



Máster en Ingeniería

Informática

# SISTEMAS EMPOTRADOS Y UBICUOS

|  |
| --- |
| PROYECTO |
| Data logger con FRDM K64F y Basic Shield |

|  |  |
| --- | --- |
| Docentes: |  |
| Alejandro Merino |  |
| Dpto. de Ingeniería Electromecánica  Área de Ingeniería de Sistemas y Automática |  |
| Alumnos:  Antonio Pérez Trapero | [Apt1003@alu.ubu.es](mailto:Apt1003@alu.ubu.es) |
| José Ignacio Huidobro del Arco | [Jhd1001@alu.ubu.es](mailto:Jhd1001@alu.ubu.es) |
| Antón Epikhin | [Aex0022@alu.ubu.es](mailto:Aex0022@alu.ubu.es) |
| Versión 1.0  Fecha 16/03/2019 19:56 |  |

|  |
| --- |
| Índice de contenidos |
|  |

Contenido

[SISTEMAS EMPOTRADOS Y UBICUOS 1](#_Toc3999054)

[Enunciado 3](#_Toc3999055)

[Componentes utilizados 3](#_Toc3999056)

[Main.c 5](#_Toc3999057)

[Events.c 5](#_Toc3999058)

[Código de componentes de JHA 6](#_Toc3999059)

[Funciones de inicialización 6](#_Toc3999060)

[PORT\_PDD\_SetPinPullSelect y PORT\_PDD\_SetPinPullEnable 6](#_Toc3999061)

[FAT\_E1\_Init 6](#_Toc3999062)

[FAT\_E1\_mount 6](#_Toc3999063)

[JHA\_Run() 6](#_Toc3999064)

[Funciones de tratamiento 6](#_Toc3999065)

[Código de JHA 6](#_Toc3999066)

[char \* getHora(void) 6](#_Toc3999067)

[void JHA\_Factoriza(void) 7](#_Toc3999068)

[double getFactor(void) 7](#_Toc3999069)

[void JHA\_SetHora(void) 7](#_Toc3999070)

[void JHA\_OnFullRxBuf() 8](#_Toc3999071)

[void parseHora(char \*datos, TIMEREC \*hora) 8](#_Toc3999072)

[Código de AEX 9](#_Toc3999073)

[void CloseFP() 9](#_Toc3999074)

[int LogToSDCard(int16\_t x, int16\_t y, int16\_t z) 9](#_Toc3999075)

[int LogToSDCardL(int16\_t x, int16\_t y, int16\_t z, int16\_t luminocidad) 9](#_Toc3999076)

[int LogToSDCardT(int16\_t x, int16\_t y, int16\_t z, int16\_t temperatura) 9](#_Toc3999077)

[int LogToSDCardLT(int16\_t x, int16\_t y, int16\_t z, int16\_t luminocidad, int16\_t temperatura) 9](#_Toc3999078)

[Fichero escrito en la tarjeta SD 10](#_Toc3999079)

# Enunciado

Se desea registrar la información que recibe la placa FRDM K64F y almacenarla en una tarjeta SD para su consulta y análisis posterior.

Se remite a lo descrito en la propuesta de proyecto de la asignatura.

# Componentes utilizados

A continuación, se describen los distintos componentes utilizados, así como sus características más relevantes para el trabajo a realizar y el alumno que lo ha incluido, configurado y programado.

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo | Alumno | Característica de la placa |
| TmDt\_H1 | TimeDate | Jhd1001 | Periodic Interrupt: FTM0\_MOD |
| Uso | Permite establecer y consultar la fecha y la hora almacenada en la placa | |
| Características | Recolución de 25 ms.  Uso de métodos GetTime y SetTime | |
| UTIL\_H1 | Utility | Jhd1001 |  |
| Uso | Permite utilizar funciones comunes de formateo de números a cadenas, concatenarlas, etc. | |
| Características |  |  |
| AD1 canal 2 | ADC | Jhd1001 | A/D converter: ADC0 |
| Uso | Controla el pontenciometro 1 encargado de calcular el factor de la frecuencia de muestreo de los sensores | |
| Características | Conversion time: 19.230769 µs. Métodos MeasureChan y GetChanValue16 | |
| TI\_H1 | TimerInt | Jhd1001 | Periodic Interrupt: FTM1\_MOD |
| Uso | Timer que lanza una medición del potenciómetro | |
| Características | Interrupt period: 25 ms |  |
| EInt\_H1 | ExtInt | Jhd1001 | Pin: PTB23 |
| Uso | Controla el pulsador D4[[1]](#footnote-1) | |
| Características | Generate interrupt on: both edges. Métodos: GetVal | |
| FC16-H1 | FreeCntr16 | Jhd1001 | Periodic Interrupt: FTM2\_MOD |
| Uso | Se utiliza para controlar el tiempo de pulsación del pulsador D4 (D12 en el enunciado) | |
| Características | Resolution: 25 ms. Métodos: Reset y GetTimeMS | |
| AS\_H1 | AsynchroSerial | Jhd1001 | RxD: PTB16  TxD: PTB17 |
| Uso | Componente que permite comunicarse con el usuario por el puerto serie/USB para que introduzca la hora del sistema | |
| Características | Channel: UART0, Interrupt service activado, Input buffer size: 5, Output buffer size: 28, Paridad: none, Width: 8 bits, Stop bit: 1, Baud rate: 9600 | |
| SD\_E1 | SD\_Card | Aex0022 | Wait: WAIT\_E1  Timeout: TMOUT\_E1  HW SPI: SM\_E1 |
| Uso | Se utiliza para la conexión con la tarjeta SD | |
| Características | Utiliza el LLD (logical device driver) | |
| TMOUT\_E1 | Timeout | Aex0022 | Critical Section: CS\_E1 |
| Uso | Componente que maneja timeouts utilizando una sección critica. | |
| Características | No se ha configurado. | |
| CS\_E1 | CriticalSection | Aex0022 |  |
| Uso | Componente que maneja las secciones críticas. | |
| Características | No se ha configurado | |
| SM\_E1 | SPIMaster | Aex0022 | Input: PTE1  Output: PTE3  Clock: PTE2 |
| Uso | Maestro de la comunicación síncrona en serie. | |
| Características | Auto initialization: True; Attr set list con 2 atributos, con clock rate index a 0 y a 1; Clock rate: 375kHz; Delay after transfer: 0.095µs; CS to CLK: 0.095µs; CLK to CS: 0.095µs; | |
| FAT\_E1 | FAT\_FileSystem | Aex0022 | Utility: UTIL\_H1 |
| Uso | Modulo del sistema de ficheros FAT. | |
| Características | No se ha configurado | |
| Bit\_E1 | BitIO | Aex0022 | PTD2 |
| Uso | Entrada/salida de bit general. En este caso LED rojo. | |
| Características | Direction: Output | |
| FX1 | Acelerómetro | Apt1003 |  |
| Uso | Contiene los metodos para realizar la lectura de cada eje del acelerometro. | |
| Características |  | |
| ADC | Entradas Analógicas | Apt1003 |  |
| Uso | Contiene los metodos para iniciar y obtener los resultados del modulo de ADC conectados al sensor de temperatura, sensor de luz y potenciometro. | |
| Características | Resolucion de 16 bits, Modulos configurados ADC0 y ADC1 | |

# 

# Main.c

El fichero main.c contiene el software de inicialización del programa y el bucle principal en funciones en modo polling.

En la cabecera se definen los ficheros de cabecera:

#include "stdio.h"

#include "JHA.h"  
#include "AEX.h"

#include "app\_Accel.h"

#include "app\_Lum.h"

#include "app\_Selector.h"

#include "app\_Temp.h"

Ya en el bloque principal de la función, se ejecutan una serie de funciones de inicialización de componentes que detallaremos más adelante:

PORT\_PDD\_SetPinPullSelect(PORTE\_BASE\_PTR, 6, PORT\_PDD\_PULL\_DOWN);

PORT\_PDD\_SetPinPullEnable(PORTE\_BASE\_PTR, 6, PORT\_PDD\_PULL\_ENABLE);

if (FAT\_E1\_Init()!=ERR\_OK)

return -1;

if (FAT\_E1\_mount(&fileSystemObject, (const TCHAR\*)"0", 1) != FR\_OK)

return -1;  
JHA\_Run();

Y, si fuera necesario, se establece un bucle infinito para su funcionamiento en modo polling:

for (;;) {}

# Events.c

En este fichero se define el código en respuesta a eventos a través de interrupciones que le llegan al sistema.

Organizándolas por el programador:

## Código de componentes de JHA

* TI\_H1\_OnInterrupt: Cada 25 ms mide el potenciómetro (Canal 0). Llama a la función AD1\_MeasureChan(FALSE, 2);
* AD1\_OnEnd: Interrupción del potenciómetro, medida disponible. Llama a la función app\_SelectorTask();
* EInt\_H1\_OnInterrupt: Interrupción del pulsador D4, cuando se pulsa o cuando se suelta. Llama a la función JHA\_SetHora();
* AS\_H1\_OnFullRxBuf: El buffer de recepción en el componente de comunicación asíncrona AS\_H1, se ha llenado. Se ejecuta JHA\_OnFullRxBuf();

# Funciones de inicialización

## PORT\_PDD\_SetPinPullSelect y PORT\_PDD\_SetPinPullEnable

Se activa el pull-down para la detección de la tarjeta SD en el PTE6.

## FAT\_E1\_Init

Inicializa el controlador del sistema de ficheros FAT.

## FAT\_E1\_mount

Monta el sistema de ficheros FAT.

## JHA\_Run()

Inicializa el reloj interno al momento 00:00:00, el factor de frecuencia a 1 y el flag para recibir caracteres por el puerto serie a FALSE:

void JHA\_Run(void) {

TmDt\_H1\_SetTime(0,0,0,0);

factor = 1.0;

recibir = FALSE;

}

# Funciones de tratamiento

Este es el grupo de funciones que programan el comportamiento general del programa. También las detallaremos organizadas por programador.

## Código de JHA

Disponibles en el fichero JHA.c son llamadas desde funciones de interrupción o desde otras funciones, del propio programador o de otros.

### char \* getHora(void)

Proporciona la hora del sistema en un formato texto ajustado a los requerimientos del proyecto.

### void JHA\_Factoriza(void)

Calcula el factor que hay que aplicar a la periodicidad de las medidas. Los valores posibles son:

* 0 si el potenciómetro p = 0 o p < 32767 y va creciendo
* 1 si p < 32767 cuando va decreciendo, si p = 32767 o si p > 32767 cuando va creciendo
* 2 si p = 65535 o si p > 32767 cuando va decreciendo

En primer lugar se obtiene el valor del potenciómetro:

word valores;

AD1\_GetChanValue16(&valores, 2);

A cotinuación, se calcula el factor que se almacena en una variable global

if (valores==0) factor = 0;

else if (valores==32767) factor = 1;

else if (valores==65535) factor = 2;

else if (valores < 32767 && factor == 2) factor = 1;

else if (valores > 32767 && factor == 0) factor = 1;

else factor = factor;

### void JHA\_SetHora(void)

Trata el establecimiento de la hora mediante el pulsador D4.

Cuando se presiona D4, se resetea el contador FC16\_H1:

if (EInt\_H1\_GetVal()) {

FC16\_H1\_Reset();

}

Si se suelta el pulsador, y el tiempo transcurrido es más de 2 segundos, se solicita la hora.

else {

// se obtiene el valor del contador

FC16\_H1\_GetTimeMS(&time);

if (time>=2000) { // si han pasado más de 2000 ms

// se limpian los buffers

AS\_H1\_ClearTxBuf();

AS\_H1\_ClearRxBuf();

// se manda mensaje a la terminal

while (AS\_H1\_SendBlock(mensaje, strlen(mensaje),&env)!=ERR\_OK) {}

// y se activa la recepción por interrupción

recibir = TRUE;

}

}

### void JHA\_OnFullRxBuf()

Trata la interrupción de llenado del buffer de entrada. Se produce cuando se han tecleado 5 caracteres (tamaño del buffer de entrada) en el terminal. Su funcionalidad solo se ejecutará cuando el flag recibir es TRUE, por lo que, previamente, se debe haber ejecutado la función JHA\_SetHora explicada anteriormente.

Se comienza obteniendo los datos desde el buffer:

// carga en datos el contenido del buffer

AS\_H1\_RecvBlock((byte\*)&datos, sizeof(datos), &Recibido);

A continuación, se adaptan los caracteres recibidos al formato adecuado de la hora:

// transforma lo tecleado en la hora

parseHora(&datos, &hora);

Establece el reloj del sistema con la hora recibida

// y establece la hora del sistema

TmDt\_H1\_SetTime(hora.Hour, hora.Min, hora.Sec, hora.Sec100);

Se desactiva el flag para que no se vuelva a tratar lo recibido desde el terminal

// desactiva la recepción evitando que nuevas pulsaciones cambien

// indeseadamente la hora

recibir = FALSE;

Y se envía un mensaje confirmando el cambio de hora

// y emite un mensaje de que la hora ha sido cambiada.

while (AS\_H1\_SendBlock(mensaje2, strlen(mensaje2),&env)!=ERR\_OK) {}

### void parseHora(char \*datos, TIMEREC \*hora)

Adapta los datos recibidos como cadena de caracteres en formato hh:mm al tio de datos TIMEREC que maneja las funciones del reloj del sistema.

Se forma la subcadena con la hora y se convierte a entero

char h[] = {datos[0],datos[1],'\0'};

hora->Hour = atoi(h);

Se forma la subcadena con los minutos y se convierte a entero

char m[] = {datos[3],datos[4],'\0'};

hora->Min = atoi(m);

Se asigna el valor 0 a segundos y milisegundos

hora->Sec = 0;

hora->Sec100 = 0;

## Código de AEX

### void CloseFP()

Realiza el cierre del objeto de fichero que es responsable de grabar en la tarjeta SD.

También se ocupa de apagar el led rojo.

Esta función se llama desde todas las funciones responsables de escribir en la tarjeta SD.

### int LogToSDCard(int16\_t x, int16\_t y, int16\_t z)

### int LogToSDCardL(int16\_t x, int16\_t y, int16\_t z, int16\_t luminocidad)

### int LogToSDCardT(int16\_t x, int16\_t y, int16\_t z, int16\_t temperatura)

### int LogToSDCardLT(int16\_t x, int16\_t y, int16\_t z, int16\_t luminocidad, int16\_t temperatura)

Todas las funciones de escribir en la tarjeta SD actúan de la misma manera.

Se declara el buffer que llevará la información.

uint8\_t write\_buf[80];

Y el entero sin signo que recogerá el número de bytes escritos en el fichero.

UINT bw;

Posteriormente se realizan operaciones de abrir el fichero y desplazarse al final para empezar a escribir. Las funciones utilizadas son:

FAT\_E1\_open(&fp, "./log.txt", FA\_OPEN\_ALWAYS|FA\_WRITE)  
 FAT\_E1\_lseek(&fp, f\_size(&fp))

Para la apertura se indica el objeto del fichero de tipo **FIL**, el nombre del fichero y los permisos.  
Para ir al final se pasa de nuevo el objeto del fichero, y el tamaño del mismo.

Posteriormente se empieza a rellenar el buffer con los datos necesarios.

Se obtiene la hora:

strcpy(write\_buf, getHora());

Y utilizando el objeto UTIL, se rellena el buffer con la información necesaria separada por tabulaciones.

UTIL\_H1\_strcat(write\_buf, sizeof(write\_buf), (unsigned char\*)"\tX:");

UTIL\_H1\_strcatNum16s(write\_buf, sizeof(write\_buf), x);

Cuando la información ya está completa, se enciende el led rojo y se escribe en la tarjeta SD:

FAT\_E1\_write(&fp, write\_buf, UTIL\_H1\_strlen((char\*)write\_buf), &bw)

La función recoge, el objeto del fichero, el buffer con los datos a escribir, el tamaño del buffer y el entero que indicará el tamaño en bytes de la información que se escribió.

Posteriormente se cierra el objeto del fichero llamando a la función CloseFP.

Cada función de escritura en la tarjeta devuelve 0 en caso de éxito o -1 si ha tenido algún problema.

# Fichero escrito en la tarjeta SD

El fichero escrito en la tarjeta SD se muestra a continuación (con datos de prueba):

0:5:59:77 X:99 Y:99 Z:25 Luminosidad:92

0:6:2:27 X:99 Y:99 Z:25

0:6:4:77 X:99 Y:99 Z:25

0:6:7:27 X:99 Y:99 Z:25 Temperatura:304ºC

0:6:9:77 X:99 Y:99 Z:25

0:6:12:27 X:99 Y:99 Z:25

0:6:14:77 X:99 Y:99 Z:25

0:6:17:27 X:99 Y:99 Z:25

0:6:19:77 X:99 Y:99 Z:24 Luminosidad:92

0:6:22:27 X:99 Y:99 Z:25

0:6:24:77 X:99 Y:99 Z:24

0:6:27:27 X:99 Y:99 Z:24

0:6:29:77 X:99 Y:99 Z:25

0:6:32:27 X:99 Y:99 Z:25

0:6:34:77 X:99 Y:99 Z:25

0:6:37:27 X:99 Y:99 Z:25

0:6:39:77 X:99 Y:99 Z:24 Luminosidad:92

0:6:42:27 X:99 Y:99 Z:25

0:6:44:77 X:99 Y:99 Z:25

0:6:47:27 X:99 Y:99 Z:25

0:6:49:77 X:99 Y:99 Z:25

0:6:52:27 X:99 Y:99 Z:24

0:6:54:77 X:99 Y:99 Z:24

0:6:57:27 X:99 Y:99 Z:25

0:6:59:77 X:99 Y:99 Z:25 Luminosidad:84 Temperatura:304ºC

## Código de APT

### Lectura del acelerómetro (app\_Accel.c)

**app\_AccelTask():**

Se encarga llamar a los métodos necesarios para la lectura de cada eje del acelerómetro. Una vez obtenidos, se envían mediante UART junto con texto de referencia. Los resultados se almacenan en variables públicas para su lectura posterior

bool app\_AccelTask(void)

{

//Revisa si ha pasado el timpo para ejecutarse

if(rub\_CallTimes > raub\_AccelExecTime[factor])

{

rub\_CallTimes = 0u;

rub\_Jump |= TRUE;

x = FX1\_MeasureGetRawX();

y = FX1\_MeasureGetRawY();

z = FX1\_MeasureGetRawZ();

return TRUE;

}

else

{

rub\_CallTimes++;

return FALSE;

}

}

### Lectura de Canales ADC

**AD1\_GetMeasure(uint16\_t\* buffer):**

Obtiene los valores sin procesar de todos los canales de ADC y los almacena en un buffer.

void app\_SelectorTask(void) {

rub\_Jump = FALSE;

word valores;

AD1\_GetChanValue16(2, &valores);

if (valores==0) factor = 0;

else if (valores==32767) factor = 1;

else if (valores==65535) factor = 2;

else if (valores < 32767 && factor == 2) factor = 1;

else if (valores > 32767 && factor == 0) factor = 1;

else factor = factor;

}

**app\_TemperaturaTask()**

Toma el valor correspondiente del buffer y lo somete a una conversión matemática para representar temperatura. Una vez calculado, se envía mediante UART junto con texto de referencia. Los resultados se almacenan en variables públicas para su lectura posterior

bool app\_TempTask(void)

{

if(rub\_CallTimes > raub\_TempExecTime[factor])

{

rub\_CallTimes = 0u;

rub\_Jump |= TRUE;

// mide la temperatura

AD1\_MeasureChan(TRUE,0);

(void)AD1\_GetChanValue16(0, &rauw\_ADCRaw);

Temperatura = rauw\_ADCRaw / 200u;

return TRUE;

}

else

{

rub\_CallTimes++;

return FALSE;

}

}

**app\_LumTask()**

Toma el valor correspondiente del buffer y lo somete a una conversión matemática para representar luminosidad relativa. Una vez calculado, se envía mediante UART junto con texto de referencia. Los resultados se almacenan en variables públicas para su lectura posterior

bool app\_LumTask(void)

{

if(rub\_CallTimes >= raub\_LumExecTime[factor])

{

rub\_CallTimes = 0u;

rub\_Jump |= TRUE;

// mide la temperatura

AD1\_MeasureChan(TRUE,1);

(void)AD1\_GetChanValue16(1, &rauw\_ADCRaw);

Luminosidad = rauw\_ADCRaw / 655u;

return TRUE;

}

else

{

rub\_CallTimes++;

return FALSE;

}

}

1. Se ha utilizado el pulsador D4 en lugar del propuesto D12 porque en la placa de pruebas, no funcionaba dicho pulsador, es posible que por un defecto del mismo. [↑](#footnote-ref-1)