

Monitorización de Procesos vía Internet

Idea de Proyecto

Nuestra idea consiste en realizar un sistema embebido que monitorice un proceso automático vía internet.

El sistema debe poder tomar medidas desde los sensores disponibles, crear un informe, y enviarlo por el puerto ethernet. Este informe podrá ser personalizado a través de una HMI.

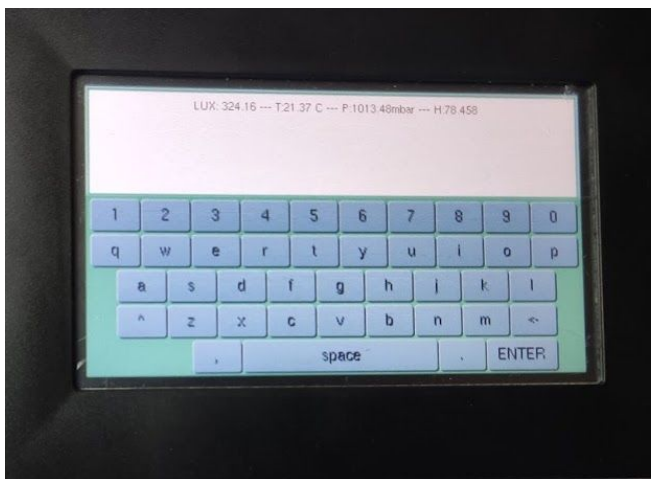
La HMI contará de una consola, y un teclado táctil. Esto permitirá enviar comandos a nuestro sistema, que al ser analizados, permitirán una alta personalización del informe periódico.

De esta forma, nuestro sistema se podría montar en varios sistemas con carácter general, y adaptar el informe al proceso que queramos medir.

Inicialmente vamos a medir la temperatura, presión, humedad y luminosidad del entorno, y enviar informes periódicos cada aproximadamente 5s.

Este informe sufrirá codificación URL y se enviará a una API de telegram, haciendo que se muestre en un chat de texto normal.

Como hemos dicho, todo ello será configurable desde la HMI, permitiendo modificar el nombre del dispositivo, las variables que mostramos, el nombre de las magnitudes y el tiempo entre informe.



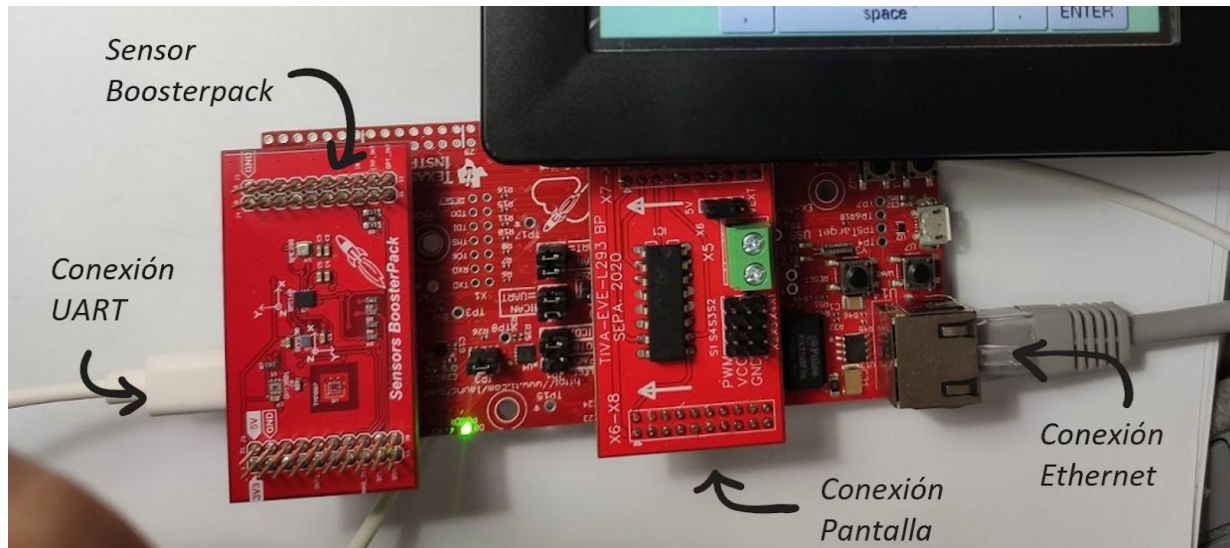
El Microcontrolador TM4C1294

Consideramos que la idea de proyecto resulta muy adecuada para este microcontrolador debido a que cuenta con un periférico ethernet que permite realizar la comunicación que deseamos, y disponen de amplios recursos que llevan a cabo todas las tareas que le imponemos con soltura.

Usamos ambos *boosterpacks*, para conectarnos con el “*Sensor Boosterpack*” y la pantalla “VM800Bxx”. Esto incluye la configuración de los buses I2C y SPI respectivamente.

Utilizamos un periférico del tipo UART que permite establecer una conexión con el PC. La información que mostramos por él no está pensada para el usuario final, simplemente se muestran datos de debug (IP local, IP del servidor, estado de la máquina de estados, etc).

Además, este microcontrolador cuenta con más recursos que podrían ser útiles en posibles ampliaciones del proyecto.



Reparto de Tareas

Las tareas que hemos tenido que realizar para llevar a cabo el proyecto, se pueden dividir en tres subgrupos:

T1: Envío de mensajes HTTP a la API de Telegram vía Ethernet.

- T1.1: Investigar el funcionamiento del ethernet y conseguir ejecutar ejemplos.
- T1.2: Desarrollo de un programa básico que envíe un único mensaje HTTP.
- T1.3: Consequir establecer conexión con el servidor con protocolo https.
- T1.4: Desarrollo de una máquina de estados que permita enviar varios mensajes.

T2: Desarrollo de la HMI

- T2.1: Desarrollo de la apariencia de la consola y el teclado.
- T2.2: Introducción de la edición de texto, aspectos visuales, y preparación para el envío de comandos.
- T2.3: Integrar el código necesario en una librería.

T3: Envío de informes personalizados:

- T3.1: Desarrollo de funciones que permitan crear un informe con codificación URL.
- T3.2: Lectura de comandos desde la pantalla.
- T3.3: Personalización completa del informe a través de los comandos introducidos.

T4: Documentación del proyecto

En la siguiente tabla, podemos ver la distribución de tareas aproximada:

	T1	T1.1	T1.2	T1.3	T1.4	T2	T2.1	T2.2	T2.3	T3	T3.1	T3.2	T3.3	T4
David	40	x			x	60	x	x	x	50		x	x	50
Miguel	60	x	x	x		40		x	x	50	x		x	50
	%					%				%				%

La Tarea 1 ha sido la que más tiempo no has llevado con diferencia, dedicando alrededor de 2 semanas solo al desarrollo de esta.

La Tarea 2 y 3 ha sido un trabajo que hemos podido realizar de manera más paralela.

A pesar de esta división de tareas ha habido un intercambio de información constante en todas las partes realizadas. Siempre hemos estado validando y corroborando lo programado el uno con el otro. Podemos decir que creemos que el desarrollo ha sido muy equitativo.

Manejo de Información
















Dado que ambos componentes del grupo tenemos todo el hardware necesario para la implementación, lo único que hemos tenido que compartir es el código que hemos ido desarrollando.

Para ello, hemos creado un repositorio privado en *GitHub*, donde usando *git* hemos ido subiendo y fusionando el trabajo realizado por cada uno de nosotros.

Pensamos que esta ha sido una opción muy acertada, ya que nos ha permitido llevar a cabo un control de versiones del proyecto, y volver a versiones funcionales cuando algo salía mal.

Para compartir documentos de interés y llevar a cabo la documentación requerida, hemos hecho uso de google drive.



	.gitignore	Modifico el gitignore	5 days ago
	FT800_TIVA.h	Versión integrada de envío de mensajes custom y HMI personalizada	2 days ago
	HAL_I2C.c	Versión integrada de envío de mensajes custom y HMI personalizada	2 days ago
	HAL_I2C.h	Versión integrada de envío de mensajes custom y HMI personalizada	2 days ago
	SENSORLIB2.lib	Versión integrada de envío de mensajes custom y HMI personalizada	2 days ago
	Sensor_HMI.c	pequeños cambios finales	16 hours ago
	Sensor_HMI.h	Versión comentada y ordenada. Pantalla inicial añadida. Modo de tiemp...	18 hours ago
	driverlib2.h	Dejo solo los archivos principales e ingoro el resto	5 days ago
	eth_client_lwip.c	Dejo solo los archivos principales e ingoro el resto	5 days ago
	eth_client_lwip.h	Dejo solo los archivos principales e ingoro el resto	5 days ago
	ft800_TIVA.c	Versión integrada de envío de mensajes custom y HMI personalizada	2 days ago
	lwipopts.h	Dejo solo los archivos principales e ingoro el resto	5 days ago
	main_FSM.c	pequeños cambios finales	16 hours ago
	sensorlib2.h	Versión integrada de envío de mensajes custom y HMI personalizada	2 days ago
	startup_ccs.c	Dejo solo los archivos principales e ingoro el resto	5 days ago

Guía de Usuario

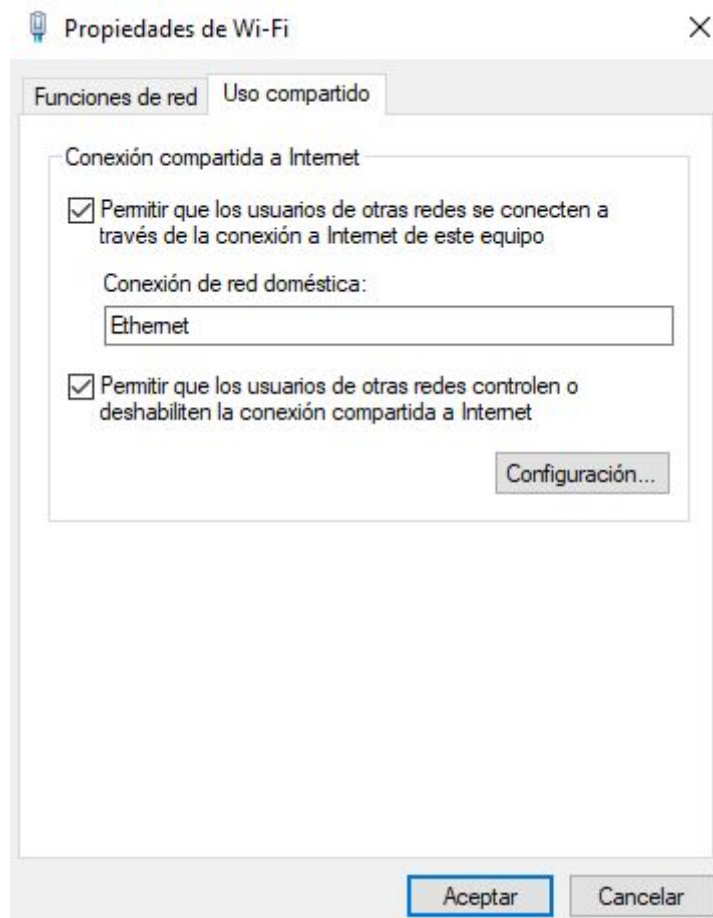
Conexión

Antes de comenzar a interactuar con la pantalla es necesario cerciorarse de que estamos conectados correctamente con la placa.

Si se conecta vía *Ethernet* directamente con el Router, no debería ser necesario ninguna configuración adicional.

Si conectamos a nuestro ordenador, tenemos que habilitar la compartición de internet, para ello realizamos lo siguiente (*Windows 10*):

- ❑ Entramos en **conexiones de Red** en el panel de control. Y seleccionamos el adaptador con el cual nuestro ordenador está conectado a la red. Click derecho propiedades.
- ❑ Ahí debemos ir a la pestaña de **Uso compartido** y marcar la casilla, así como seleccionar el adaptador conectado al microcontrolador.



Uso Pantalla

La interacción con el usuario se realiza completamente a través del teclado disponible en la pantalla táctil. Este se usa para enviar comandos a la placa, editando así los parámetros de configuración del envío de informes.

La sintaxis es la siguiente:

[COMANDO] [PARÁMETRO 1],[PARÁMETRO 2]

Primero escribimos el comando a realizar y posteriormente un espacio y los parámetros requeridos separados por comas.

Los comandos disponibles y sus parámetros son:

- ❑ **NAME [nombre]** : Cambia el nombre del dispositivo. Por ejemplo:

NAME TIVA_SEPA

Así el nuevo nombre en los informes enviados a telegram será TIVA_SEPA. Es importante tener en cuenta de que no podemos poner un nombre con espacios, ya que el segundo término sería ignorado.

- ❑ **INFO [índice medida],[info]** : Cambia la información o nombre de la medida seleccionada con el índice.

INFO 1,Temp

Ahora en el informe la medida de la temperatura para a mostrarse con el formato Temp [medida]. Por defecto es Temperatura [medida].

- ❑ **CONF [índice medida],[ON/OFF (1/0)]** : Habilita o deshabilita el envío de una medida en concreto.

CONF 1,0

Deshabilitamos el envío de la medida con índice 1, es decir, la luz.

- ❑ **FREC [frecuencia en segundos]** : Cambia la frecuencia de envío de informes.

FREC 10

Ahora se enviarán informes cada 10 segundos. Por defecto son 5 segundos.

Telegram

Ya solo queda entrar al grupo de telegram que tenemos programado para ver cómo se reciben periódicamente los datos.

Para esto hemos habilitado un Bot de Telegram, el cual se encarga de enviar mensajes por el grupo cuando recibe peticiones.

Desarrollo del Proyecto

Conexión con Telegram

Para comunicarnos con la API de telegram simplemente debemos enviar una Request HTTPS. Para más información ver [Telegram Bot API](#).

En nuestro caso queríamos hacer que el bot enviara un [mensaje](#). Para hacer esto debemos seguir la estructura indicada en la Web de Telegram, la cual es la siguiente:

```
api.telegram.org/bot[key]/sendMessage?chat_id=@[id]&text=[texto]
```

Hay tres variables aquí:

- ❑ **Key:** Key del bot de Telegram habilitado
- ❑ **ID:** Identificador del chat al que queremos enviar los mensajes. El bot debe ser miembro de este
- ❑ **Texto:** Mensaje a enviar. Debe estar codificado en [formato URL](#).

Una vez tenemos clara la comunicación con la API de Telegram debemos afrontar el problema de que con las librerías disponibles utilizamos el protocolo HTTP, el cual no es compatible con HTTPS.

Para evitar tener que programar este nuevo protocolo con encriptación y métodos de seguridad (TLS/SSL) haremos uso de una API intermedia que nos permite transformar HTTP en HTTPS.

httpstohttps.mrtimcakes.com

Esto nos permite enviar una request HTTP a este primer servidor con la petición real a Telegram como argumento, y este a su vez se encarga de transformar esta petición al protocolo aceptado por Telegram y enviarnos la respuesta.

Ya solo debemos sustituir la key de nuestro bot, ID de nuestro chat e introducir el mensaje a enviar. Para formar el mensaje hemos realizado una serie de funciones.

- ❑ **informeSensores():** Crea una cadena con la estructura básica del informe, modificando las variables (nombres, medidas, etc).
- ❑ **separaDecimales():** Las medidas tienen formato float, pero esto no podemos introducirlo directamente en el mensaje ya que el '.' que separa la unidades de los decimales invalidaría la codificación URL. Por esto se separan en dos cadenas de caracteres, una para las unidades y otra para los decimales.
- ❑ **makeRequest():** Introduce el mensaje en el request final, para ser enviado por TCP.

Un ejemplo de petición final sería:

```
GET
/https://api.telegram.org/bot[key]/sendMessage?chat_id=@[id]&text=[mensaje
URL] HTTP/1.1
Host: httpstohttps.mrtimcakes.com
Connection: close
```


Conexión Ethernet:

En la zona de setup de nuestro main, se realiza la configuración de la conexión ethernet, en la que se realizan las siguientes tareas importantes:

Primero configuramos que no haya un servidor proxy intermedio:

```
EthClientProxySet(0,0);
```

Inicializamos el proceso de ethernet como clientes, donde EnetEvents será el manejador de interrupciones.

```
EthClientInit(g_ui32SysClock,EnetEvents);
```

Actualizamos la dirección MAC, y la dirección IP de nuestra máquina. Este último proceso se realiza en bucle hasta que nuestra red le asigne una dirección IP.

Fijamos el host y resolvemos el DNS, obteniendo la IP del servidor al que queremos conectarnos. Esperamos hasta que esto se realice de igual forma que con la IP.

Es decir, se le impedirá evolucionar a nuestro sistema, hasta que la configuración inicial se haya realizado de manera correcta.

La lógica de control Internet se lleva a cabo por medio de una máquina de estados, que gestiona el estado de la conexión. Así como una rutina de interrupciones Ethernet que altera la máquina de estado en función del tipo de interrupción recibida.

Los estados posibles para la máquina son:

- ❑ **NOT_CONNECTED:** No estamos conectados a ningún Endpoint, por ello intentamos conectarnos a uno y pasamos al estado de espera de conexión.
- ❑ **WAIT_CONNECTION:** No hacemos nada hasta que se sobrepase un timeout. En cuyo caso volvemos a *NOT CONNECTED*. Si la conexión que esperamos es fructífera el manejador de interrupciones nos lleva a *NEW CONNECTION*.

```

else if(g_iState == STATE_NOT_CONNECTED){
    conexionTelegram = EthClientTCPConnect();
    g_iState = STATE_WAIT_CONNECTION;
    g_ui32Delay = 1000;
}

```

- ❑ **NEW_CONNECTION:** Se ha establecido una conexión nueva, ponemos la conexión en estado ocioso. *CONNECTED_IDLE*.

- ❑ **CONNECTED_IDLE:** Aquí es donde agrupamos todos los datos y realizamos la petición *HTTP*. Para ello debemos tener en cuenta el formato del protocolo *HTTP 1.1* que explicamos en el apartado de comunicación con Telegram.

```

else if(g_iState == STATE_CONNECTED_IDLE)
{
    g_ui32Delay = 1000; // Timeout de 10s
    g_iState = STATE_WAIT_DATA;

    // Preparamos envío de REQUEST a telegram
    //
    separaDecimales(medidasSensores, unidades, decimales,
NUMERO_SENSORES);

    informeSensores(nombreDispositivo, textoRequest, infoSensores,
unidades, decimales, configInforme, NUMERO_SENSORES);

    makeRequest(textoRequest, myRequest);

    // Realizamos el envío
    //
    resEnvio=EthClientSend(myRequest, sizeof(myRequest));

    // Debug
    //
    UARTprintf("\n%s", myRequest);
    if(!conexionTelegram)
    UARTprintf("\n\n>Conexion TCP establecida");
    if(!resEnvio)
    UARTprintf("\n\n>Mensaje Enviado");
}

```

Una vez hecha la request realizamos el envío con `EthClientSend` y pasamos al estado `WAIT_DATA`.

- ❑ **WAIT_DATA:** Aquí simplemente esperamos el *ACK* del servidor, para validar el envío. Si se sobrepasa un timeout, desconectamos y vamos a *NOT_CONNECTED*. Si el envío es exitoso el manejador de interrupciones cambia el estado a *UPDATE*
- ❑ **UPDATE:** En este estado simplemente inicializamos el contador de tiempo de espera entre mensajes y nos vamos al estado *WAIT_NICE*
- ❑ **WAIT_NICE:** Cuando se alcance el timeout desconectamos y vamos a *NOT_CONNECTED* para volver a iniciar la conexión.

Ahora pasamos a comentar el *manejador de interrupciones*.

```
void EnetEvents(uint32_t ui32Event, void *pvData, uint32_t ui32Param)
{
    // Evento de conexión al endpoint.
    if(ui32Event == ETH_CLIENT_EVENT_CONNECT)
    {
        g_iState = STATE_NEW_CONNECTION;

        UpdateIPAddress(g_pcIPAddr, EthClientAddrGet());
    }

    // Desconexión del endpoint.
    else if(ui32Event == ETH_CLIENT_EVENT_DISCONNECT)
    {
        g_iState = STATE_NOT_CONNECTED;

        UpdateIPAddress(g_pcIPAddr, 0);
    }

    // Recibimos el ACK de respuesta. Usamos TCP así que significa que el paquete
    se ha enviado correctamente.
    else if(ui32Event == ETH_CLIENT_EVENT_SEND){
        g_iState = STATE_UPDATE;
    }
}
```

Es llamado por la librería *eth_client_lwip*, nosotros simplemente debemos pasarlo como argumento cuando iniciamos Ethernet.

Esta función inicia toda la lógica de la librería y toma como argumentos el reloj del sistema y el manejador de interrupciones.

Como vemos este simplemente cambia el estado en función del evento recibido. No todos los eventos están considerados, ya que no todos son relevantes para nuestro problema.

- ❑ **EVENT_CONNECT**: Se produce cuando nos conectamos a un Endpoint.
- ❑ **EVENT_DISCONNECT**: Cuando nos desconectamos de un Endpoint.
- ❑ **EVNT_SEND**: Cuando recibimos *ACK* del servidor para la request HTTP.

HMI y Sensores:

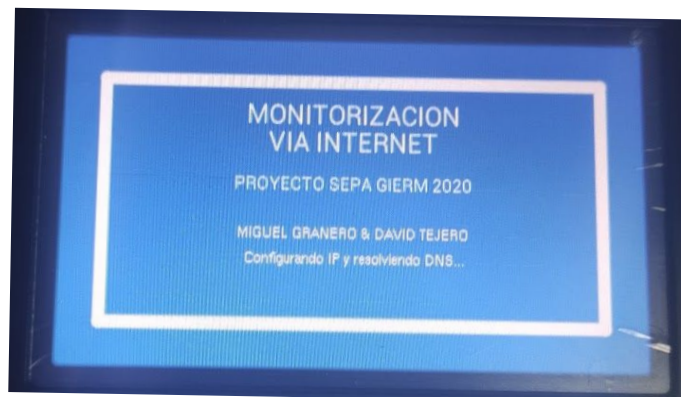
Para facilitar la coordinación de esta parte con el resto del programa hemos desarrollado una librería (*Sensor_HMI.c*, *Sensor_HMI.h*), en la que se encuentran las 4 funciones siguientes:

void configuraSensor(int frecReLoj);

Código sin mucha relevancia, ya que su contenido ha sido usado en prácticas. Simplemente configuramos los sensores de interés del boosterpack.

void configuraPantalla(int BP, int frecReLoj);

Se realiza la configuración de la pantalla, y pintamos la pantalla inicial. Esta pantalla se mantiene mientras se realiza toda la configuración de internet.



void LeeSensores(void);

Se leen los valores de los sensores. Código de prácticas.

void HMI(void);

En esta función se encuentra toda la gestión de la HMI. Comenzamos creando un aspecto visual primario compuesto por fondo, recuadro de la consola, etc.

A continuación pintamos las letras del teclado. Esto se realiza con un par de bucles *for*, que recorren nuestra matriz *teclado*. Esta matriz contiene los caracteres ASCII o el valor cero si no queremos que se pinte nada en esa celda.

La variable *cursor* es muy usada, ya que indica la posición dentro de la cadena *strCons* en la que vamos a escribir.

Mencionar el uso de la variable *flanco*. Si un botón se está pulsando y es la primera vez que leemos esa pulsación *flanco* valdrá cero, lo cual indica que estamos en un flanco de subida, y realizamos las actuaciones necesarias (guardamos el carácter en la cadena *strCons* y aumentamos el valor de *cursor*).

Tras actualizarse el valor de *strCons*, *flanco* tomará un valor no nulo que identificará la tecla pulsada. Esta identificación es necesaria para poder desactivar la variable *flanco*. Esto ocurre cuando la misma tecla que la ha activado deja de estar pulsada.

De esta forma conseguimos dos objetivos, que solo se guarde una única vez el carácter introducido en la cadena, de manera independiente a la duración de la pulsación, y que no se permita la lectura simultánea de varias teclas.

```

for (j=0;j<4;j++)
{
    if (j>1) // Identamos las dos filas de abajo
        indent=HSIZE/20;
    for (i=0;i<10;i++)
    {
        if(teclado[j][i])
        {
            if (mayuscAct && j!=0)
                sprintf(strTec1,"%c",teclado[j][i]+'A'-'a');
            else
                sprintf(strTec1,"%c",teclado[j][i]);

            if(Boton(indent+6+i*(HSIZE/10-1),((VSIZE*3)/8+5)+(VSIZE/8-1)*j,
                HSIZE/10-3, VSIZE/8-5, 21,strTec1))
            {
                //Si se da un flanco de subida
                if (!flanco)
                {
                    strCons[cursor]=strTec1[0];
                    flanco=j*10+i;
                    cursor++;
                }
            }
            else if (j*10+i==flanco)
                flanco = 0;
        }
    }
}

```

Simplemente, quedaría mencionar que la variable *indent* se suma a la posición x de todas las teclas, tomando un valor nulo para las primera tecla, y un valor de media tecla para las dos siguientes, obteniendo así, el efecto deseado de teclado.

Si están activadas las mayúsculas, el teclado cambiará su apariencia, y se introducirán teclas mayúsculas en la cadena.

Se repite la misma estructura fuera de bucle, para el resto de teclas de edición. Con especial mención a la tecla de *ENTER*, que es la que se encarga de activar la variable *comandoEnviado*, y copiar el contenido de la consola en la cadena *strConsOutput*, facilitando así la integración con el resto del programa.

El cursor se muestra en la pantalla gracias a la variable *muestraCursor*, una variable que debe ser activada alternativamente de manera externa para emular el parpadeo del cursor.

Mencionar, que toda la apariencia física está hecha a través de los valores de los macros *VSIZE* y *HSIZE*. Lo que hace que toda la apariencia se reescale en función de la pantalla usada.

Hemos probado el funcionamiento en ambas resoluciones de pantallas, y el aspecto visual es adecuado en ambas.

UART - Debug:

Cabe destacar que también hemos utilizado la UART pero esta no está destinada al usuario, sino al debug de fallos por nuestra parte.

Imprime información que puede ser útil a la hora de analizar el funcionamiento interno del programa.



```
COM6 - PuTTY
INIT
>Mi IP: 192.168.137.54
>Server IP: 82.25.221.41
GET /https://api.telegram.org/bot1435063235:AAHmzw5HcVKpZvCxf3kgwenqtsIXOHZ167E/
sendMessage?chat_id=@sepaGIERM&text=--ENVIADO%20DESDE%20TIVA%20--%0A%0ATemperatu
ra%2020%2E39%0ALuz%2014%2E6%
ESTADO: WAIT NEXT
ESTADO: WAIT NEXT
```

Dificultades encontradas

Conexión Ethernet

Este ha sido, sin duda alguna, el apartado del desarrollo del proyecto con el que más quebraderos de cabeza hemos tenido.

Esto es debido a que enviar un mensaje por Ethernet conlleva la coordinación de varias capas de abstracción, y para ello debemos entender el funcionamiento de múltiples librerías que se construyen una encima de otra.

Además hemos tenido que lidiar con una notable falta de documentación que explique el funcionamiento de las librerías existentes.

El único ejemplo que hemos podido usar de referencia ha sido el proyecto de Texas Instrument *enet_weather*. Entender su código ha llevado un tiempo considerable, ya que no sigue un esquema de programación muy secuencial, existiendo varias máquinas de estados que interactúan entre sí, cambios del manejador de interrupciones del ethernet, etc.

Además, se apoya en la librería *eth_client.c*, una versión notablemente modificada de la librería base que usamos nosotros, *eth_client_lwip.c*, orientando todo el código a realizar el proceso concreto de leer la información meteorológica de ciudades. Esto hace que sea casi imposible usar el código disponible para realizar una petición general.

Resulta interesante mencionar que hemos usado el analizador de paquetes *Wireshark*, que permite ver todos los mensajes que envía y recibe nuestro sistema vía ethernet. Esto ha sido crucial para identificar los errores de transmisión que se estaban cometiendo.

Analizar y comprender el funcionamiento de este así como conseguir debuggear errores que han ido surgiendo con el mismo ha sido la tarea más desafiante.

Peticiones HTTP → HTTPS

Encontrar la página que nos permite transformar estas peticiones no fue tarea sencilla tampoco. Por lo que hemos podido ver no hay más herramientas similares accesibles. El código fuente de la página web está disponible en [GitHub](#).

Anexo: Código Usado

Los archivos que incluye nuestro proyecto son los siguientes:

- ❑ **eth_client_lwip**: Manejo de internet
- ❑ **ft800_TIVA**: Control de la pantalla.
- ❑ **HAL_I2C**: Manejo sensores

- ❑ Dependencias extra requeridas por estas librerías. Eg: lwipopts, pinout, etc

- ❑ **main_FSM**: “Main” de nuestro proyecto
- ❑ **Sensor_HMI**: agrupa las funciones de los sensores y pantalla

Lo realmente programado por nosotros se encuentra en estas dos últimas, que son las que añadimos a continuación. Para ver con mayor claridad sugerimos abrir los *ficheros de código adjuntos* (Se desconfigura el formato al copiar el código en el documento).

Sensor_HMI:

```
#include <stdint.h>

#include <stdbool.h>

#include "driverlib2.h"

#include "FT800_TIVA.h"

#include <stdio.h>

#include <string.h>

#include "HAL_I2C.h"

#include "sensorlib2.h"

#include <math.h>

////////////////////////////////////

// Variables para el control de la pantalla

//

char chipid = 0; // Holds value of Chip ID read from the FT800

unsigned int CMD_Offset = 0;

unsigned long cmdBufferRd = 0x00000000; // Store the value read from the REG_CMD_READ
register
```



```
unsigned long cmdBufferWr = 0x00000000;           // Store the value read from the REG_CMD_WRITE
register

unsigned int t=0;

const int32_t REG_CAL[6]={21696,-78,-614558,498,-17021,15755638};

const int32_t REG_CAL5[6]={32146, -1428, -331110, -40, -18930, 18321010};

unsigned long POSX, POSY, BufferXY;

////////////////////////////////////

////////////////////////////////////

// Variables para el control de los sensores
//

uint8_t Sensor_OK=0;

uint8_t Opt_OK, Bme_OK;

// BME280

int returnRslt;

int g_s32ActualTemp  = 0;

unsigned int g_u32ActualPress  = 0;

unsigned int g_u32ActualHumity = 0;

float T_act,P_act,H_act;

//OPT

float lux;

//Vector Medidas

float vectorMedidas[4];

////////////////////////////////////
```

```

////////////////////////////////////

// Variables de control de la HMI

//

int cursor = 0;           //Posición del cursor dentro de la cadena

char strTecl[2];          //Cadena para pintar el teclado

char strConsMedidas[60];  //Cadena que muestra las variables de los sensores en la consola

int indent;

const char teclado[4][10] = {{'1','2','3','4','5','6','7','8','9','0'}, //Teclado alfanumérico
                              {'q','w','e','r','t','y','u','i','o','p'},
                              {'a','s','d','f','g','h','j','k','l', 0},
                              { 0 , 'z','x','c','v','b','n','m', 0 , 0 }};

int flanco = 0;           //Almacenará un numero que identificará el botón en el que acaba de
//haber un flanco

bool mayuscAct=0;         //Bloqueo de mayusculas activado

bool muestraCursor=0;      //Variable que debe cambiarse alternativamente para mostrar el
//parpadeo del cursor

char strCons[52];         //Texto de la consola

char strConsOutput[52]={""}; //Texto que sale de la consola cuando se presiona enter

bool comandoEnviado = 0;  //Se queda a 1 cuando se pulsa enter

////////////////////////////////////

//*****

//

// FUNCIÓN DE CONFIGURACIÓN DE LOS SENSORES

// frecReloj -> frecuencia configurada del reloj

//

//*****

```

```
void configuraSensor(int frecRelej)
{
    if(Detecta_BP(1))
        Conf_Boosterpack(1, frecRelej);
    else if(Detecta_BP(2))
        Conf_Boosterpack(2, frecRelej);

    //OPT3001
    Sensor_OK=Test_I2C_Dir(OPT3001_SLAVE_ADDRESS);

    if(!Sensor_OK)
    {
        Opt_OK=0;
    }
    else
    {
        OPT3001_init();
        Opt_OK=1;
    }

    //BME280
    Sensor_OK=Test_I2C_Dir(BME280_I2C_ADDRESS2);

    if(!Sensor_OK)
    {
        Bme_OK=0;
    }
    else
    {
        bme280_data_readout_template();

        bme280_set_power_mode(BME280_NORMAL_MODE);

        Bme_OK=1;
    }
}
```

```

}

//*****

//

// FUNCIÓN DE CONFIGURACIÓN DE LA PANTALLA

// BP      -> Boosterpack de conexión

// frecRelej -> Frecuencia de reloj configurada

//

//*****

void configuraPantalla(int BP, int frecRelej)
{
    int i;

    HAL_Init_SPI(BP, frecRelej);

    Inicia_pantalla();

    SysCtlDelay(frecRelej/3);

    //Calibración de la pantalla

    //

#ifdef VM800B35

    for(i=0;i<6;i++) Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL[i]);

#endif

#ifdef VM800B50

    for(i=0;i<6;i++)    Esc_Reg(REG_TOUCH_TRANSFORM_A+4*i, REG_CAL5[i]);

#endif

    //Pintamos pantalla inicial

    //

    Nueva_pantalla(0x10,0x10,0x10);

    ComColor(0x02,0x5E,0x80);

    ComLineWidth(5);

```

```

Comando(CMD_BEGIN_RECTS);

ComVertex2ff(10,10);

ComVertex2ff(HSIZE-10,VSIZE-10);

ComColor(0x08,0x8E,0xC0);

ComVertex2ff(12,12);

ComVertex2ff(HSIZE-12,VSIZE-12);

Comando(CMD_END);


ComColor(0xff,0xff,0xff);

ComTXT(HSIZE/2,VSIZE/5      , 28, OPT_CENTERX,"MONITORIZACION");

ComTXT(HSIZE/2,VSIZE/5 + 22 , 28, OPT_CENTERX,"VIA INTERNET");

        ComTXT(HSIZE/2,60+VSIZE/5      , 26, OPT_CENTERX," PROYECTO SEPA GIERM 2020 ");

ComTXT(HSIZE/2,100+VSIZE/5    , 20, OPT_CENTERX,"MIGUEL GRANERO & DAVID TEJERO");

ComTXT(HSIZE/2,120+VSIZE/5    , 20, OPT_CENTERX," Configurando IP y resolviendo DNS...");


Comando(CMD_BEGIN_LINES);

ComVertex2ff(40,40);

ComVertex2ff(HSIZE-40,40);

ComVertex2ff(HSIZE-40,40);

ComVertex2ff(HSIZE-40,VSIZE-40);

ComVertex2ff(HSIZE-40,VSIZE-40);

ComVertex2ff(40,VSIZE-40);

ComVertex2ff(40,VSIZE-40);

ComVertex2ff(40,40);

Comando(CMD_END);

        Dibuja();


        SysCtlDelay(frecReloj); //Mostramos el mensaje 1s

}

```

```
//*****
```

```
//  
  
// FUNCIÓN PARA LEER EL VALOR DE LOS SENSORES  
  
//  
  
//*****  
  
void leeSensores(void)  
{  
  
    //LECTURA DE DATOS  
  
    // OPT3000  
  
    //  
  
    if(Opt_OK)  
    {  
  
        lux=OPT3001_getLux();  
  
    }  
  
    // BME280  
  
    //  
  
    if(Bme_OK)  
    {  
  
        returnRslt = bme280_read_pressure_temperature_humidity(&g_u32ActualPress, &g_s32ActualTemp,  
&g_u32ActualHumity);  
  
        T_act=(float)g_s32ActualTemp/100.0;  
  
        P_act=(float)g_u32ActualPress/100.0;  
  
        H_act=(float)g_u32ActualHumity/1000.0;  
  
    }  
  
    // Guardamos en el vector de medidas  
  
    //  
  
    vectorMedidas[0] = T_act;  
  
    vectorMedidas[1] = lux;  
  
    vectorMedidas[2] = P_act;  
  
    vectorMedidas[3] = H_act;
```

```
}

//*****

//

//  FUNCIÓN PARA PINTAR Y EJECUTAR LA HMI

//

//*****

void HMI (void)
{

    int i,j;          //índices

    //  Pintamos la pantalla

    //

    Nueva_pantalla(0,0,0);

    ComLineWidth(1);

    //  Fondo+Marco

    //

    ComColor(0x7F,0xCC,0xA2);

    Comando(CMD_BEGIN_RECTS);

    ComVertex2ff(2,2);

    ComVertex2ff(HSIZE-2,VSIZ-2);

    //  Ventana de texto

    //

    ComColor(0xFF,0xFF,0xFF);

    ComVertex2ff(3,3);

    ComVertex2ff(HSIZE-3,(VSIZ*3)/8);

    Comando(CMD_END);
```



```

ComColor(0x7F,0xCC,0xA2);

Comando(CMD_BEGIN_LINES);

ComVertex2ff(4,4);

ComVertex2ff(HSIZE-4,4);

ComVertex2ff(HSIZE-4,4);

ComVertex2ff(HSIZE-4,(VSIZE*3)/8);

ComVertex2ff(HSIZE-4,(VSIZE*3)/8);

ComVertex2ff(4,(VSIZE*3)/8);

ComVertex2ff(4,(VSIZE*3)/8);

    ComVertex2ff(4,4);

Comando(CMD_END);

//Pintamos el teclado y vemos que teclas están pulsadas

// Letras

//

ComColor(0x00,0x2B,0x38);

ComFgcolor(0x96,0xB7,0xB5);

    indent=0;

    for (j=0;j<4;j++)

    {

        if (j>1) // Identamos las dos filas de abajo

            indent=HSIZE/20;

        for (i=0;i<10;i++)

        {

            if(teclado[j][i])

            {

                if (mayuscAct && j!=0)

                    sprintf(strTecl,"%c",teclado[j][i]+('A'-'a'));

                else

                    sprintf(strTecl,"%c",teclado[j][i]);

```

```

        if(Boton(indent+6+i*(HSIZE/10-1),((VSIZE*3)/8+5)+(VSIZE/8-1)*j, HSIZE/10-3, VSIZE/8-5,
21,strTec1))

        {

            //Si se da un flanco de subida

            if (!flanco)

            {

                strCons[cursor]=strTec1[0];

                flanco=j*10+i;

                cursor++;

            }

        }

        else if (j*10+i==flanco) //Si el botón para el cuál activamos el flanco se ha dejado
de pulsar...

            flanco = 0;

    }

}

}

//Edición de texto y caracteres especiales

// Espacio

//

        if(Boton(6+3*(HSIZE/10-1),((VSIZE*3)/8+5)+(VSIZE/8-1)*4, 4*(HSIZE/10)-6, VSIZE/8-5,
21,"space"))

        {

            if (!flanco)

            {

                strCons[cursor]=' ';

                cursor++;

                flanco=100;

            }

        }

```

```
else if (flanco==100)

    flanco = 0;

    // Coma

    //

    if(Boton(6+2*(HSIZE/10-1),((VSIZE*3)/8+5)+(VSIZE/8-1)*4, (HSIZE/10)-3, VSIZE/8-5, 21,","))

    {

        if (!flanco)

        {

            strCons[cursor]=',';

            cursor++;

            flanco=101;

        }

    }

    else if (flanco==101)

    flanco = 0;

    // Punto

    //

    if(Boton(6+7*(HSIZE/10-1),((VSIZE*3)/8+5)+(VSIZE/8-1)*4, (HSIZE/10)-3, VSIZE/8-5, 21,"."))

    {

        if (!flanco)

        {

            strCons[cursor]='.';

            cursor++;

            flanco=102;

        }

    }

    else if (flanco==102)

    flanco = 0;
```

```

// Mayusc

//

if (mayuscAct)

ComFgcolor(0x56,0x87,0x85);

else

ComFgcolor(0x96,0xB7,0xB5);


if (Boton(indent+6,((VSIZE*3)/8+5)+(VSIZE/8-1)*3, HSIZE/10-3, VSIZE/8-5, 21,"^"))

{

if (!flanco)

{

mayuscAct=!mayuscAct;

flanco=103;

}

}

else if (flanco==103)

flanco = 0;


ComFgcolor(0x96,0xB7,0xB5);


// Borrar

//

if (Boton(indent+6+8*(HSIZE/10-1),((VSIZE*3)/8+5)+(VSIZE/8-1)*3, HSIZE/10-3, VSIZE/8-5,
21,"<-"))

{

if (!flanco && cursor>0)

{

cursor --;

flanco=104;

}

}

```

```
}

else if (flanco==104)

    flanco = 0;

// Enter

//

if (Boton(6+8*(HSIZE/10-1),((VSIZE*3)/8+5)+(VSIZE/8-1)*4, (3*(HSIZE/10))/2-3, VSIZE/8-5,
21,"ENTER"))

{

    if (!flanco)

    {

        comandoEnviado=1;

        strcpy(strConsOutput,strCons);

        strConsOutput[cursor] ='\0';

        cursor=0;

        flanco=105;

    }

}

else if (flanco==105)

    flanco = 0;

// Cursor

//

if (cursor>50) //No podemos sobrepasar el cursor de la posición penúltima

    cursor=50;

if (muestraCursor)

    strCons[cursor]='_';

else

    strCons[cursor]=' ';
```

```
    strCons[cursor + 1]='\0';

    // Mostramos el texto por consola
    //

    ComColor(0,0,0);

    ComTXT(HSIZE/32,VSIZE/4, 27, OPT_MONO,strCons);

    // Mostramos las medidas ambientales por pantalla
    //

    if(Opt_OK && Bme_OK)

    {

        ComColor(0xA6,0x39,0x2C);

        sprintf(strConsMedidas,"LUX: %.2f --- T:%.2f C --- P:%.2fmbars ---
H:%.3f",lux,T_act,P_act,H_act);

        ComTXT(HSIZE/2,VSIZE/16, 20, OPT_CENTER,strConsMedidas);

    }

    // Dibujamos toda la pantalla
    //

    Dibuja();
}
```

main_FSM:

```
#include <stdint.h>

#include <stdbool.h>

#include "driverlib2.h"

#include "drivers/pinout.h"

#include "utils/cmdline.h"

#include "utils/locator.h"

#include "utils/flash_pb.h"

#include "utils/lwiplib.h"

#include "utils/uartstdio.h"

#include "utils/ustdlib.h"

#include "eth_client_lwip.h"

#include <string.h>

#include <stdio.h>

#include "Sensor_HMI.h"


#define SYSTEM_TICK_MS      10

#define SYSTEM_TICK_S      100


#define APP_INPUT_BUF_SIZE      1024

#define MAX_REQUEST_SIZE      1024


#define ETH_CLIENT_EVENT_DHCP      0x00000001

#define ETH_CLIENT_EVENT_DISCONNECT      0x00000002

#define ETH_CLIENT_EVENT_DNS      0x00000003

#define ETH_CLIENT_EVENT_CONNECT      0x00000004
```



```
#define ETH_CLIENT_EVENT_RECEIVE    0x00000005

#define ETH_CLIENT_EVENT_SEND       0x00000006

#define ETH_CLIENT_EVENT_ERROR      0x00000007

////////////////////////////////////

// CONFIGURACIÓN

//

#define BP_PANTALLA      1 // BOOSTERPACK Pantalla

#define NUMERO_SENSORES  4 // Numero de sensores que se van a monitorizar

                           // Lo hemos hecho así con la idea de poder ajustar el
codigo

                           // a otro set de sensores más facilmente.

////////////////////////////////////

////////////////////////////////////

// CONEXIÓN Y VARIABLES DE RED

//

//Estados de la máquina de estados de internet

enum

{

    STATE_NOT_CONNECTED,

    STATE_NEW_CONNECTION,

    STATE_CONNECTED_IDLE,
```

```

    STATE_WAIT_DATA,

    STATE_UPDATE,

    STATE_WAIT_NICE,

    STATE_WAIT_CONNECTION,
}

g_iState = STATE_NOT_CONNECTED;

const char* stateName[] = {"NOT CONNECTED", "NEW CONNECTION", "IDLE", "WAIT
DATA", "UPDATING", "WAIT NEXT", "WAIT FOR CONNECTION"};

//Dominio, puerto y request para la conexión con nuestra API

const char* nombreDominio = "httptohttps.mrtimcakes.com"; // Telegram necesita
HTTPS, utilizamos esta pagina intermedia para

// la transformacion
HTTP -> HTTPS

uint16_t puertoConexion = 80;

int8_t g_exampleRequest[] =

    "GET
/https://api.telegram.org/bot1435063235:AAHmzw5HcVKpZvCxf3kgwenqtsIX0Hz167E/sendMe
ssage?chat_id=@sepaGIERM&text=TIVAHEY HTTP/1.1\r\nHost:
httptohttps.mrtimcakes.com\r\nConnection: close\r\n\r\n";

//Partes que conforman el Request

const char* requestHeader = "GET /https://api.telegram.org/bot";

char* APIkey = "1435063235:AAHmzw5HcVKpZvCxf3kgwenqtsIX0Hz167E";

const char* sendMessage = "sendMessage?chat_id=@";

char* chatName = "sepaGIERM";

const char* textHeader = "&text=";

```

```

char* messageToSend;

const char* endRequest = " HTTP/1.1\r\nHost:
httpstohttps.mrtimcakes.com\r\nConnection: close\r\n\r\n";

char myRequest[MAX_REQUEST_SIZE];

char textoRequest[200];

////////////////////////////////////

////////////////////////////////////

// VARIABLES DE SENSORES Y DEL INFORME
//

bool configInforme[NUMERO_SENSORES] = {true,true,true,true};
//Configuracion por defecto

float medidasSensores[NUMERO_SENSORES] = {20.1, 385.7, 1024.1, 12.6};
//Valores por defecto (DEBUG)

char infoSensores[NUMERO_SENSORES][20]={ "Temperatura", "Luz", "Presion", "Humedad"};
//Nombre por defecto

char nombreDispositivo[20] = "TIVA"; //Por defecto TIVA

char unidades[NUMERO_SENSORES][10];

char decimales[NUMERO_SENSORES][10];

int tiempoEntreMensajes = 500; //Ciclos de 10 ms entre requests (por defecto 5s)

////////////////////////////////////

////////////////////////////////////

// RELOJ Y PRIORIDADES DE INTERRUPTIÓN
//

```

```
uint32_t g_ui32Delay;

uint32_t g_ui32SysClock;

#define SYSTICK_INT_PRIORITY    0x80

#define ETHERNET_INT_PRIORITY   0xC0

////////////////////////////////////

////////////////////////////////////

//  DIRECCIONES
//

char g_pcMACAddr[40];

uint32_t g_ui32IPAddr;

char g_pcIPAddr[20];

////////////////////////////////////

////////////////////////////////////

//  UART
//

uint16_t actualizarUART;

uint32_t g_ui32UARTDelay;

////////////////////////////////////
```

```
////////////////////////////////////
// VARIABLES PARA EL CONTROL TEMPORAL DE LA HMI
//

int tAct=0,tCursor=0;
bool actualizaHMI=1;

////////////////////////////////////

////////////////////////////////////
// VARIABLES PARA EL CONTROL DE COMANDOS DE LA HMI
//

char comandoHMI[5];
char parametrosHMI[10];

////////////////////////////////////

////////////////////////////////////
// VARIABLES DEBUG
//

int32_t DNSResuleto;
int32_t conexionTelegram;
int32_t resEnvio;
int32_t telegramIP;
```

```
////////////////////////////////////

//*****

//

//  FUNCIÓN QUE GUARDA EN UN STRING LA IP LOCAL

//

//*****

void UpdateIPAddress(char *pcAddr, uint32_t ipAddr)
{
    uint8_t *pui8Temp = (uint8_t *)&ipAddr;

    if(ipAddr == 0)
    {
        strncpy(pcAddr, "IP: ---.---.---.---", sizeof(g_pcIPAddr));
    }
    else
    {
        sprintf(pcAddr, "IP: %d.%d.%d.%d", pui8Temp[0], pui8Temp[1],
            pui8Temp[2], pui8Temp[3]);
    }
}

//*****

//

// ACTUALIZA LA DIRECCIÓN MAC
```

```
//  
//*****  
  
void UpdateMACAddr(void)  
{  
    uint8_t pui8MACAddr[6];  
  
    EthClientMACAddrGet(pui8MACAddr);  
  
    usprintf(g_pcMACAddr, "MAC: %02x:%02x:%02x:%02x:%02x:%02x",  
            pui8MACAddr[0], pui8MACAddr[1], pui8MACAddr[2], pui8MACAddr[3],  
            pui8MACAddr[4], pui8MACAddr[5]);  
}  
  
//*****  
  
//  
// ESCRIBE POR LA UART NUESTRA DIRECCIÓN IP  
//  
//*****  
  
void PrintIPAddress(char *pcAddr, uint32_t ipaddr)  
{  
    uint8_t *pui8Temp = (uint8_t *)&ipaddr;  
  
    UARTprintf("%d.%d.%d.%d\n", pui8Temp[0], pui8Temp[1], pui8Temp[2],  
            pui8Temp[3]);  
}
```



```

/*****
//
// FUNCIÓN HANDLER DE LAS INTERRUPCIONES SysTick (CADA 10 MS)
//
*****/

void SysTickIntHandler(void)
{
    // Call the lwIP timer handler.
    EthClientTick(SYSTEM_TICK_MS);

    //Timeout para actualizar la UART
    if(actualizarUART > 0)
        actualizarUART--;

    //Variable para realizar timeout y delays en la conexión
    if(g_ui32Delay > 0)
    {
        g_ui32Delay--;
    }

    //Refrescamos la pantalla cada 100 ms
    tAct++;
    if(tAct > 10)
    {
        tAct=0;
    }
}
```

```
        actualizaHMI=1;

    }

    //Hacemos que el cursor de la pantalla aparezca y desaparezca cada 0.5s
    tCursor++;

    if (tCursor > 50)
    {
        tCursor=0;

        muestraCursor=!muestraCursor;
    }
}

//*****
//
// MANEJADOR DE INTERRUPCIONES DE LOS EVNETOS DE INTERNET
//
//*****

void EnetEvents(uint32_t ui32Event, void *pvData, uint32_t ui32Param)
{
    // Evento de conexión al endpoint.

    if(ui32Event == ETH_CLIENT_EVENT_CONNECT)
    {
        g_iState = STATE_NEW_CONNECTION;

        UpdateIPAddress(g_pcIPAddr, EthClientAddrGet());
    }
}
```

```
}

// Desconexión del endpoint.

else if(ui32Event == ETH_CLIENT_EVENT_DISCONNECT)

{

    g_iState = STATE_NOT_CONNECTED;

    UpdateIPAddress(g_pcIPAddr, 0);

}

// Recibimos el ACK de respuesta.

else if(ui32Event == ETH_CLIENT_EVENT_SEND){

    g_iState = STATE_UPDATE;

}

}

//*****

//

// SEPARA UN FLOAT EN UNIDADES Y DECIMALES (2 DEC DE PRECISIÓN)

//

//*****

void separaDecimales(float *in, char unidad[][10], char decimal[][10], int N){

    int i;

    int ud,dec;
```

```

    for(i = 0; i < N; i++){

        char aux[10];

        ud = in[i];

        dec = (in[i]-ud)*100;

        sprintf(aux,"%d",ud);

        strcpy(unidad[i],aux);

        sprintf(aux,"%d",dec);

        strcpy(decimal[i],aux);

    }
}

//*****

//

//  CREA EL INFORME DE LA LECTURA DE LOS SENSORES - COFIFICACIÓN URL

//

//*****

void informeSensores(char *nombreDisp ,char *cadenaOut, char info[][20], char
unMedidas[NUMERO_SENSORES][], char decMedidas[NUMERO_SENSORES][], bool *config,
int numeroMedidas){

    int i;

    ///////////////////////////////////

    //  URL ENCONDING  //

    //  ' ' ->%20      //

    //  '\n' ->%0A     //

    //  '.' ->%2E      //

```

```
////////////////////////////////////

strcpy(cadenaOut, "--ENVIADO%20DESDE%20");

strcat(cadenaOut, nombreDisp);

strcat(cadenaOut, "%20--%0A");

for(i = 0; i < numeroMedidas; i++){
    if(config[i]){
        strcat(cadenaOut, "%0A");
        strcat(cadenaOut, info[i]);
        strcat(cadenaOut, "%20");
        strcat(cadenaOut, unidades[i]);
        strcat(cadenaOut, "%2E");
        strcat(cadenaOut, decimales[i]);
    }
}

}

//*****

//

//  CONSTRUYE EL MENSAJE DE REQUEST HTTP 1.1 PERSONALIZADO

//

//*****

void makeRequest(char *text, char *finalRequest){
```

```
        sprintf(finalRequest, "GET
/https://api.telegram.org/bot%s/sendMessage?chat_id=%s&text=%s HTTP/1.1\r\nHost:
httphttps.mrtimcakes.com\r\nConnection: close\r\n\r\n", APIkey, chatName, text);
    }

//*****

//

//  ACTUALIZA EL VECTOR DE MEDIDAS

//

//*****

void actualizarMedidas(float *nuevosValores, float *arrayOut, int N){
    int i;
    for(i = 0; i < N; i++) arrayOut[i] = nuevosValores[i];
}

//*****

//

//  ACTUALIZA EL NOMBRE DEL DISPOSITIVO

//

//*****

void actualizaNombreDisp(char *nuevoNombre, char *stringOut){
    strcpy(stringOut, nuevoNombre);
}

//*****
```

```
//  
  
// CAMBIA LA CONFIGURACIÓN DEL ENVÍO DE MEDIDAS  
  
//  
  
//*****  
  
void cambiaConfig(bool nuevaConfig, bool *arrayOut, int opcion){  
    arrayOut[opcion] = nuevaConfig;  
}  
  
//*****  
  
//  
  
// CAMBIA EL NOMBRE DE LOS PARÁMETROS  
  
//  
  
//*****  
  
void cambiaInfo(char *nuevaConfig, char arrayOut[][20], int opcion){  
    strcpy(arrayOut[opcion], nuevaConfig);  
}  
  
//*****  
  
//  
  
// PARSEAR PARA LOS COMANDOS DEL TECLADO DE LA PANTALLA  
  
//  
  
//*****  
  
int commandParser(char *buffer, char *orden, char *parametros){  
    int resultado;
```

```
        resultado = sscanf(buffer,"%s %s",orden,parametros); // Dividimos en
comando y parametros

        //if(parametros[strlen(parametros)-1] == '_')
parametros[strlen(parametros)-1] = '\0';

        UARTprintf("\n%d -> ESCANEADO: %s %s",resultado,orden,parametros);

        return resultado; //Si devuelve 2 significa que se ha parseado
correctamente
}

//*****

//

// PROCESADO DE COMANDOS DE LA PANTALLA

//

//*****

void commandAction(char *orden, char *parametros){

    int opcion;

    int valorBool;

    bool finalBool;

    char valorString[20];

    int res;

    // INFO

    if(!strcmp(orden,"INFO")){ // Cambia el "nombre" de una medida. Eg:
Temperatura, Luz, ...

                                // INFO [Indice medida],[nueva info]
```



```
res = sscanf(parametros,"%d,%s",&opcion,valorString);

if(res){

    cambiaInfo(valorString, infoSensores, opcion);

    UARTprintf("\n%d -> CAMBIADA INFO[%d]: %s ",res,opcion,valorString);

}

}

// NAME

else if(!strcmp(orden,"NAME")){ // Cambia el nombre del dispositivo

    // NAME [nuevo nombre]

    res = sscanf(parametros,"%s",valorString);

    if(res){

        UARTprintf("\n%d -> CAMBIADO NOMBRE: %s ",res,valorString);

        actualizaNombreDisp(valorString, nombreDispositivo);

    }

}

// CONF

else if(!strcmp(orden,"CONF")){ // Cambia la configuracion del informe. Qué
medidas enviar y cuales no

    // CONF [Indice medida],[ON/OFF (1/0)]

    res = sscanf(parametros,"%d,%d",&opcion, &valorBool);

    if(res){

        UARTprintf("\n%d -> CAMBIADA CONF[%d]: %d ",res,opcion,valorBool);

        finalBool = (valorBool != 0);

        cambiaConfig(finalBool, configInforme, opcion);

    }

}
```

```
    }

    }

    //FREC

    else if(!strcmp(orden,"FREC")) { // Cambia la frecuencia de envio de
informes

                                // FREC [frecuencia]

    res = sscanf(parametros,"%d",&opcion);

    if(res){

        UARTprintf("\n%d -> CAMBIADA FRECUENCIA A: %d S ",res,opcion);

        tiempoEntreMensajes = 100*opcion;

    }

    }

}

//*****

//

// MAIN

//

//*****

int

main(void)

{

    SysCtlMOSCConfigSet(SYSCTL_MOSC_HIGHFREQ);
```

```
g_ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |  
    SYSCTL_OSC_MAIN |  
    SYSCTL_USE_PLL |  
    SYSCTL_CFG_VCO_480), 120000000);  
  
// Configuramos el sensor y la pantalla  
  
//  
configuraSensor(g_ui32SysClock);  
  
configuraPantalla(BP_PANTALLA,g_ui32SysClock); //Pantalla en el BoosterPack  
definido en la config  
  
PinoutSet(true, false);  
  
// UART  
  
//  
UARTStdioConfig(0, 115200, g_ui32SysClock);  
UARTprintf("\nINIT ");  
  
// SysTick a 10 ms  
  
//  
SysTickPeriodSet((g_ui32SysClock / 1000) * SYSTEM_TICK_MS);  
SysTickEnable();  
SysTickIntEnable();  
  
// Poner la IP a 0.0.0.0.  
  
//  
UpdateIPAddress(g_pcIPAddr, 0);
```

```
// Interrupciones

//

IntMasterEnable();

IntPriorityGroupingSet(4);

IntPrioritySet(INT_EMAC0, ETHERNET_INT_PRIORITY);

IntPrioritySet(FAULT_SYSTICK, SYSTICK_INT_PRIORITY);


// Inicializacion Ethernet

//

EthClientProxySet(0,0);

EthClientInit(g_ui32SysClock,EnetEvents);


// Actualizamos la dirección MAC

//

UpdateMACAddr();


// Actualizamos la dirección IP

//

do{

    g_ui32IPAddr = EthClientAddrGet();

    SysCtlDelay(50000);

}while(g_ui32IPAddr == 0 || g_ui32IPAddr == 0xffffffff); // Debemos esperar
hasta recibir IP buena
```

```
UARTprintf("\n\nMi IP: ");

PrintIPAddress(0, g_ui32IPAddr);    //IP Local ok


// Fijamos el servidor y resolvemos el DNS
//

EthClientHostSet(nombreDominio, puertoConexion);


do{

DNSResuleto = EthClientDNSResolve();

SysCtlDelay(50000);

}while(DNSResuleto != 0); // De nuevo esperaamos a que se resuelva


telegramIP = EthClientServerAddrGet();

UARTprintf("\n\nServer IP: ");

PrintIPAddress(0, telegramIP);    //IP Telegram ok


// Inicializamos el contador para la actualización de la UART
//

actualizarUART = 50;


// Configuramos el modo bajo consumo
//

SysCtlPeripheralClockGating(true);
```

```
    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_EMAC0);

    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_EPHY0);

    SysCtlPeripheralSleepEnable(SYSCTL_PERIPH_UART0);


    // Bucle infinito principal

    //

    while(1)
    {
        if(g_iState == STATE_NEW_CONNECTION)
        {
            g_iState = STATE_CONNECTED_IDLE;
        }

        else if(g_iState == STATE_CONNECTED_IDLE)
        {
            g_ui32Delay = 1000; // Timeout de 10s

            g_iState = STATE_WAIT_DATA;


            // Preparamos envio de REQUEST a telegram

            //

            separaDecimales(medidasSensores, unidades, decimales,
NUMERO_SENSORES);

            informeSensores(nombreDispositivo, textoRequest,infoSensores,
unidades, decimales, configInforme, NUMERO_SENSORES);

            makeRequest(textoRequest, myRequest);


            // Realizamos el envío

            //
```

```
        resEnvio=EthClientSend(myRequest,sizeof(myRequest));

        //  Debug
        //
        UARTprintf("\n%s",myRequest);

        if(!conexionTelegram)

            UARTprintf("\n\n>Conexion TCP establecida");

        if(!resEnvio)

            UARTprintf("\n\n>Mensaje Enviado");

    }

    else if(g_iState == STATE_UPDATE)

    {

        // Entraremos en este estado desde el manejador de eventos si todo
sale bien

        g_iState = STATE_WAIT_NICE;

        // Inicializamos el delay de espera entre mensajes

        g_ui32Delay = tiempoEntreMensajes;

    }

    else if(g_iState == STATE_WAIT_NICE)

    {

        if(g_ui32Delay == 0) // Hasta que no haya pasado el tiempo de espera
no hacemos nada

        {

            EthClientTCPDisconnect();

            g_iState = STATE_NOT_CONNECTED;

        }

    }

}
```

```
else if(g_iState == STATE_WAIT_DATA)
{

    if(g_ui32Delay == 0) // Esperamos ACK, si hay timeout desconectamos
    {
        UARTprintf("\n\n>Ha habido algun fallo, cerramos la conexion");
        EthClientTCPDisconnect();
        g_iState = STATE_NOT_CONNECTED;
    }
}

else if(g_iState == STATE_NOT_CONNECTED){ // Intentamos conectar y vamos a
estado de espera
    conexionTelegram = EthClientTCPConnect();
    g_iState = STATE_WAIT_CONNECTION;
    g_ui32Delay = 1000;
}

else if(g_iState == STATE_WAIT_CONNECTION){ // Esperamos conexion hasta
timeout
    if(g_ui32Delay == 0)
    {
        UARTprintf("\n\n>Ha habido algun fallo, cerramos la conexion");
        EthClientTCPDisconnect();
        g_iState = STATE_NOT_CONNECTED;
    }
}

if(actualizarUART == 0){ // Actualizamos la UART
    UARTprintf("\n\nESTADO: %s",stateName[g_iState]);
```



```
        actualizarUART = 50;

    }

    //Leemos los sensores y actualizamos la pantalla
    //
    if (actualizaHMI)
    {
        actualizaHMI = 0;
        leeSensores();
        HMI();
        actualizarMedidas(vectorMedidas, medidasSensores, NUMERO_SENSORES);
        if(comandoEnviado){
            // Procesamos el comando recibido
            //
            comandoEnviado = 0;
            commandParser(strConsOutput, comandoHMI, parametrosHMI);
            commandAction(comandoHMI, parametrosHMI);
        }
    }

    //SysCtlDelay(500);
    SysCtlSleep();
}
}
```