibe la respuesta se analizan los posibles errores
  - Si todo correcto se llama a la función de decodificación
    - processResponse()
  - Una vez decodificada la respuesta se llama a la función que ha pasado el usuario como parámetro
    - callBack() → En este ejemplo sería plotADGraph()

```
function responseHandler(){
 if(xmlHttp.readyState == 4) { //response ready
  if(xmlHttp.status == 200) { //handle received data
   var xmlDoc = xmlHttp.responseXML;
   if(xmlDoc == null)
    return 0;
   try {  //catching IE bug
    processResponse(xmlDoc);
   }
   catch(e) {
    return 0;
   }
   /* Callback function for custom update. */
   if (callBack != undefined)
    callBack();
  }
  else if(xmlHttp.status == 401)
   alert("Error code 401: Unauthorized");
  else if(xmlHttp.status == 403)
   alert("Error code 403: Forbidden");
  else if(xmlHttp.status == 404)
   alert("Error code 404: URL not found!");
 }
}
```

TCPnet: HTTP server

```
function processResponse(xmlDoc) {
 textElementArr = xmlDoc.getElementsByTagName("text");
 for(var i = 0; i < textElementArr.length; i++) {
  try {
   elId = textElementArr[i].childNodes[0].childNodes[0].nodeValue;
   elValue = textElementArr[i].childNodes[1].childNodes[0].nodeValue;
   document.getElementById(elId).value = elValue;
  }
  catch(error) {
   if(elId == undefined){
    continue;
   }
   else if(elValue == undefined) {
    elValue = "";
    document.getElementById(elId).value = elValue;
   }
  }
 }
```

```
<form>
  <text>
    <id>ad_value</id>
    <value>0x7FF</value>
  </text>
</form>
.
```

```
t <form>
t <text>
t <id>ad_value</id>
c x<value>0x%03X</value>
t </text>
t </form>
.
```

Para cada elemento <text> se actualiza el objeto de identificador <id> con el valor <value>

TCPnet: HTTP server

```
checkboxElementArr = xmlDoc.getElementsByTagName("checkbox");
for(var i = 0; i < checkboxElementArr.length; i++) {
 try {
  elId = checkboxElementArr[i].childNodes[0].childNodes[0].nodeValue;
  elValue = checkboxElementArr[i].childNodes[1].childNodes[0].nodeValue;
  if(elValue.match("true"))
   document.getElementById(elId).checked = true;
  else
   document.getElementById(elId).checked = false;
 }
 catch(error) {
  if(elId == undefined) {
   continue;
  }
  else if(elValue == undefined) //we leave current state
   continue;
 }
}
```

TCPnet: HTTP server

```
selectElementArr = xmlDoc.getElementsByTagName("select");
for(var i = 0; i < selectElementArr.length; i++) {
 try {
  elId = selectElementArr[i].childNodes[0].childNodes[0].nodeValue;
  elValue = selectElementArr[i].childNodes[1].childNodes[0].nodeValue;
  document.getElementById(elId).value = elValue;
  if(elValue.match("true"))
   document.getElementById(elId).selected = true;
  else
   document.getElementById(elId).selected = false;
 }
 catch(error) {
  if(elId == undefined) {
   continue;
  }
  else if(elValue == undefined) {
   elValue = "";
   document.getElementById(elId).value = elValue;
  }
 }
}
```

TCPnet: HTTP server

```
radioElementArr = xmlDoc.getElementsByTagName("radio");
for(var i = 0; i < radioElementArr.length; i++) {
 try {
  elId = radioElementArr[i].childNodes[0].childNodes[0].nodeValue;
  elValue = radioElementArr[i].childNodes[1].childNodes[0].nodeValue;
  if(elValue.match("true"))
   document.getElementById(elId).checked = true;
  else
   document.getElementById(elId).checked = false;
 }
 catch(error) {
  if(elId == undefined) {
   continue;
  }
  else if(elValue == undefined) //we leave current state
   continue;
 }
}
```

TCPnet: HTTP server

```
/* XMLHttpRequest object specific functions */
function GetXmlHttpObject() { //init XMLHttp object
 var xmlHttp=null;
 try {
  xmlHttp=new XMLHttpRequest(); // Firefox, Opera 8.0+, Safari
 }
 catch (e) {
  try {   // Internet Explorer
   xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch (e) {
   xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
  }
 }
 return xmlHttp;
}

/* Objects templates */
function periodicObj(url, period) {
 this.url = url;
 this.period = (typeof period == "undefined") ? 0 : period;
}
```

TCPnet: Librería para sockets

- Se pueden emplear para construir protocolos de aplicación propios.
- UDP:
  - Establece una comunicación half dúplex entre dos IP y dos puertos.
  - Para abrir el socket:

```
socket_udp = udp_get_socket (0, UDP_OPT_SEND_CS | UDP_OPT_CHK_CS, udp_callback);
if (socket_udp != 0) {
  udp_open (socket_udp, PORT_NUM);
}
```

  - Cuando se reciba un paquete por este puerto, automáticamente se ejecuta la función *udp_callback()*

```
U16 udp_callback (U8 soc, U8 *rip, U16 rport, U8 *buf, U16 len) {
  rip  = rip;
  rport= rport;
  len  = len;

  if (soc != socket_udp) {
    /* Check if this is the socket we are connected to */
    return (0);
  }
  procrec(buf);
  return (0);
}
```

TCPnet: Librería para sockets

- UDP envío:
  - Se abre un socket
  - Se crea un buffer para introducir la información a enviar con *udp_get_buf()*
  - Se manda la información con la función *udp_send*
  - Ejemplo:

```
if (socket_udp != 0) {
  /* Start Connection */
  sendbuf = udp_get_buf (SENDLEN);
  sendbuf[0] = BLINKLED;
  sendbuf[1] = p2val;
  udp_send (socket_udp, Rem_IP, 1001, sendbuf, SENDLEN);
}
```

  - Ver ejemplo LEDSwitch y LEDClient en
    - ….keil\ARM\Boards\keil\MCB1700\RL\TCPnet

TCPnet: Librería para sockets

- TCP:
  - Soporta comunicaciones full dúplex.
  - Configura un equipo como servidor y otro como cliente.
  - Utiliza dos puertos de cada equipo.
  - Garantiza el envío de los datos de paquetes.
    - Existe reconocimiento de paquetes.
    - Se reenvían los paquetes con errores.
    - Los paquetes se envían en el orden correcto.
  - Para establecer una conexión TCP:

```
/* Initialize TCP Socket and start listening */
socket_tcp = tcp_get_socket (TCP_TYPE_SERVER, 0, 10, tcp_callback);
if (socket_tcp != 0) {
  tcp_listen (socket_tcp, PORT_NUM);
}

socket_tcp = tcp_get_socket (TCP_TYPE_CLIENT, 0, 10, tcp_callback);
```

    - O cliente activo (es posible ser cliente y servidor a la vez)

TCPnet: Librería para sockets

- Envío: antes de enviar datos, debe consultar el estado de la conexión y si los datos anteriores han sido confirmados.

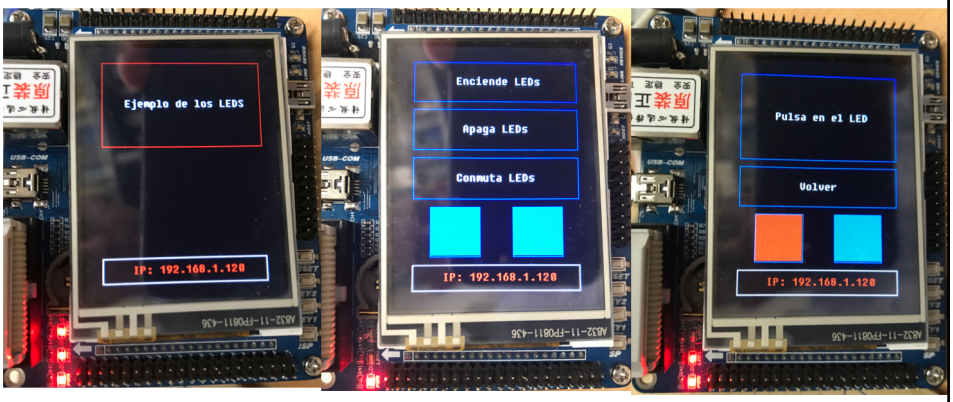
```
void send_data (U8 p2val) {
  U8 *sendbuf;
  U8 p2;

  /* UDP */
  if (socket_udp != 0) {
    /* Start Connection */
    sendbuf = udp_get_buf (SENDLEN);
    sendbuf[0] = BLINKLED;
    sendbuf[1] = p2val;
    udp_send (socket_udp, Rem_IP, 1001, sendbuf, SENDLEN);
  }

  /* TCP */
  if (socket_tcp != 0) {
    /* Start Connection */
    p2 = p2val;
    switch (tcp_get_state(socket_tcp)) {
      case TCP_STATE_FREE:
      case TCP_STATE_CLOSED:
        tcp_connect (socket_tcp, Rem_IP, PORT_NUM, 0);
        break;
      case TCP_STATE_CONNECT:
        if (tcp_check_send (socket_tcp) == __TRUE) {
          sendbuf = tcp_get_buf(SENDLEN);
          sendbuf[0] = BLINKLED;
          sendbuf[1] = p2;
          tcp_send (socket_tcp, sendbuf, SENDLEN);
        }
        break;
    }
  }
}
```

TCPnet: Librería para sockets

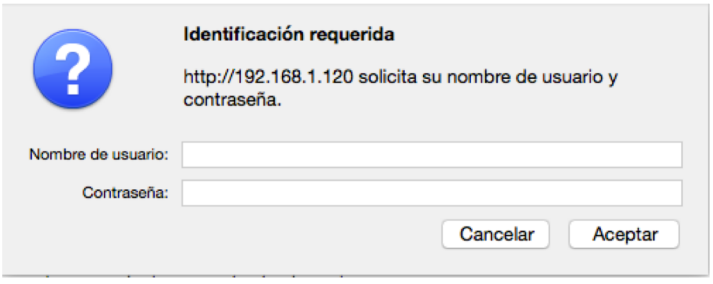
- En el caso de recibir datos debe escribirse la función tcp_callback

```
U16 tcp_callback (U8 soc, U8 evt, U8 *ptr, U16 par) {
  /* This function is called by the TCP module on TCP event */
  /* Check the 'Net_Config.h' for possible events.          */
  par = par;

  if (soc != socket_tcp) {
    return (0);
  }

  switch (evt) {
    case TCP_EVT_DATA:
      /* TCP data frame has arrived, data is located at *par1, */
      /* data length is par2. Allocate buffer to send reply.   */
      procrec(ptr);
      break;

    case TCP_EVT_CONREQ:
      /* Remote peer requested connect, accept it */
      return (1);

    case TCP_EVT_CONNECT:
      /* The TCP socket is connected */
      return (1);
  }
  return (0);
}
```

Ejemplo Menu_LEDs_HTTP



Ejemplo Menu_LEDs_HTTP



Referencias

- TCP/IP Embedded Internet Applications. Edward Insam. Elsevier.
- Getting Started. Builing Applications with RL-ARM. Keil 2009.