UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Dispositivo basado en Arduino para intercambio de ficheros en memorias flash

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Daniel Parra López

Tutor: Floreal Acebrón Linuesa

2014-2015

Me gustaría agradecerle a mi tutor Floreal por su implicación con el proyecto y la ayuda aportada en la resolución de problemas y organización del proyecto. También quería agradecerle a mi padre Juan José por su apoyo y gran ayuda en la parte electrónica del proyecto y a Mireia por su aportación con los gráficos de la interfaz. Para terminar me gustaría darle las gracias a Andrew y a su equipo, desarrolladores de la librería USB, por el soporte brindado.

Resumen

El proyecto ha consistido en desarrollar un dispositivo basado en Arduino capaz de transferir información entre memorias de almacenamiento flash USB. Se ha utilizado un HUB USB conectado a una placa Arduino, una batería que suministra corriente a la propia placa y a las memorias flash, una pantalla táctil, a través de la cual el usuario podrá interactuar con la aplicación de intercambio de información y varios integrados de memoria SRAM.

Palabras clave: Arduino, USB, memorias flash, pantalla táctil, transmitir información, intercambio ficheros.

Abstract

The project has been to develop a device based on Arduino capable of transferring information between memories of USB flash. It has been used a USB hub connected to an Arduino board, a battery that supplies power to the board itself, a touch screen through which the user can interact with the application for the exchange of information and several SRAM memory integrated circuits.

Keywords : Arduino, USB, flash memories, touch screen, information transmission, exchange of files.

Tabla de figuras

Acrónimos

USB	*Universal Serial Bus*
FAT	*File Allocation Table*
NTFS	*New Technology File System*
ext4	*Fourth Extended Filesystem*
LCD	*Liquid Crystal Display*
USB OTG	*USB On The Go*
FTDI	*Future Technology Devices International*
SD	*Secure digital*
RAM	*Random Access Memory*
SRAM	*Static Random Access Memory*
DDR	*Double Data Rate*
LPDDR	*Low Power DDR*
B	*Byte*
KB	*Kilobyte*
GB	Gigabyte
MHz	Megahercio
GHz	Gigahercio
EEPROM	*Electrical Erase Programmable Read-Only Memory*
IDE	*Integrated Development Environment*
ICSP	*In-Circuit Serial Programming*
CS	*Chip Select*
SO	Sistema Operativo
PID	*Packet Identifier Field*
CRC	*Cyclic Redundancy Check*
JPEG	*Joint Photographic Expert Group*

Tabla de contenidos

1. INTRODUCCIÓN

Todos nos hemos visto en la problemática de transferir datos entre distintas memorias flash USB o nos hemos quedado sin espacio en la memoria del teléfono móvil y no hemos podido almacenar más archivos. Con el dispositivo desarrollado en el proyecto es posible copiar información entre distintos *pendrives* sin necesidad de un ordenador.

Las motivaciones que han llevado a desarrollar este proyecto son varias, una de las principales es desarrollar un dispositivo capaz de transferir información entre múltiples *pendrives* y teléfonos móviles sin la necesidad de un ordenador. Otra de las motivaciones ha sido ampliar y mejorar los conocimientos sobre la plataforma Arduino.

Los objetivos que se propusieron en un primer momento fueron: desarrollar un dispositivo basado en Arduino capaz de intercambiar ficheros entre memorias flash, sin intervención de ningún *hardware* y *software* adicional. Este dispositivo debía de soportar los diferentes sistemas de ficheros, FAT, NTFS, ext4, etc. y también ser capaz de comunicarse con dispositivos móviles sin importar el sistema operativo. Dada la complejidad del proyecto los objetivos fueron reducidos de forma que el dispositivo es capaz de comunicarse con múltiples *pendrives y* soporta el sistema de archivos FAT. Se deja como posibles ampliaciones el soporte de los sistemas de ficheros NTFS, ext4, etc., así como la comunicación con dispositivos móviles con distinto sistema operativo.

El proyecto se encuentra dividido en 10 partes. Las dos primeras conforman la introducción y estado del arte donde veremos que hay en el mercado y las posibles alternativas. En la tercera, cuarta y quinta parte, se verá que es Arduino y los protocolos USB y FAT. En la sexta se analizará el entorno de trabajo utilizado para el proyecto. En la séptima se describirá el desarrollo que se ha seguido para el proyecto, tanto a nivel *hardware* como *software*. En la octava se expondrá la problemática encontrada en el desarrollo del proyecto. En la novena se comentarán posibles ampliaciones al proyecto. Por último se encuentra la conclusión, donde una vez finalizado el proyecto se han comentado los resultados.

2. ESTADO DEL ARTE

Se ha analizado el mercado y actualmente no se ha encontrado ningún dispositivo comercial que realice operaciones de copiado entre dos memorias de almacenamiento flash USB, sin la ayuda de un tercer dispositivo de cómputo, como un ordenador, teléfono móvil, PDA, etc.

Existe un dispositivo comercial para transferir datos entre dispositivos móviles y memorias USB denominado USB On-The-Go (USB OTG), este dispositivo permite conectar una memoria flash USB a un dispositivo móvil con sistema Android. Se ha podido observar que este dispositivo solo funciona con sistemas Android y no es compatible con todos los dispositivos. Su precio varía entre 1€ y 3€.

Figura 1: Cable USB OTG

Existe otro dispositivo comercial llamado *Leef Bridge* para su versión Android y *Leef iBridge* para su versión iOS. Este dispositivo permite transferir datos desde el dispositivo móvil al dispositivo en cuestión. Su precio varía entre 15€ y 45€, según la capacidad del mismo, para la versión Android y entre 55€ y 370€ para la versión iOS.

Figura 2: Leef Bridge - Android

Figura 3: Leef iBridge - iOS

3. ARDUINO

Arduino es una plataforma compuesta por *hardware* y *software* libre, basada en una placa con un microcontrolador Atmel AVR, aunque actualmente también se utilizan los microcontroladores CortexM3 de ARM. Esta plataforma utiliza su propio IDE de desarrollo y se programa con el lenguaje Arduino que está basado en el lenguaje de alto nivel Processing, que a su vez está basado en Java. Sin embargo, se pueden utilizar otros lenguajes de programación gracias a la transmisión serie de datos que utiliza Arduino y es soportada por la mayoría de los lenguajes, también es posible utilizar software intermediario que traduzca los mensajes enviados por ambas partes. Algunos de los lenguajes son: C, C++, C#, Cocoa, Flash, Java, Mathematica, Matlab, Perl, PHP, Python, etc.

Esta plataforma cuenta con gran cantidad de placas auxiliares o *shields* para dotar a Arduino de múltiples funcionalidades. Arduino GSM *Shield* es una de estas *shield*, la cual proporciona funcionalidades GPRS que permiten recibir y enviar llamadas y mensajes SMS, Arduino WiFi *Shield* es otro ejemplo que permite conectarse a la red inalámbrica, Arduino USB Host *Shield* permite conectar dispositivos USB, otra *shield* como la Arduino Motor *Shield* permite controlar múltiples motores. También es posible conectar accesorios con Arduino, como pantallas LCD, teclados serie y prácticamente todo lo que necesitemos y sea electrónicamente compatible.

3.1. Alternativas

Existen varias alternativas a Arduino con distintas características y sistemas operativos, algunas de ellas son BeagleBone, Raspberry Pi, Gizmo 2 y MinnowBoard. A continuación se comentan las principales características de cada una de ellas.

BeagleBone es un ordenador basado en Linux del tamaño de una tarjeta de crédito, que puede ejecutar un sistema operativo como Ubuntu o Android 4.0. Puede utilizar lenguajes de programación de alto nivel como C++, Java y Node.js. El modelo BeagleBone Black utiliza un procesador AM335x 1 GHz ARM cortex A8, dispone de una memoria RAM DDR3 de 512 MB, acelerador grafico 3D, conexión Ethernet, HDMI y un puerto USB. Su precio es de 40€.

Figura 4: BeagleBone

Raspberry Pi, como el anterior, es un miniordenador que puede trabajar con sistemas basados en Linux y se está trabajando para que funcione con Windows 10. Puede utilizar lenguajes de programación de alto nivel como C++, Java y Python. El modelo Raspberry Pi 2 model B utiliza un procesador *quad-core* a 900 MHz, ARM Cortex-A7, dispone de una memoria RAM LPDDR 2 de 1 GB, acelerador gráfico, cuatro puertos USB, HDMI, Ethernet y tiene un *slot* para una tarjeta micro SD. Su precio es de 33 €.

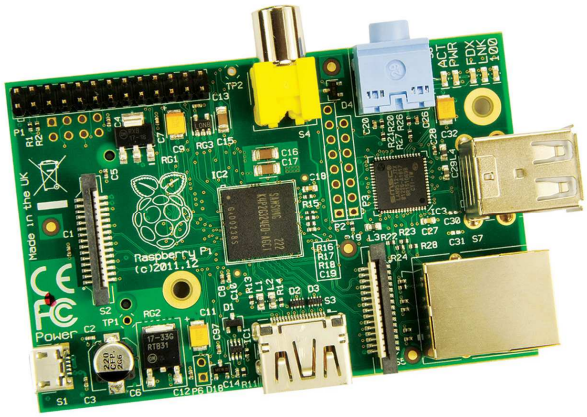

Figura 5: Raspberry Pi 2 model B

Gizmo 2, se trata del miniordenador de AMD el cual trabaja con Windows y Linux. Utiliza un procesador AMD de doble núcleo a 1 GHz, dispone de una memoria RAM DDR3 de 1 GB, el chip gráfico es un Radeon HD 8210E, incluye un puerto HDMI, un conector mSATA, un puerto Ethernet, un *slot* para una tarjeta Micro SD, dos puertos USB 3.0 y dos puertos USB 2.0. Este miniordenador es el único que requiere de disipador de calor. Su precio es de 160€.

Figura 6: Gizmo 2

MinnowBoard, miniordenador de Intel, el cual trabaja con Debian, Windows 8.1, Android 4.4 y es compatible con el proyecto Yocto. El modelo MinnowBoard MAX dispone de dos versiones, la versión económica que utiliza el procesador de un solo núcleo Intel Atom E3815 a 1.46 GHz y dispone de una memoria RAM DDR3 de 1GB mientras que la otra versión utiliza un procesador doble núcleo Intel Atom E3825 a 1.33 GHz y dispone de una memoria RAM DDR3 de 2 GB. Ambas versiones incluyen un puerto HDMI, un conector SATA, un puerto Ethernet, un *slot* para una tarjeta Micro SD, un puerto serie FTDI, un puerto USB 3.0 y un puerto USB 2.0. El precio de estos dos miniordenadores son de 90 € y 130 € respectivamente.

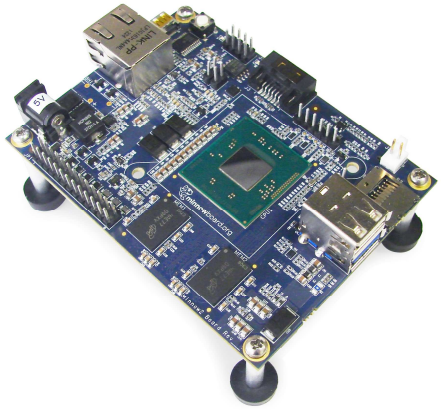

Figura 7: MinnowBoard MAX

Después de analizar las distintas alternativas se decidió utilizar la plataforma Arduino, aunque las alternativas tienen un hardware más potente, existe una gran comunidad de desarrolladores y gran variedad de componentes *hardware* y librerías *software* de código abierto que se pueden utilizar con Arduino. El precio también fue un factor determinante ya que el precio de la placa Arduino Mega 2560, utilizada en el proyecto y que detallaremos en la sección de hardware utilizado, es de 35€, esto es un factor importante de cara a una futura comercialización del producto.

4. USB

USB es un estándar que define los cables, conectores y protocolos usados para comunicar computadores y periféricos mediante el bus USB. El estándar nació de un grupo de empresas del sector, entre las que estaban Compaq, Nortel, Intel, IBM, Microsoft y NEC. El protocolo USB permite conectar dispositivos en caliente, sin necesidad de reiniciar el sistema, además los dispositivos conectados se configuran automáticamente, algunos de estos periféricos son: teclado, ratón, memorias flash USB, cámaras y *joysticks*.

Existen distintas velocidades de transmisión según la versión, son estas:

- Baja velocidad (1.0): Tiene una tasa de transferencia de hasta 1.5 Mbits/s. Se utiliza en su mayoría por dispositivos de interfaz humana como los ratones, teclados, cámaras, etc.
- Velocidad completa (1.1): Tiene una tasa de transferencia de hasta 12 Mbits/s. Los dispositivos dividen el ancho de banda entre ellos, usando un algoritmo LIFO.
- Alta velocidad (2.0): Tiene una tasa de transferencia de hasta 480 Mbits/s, aunque la tasa real está sobre los 280 Mbits/s. El cable USB 2.0 dispone de dos líneas para datos y dos para alimentación. La mayoría de dispositivos actuales trabajan a esta velocidad.
- Súper velocidad (3.0): Tiene una tasa de transferencia de hasta 4.8 Gbits/s. Es posible alcanzar esta velocidad gracias a que se han añadido 4 líneas de datos. Actualmente se está trabajando en la versión 3.1 que soportara una tasa de transferencia de como mínimo 10 Gbits/s.

Para el dialogo entre el equipo del usuario y el dispositivo USB se emplean tres tipos de tramas o paquetes. Cada una de ellas se utiliza según el tipo de comunicación:

Los paquetes de datos se utilizan para el intercambio de información. Están formados por un PID, un campo con los datos, que puede variar de 0 a 1023 B, y un campo CRC generado a partir de los datos.

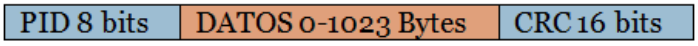

Figura 8: Ejemplo de trama de datos

Los paquetes *Handshake* solo contienen un PID, se utilizan para informar del estado de la transmisión y pueden devolver valores indicando la correcta recepción de datos, comandos y situaciones de parada y control de flujo.

Los paquetes *Token* se utilizan para la identificación del destinatario y origen. Están constituidos por un PID y los campos: ADDR, ENDP y CRC. En transacciones de OUT y SETUP los campos ADDR y ENDP identifican al destinatario que recibirá los paquetes de datos. En transacciones de IN identifican al dispositivo USB que transmite la información. Los *token* del tipo STO, proporcionan información de sincronización.

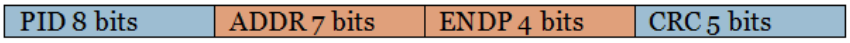

Figura 9: Ejemplo de trama token

La información del PID, es utilizada para identificar a la trama, formato del paquete, mecanismos de detección de errores empleados e interfaz USB, consta de 8 bits, 4 de ellos son utilizados para codificar la información nombrada y los 4 restantes para comprobación de errores del PID. En la figura 11 podemos ver la codificación utilizada.

Tipo PID	Nombre PID	Codificación	Descripción
Data	DATA0 DATA1	0011 1011	Paquete de datos PAR. Paquete de datos IMPAR.
Handshake	ACK NAK STALL	0010 1010 1110	Recepción de paquete libre de errores. El receptor no puede aceptar . No puede soportar el comando.
Token	OUT IN SOF SETUP	0001 1001 0101 1101	Dirección del dispositivo destinatario. Dirección del dispositivo transmisor. Indicador de comienzo de trama. Dirección del dispositivo destinatario de comandos de control.
Special	PRE	1100	Preámbulo para la comunicación con dispositivos de baja velocidad.

Figura 10: Codificación PID

5. FAT

FAT es el sistema de archivos creado por Bill Gates y Marc McDonald en 1977, su principal característica es que el dispositivo tiene reflejado el estado de cada unidad de almacenamiento en una tabla llamada *File Allocation Table*, de ahí su nombre. Esta tabla contiene el índice del contenido del disco y esta duplicada para evitar pérdidas, ambas copias ocupan los primeros sectores del disco.

Los sistemas FAT utilizan un método de grabación que agrupa varios sectores en un mismo *cluster*. Un sector es la unidad mínima de almacenamiento y siempre es de 512 B, esto es debido a la estructura física del dispositivo. Sin embargo, el tamaño de un *cluster* puede variar según el tamaño de la unidad, para asignar un nuevo tamaño necesitamos formatear el dispositivo, ya que esto forma parte de la estructura lógica de la unidad.

En la tabla de ficheros se almacenan los *clusters* que ocupa cada fichero, permitiendo así que el archivo ocupe espacios no consecutivos, esto también ralentizará el acceso al contenido del mismo. Un archivo puede tener un tamaño máximo de 4 GB y un tamaño de partición de 32 GB, esto es debido a una imposición de diseño por parte de Windows.

6. ENTORNO DE TRABAJO

En este apartado se describen el *hardware* y software utilizado para llevar a cabo este proyecto. En primer lugar se describen cada uno de los dispositivos *hardware* empleados y a continuación los programas *software* utilizados.

6.1. Hardware utilizado

6.1.1. Arduino Mega 2560

Este dispositivo contendrá el programa y será el encargado de ejecutarlo y establecer comunicación con el resto de dispositivos *hardware*. Se trata de una placa Arduino, la cual utiliza el microcontrolador ATmega2560, dispone de 54 pines de entrada/salida digital, 16 entradas analógicas, 4 controladores para comunicación serie, un conector ICSP, un cristal oscilador de 16 MHz para sincronizar los dispositivos conectados, un puerto USB para programarla y un conector para conectar un alimentador externo. Tiene 256 KB de memoria flash, de los cuales 8 KB los utiliza el cargador del programa y el resto son utilizados para almacenar el código del programa, 8 KB de memoria SRAM y 4 KB de memoria EEPROM. Como ya vimos en el apartado de alternativas, su precio es de 35€. Para más información consultar el anexo I.

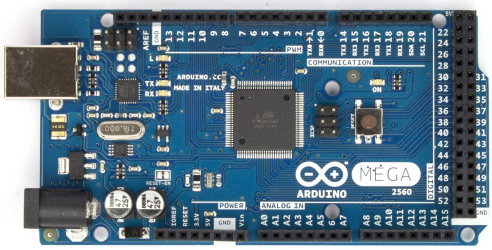

Figura 11: Arduino Mega 2560 frontal

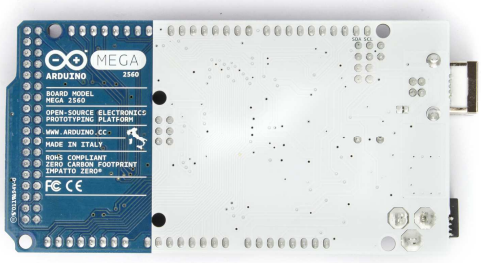

Figura 12: Arduino Mega 2560 trasera

6.1.2. USB *Host Shield*

Este dispositivo será el encargado de comunicar los diferentes *pendrives* o dispositivos móviles conectados a él con la placa Arduino. Además de soportar estos periféricos también es capaz de comunicarse con joysticks, cámaras digitales, dispositivos *bluetooth* y otros dispositivos. Esta *shield* incorpora el microcontrolador MAX3421E, que es el encargado de establecer comunicación con la placa Arduino mediante el protocolo SPI que veremos en el apartado de desarrollo, la versión del protocolo USB que utiliza es la 2.0. Su precio es de 25€. Para más información consultar el anexo II.

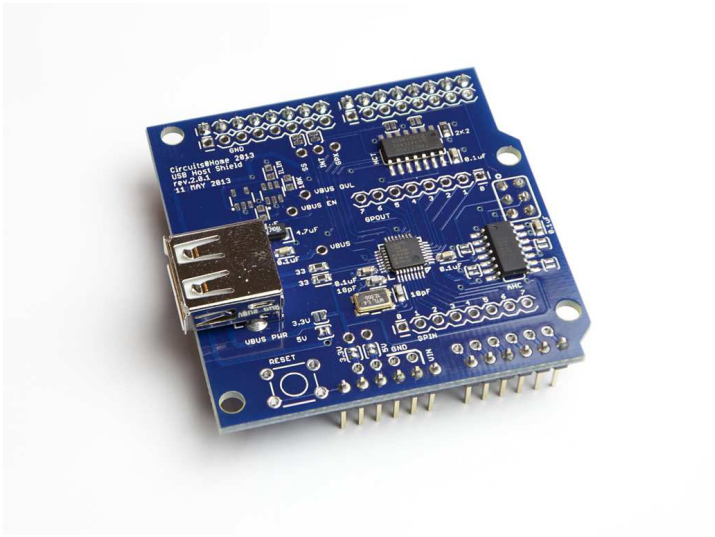

Figura 13: USB Host Shield

6.1.3. Display SmartGPU 2

SmartGPU 2 es una pantalla táctil de 3.5" que utilizaremos como interfaz para la comunicación entre el dispositivo y el usuario. Este dispositivo utiliza una comunicación serie para comunicarse con la placa Arduino. Es capaz de reproducir video y audio, además de mostrar imágenes almacenadas en una memoria SD y dibujar primitivas como líneas, rectángulos, círculos, etc. Su precio es de 115€. Para más información consultar el anexo III.

Figura 14: SmartGPU 2 frontal

Figura 15: SmartGPU 2 trasera

6.1.4. Integrados

Para este proyecto se han utilizado 3 tipos de integrados:

- Tres chips de memoria para ampliar la memoria SRAM de la placa Arduino.
- Un decodificador para gestionar los distintos chips de memoria.
- Un driver para aislar de posibles cortocircuitos la placa Arduino y los chips de memoria.

El precio de estos componentes suma 5€.

El chip de memoria que se ha utilizado es el 23LC1024, que proporciona 128 KB de memoria SRAM extra. Este chip se comunica con la placa Arduino utilizando el conector ICSP que implementa el protocolo SPI y se activa a nivel bajo. La conexión con Arduino la veremos en la sección desarrollo.

Estos chips se han utilizado de la siguiente forma:

- El primero de ellos se utiliza para cargar los nombres del directorio actualmente mostrado.
- El segundo se utiliza para almacenar la ruta y nombre de los archivos que van a ser copiados.
- El tercero se pensó que funcionara como *buffer* de lectura y escritura de archivos pero actualmente no está en uso ya que no se requiere.

Para más información consultar el anexo IV.

El decodificador utilizado es el HCF4556BE, este integrado es el encargado de habilitar el chip de memoria correspondiente en cada momento y asegurarse de que solo uno de ellos está activo a la vez. Aquí podemos ver la tabla de funcionamiento de este integrado. Para más información consultar el anexo V.

Entradas			Salidas			
/E	B	A	/Q3	/Q2	/Q1	/Q0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	X	X	1	1	1	1

Figura 16: Tabla de funcionamiento HCF4556BE

El driver SN74HCT245 se encarga de aislar los chips de memoria citados en este apartado y la placa Arduino protegiéndola así de posibles problemas eléctricos. Este transceptor (Transmisor-receptor) de bus octal está diseñado para la comunicación asíncrona entre buses de datos.

Los dispositivos HCT245 permiten la transmisión de datos desde el bus A al bus B o desde el bus B al bus A, dependiendo del nivel lógico en la entrada del control de dirección (DIR). La entrada de activación de salidas (OE) se puede utilizar para desactivar el dispositivo de modo que los buses A y B quedan aislados. Para más información consultar el anexo VI.

Entradas		Operación
OE	DIR	
0	0	Transmite de B a A
0	1	Transmite de A a B
1	X	Aislamiento

Figura 17: Tabla de funcionamiento SN74HCT245

6.2. Software utilizado

Cuando se empezó el desarrollo de este proyecto las versiones del IDE de Arduino no permitían compilar la librería USB directamente y teníamos que utilizar archivos *Makefile*, por lo que se comenzó trabajando con el SO Linux Ubuntu, ya que facilitaba bastante el uso de estos archivos y proporcionaba una fácil instalación de las librerías AVR, utilizadas para cargar el programa compilado en la placa Arduino.

Para la edición de código se utilizaba el editor Geany, un editor de textos para sistemas basados en Linux, capaz de entender y dar formato a múltiples lenguajes, entre ellos C, C++, Java, HTML, CSS, PHP, Pascal, etc. Se eligió este editor por su sencillez y por experiencia anterior con el mismo.

Figura 18: Geany

Después de la actualización a la versión 3.0 de la librería USB utilizada, se comenzó a trabajar con Windows y con la última versión del IDE de Arduino, 1.6.3, que permitía editar el programa principal y cargarlo sobre la placa Arduino. Como editor de código se pasó a utilizar Notepad++ por su sencillez y código de colores que ayudan a leer el código. Se probó Visual Studio 2013 que además de contar con un código de colores, dispone de funciones de autocompletado y ayuda, se descartó este IDE por la pesadez del mismo.

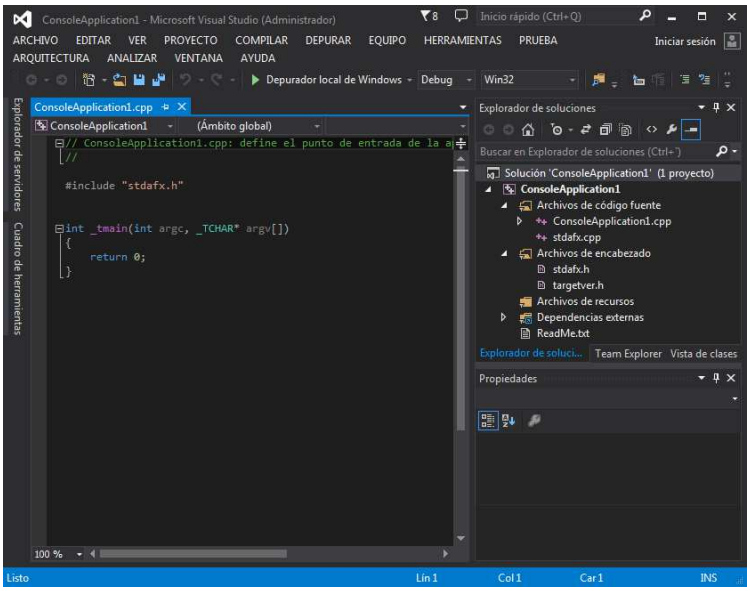

Figura 19: Visual Studio

Figura 20: Arduino IDE

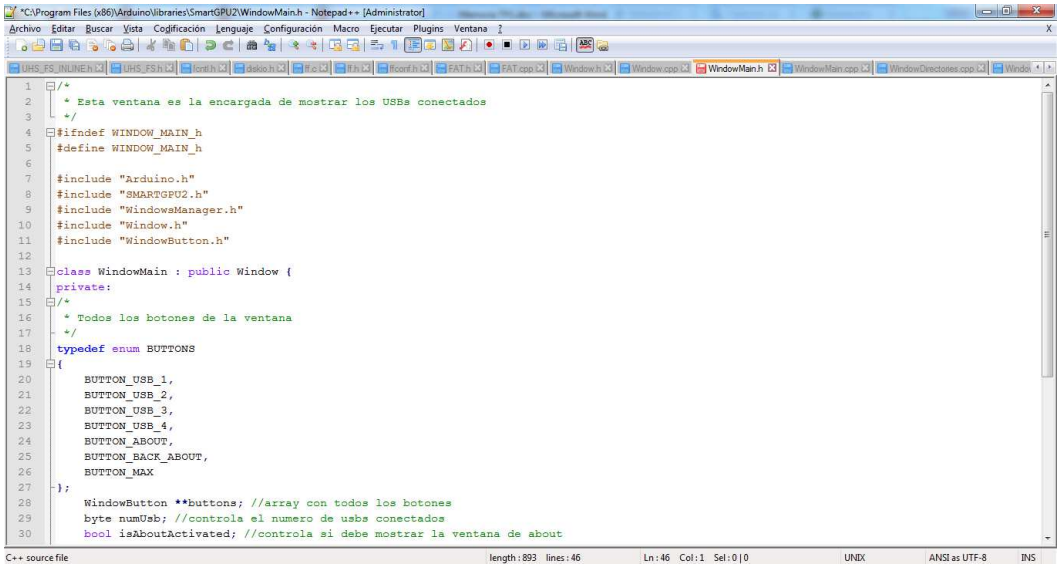

Figura 21: Notepad++

7. DESARROLLO

En esta sección se describe como se relacionan y comunican todos los dispositivos *hardware* entre ellos y así como las librerías utilizadas.

7.1. Hardware

En primer lugar se describe cómo está conectado el *hardware*. Para el proyecto se ha desarrollado una *shield* con todos los integrados, que nombramos en el apartado de *hardware* utilizado, y las conexiones necesarias para comunicar con la placa USB y el *display*. La conexión de todos los componentes es la que podemos ver en la figura 22.

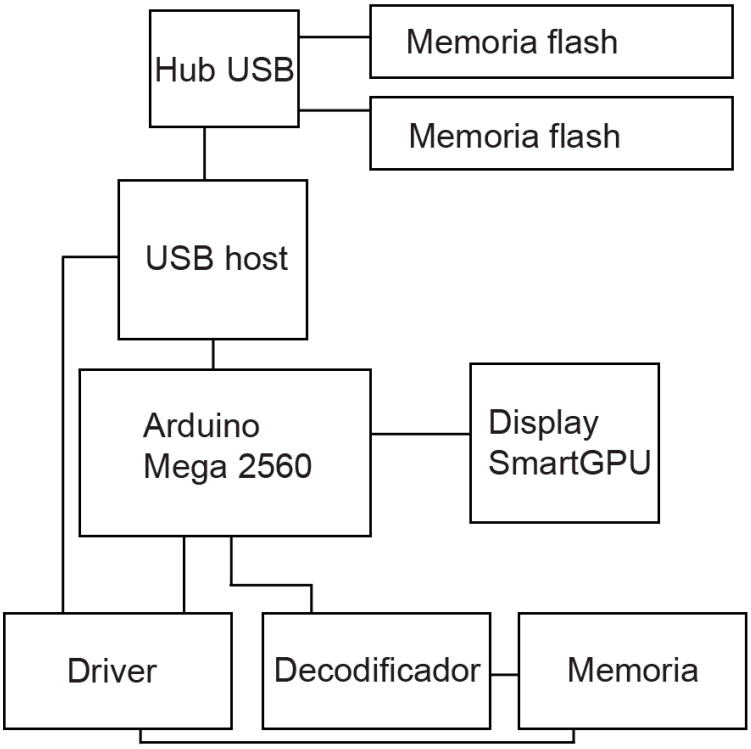

Figura 22: Diagrama de conexión

Como vemos en la figura 23, conectamos el pin de selección de cada chip de memoria 23LC1024 a las distintas salidas del decodificador HCF4556BE, los pines de salida de datos de las memorias al pin 18 del driver SN74HCT245, los pines de entrada de datos de las memorias al pin 3 del driver, el pin MOSI de Arduino al pin 17 del driver, el pin MISO de Arduino al pin 2 del driver y los relojes de los chips de memoria los conectamos con el reloj del conector ICSP de Arduino. También conectamos la alimentación y tierra de cada componente con la placa Arduino.

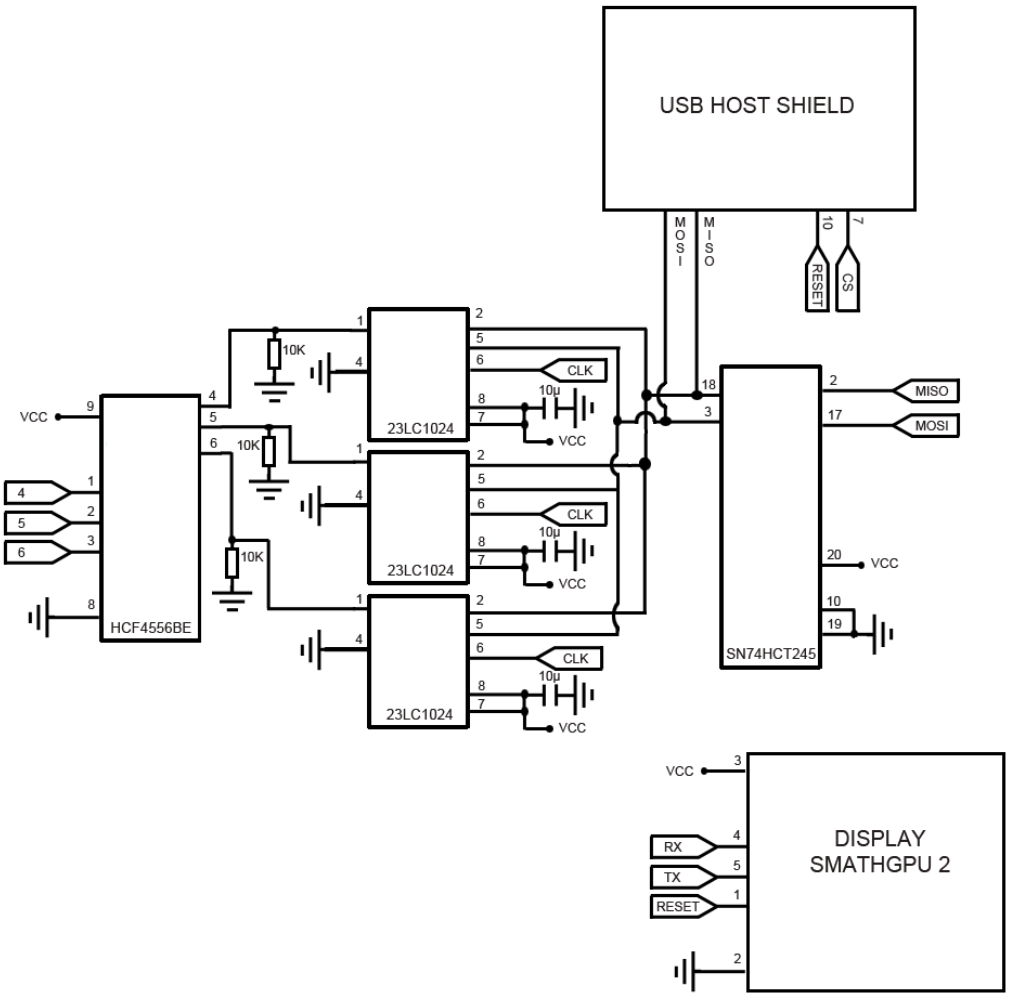

Figura 23: Esquema de conexión

Para controlar el decodificador conectamos las entradas de control 1, 2 y 3 a los pines 4, 5 y 6, respectivamente, de la placa Arduino.

Los pines RX y TX del *display* SmartGPU 2 lo conectamos a los pines los pines RX y TX del puerto serie 2, además de los conectores para la alimentación, toma de tierra y *reset*.

La USB host *shield,* igual que los chips de memoria 23LC1024, utilizan los conectores ICSP. Para evitar conflictos se ha redirigido el pin MISO al puerto A4 y el pin MOSI al puerto A5 de la placa Arduino y estos se conectan al driver junto a los pines de entrada y salida de datos de las memorias.

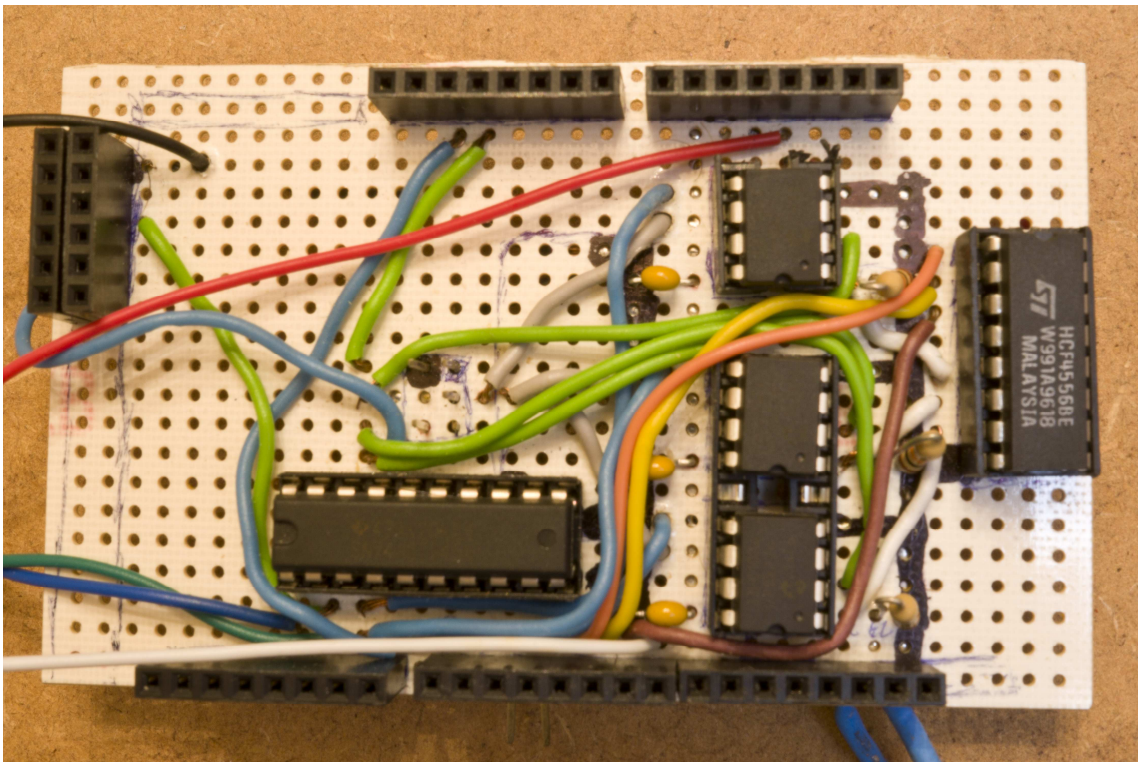

Figura 24: Shield con los integrados

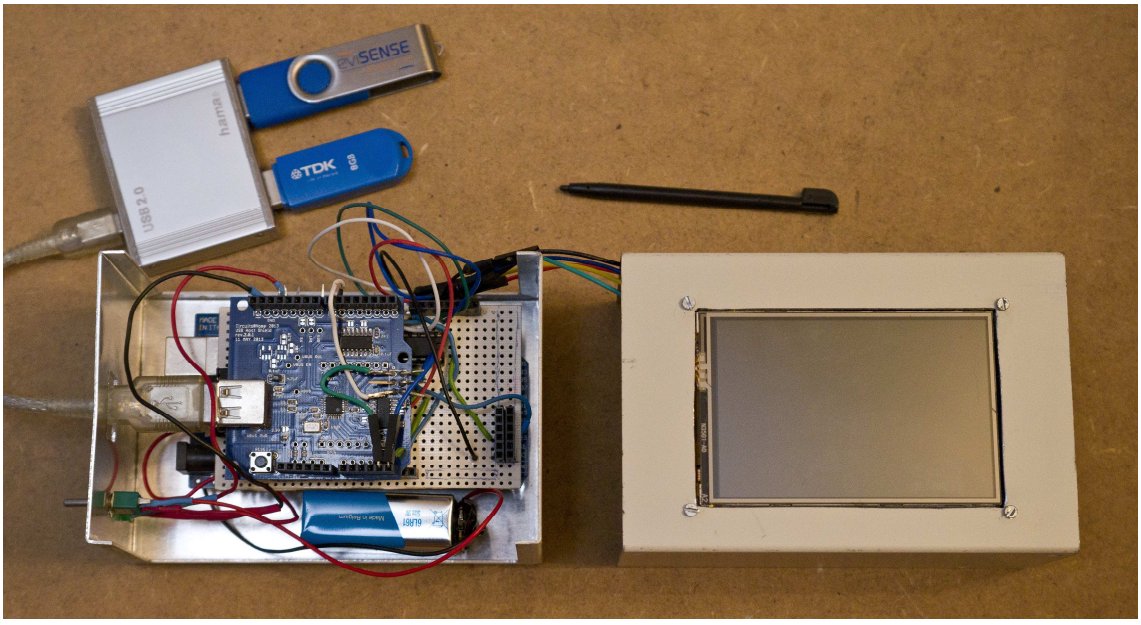

Figura 25: Dispositivo desarrollado

7.2. Software

7.2.1. Librerías libres utilizadas

7.2.1.1. Serial

Esta librería es la encargada de leer y escribir sobre el puerto serie. Esto se puede utilizar para comunicar con el ordenador u otros dispositivos. En el proyecto se ha utilizado para depurar la aplicación mediante mensajes de consola y para comunicar la placa Arduino con el *display* SmartGPU.

7.2.1.2. SPI

Es un protocolo de comunicación serie síncrono, utilizado para comunicar varios dispositivos de forma que uno actúa como maestro, en este caso la placa Arduino. Para la comunicación se utilizan las siguientes líneas:

- MISO (*Master In Slave Out*): Salida de datos del dispositivo esclavo hacia el maestro.
- MOSI (*Master Out Slave In*): Entrada de datos del dispositivo maestro hacia el esclavo.
- SCK (*Serial Clock*): Pulso de reloj utilizado por el maestro para sincronizar la transmisión.
- SS (*Slave Select*): Pin en cada dispositivo esclavo utilizado por el maestro para activar y desactivarlo.

Cuando el pin SS del dispositivo está a nivel bajo se comunica con el maestro, si está a nivel alto lo ignora. Esto es así debido a que las líneas MISO, MOSI y SCK son compartidas.

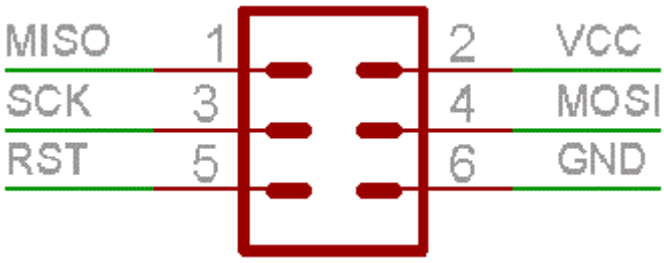

Figura 26: Conexión SPI

7.2.1.3. USB y FAT

La librería USB es la encargada de comunicar Arduino con la placa USB h*ost shield*, esta inicializa el dispositivo y se encarga de establecer y gestionar la comunicación entre la placa y los dispositivos USB conectados.

La librería FAT hace uso de la librería USB y es la encargada de mantener las estructuras necesarias para enviar los mensajes pertinentes a los pendrives conectados. También se encarga de leer y escribir los ficheros, el proceso para esto es el siguiente: se pide al pendrive un *cluster* de 512 B, este lo envía a la placa y se mantiene en memoria, una vez en memoria se modifica y se envía al pendrive para que lo escriba, si lo que queremos es modificar o volcar información que está a mitad de dos clúster lo que debemos hacer es leer el primer clúster, modificarlo y volver a escribirlo, después leemos el segundo clúster y repetimos el proceso.

Estas peculiaridades son totalmente transparentes para el usuario que hace uso de esta librería, como veremos a continuación el usuario solo se tiene que preocupar de proporcionar el nombre del archivo, situarse en la posición dentro del archivo que se quiera escribir o leer y enviar el buffer con los datos. A continuación se enumeran brevemente algunas de las funciones de la librería.

Init_Generic_Storage: Esta función es la encargada de inicializar la librería USB y crear las estructuras necesarias para la comunicación.

fs_ready: Recibe como parámetro una ruta dentro del pendrive y devuelve si esta lista para utilizar o no.

fs_open: Abre el archivo indicado en modo lectura o escritura según se indique.

fs_opendir: Abre el directorio indicado y devuelve una estructura al mismo.

fs_close: Cierra un fichero abierto anteriormente.

fs_closedir: Cierra un directorio abierto eliminando la estructura de la memoria.

fs_read: Lee la cantidad de bytes indicados del archivo que se pase como parámetro.

fs_readdir: Lee el siguiente archivo o carpeta dentro del directorio abierto y devuelve una estructura con la información leída.

fs_write: Escribe la cantidad de datos indicados sobre un archivo abierto.

fs_unlink: Elimina un archivo o directorio vacio.

fs_sync: Fuerza el volcado de los datos que estaban pendientes.

fs_mkdir: Crea un directorio en la ruta indicada.

7.2.1.4. *Display* SmartGPU 2

Esta librería es la utilizada para comunicar con el *display* y así facilitar el uso de las funciones que ofrece la pantalla. Las funciones que hemos utilizado para el proyecto son las siguientes:

Init: Esta función envía las ordenes necesarias para inicializar la pantalla.

Erase: Esta función es utilizada para borrar la pantalla según el color configurado con la función s*etEraseBackColour*.

BaudChange: Sirve para cambiar la velocidad de comunicación entre la placa Arduino y el *display*. Esto afecta a la velocidad con la que se ejecuta las ordenes de dibujado.

String: Recibe como parámetros las coordenadas donde se dibujara el *string* que le pasemos como parámetro y lo dibuja según la configuración asignada.

SetTextColour, setTextBackColour, setTextSize y setTextBackFill: Sirven para configurar los parametros con los que se mostraran los *strings,* respectivamente se utilizan para: asigna el color del texto, configura el color de fondo del texto, configurar el tamaño de la fuente y asigna el modo en el que se debe pintar el fondo.

Image BMPSD: Dibuja una imagen en formato BMP que tengamos almacenados en la memoria SD. Para esto recibe las coordenadas donde la pintará y el nombre del archivo.

ObjProgressBar: Dibuja una barra de progreso con el porcentaje indicado.

ObjScrollBar: Dibuja una barra de desplazamiento con las divisiones y orientación que le indiquemos.

TouchScreen: Recibe como parámetros un puntero a un objeto *POINT* donde almacenara las coordenadas de la pulsación en el caso de que exista.

7.2.2. Librerías desarrolladas

En esta sección se van a detallar las clases desarrolladas, su funcionamiento y los métodos utilizados. Consultar el anexo VII para ver todo el código.

7.2.2.1. Programa principal

El programa principal es el encargado de gestionar y unificar todas las librerías. Consta de una función principal *setup* que es la primera función que se llama al arrancar el programa y se encarga de configurar todos los dispositivos y crear la ventana principal, después de esta función se llama a la función *loop* que se ejecutara en bucle durante toda la ejecución del programa. Esta función es la encargada de gestionar los mensajes entre las ventanas y ejecutar las funciones del programa. Estas funciones son las siguientes:

deleteCurrentWindow: Elimina la ventana actual.

createWindowMain: Crea la ventana principal.

createWindowDirectories: Crea la ventana donde se mostraran los directorios.

createWindowDelete: Crea la ventana de borrado de archivos.

createWindowCopy: Crea la ventana de progreso de copiado de archivos.

createWindowKeyboard: Crea la ventana que muestra un teclado en pantalla.

createWindowDetails: Crea la ventana que mostrara los detalles del archivo o directorio.

actionDetails: Obtiene los detalles del archivo o directorio indicado y se los comunica a la ventana WindowDetails

actionOK: Es llamada cuando se pulsa el botón aceptar en la ventana WindowKeyboard y obtiene la cadena introducida para crear un directorio o renombrar un archivo.

changeInUSBs: Es llamada en cada pasada del bucle principal y se encarga de detectar s ha habido cambios en las conexiones USB.

actionUSB: Se llama cuando se pulsa sobre la etiqueta de un pendrive para abrirlo, se encarga de detectar sobre que dispositivo se ha pulsado y comunicárselo a la ventana WindowDirectories para que muestre el directorio en cuestión.

actionDelete: Es llamada cuando se pulsa sobre la acción de borrar y se encarga de eliminar los archivos y directorios seleccionados para eliminar.

deleteDir: Se encarga de eliminar el contenido de un directorio y el mismo directorio.

actionOpen: Es llamada cuando se pulsa sobre la acción de abrir un directorio y se encarga de comunicarle a la funcion showDir el directorio que tiene que mostrar.

actionPaste: Es llamada cuando se pulsa sobre la acción de copiar y se encarga de copiar los archivos y directorios seleccionados.

copyDir: Es una función auxiliar llamada por actionPaste para copiar el contenido del directorio indicado.

copy: Es una función auxiliar llamada por actionPaste para copiar el fichero indicado.

getFileSize: Obtiene el tamaño en bits del archivo indicado.

numFiles(char *path): Obtiene el numero de ficheros que contiene un directorio.

showDir(char *path): Es la función encargada de leer un directorio y comunicarle a la ventana WindowDirectories los archivos que debe mostrar.

openDir(char *dir): Es una función auxiliar utilizada para mostrar un directorio, internamente esta función llama a showDir.

backDir(): Es una función auxiliar utilizada para volver atrás un directorio o en el caso que este en la raíz volver a la ventana principal donde se mostraran los dispositivos conectados.

7.2.2.2. Gestión de ventanas

Esta librería es la encargada de gestionar las ventanas y eventos del usuario. Esto implica dibujar las ventanas o interfaz, analizar las pulsaciones e informar de los eventos. A continuación vamos a ver con más detalle las clases que la componen.

7.2.2.2.1. WindowButton

Esta clase tiene implementado la funcionalidad de un botón, detecta las pulsaciones, dibuja el contenido del mismo y almacena las acciones que puede realizar un botón.

Init: Inicializa el botón con un rectángulo de interacción donde detectará las pulsaciones, la acción que realiza y un puntero al objeto LCD.

IsTouched: Devuelve la acción que realiza el botón en el caso de que se pulse dentro del rectángulo de interacción.

Draw: Dibuja sobre el LCD el texto a mostrar, colorea el fondo según su estado y dibuja el icono correspondiente si se requiere.

DrawDebug: Esta función se utiliza solo en modo depuración para dibujar el rectángulo de interacción del botón.

SetText: Existen distintas variantes de esta función según los parámetros que recibe. Se utiliza para asignar el texto mostrado, su posición, el tamaño de la fuente y su color. Esta función es utilizada para mostrar texto dinámico, que puede cambiar durante la ejecución del programa. Dentro de estas funciones tenemos una que recibe la memoria SRAM que contiene el texto y la información necesaria para acceder.

SetTextInFlash: Se utiliza para asignar el índice del texto almacenado en memoria flash que se mostrara. Esta función es utilizada para mostrar texto estático, que no cambia durante la ejecución del programa.

SetActive: Activa y desactiva el botón.

SetIsDrawText: Indica si se debe pintar el texto o no. Existen botones que no tienen texto.

setIsDrawBackground: Indica si se debe pintar el fondo del botón.

SetIsSelected: Indica si el botón esta seleccionado. Utilizado para configurar el dibujado.

SetIsFocused: Indica si el botón tiene el foco. Utilizado para configurar el dibujado.

setImatgeType: Indica el icono que se debe mostar.

getDynamicText: Devuelve una copia del texto dinámico que se muestra, esto es utilizado para saber el nombre del fichero sobre el que se pulsa.

setAction: Recibe la acción que realiza el botón y la almacena.

7.2.2.2.2. Window

Se trata de la clase abstracta de la que heredaran todas las ventanas. Define las funciones *init*, *loop*, *reset y drawDebug*, además de implementar la función *draw*, utilizada para dibujar la parte común de todas las ventanas. También establece algunos *define* de la configuración y contiene la enumeración de todas las acciones posibles.

Init: Crea e inicializa los botones que componen la ventana.

Reset: Esta función se llama cada vez que se activa la ventana, se encarga de establecer las variables y botones a su valor por defecto.

Loop: Esta función es llamada en cada pasada del bucle principal, se encarga de llamar al dibujado de la ventana, gestionar las pulsaciones sobre la ventana e informar al programa principal.

DrawDebug: Llama a dibujar el rectángulo de interactuación de los botones.

Draw: Dibuja la ventana y los botones.

7.2.2.2.3. WindowMain

Esta clase es la encargada de dibujar la ventana principal donde se mostraran los distintos dispositivos USB conectados.

SetNumUsb: Recibe el número de dispositivos USB conectados y su nombre para posteriormente mostrarlos.

DrawAbout: Dibuja la ventana *about* dentro de la ventana principal.

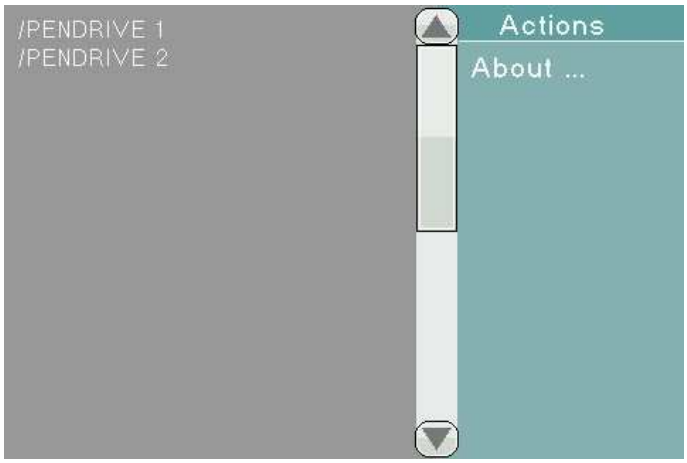

Figura 27: Ventana principal

7.2.2.2.4. WindowDirectories

Esta clase es la encargada de dibujar la ventana de directorios donde se mostraran los distintos ficheros y carpetas del dispositivo USB y las acciones que podemos realizar sobre este.

SetFiles: Recibe las posiciones de la memoria SRAM donde están almacenados los nombres del directorio a mostrar y asigna los nombres correspondientes a los botones.

ActionFile: Gestiona sobre que archivo o directorio se ha pulsado.

Copy: Almacena los nombres de los archivos y directorios a copiar en la estructura de copiado y los asigna a la memoria correspondiente.

DrawConfirmDeleteFiles: Dibuja la sub-ventana de confirmado de borrado.

GetSelectedFile: Devuelve el nombre del directorio o fichero seleccionado, esto es utilizado por el programa principal para acceder a las carpetas o renombrar ficheros.

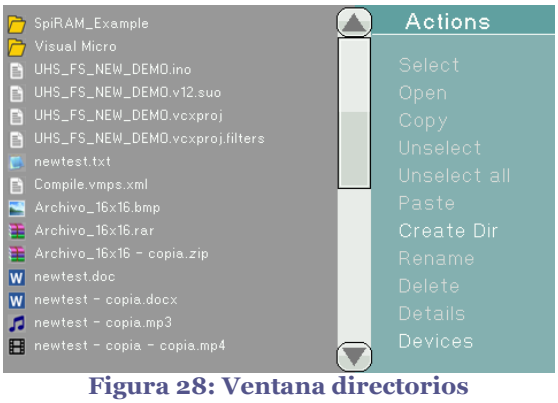

Figura 28: Ventana directorios

Figura 29: Ventana seguro borrar

7.2.2.2.5. WindowDelete

Esta clase se encarga de dibujar la ventana de borrado de ficheros donde mostrara el proceso de la acción.

drawDeleteProgress: Dibuja el progreso de borrado indicando el fichero actual y ficheros restantes.

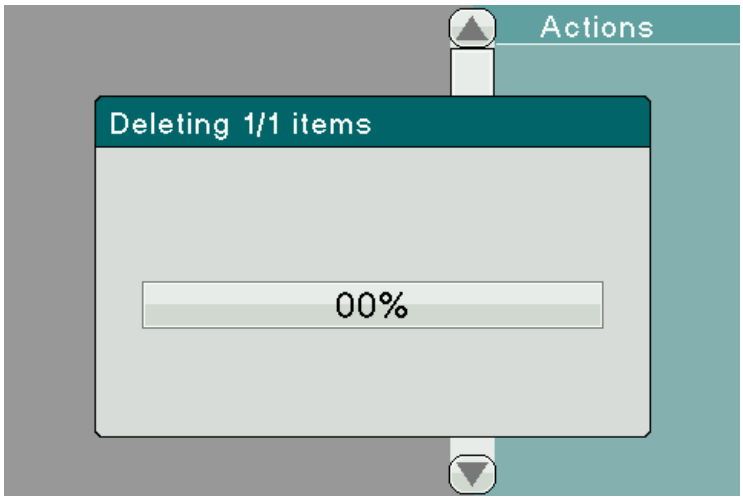

Figura 30: Ventana progreso borrar

7.2.2.2.6. WindowCopy

Esta clase es la encargada de dibujar las distintas ventanas que se pueden mostrar durante el proceso de copiado.

drawPaste: Dibuja la ventana del proceso de copiado indicando el fichero actual, cuantos ficheros faltan por copiar y una barra con el porcentaje de este proceso.

isPasteCanceled: Detecta si se ha pulsado cancelar y lo devuelve al programa principal.

drawConfirmPasteCanceled: Muestra la ventana de confirmación del cancelado del proceso.

drawFileExist: Dibuja la ventana que se muestra cuando un fichero ya existe y devuelve si el usuario ha decidido omitir el archivo, reemplazarlo, renombrarlo o cancelar el proceso de copiado.

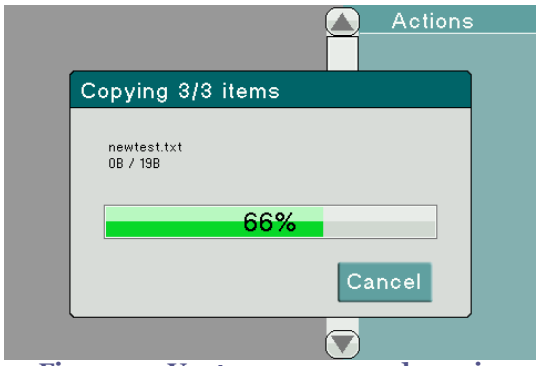

Figura 31: Ventana progreso de copia

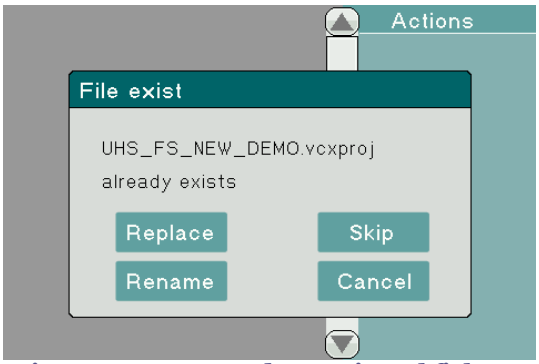

Figura 32: Ventana de ya existe el fichero

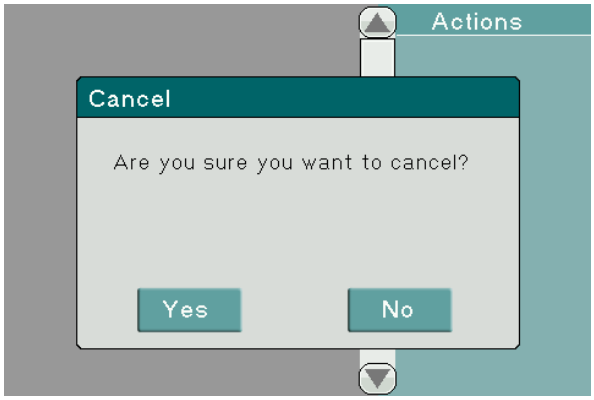

Figura 33: Ventana de cancelar copia

7.2.2.2.7. WindowKeyboard

Esta clase se encarga de mostrar la ventana teclado utilizada para asignarle un nombre al directorio creado o renombrar un fichero.

drawSingleKey: Dibuja una sola tecla del teclado, esto se utiliza para cuando pulsas sobre una tecla resaltarla como pulsada.

drawAllKeyboard: Dibuja el teclado indicado, siendo este el teclado de minúsculas, mayúsculas o de símbolos.

getKeyTouch: Devuelve el carácter pulsado o el identificador del botón en el caso de los botones de cambio de teclado, cancelar y aceptar.

setText: Muestra un texto inicial en el teclado.

getText: Devuelve la cadena introducida.

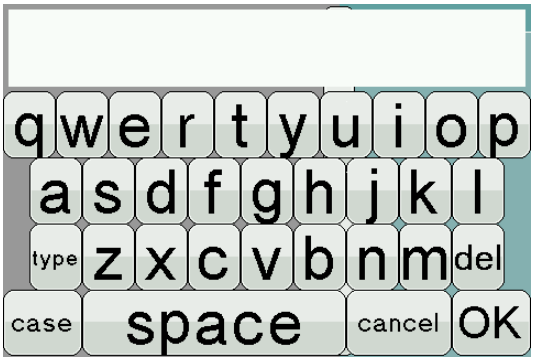

Figura 34: Ventana teclado

7.2.2.2.8. WindowDetails

Esta clase se encarga de mostrar la ventana de detalles de un fichero o directorio.

drawDetails: Dibuja la ventana de detalles, la información mostrada es: el nombre del fichero o directorio, su tamaño, los atributos de este y la ultima fecha de modificación.

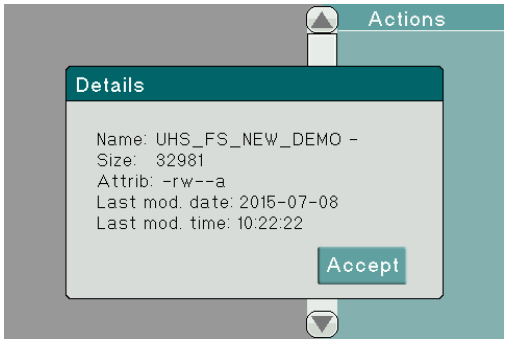

Figura 35: Ventana detalles

8. PROBLEMÁTICA

En este apartado se describen los problemas surgidos durante el desarrollo y la solución adoptada.

Cuando el proyecto estuvo en una etapa avanzada se empezó a tener problemas de fragmentación de memoria, quedaba memoria libre, pero estaba dividida, de forma no consecutiva y no existía una cantidad mínima para alojar las estructuras que se requieren para abrir los directorios en los *pendrives*.

Para solucionar este problema primero se hicieron unas pruebas para ver si existían pérdidas de memoria por parte de la librería USB, una vez demostrado que no habían perdidas se pasó a analizar la librería desarrollada para el proyecto. Se pudo demostrar que no había pérdidas de memoria pero si una utilización de variables excesiva en el bucle principal del programa, para solucionar este problema se optó por subdividir el bucle principal en pequeñas funciones con variables locales a esta función. Además de solucionar el problema de fragmentación se ganaron varios bytes de memoria.

Cuando la librería principal creció en funcionalidad se volvió a tener problemas de memoria, esta vez se optó por reducir la cantidad de objetos en memoria. Al inicio del programa se creaban todas las ventanas del programa y se mantenían en memoria, de forma que solo se tenían que mostrar u ocultar según conviniera. La solución consistió a este problema en destruir la ventana que se quería ocultar y crear la nueva ventana que se quería mostrar reduciendo así la cantidad de memoria utilizada.

9. TRABAJO FUTURO

Este proyecto se podría ampliar con más funcionalidades, algunas de ellas podrían ser las siguientes:

Ampliar los sistemas de archivos reconocidos, como NTFS o ext. Permitir conectar dispositivos móviles mediante USB, con sistema Android u otros. Añadir funcionalidades como sincronizar carpetas o dispositivos. Añadir funcionalidades de verificación de archivos copiados.

10. CONCLUSIONES

Para concluir diremos que se ha alcanzado el objetivo propuesto para las memorias flash con formato FAT, no solo se pueden copiar ficheros, también es posible copiar directorios de forma recursiva, además se le ha dotado al dispositivo de funcionalidades como: renombrar archivos y directorios, obtener información detallada sobre el fichero, eliminar recursivamente ficheros y directorios y la posibilidad de crear directorios.

También comentar que ha sido un proyecto muy interesante para el desarrollador ya que planteaba un gran reto, teniendo que investigar en profundidad el funcionamiento de las memorias flash y el formato FAT, así como ampliar los conocimientos sobre el sistema Arduino y los lenguajes de programación C y C++.

11. BIOGRAFÍA

Arduino. Wikipedia. Fecha consulta: 10 de Abril de 2015. URL: http://es.wikipedia.org/wiki/Arduino

Processing. Wikipedia. Fecha consulta: 10 de Abril de 2015. URL: http://es.wikipedia.org/wiki/Processing

Raspberrypi. Fecha consulta: 20 de Abril de 2015. URL: http://www.raspberrypi.org

Beagleboard. Fecha consulta: 20 de Abril de 2015. URL: http://beagleboard.org/

Omicrono. Minnowboard, Intel se atreve contra la raspberry pi con un ordenador libre por 199 dolares. Fecha consulta: 20 de Abril de 2015. URL: http://www.omicrono.com/2013/08/minnowboard-intel-se-atreve-contra-la-raspberry-pi-con-un-ordenador-libre-por-199-dolares

Omicrono. Gizmo 2 es la alternativa de amd a la raspberry pi. Fecha consulta: 20 de Abril de 2015. URL: http://www.omicrono.com/2014/11/gizmo-2-es-la-alternativa-de-amd-a-la-raspberry-pi

Blogthinkbig. Cuatro alternativas Arduino: Beaglebone, Raspberrypi, Nanode y Waspmote. Fecha consulta: 20 de Abril de 2015. URL: http://blogthinkbig.com/4-alternativas-arduino-beaglebone-raspberrypi-nanode-waspmote

PCWorld. Open-source Gizmo 2 PC packs an AMD CPU, Radeon graphics. Fecha consulta: 20 de Abril de 2015. URL: http://www.pcworld.com/article/2846112/gizmosphere-focuses-on-graphics-in-opensource-computer.html

Minnowboard. Fecha consulta: 20 de Abril de 2015. URL: http://www.minnowboard.org

Leefco. Ibridge. Fecha consulta: 5 de Mayo de 2015. URL: http://www.leefco.com/ibridge

Circuits at home. Fecha consulta: 6 de Mayo de 2015. URL: https://www.circuitsathome.com/

Torres Rodriguez, Sergio Iban. Universidad de las Palmas de Gran Canaria. Universal Serial Bus. Fecha consulta: 8 de Mayo de 2015. URL: http://www.iuma.ulpgc.es/~avega/int_equipos/trab9899/usb_1/index.html

Martín Gomez, Pablo. Simposio Argentino de Sistemas Embebidos. USB. Fecha consulta: 8 de Mayo de 2015. URL: http://www.sase.com.ar/2013/files/2013/09/SASE2013-USB-P-Gomez.pdf

Zator System. Tecnología del PC. Fecha consulta: 10 de Mayo de 2015. URL: http://www.zator.com/Hardware/H8_1_2a1.htm

García Fonseca, Francisco Javier. Rincón del vago. Introducción a los sistemas de archivos. Fecha consulta: 10 de Mayo de 2015. URL: http://html.rincondelvago.com/sistema-de-archivos-fat.html