SISTEMAS EMBEBIDOS

INFORME
PROYECTO 2009:

CONTROLADOR DE TARJETA SD

GRUPO 1

Nombre	CI	email
Luis Ignacio de León Echarri	4246997-1	roverano8@gmail.com
María Cecilia San Román Rincón	3772285-5	cecisr@gmail.com
Gonzalo Eduardo Sotta Medina	3817621-9	gsotta@gmail.com

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería
Universidad de la República

RESUMEN

El presente trabajo describe la implementación de un interfaz maestro esclavo entre un microcontrolador (Atmega 32) y una tarjeta de memoria SD (Secure Digital).

La comunicación entre el microcontrolador y la tarjeta de memoria SD se realizó con una interfaz SPI y dado que la escritura de los datos en la tarjeta no se realizó con un sistema de archivos compatible con Windows/Linux se utilizó la UART, y como interfaz con el usuario una terminal de comunicación serial. El software fue programado en lenguaje C, y se utilizó el "IAR Embedded Workbench" como ambiente de desarrollo. Para la programación del microcontrolador se utilizó el AVR Dragon.

En el desarrollo de este trabajo se expondrá el diseño del hardware y los procedimientos para poder inicializar la tarjeta SD, leer y escribir un bloque simple de datos así como también las dificultades que surgieron debido a que en la documentación de las tarjetas SD no se encontraba toda la información necesaria para llevar a cabo las mismas.

TABLA DE CONTENIDO

INTRODUCCIÓN

Una de las ventajas en la utilización de tarjetas SD es la de dotar a cualquier sistema embebido con una fuente de memoria de gran capacidad, sin alterar en forma significativa las dimensiones físicas del diseño ni su consumo.
Cabe destacar la posibilidad de utilizar una interfaz SPI (Serial Protocol Interface) para la comunicación con la tarjeta. Dicha interfaz ya esta implementada en el microcontrolador utilizado lo cual es una gran ventaja, como contrapartida la comunicación es más lenta que si se utilizara la interfaz propia de la tarjeta SD.
La idea fundamental está basada en la comunicación mediante el protocolo SPI, lo cual facilita la implementación pese a reducir la velocidad de transmisión.

A continuación se expone los objetivos y alcance del proyecto, la arquitectura de software y hardware utilizada, como fue implementado y las pruebas realizadas.
Al final se anexa una breve descripción del funcionamiento de las tarjetas SD: su inicialización, la activación/desactivación del CRC (Cyclic Redundancy Check), como se realiza una escritura/lectura de un bloque de datos. También se anexa información sobre la interfaz SPI y el sistema de archivos FAT.

OBJETIVOS

Comprender el principio de funcionamiento de las tarjetas SD, la interfaz SPI y del formato de archivos FAT. Realizando la documentación de los temas aprendidos.
Generar un controlador de memorias SD que permita la lectura y escritura de la misma mediante la utilización de la interfaz SPI.

ALCANCE

El proyecto incluye la construcción del hardware requerido para la realización del mismo, así como el software necesario para controlar la tarjeta SD.
El software implementa la inicialización, escritura y lectura de un bloque de datos de tamaño variable en una dirección de memoria determinada por el usuario. A su vez permite ver las respuestas a las acciones realizadas, mediante la UART en una terminal. También se implementan otras funciones que fijan distintos parámetro de la tarjeta, como la utilización de códigos CRC en la comunicación y el tamaño de los bloques de lectura y escritura.
Tanto el software y el hardware son diseñados para comunicarse con la tarjeta por medio de la interfaz SPI.
El manejo de la memoria de la tarjeta utilizando el sistema de archivos FAT no fue implementado.
La tarjeta SD permite, además de la comunicación SPI, la comunicación SD. La cual es propietaria y por tanto no se incluye.

DISEÑO

Arquitectura Software

Debido a que la tarjeta SD solo se activa cuando se desea inicializarla, y al leer o escribir un dato; para llevar a cabo dichas tareas se espera que el usuario mediante una terminal ingrese la acción que desee realizar. Es por esto que se utilizó Round-Robin sin interrupciones.
A partir del diagrama de flujo de la Figura 1, se puede ver que primero se realiza "polling" de un único evento, pues se espera a que el usuario mande el comando para inicializar la tarjeta, y luego se realiza "polling" para leer o escribir un bloque de datos en la tarjeta.

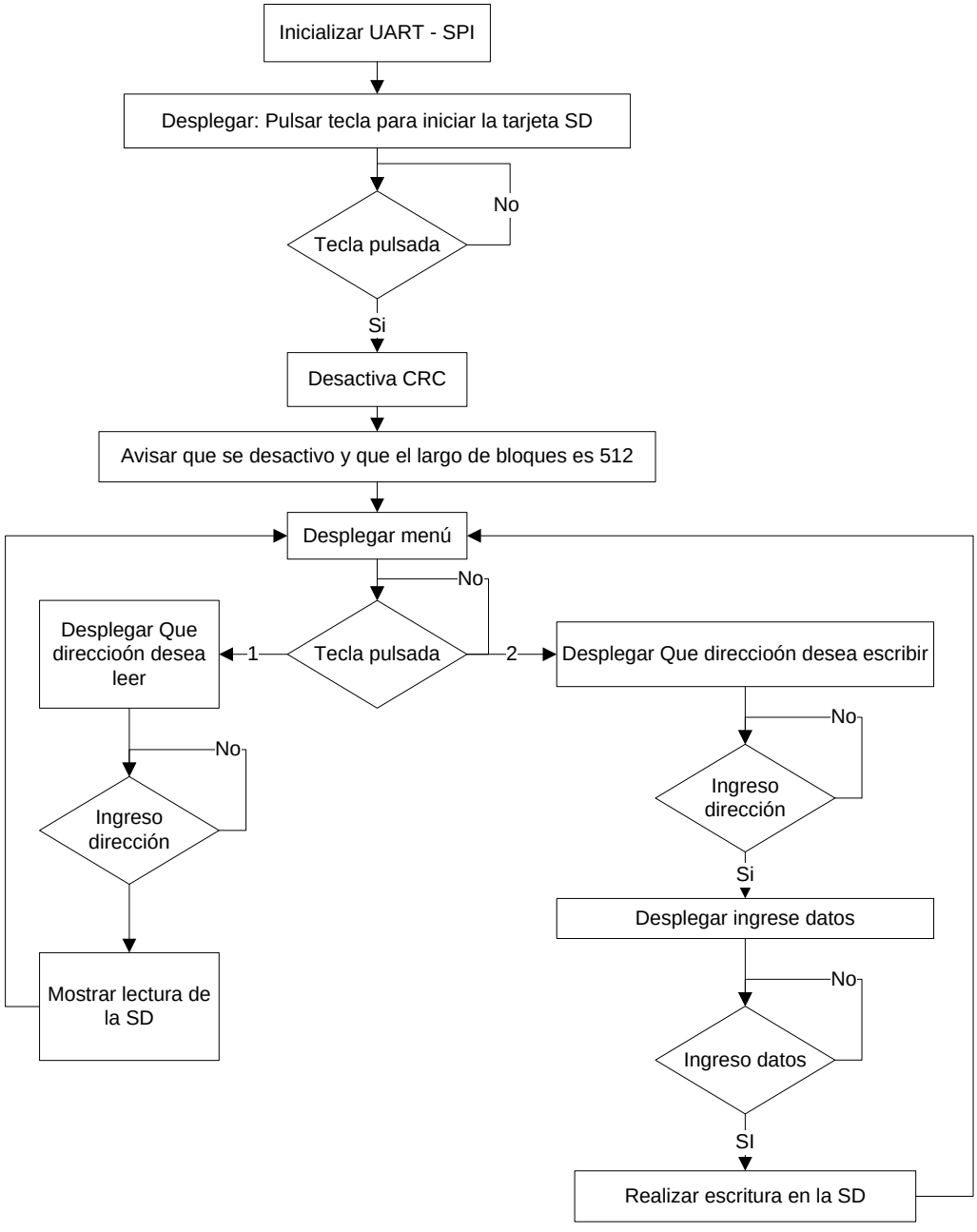

Figura 1. Diagrama de flujo del software desarrollado.

Arquitectura Hardware

El sistema hardware se divide en tres grandes bloques, la tarjeta SD, el microprocesador y el interfaz PC – microcontrolador implementado por el Max232. Ver Figura 2.
Se utilizó el microcontrolador Atmega32, y en lugar de utilizar una tarjeta SD se uso una microSD de 2GB y un adaptador para SD.

La idea principal es que el usuario ingrese la tarea a realizar en la PC, y mediante el puerto RS232 se la envía al microprocesador y éste se encarga de realizar las funciones necesarias sobre la memoria SD para cumplir con dicha tarea. La tarjeta envía una respuesta a dicha orden y siguiendo el camino inverso se visualiza en el computador.

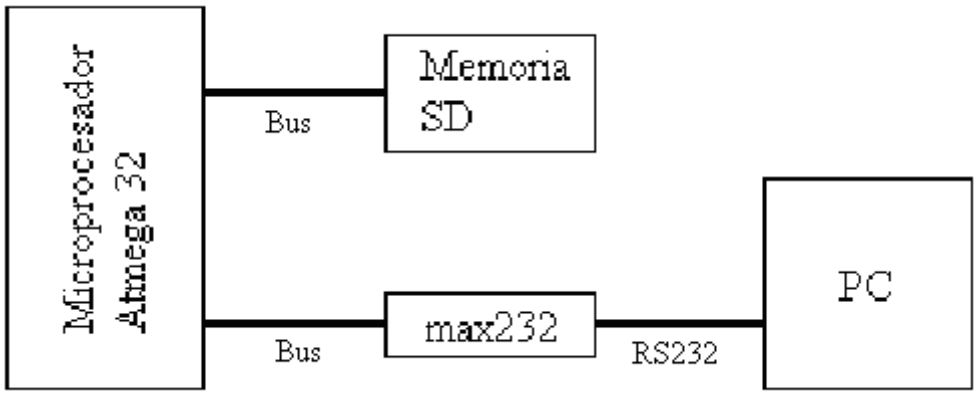

Figura 2. Diagrama de bloques de la arquitectura Hardware implementada.

IMPLEMENTACIÓN

Una vez realizados los estudios de las partes involucradas: protocolo SPI, tarjeta SD, formato FAT. Se procedió de la siguiente manera:

- Construcción de hardware.
- Código de prueba UART.
- Código de prueba UART & SPI.
- Código de inicialización de SD.
- Código de escritura y lectura de SD.
- Interfaz con usuario.

Construcción de hardware

El hardware implementado se muestra en la Figura 3. El cual se compone básicamente de cinco elementos:

- Microprocesador Atmega32.
- Tarjeta SD.
- Convertidor de niveles, Max232.
- Regulador de tensión LM317.
- Conector DB9 hembra.
- Capacitores, resistencias.

Todos ellos fueron ensamblados en un "Protoboard" y se utilizó el AVR Dragon para la programación y depurado del software en el Atmega32.

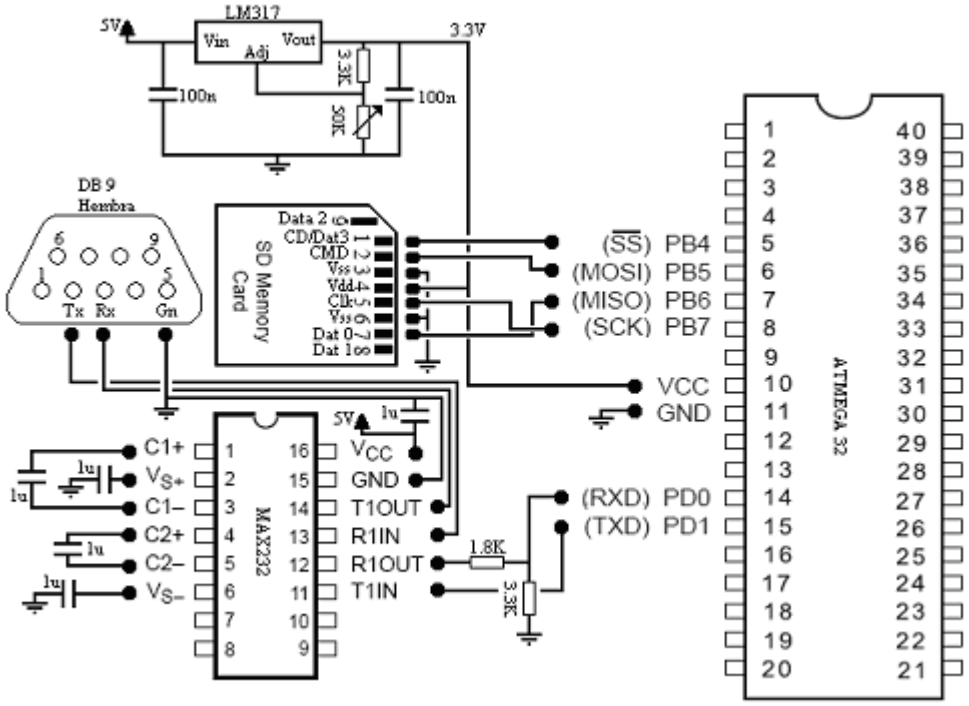

Figura 3. Hardware implementado.

La alimentación principal del sistema es de 5V y es provista por el AVR Dragon. Ésta alimenta al Max232 y al regulador de tensión (LM317) el cual da como salida 3.3V que son usados para alimentar, la tarjeta SD y el microprocesador.
Producto de la existencia de dos tensiones diferentes es que se coloca un divisor resistivo entre el Max232 y el microcontrolador, con el fin de igualar sus niveles lógicos.

En la Tabla 1 se muestra el conexionado de pines.

ATMEGA 32	SD	
5	1	
6	2	
7	7	
8	5	ALIMENTACIÓN
10	4	3,3V
11	3 y 6	GND

	MAX232	
14	12 (div R 1.8K/3.3K)	
15	11	ALIMENTACIÓN
	16	5V
	15	GND

Tabla 1. Lista de conexiones.

Código de la UART

Se llevó a cabo a partir de de la información brindada por el manual del Atgema32 [1] y los laboratorios realizados en el curso.
El código se basa en la configuración de determinados bits de registros específicos, con el fin de habilitar cuando leer el registro de datos de entrada, o cuando colocar un dato en el puerto de salida.

Código de la interfaz SPI[1]

Al igual que en el caso de la UART, el manual del ATmega32 [1] proporciona código de ejemplo para la configuración correcta del puerto SPI del microcontrolador.
La configuración correcta del puerto SPI está basada en el mismo concepto de la UART, configurar correctamente bits específicos de registros determinados. Para más información sobre la interfaz SPI y su configuración en el ATmega32[2].

Código de inicialización de SD[3]

Éste código únicamente se encarga de la inicialización de la tarjeta SD. Una vez que el sistema se pone en funcionamiento, es preciso dar aviso a la SD que va a ser usada en modo SPI. De esta manera, una vez configurada, la tarjeta queda en estado "idle" esperando un comando.
Para llevar a cabo la inicialización de la tarjeta SD se deben realizar los siguientes pasos:

- Se debe activar la señal del conector 5 de la SD (CLK) 80 ciclos, mientras se tiene el conector 1 en nivel alto.
- Enviar el comando 0[4]
- Enviar el comando 1 varias veces (no está predefinido, es variable).

[1] Las funciones que se encargan de inicializar y manejar la UART y el puerto SPI se agruparon en los archivos fun_UART.h y SPI.h respectivamente.
[2] Por más información ver Anexo B
[3] La función encargada de realizar la inicialización es SD_Init(), se encuentra en el archivo SD_Fun.h.
[4] Para más detalles ver Anexo A

Código de escritura y lectura de SD[5].

Una vez inicializada la tarjeta, este código se encarga de leer/escribir un bloque de datos en la dirección de memoria que ingrese el usuario. En caso de realizarse una escritura, también se debe enviar el bloque de datos a escribir.

Para llevar a cabo la lectura de la tarjeta SD, se debe enviar el comando 17, esperando la respuesta de la tarjeta SD. Está respuesta informa si a continuación envía el bloque de datos o si se produjo un error. Si no se produjo un error, se debe esperar a recibir un "Data Token", este es el byte 11111110, o un "Data Error", este es el byte 0000xxxx. Si se recibe un "Data Token", a continuación se reciben los bytes de datos y dos bytes de CRC (Cyclic Redundancy Check). Por mas información respecto a los comandos y respuestas recibidas[6].

Para realizar una escritura en la tarjeta SD, primero se envía el comando 24, y se espera hasta recibir una respuesta de la tarjeta. Al igual que para realizar una lectura, esta respuesta informa si se produjo un error o si se puede comenzar a enviar los datos que se desea escribir. Una vez recibida esta respuesta se verifica que no se produjo ningún error, se envía un "Data Token" (el byte 11111110), los bytes de datos a almacenar y dos bytes de CRC.

Ahora se debe esperar que la tarjeta responda nuevamente, informando si los datos fueron aceptados o si se produjo un error en la escritura. Por último, falta esperar que la tarjeta deje de estar ocupada y verificar el status de la tarjeta. Esto se hace enviando el comando 13.

Interfaz con usuario[7].

Se realizó un programa que simplemente muestra en pantalla un menú donde el usuario debe elegir que acciones a realizar con la tarjeta.
Estas acciones tienen un orden lógico el cual debe ser seguido. A saber, primero debe ser inicializada la tarjeta, luego escribir o leer.
El menú funciona mediante un "Switch – Case".

Las funciones de los distintos bloques se utilizaron luego para crear una interfaz de usuario que realiza las tareas que éste requiera.

[5] Las funciones que se encargan de realizar las operaciones básicas de comunicación con la tarjeta, así como también las funciones que ejecutan comandos específicos se agruparon en el archivo SD_Fun.h.
[6] Ver Anexo A
[7] Las funciones que se encargan de desplegar el menú, de armar la dirección y decidir si leer o escribir se encuentran en SD_Interfaz.h

PRUEBAS

Todas las pruebas se realizaron con el hardware completo y utilizando una terminal virtual en el computador para visualizar los datos en el puerto serie. Para la terminal virtual se usó la "Hyperterminal" de Windows y la Terminal de "Bray++".
Para comprobar el correcto funcionamiento de la comunicación física entre la PC y la UART del microcontrolador, es necesario activar el eco de la terminal, y modificar el circuito conectando los siguientes pines, de uno por vez:

- Pines: 12 y 11 del Max232.
- Pines: 14 y 13 del Max232.

Código de prueba UART

Para la realización de esta prueba se enviaron datos desde la terminal y mediante el depurado del sistema se visualiza si los datos llegan al buffer de entrada de la UART en forma correcta. Luego se carga el buffer de salida de la UART para enviar el carácter recibido previamente, corroborando su envío con la hyperterminal.

Código de prueba UART & SPI.

Para verificar que el puerto SPI se configuró correctamente se realizó el siguiente ensayo[8]:

- Se cortocircuitaron los pines 6 y 7 del Atmega32 (MOSI y MISO).
- Se envió un dato a través de la terminal.
- El dato recibido por la UART se envió por el puerto SPI
- El dato recibido por el puerto SPI se envió por la UART a la terminal.
- El dato recibido en la terminal efectivamente era el dato que se había enviado.

Código de inicialización de SD

Una vez comprobada la correcta funcionalidad de los casos anteriores, se procedió a inicializar la tarjeta[9].
En la terminal se observaba la respuesta de la tarjeta, de manera de poder corroborar si estaba siendo correctamente inicializada.

Código de escritura y lectura de SD.

Una vez inicializada la tarjeta se le envió un dato fijo a escribir en ella. Se visualizaba si el comando de escritura así como el dato habían llegado correctamente observando la respuesta de la SD en la terminal.
De igual modo se procedió para la lectura.

Interfaz con el usuario.

La prueba se basó simplemente en la comprobación del correcto flujo del "Switch – Case".
Es de notar que en todos los casos se usó el AVR Dragon, para el depurado de los códigos.

[8] La tarjeta no debe estar conectada.
[9] Ver Anexo A

CONCLUSIONES

Se logró la comunicación con la tarjeta SD utilizando la interfaz SPI, a pesar que los manuales de la misma no brindan toda la información necesaria para realizarla.
La falta de información en los manuales referente a la SD dificultó notoriamente el desarrollo del proyecto.
La utilización de la interfaz SPI facilita la comunicación con la tarjeta, dado que el hardware necesario ya está implementado internamente en el microcontrolador. Haciendo que este modo de comunicación sea el preferido para el proyecto.
Se estima que es posible trabajar con la SD mediante un sistema de archivos FAT32, dado que las funciones básicas de manejo de la tarjeta fueron realizadas satisfactoriamente. Sería cuestión de analizar su correcta utilización para lograr trabajar con dicho sistema de archivos. Vale mencionar que se encontraron trabajos anteriores que utilizan la tarjeta con FAT32.

AGRADECIMIENTOS

A los señores profesores Pablo Mazzara y Leonardo Steinfeld y al señor Jorge Villaverde por el apoyo brindado.

REFERENCIAS

[1] 2503NS-AVR-06/08, 8-bit Microcontroller with 32K Bytes In-System Programmable Flash, Data sheet, Atmel Corpotation. 2008
[2] Product Manual, Versión 2.2, SanDisk SD Card. SanDisk Corporation. 2004
[3] SD Specifications Part 1 Physical Layer Simplified Specification. Version 2.00. SanDisk Corporation. 2006
[4] The new Peter Norton programmers guide to the IBM PC & PS/2 Second Edition 1988. Peter Norton y Richard Wilton. Microsoft Press ISBN 1-55615-131-4

ANEXOS

A. SD (Secure Digital)

La tarjetas SD están basadas en una memoria flash controladas por un microprocesador interno.
Las mismas fueron diseñadas con el fin de tener gran capacidad de almacenamiento, seguridad y para su uso en tecnologías móviles.
A su vez estas tarjetas tienen un mecanismo de protección y autentificación contra robo.

La comunicación con la tarjeta no se hace directamente con la memoria, sino con el microprocesador que incluye, como anteriormente se dijo. Por tanto esta se realiza mediante comandos que se proporcionan en el manual de la misma.
Las tarjetas SD tienen dos maneras de comunicarse, una mediante el protocolo propietario SD y el otro mediante una interfaz SPI.

El protocolo SD utiliza todos los pines de la tarjeta. Donde los cuatro pines de datos son bidireccionales, a diferencia del SPI que para los datos se usan dos pines y estos son unidireccionales. Por lo que la comunicación SD es mucho más rápida, alcanzando 50MB/sec.
Ambas formas de comunicación utilizan una señal de reloj de entre 0 a 25MHz y alimentación entre 2.0V a 3.6V.

Comandos de relevancia.

Existen más de cincuenta diferentes comandos para comunicarse con la SD. Pero aquí se mencionaran solo los necesarios utilizados durante el proyecto. Ver Tabla 1.

Comando	Argumento	Respuesta	Acción
CMD0	[31:0] 0	R1	Reinicia la SD
CMD1	[31:0] 0	R1	Confirmación de inicio
CMD13	[31:0] 0	R2	pide registro de estado
CMD16	[31:0] largo	R1	Selecciona largo de bloque
CMD17	[31:0] dirección	R1	Leer un bloque
CMD24	[31:0] dilección	R1	Escribir un bloque
CMD59	[31:1] 0 [0:0] CRC op.	R1	Usar CRC o no

Tabla 1. Lista de comandos usados.

Armado de trama de comando

Para enviar un comando se debe armar una trama la cual presenta la siguiente estructura. Ver la Figura 1.

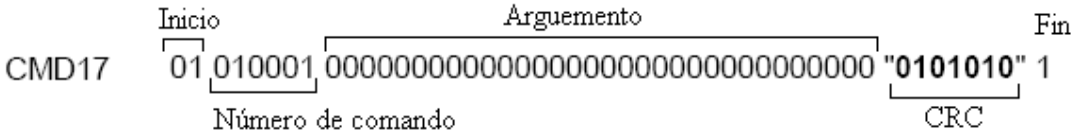

Figura 1. Formato de trama de un comando.

La trama está compuesta por 6 Bytes, de los cuales se explicita en la figura. Para enviarla, se debe enviar de a byte empezando por el más significativo.

El diagrama de tiempos para el envío de un comando se muestra en la Figura 2.

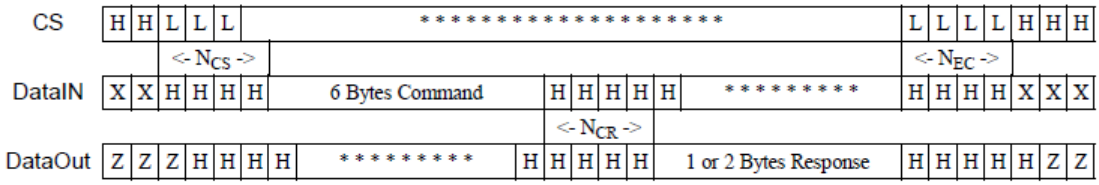

Figura 2. Diagrama de tiempos del envío de un comando.

.

Respuesta de tipo R1

La tarjeta envía una respuesta después de cada comando con la excepción del comando SEND_STATUS. La respuesta de tipo R1 es 1 byte y el MSB es siempre "0", un "1" en cualquiera de los otros bits significa un error (Ver Figura 3).

- Bit 0: Idle State – La tarjeta está en estado idle y corriendo el proceso de inicialización.
- Bit 1: Erase reset
- Bit 2: Illegal command – Se detecto un código de comando "ilegal".
- Bit 3: Communication CRC error – El CRC del último comando falló.
- Bit 4: Erase sequence error
- Bit 5: Address error
- Bit 6: Parameter error – Los argumentos del comando, como pueden ser la dirección o el largo del bloque, no están en el rango permitido para ésta tarjeta.

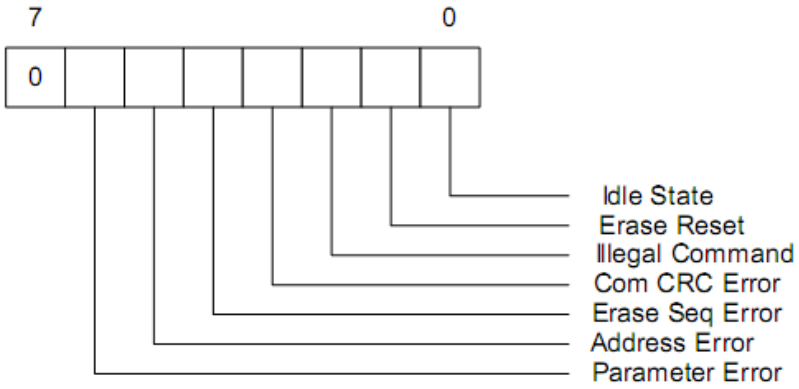

Figura 3. Datos que se obtienen de una respuesta de tipo R1.

Respuesta de tipo R2

La respuesta de tipo R2 son dos bytes, el primer byte es una respuesta de tipo R1 y el segundo byte da la siguiente información (Ver Figura 4):

- Bit 0: Card is locked – Se setea cuando el usuario bloquea la tarjeta, se resetea cuando se desbloquea.
- Bit 1: Wp erase skip|lock/unlock cmd failed - Este bit tiene dos funciones: se setea cuando se intenta borrar un sector protegido contra escritura o crea una secuencia o error en la contraseña mientras que la tarjeta se bloquea/desbloquea.
- Bit 2: Error - Ocurrió un error general o desconocido.
- Bit 3: CC error - Ocurrió un error en el controlador interno de la tarjeta.
- Bit 4: ECC failed –Hubo una falla en el código de corrección de errores.
- Bit 5: Write-protect violation – El comando intentó escribir en un bloque protegido contra escritura.
- Bit 6: Erase param – Selección inválida.
- Bit 7: Out of range | csd overwrite.

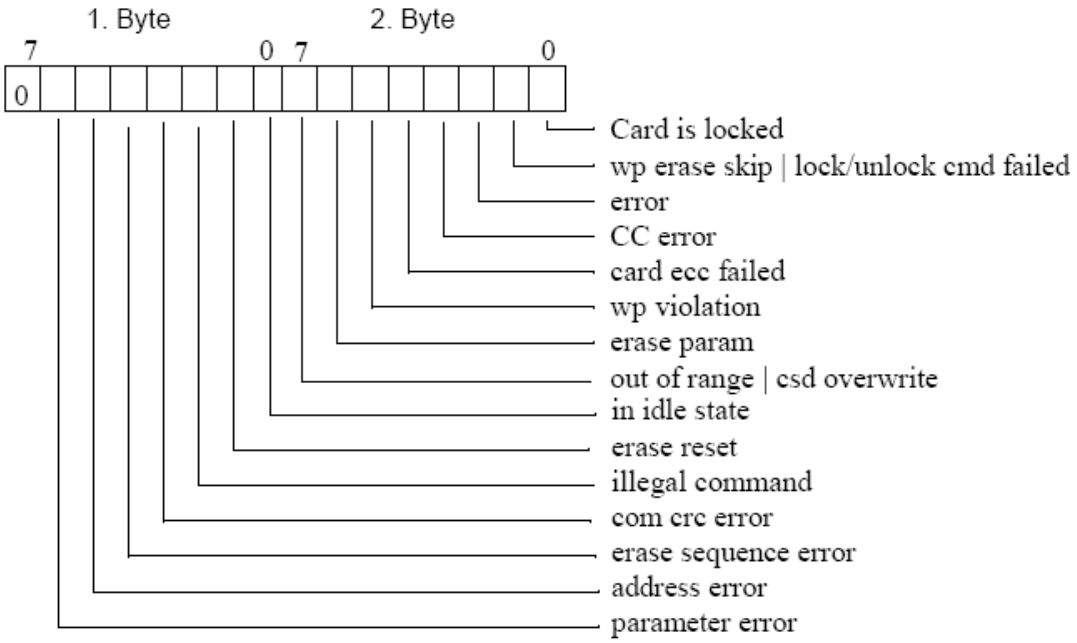

Figura 4. Respuesta de tipo R2.

Data Response

Es 1 byte y tiene el formato que se muestra en la Figura 5. El bit 0 es un '1', el bit 4 un '0' y si Status es:

- '010' – Los datos fueron aceptados.
- '101' – Los datos fueron rechazados debido a un problema con el CRC.
- '110' – Los datos fueron rechazados debido a un error en la escritura.

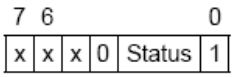

Figura 5. Formato de la respuesta después de escribir un bloque de datos.

Data Tokens

Al utilizar los comandos de escritura/lectura se tiene transferencias de datos. Estos datos se comienzan a transmitir o recibir vía data tokens.

Los bytes de datos que son tomados para escrituras y lecturas de bloques simples y múltiples tienen el siguiente formato:

- 1 byte: Comienza el bloque
- Bytes 2-513 (Depende en el largo del bloque de datos): Los datos a leer/escribir.
- 2 bytes de CRC.

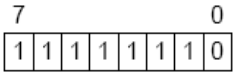

Figura 6. Formato de Data Tokens.

Data Error Token

Si la operación de lectura falla y la tarjeta no puede enviar los datos envía un Data Error Token en lugar de enviar un Data Token. El Data Error Token en un byte y tiene la forma que muestra la Figura 7, los 4 bits más significativos son '0' y los demás son los mismos errores que en la respuesta de tipo R2.

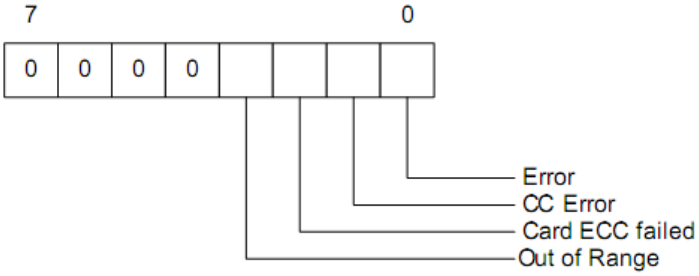

Figura 7. Formato de Data Error Token.

Escritura de bloques simples en la tarjeta SD en modo SPI.

Los pasos para escribir un bloque de datos simple en la SD son (Ver Figura 8):

- Enviando el comando 24.
- Esperar respuesta de tipo R1.
- Enviar el bloque de datos (Tener en cuenta
- Data Tokens).
- Esperar data_response
- Esperar que la tarjeta deje de estar ocupada.

En la Figura 9 se indican los tiempos que hay que respetar para que la escritura sea exitosa.

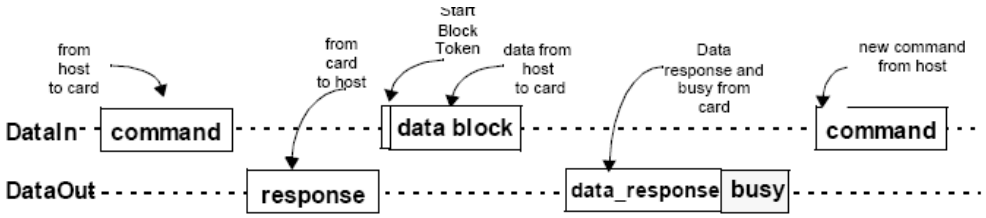

Figura 8. Diagrama de escritura de un bloque de datos.

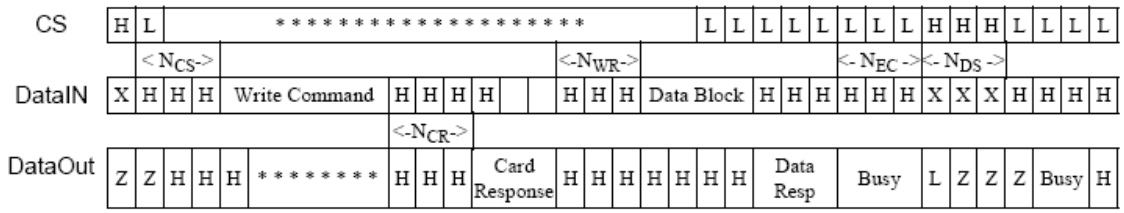

Figura 9. Diagrama de tiempos de escritura de un bloque de datos.

Lectura de bloques simples en la tarjeta SD en modo SPI.

Los pasos para leer un bloque de datos simple en la SD son (Ver Figura 10):

- Enviando el comando 17.
- Esperar respuesta de tipo R1.
- Recibir el bloque de datos[10].

[10] Ver Data Tokens

En la Figura 11 se indican los tiempos que hay que respetar para que la lectura sea exitosa.

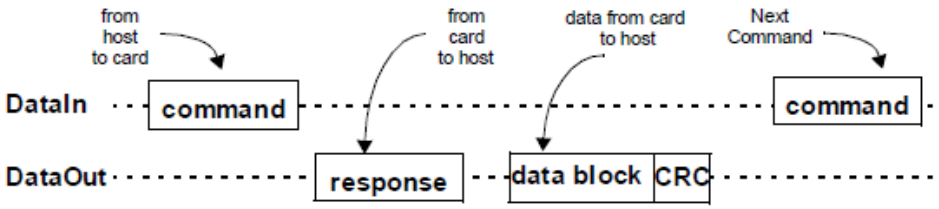

Figura 10. Diagrama de lectura de un bloque de datos.

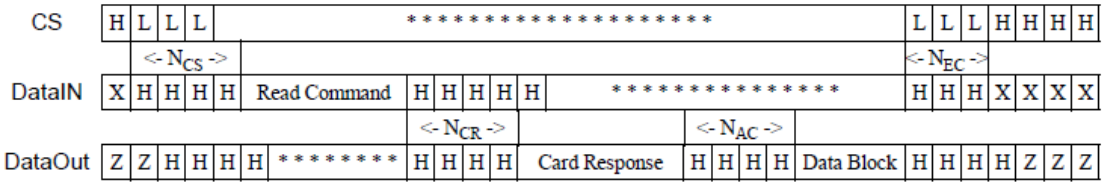

Figura 11. Diagrama de tiempos de lectura de un bloque de datos.

B. Interfaz SPI y su configuración en el ATmega32

La interconexión entre las CPUs Maestro y Esclavo en la interfaz SPI se muestra en la Figura 1. El sistema consiste de dos Shift Registers y un generador de reloj Maestro. El Maestro SPI inicia el ciclo de comunicación llevando la señal SS (Slave Select) del Esclavo elegido a cero. Maestro y esclavo preparan los datos a ser transferidos en sus respectivos Shift Registers, y el Maestro genera los pulsos de reloj requeridos en la línea SCK para intercambiar los datos.

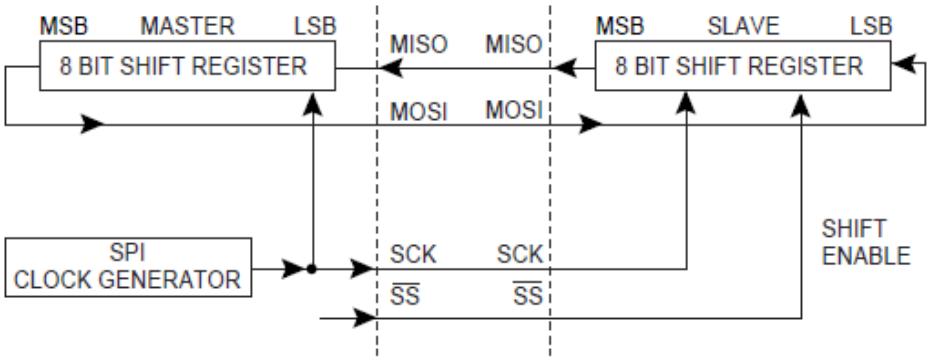

Figura 1. Interconexionado Maestro-Esclavo SPI.

Cuando se configura como Maestro, la interfaz SPI no tiene un control automático de la línea SS. Esta debe ser manejada por el software usuario antes de que la comunicación pueda comenzar. Cuando se realiza dicho manejo, el escribir un byte en el SPI Data Register activa el generador de reloj SPI y el hardware envía los ocho bits al Esclavo. Luego de enviar un byte, el generador de reloj SPI de detiene, activando la bandera de final de transmisión (SPIF).

El Maestro puede continuar enviando el siguiente byte escribiéndolo en el SPDR, o señalizar el final de transmisión del paquete llevando la señal Slave Select, SS a uno. El último byte recibido se mantiene en el Buffer Register para uso posterior.

El sistema tiene un buffer único en el sentido de transmisión y un buffer doble en el de recepción. Esto implica que los bytes a ser transmitidos no pueden escribirse al SPI Data Register antes que el ciclo completo de transmisión haya terminado. En la recepción de datos, sin embargo, un carácter recibido debe ser leído del SPI Data Register antes que el siguiente carácter sea recibido completamente. De lo contrario el primero se perderá.

Cuando se activa el SPI, el sentido de los pines de datos MOSI, MISO SCK y SS es fijado según se indica en Tabla 1.

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

Tabla 1. Sentido de los pines de datos MOSI, MISO SCK y SS.

Funcionalidad del pin SS

Modo Maestro

Cuando el SPI se configura como maestro (se setea el bit MSTR en el registro SPCR), el usuario puede determinar el sentido del pin SS.

Si SS se configura como salida, el pin es una salida general que no afecta al sistema SPI. Típicamente, este pin manejará al pin SS del Esclavo.

Si se configura como una entrada, éste debe ser mantenido en alto para asegurar la operación del Maestro SPI. Si el pin SS es llevado a cero por un periférico cuando el SPI está configurado como maestro con el pin SS definido como una entrada, el sistema SPI interpreta esto como otro Maestro eligiendo al SPI como Esclavo y comenzando a transmitir datos hacia el sistema.
Para evitar problemas en el bus el sistema SPI realiza las siguientes acciones:

1. El bit MSTR en el registro SPCR se borra y el sistema SPI se convierte en Esclavo. Resultando en la configuración de los pines MOSI y SCK como entradas.
2. La bandera SPIF en el registro se setea, y si las interrupciones están activadas y el bit I en el registro SREG está seteado, la rutina de interrupción será ejecutada. Por lo tanto, cuando las transmisiones SPI son manejadas por interrupciones en el modo Maestro, y existe la posibilidad de que SS sea llevada a cero, la interrupción debe chequear siempre que MSTR siga estando seteado. Si MSTR es llevado a cero debido a una selección de Esclavo, éste debe ser seteado por el usuario para reactivar el modo SPI Maestro.

Registros

SPI Control Register – SPCR

Bit	7	6	5	4	3	2	1	0	
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	SPCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 2. Registro SPCR.

• Bit 7 – SPIE: SPI Interrupt Enable
Este bit causa la ejecución de la interrupción del SPI si el bit SPIF en el registro SPSR es seteado y si el bit de interrupciones globales en el registro SREG está activado.

• Bit 6 – SPE: SPI Enable
Cuando el bit SPE es escrito en uno, el SPI es activado. Este bit debe ser seteado para activar cualquier operación del SPI.

• Bit 5 – DORD: Data Order
Cuando el bit DORD es llevado a uno, el LSB de la palabra dato se transmite primero.
Cuando el bit DORD es llevado a cero, el MSB de la palabra dato se transmite primero.

• Bit 4 – MSTR: Master/Slave Select
Este bit selecciona el modo SPI Maestro cuando se escribe en uno, y en modo SPI Esclavo cuando es escrito un cero lógico. Si SS es configurado como entrada y es llevado a cero mientras MSTR esta seteado, MSTR será llevado a cero, y el bit SPIF en el registro SPSR será seteado. El usuario entonces tendrá que setear MSTR para reactivarse como Maestro SPI.

• Bit 3 – CPOL: Clock Polarity

Cuando en este bit se escribe uno, SCK estará en alto mientras está inactivo. Cuando en CPOL se escribe un uno, SCK estará en nivel bajo mientras está inactivo. Referirse a Figura 5 y Figura 6 para ver un ejemplo. La funcionalidad del bit CPOL se resume a continuación:

CPOL	Leading Edge	Trailing Edge
0	Rising	Falling
1	Falling	Rising

Tabla 2. Funcionalidad de CPOL.

• Bit 2 – CPHA: Clock Phase
El seteo del bit Clock Phase (CPHA) determina si los datos son muestreados en el primer o último flanco de SCK. Referirse a Figura 5 y Figura 6 para ver un ejemplo.

La funcionalidad del bit CPHA se resume a continuación:

CPHA	Leading Edge	Trailing Edge
0	Sample	Setup
1	Setup	Sample

Tabla 3. Funcionalidad de CPHA.

• Bits 1, 0 – SPR1, SPR0: SPI Clock Rate Select 1 and 0
Estos dos bits controlan la frecuencia de SCK cuando el dispositivo se configura como Maestro. SPR1 y SPR0 no tienen efecto en el modo Esclavo. La relación entre SCK y la frecuencia del Oscilador del Reloj f_{osc} se muestra en la siguiente tabla:

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

Tabla 4. Relación entre SCK y la frecuencia del Oscilador del Reloj.

SPI Status Register –SPSR

Bit	7	6	5	4	3	2	1	0	
	SPIF	WCOL	–	–	–	–	–	SPI2X	SPSR
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figura 3. Registro SPSR.

• Bit 7 – SPIF: SPI Interrupt Flag
Cuando una transmisión serial se completa, la bandera SPIF se setea. Una interrupción se genera si el bit SPIE en el registro SPCR es uno y las interrupciones globales están activadas. Si SS es una entrada y esta es llevada a cero cuando se está en modo SPI Maestro, la bandera SPIF será seteada. SPIF es borrada por hardware cuando se ejecuta el vector de interrupción correspondiente. En forma alternativa, la bandera SPIF es borrada leyendo primero el SPI Status Register con SPIF seteada, y luego accediendo al SPI Data Register (SPDR).

• Bit 6 – WCOL: Write COLlision Flag
El bit WCOL es seteado si el SPI Data Register (SPDR) es escrito durante una transmisión de datos. El bit WCOL (y el bit SPRD) son llevados a cero al leer primero el SPI Status Register con WCOL seteado, y luego accediendo al SPI Data Register.

• Bit 5...1 – Res: Reserved Bits
Estos bits son reservados en el ATmega 32 y son siempre leídos como cero.

• Bit 0 – SPI2X: Double SPI Speed Bit
Cuando en estos bits se escribe un uno, la velocidad de la SPI (Frecuencia de SCK) es duplicada cuando la SPI está en modo Maestro. Ver “Tabla 4”. Esto significa que el mínimo periodo de SCK será dos períodos de reloj de la CPU. Cuando la SPI se configura como Esclavo, la SPI solo está garantizada para funcionar a $f_{osc}/4$ o menor.

SPI Data Register –SPDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	SPDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	X	X	X	X	X	X	X	X	Undefined

Figura 4. Registro SPDR.

El SPI Data Register es un registro de lectura/escritura utilizado para la transmisión de datos entre el Register File y el SPI Shift Register. Al escribir en el registro se inicia la transmisión de datos. La lectura del registro causa la lectura del buffer del Shift register de recepción.

Modos de Datos

Existen cuatro combinaciones para la fase y polaridad de SCK con respecto a los datos seriales, que pueden ser determinados al controlar los bits CPHA y CPOL. Los formatos de transmisión de datos de la SPI se muestran en Figura 5 y Figura 6. Los bits de datos son enviados y muestreados en flancos opuestos de la señal SDK, asegurando el tiempo suficiente para que la señal de datos se estabilice.

	Leading Edge	Trailing Edge	SPI Mode
CPOL = 0, CPHA = 0	Sample (Rising)	Setup (Falling)	0
CPOL = 0, CPHA = 1	Setup (Rising)	Sample (Falling)	1
CPOL = 1, CPHA = 0	Sample (Falling)	Setup (Rising)	2
CPOL = 1, CPHA = 1	Setup (Falling)	Sample (Rising)	3

Tabla 5. Formato de transmisión SPI con CPHA = 0

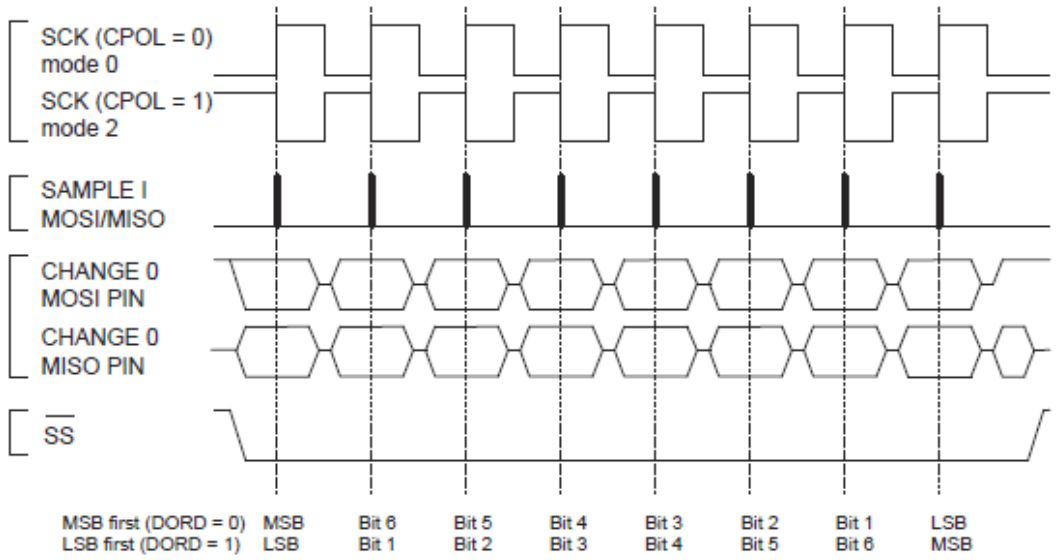

Figura 5. Formato de transmisión SPI con CPHA = 1.

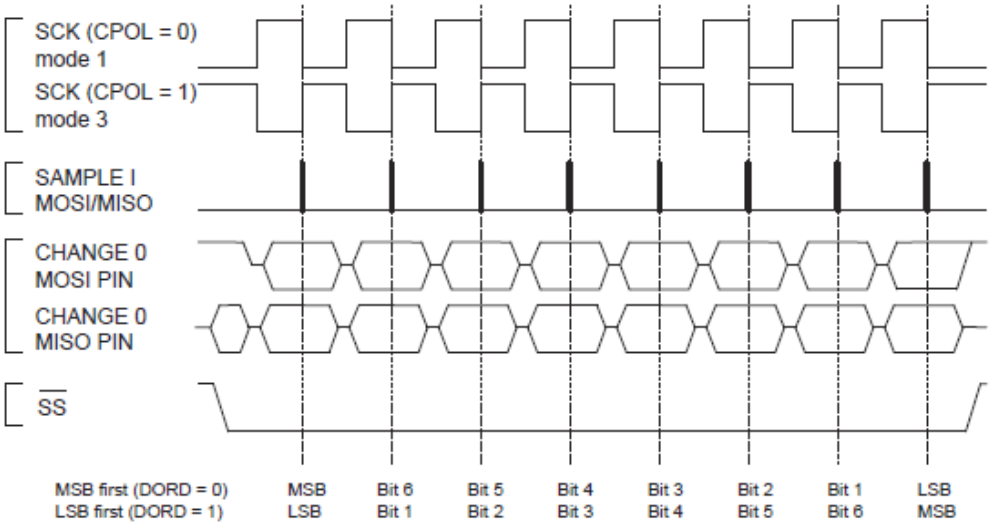

Figura 6. Formato de transmisión SPI con CPHA = 1.

Fuente: ATMEL® 8-bit AVR® Microcontroller with 32K Bytes In-System Programmable Flash. ATmega 32, ATmega 32L, © 2008 Atmel Corporation.

C. Sistemas de archivos FAT

El sistema de archivos FAT surgió por los años 70, y hoy se ha convertido en estándar en diversas áreas.
A primera impresión, una memoria podría imaginarse como un cuadro de doble entrada donde, dando las coordenadas adecuadas se puede acceder a un sector de memoria donde por ejemplo se almacene un Byte. La forma de almacenamiento sería lineal, es decir un dato a continuación del otro.
El concepto anterior presenta sus desventajas.
Inicialmente se supone la memoria vacía, entonces se almacena en ella un dato "A" el cual ocupa varios sectores, uno atrás del otro. Más tarde se dispone a guardar un dato "B", que también ocupará varios sectores, el cual se almacena a continuación del dato "A". Si a continuación se desea modificar "A" y aumentar su tamaño, se presenta un problema, o se borra parte de "B" o se continúa al lado de "B".
Éste es uno de los problemas que FAT puede resolver.

Sectores de memoria.

Toda memoria con formato FAT es dividida en cuatro áreas. Estas áreas son: "Reserved Area", "File Allocation Table" (FAT), "Root Directory", y "File Area" El tamaño de cada área puede variar pero la forma no. Ver Figura 1.
La "Reserved Area" puede contener diversa información, como ser: "Boot Sector", tamaño y número de copias de "File Allocation Table" así como el número de entradas del "Root Directory".

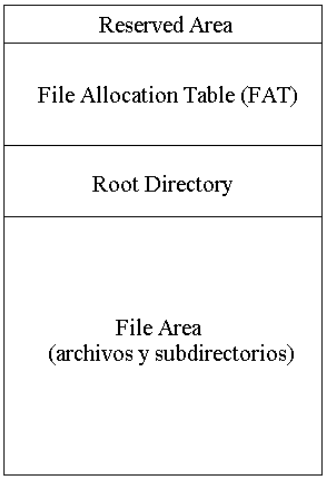

Figura 1. Áreas de una memoria con formato FAT.

Boot Sector
Este sector contiene un trozo pequeño de programa en código de máquina, para inicializar la misma en caso necesario.

Root Directory
Este es usado como tabla de contenido, identificando cada archivo del disco con un directorio que contiene información importante, como ser: nombre del archivo, tamaño, ubicación en el disco, fecha de creación, de modificación.

The File Area
El área de archivos ocupa el grueso de la memoria. Éste contiene los subdirectorios así como los archivos. Ambos son almacenados de forma lineal en la memoria en sectores llamados "clusters" (Un "cluster" es uno o más sectores consecutivos, el número de sectores que formen el "cluster" se determinan cuando se le da formato a la memoria). Se asemeja a un cuadro de doble entrada, y por tanto es aquí donde puede ocurrir lo explicado al inicio de este anexo.

File Allocation Table (FAT)
La FAT es un mapa del uso de todo el espacio del área del disco llamado "File Area", incluyendo espacio usado por los archivos, espacio no usado y espacio dañado.
De esté, dentro de la memoria hay una copia, es decir hay un respaldo de la FAT, en caso de dañarse.

La FAT contiene una entrada por cada “cluster” en “File Area”. Si el valor de la entrada no marca que el “cluster” está dañado, usado o reservado, entonces el “cluster” es parte de un archivo y es indicado por la FAT, a su vez la entrada indica el próximo “cluster” que es parte del archivo.
Esto significa que el espacio que pertenece a un determinado archivo se asigna por una cadena de entradas FAT, donde cada una apunta a la siguiente. El primer “cluster” de una archivo es identificado por su número al igual que el resto, solo que este se encuentra en la entrada guardada en el “Root Directory”. Ver Figura 2.

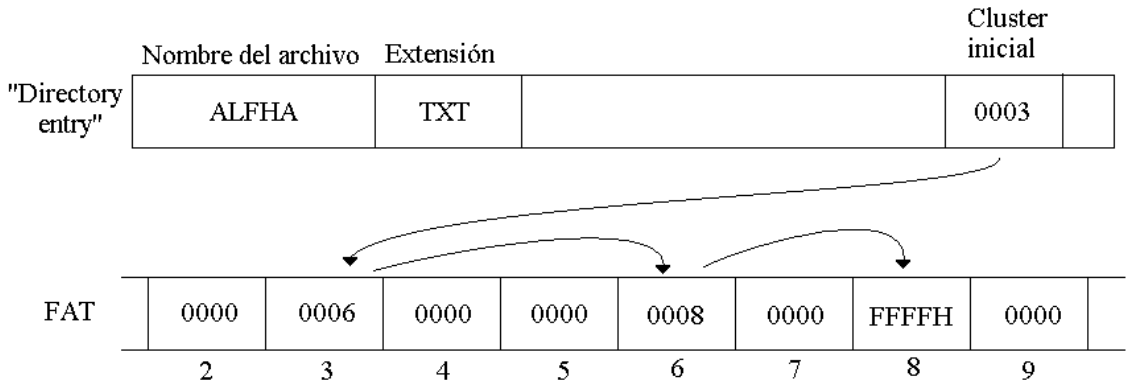

Figura 2. Uso de la FAT.

Cuando un archivo es creado o ampliado se buscan los “clusters” cuya entrada en la FAT sea cero, ya que estos son los que están vacíos. Cuando se borra un archivo se ponen en cero las entradas de la FAT que encadenan los “clusters”.
Según el tamaño que tengan las entradas en la FAT, es el sufijo que se le da a esta. Ej.: FAT16, entrada de 16 bits, y así sucesivamente.

D. Conceptos del curso aplicados al proyecto

- Conocimientos de lenguaje C.
- Round Robin - Polling. Dado que debemos esperar por un solo evento y no hay fuertes restricciones en los tiempos de respuesta es que se optó por esta opción.
- Técnicas de depuración.

E. Especificación del proyecto

Resumen

Nombre del proyecto: Vaca SD

Integrantes:
Luis Ignacio de León Echarri
María Cecilia San Román Rincón
Gonzalo Eduardo Sotta Medina

Referente: Pablo Mazzara
Fecha prevista de comienzo: 27/04/2009
Fecha prevista de finalización: 22/06/09

Descripción del problema

Desarrollar un controlador para una tarjeta de memoria SD utilizando un microcontrolador el cual reciba datos, realice las operaciones necesarias sobre la memoria SD y almacene dichos datos en una estructura de carpetas y archivos ordenados por fecha y hora.

Antecedentes

La Facultad de Agronomía de la Regional Norte de Paysandú está realizando actividades de investigación con bovinos y ovinos en la Unidad de Extensión de Cerro Largo. Algunas de ellas se basan en obtener el tiempo que dichos herbívoros se encuentran pastoreando, rumiando o ninguna de las anteriores. Para medir estos tiempos utilizan un equipo que fue desarrollado hace aproximadamente 15 años y consiste en captar los movimientos de la mandíbula, registrarlos y finalmente interpretarlos con el software propietario.
Actualmente en Facultad de Ingeniería se está desarrollando el proyecto MOSOBO (proyecto de fin de carrera) que consiste en adquirir, almacenar y procesar los sonidos que emiten los rumiantes a efectos de analizar el comportamiento ingestivo de los mismos.
En el marco de este proyecto se implementará la interfaz de almacenamiento de datos del sistema utilizando una memoria SD (Secure Digital).

Objetivos del proyecto

El objetivo de este proyecto es generar un controlador de memorias SD que permita el almacenamiento de datos para poder obtener en que momento del día y durante cuanto tiempo el rumiante realizo alguna de las actividades detalladas anteriormente. Dicha memoria será accedida desde un PC.

Alcance del proyecto

El proyecto abarca la creación del software necesario (basado en un microcontrolador a determinar) para cumplir con los objetivos especificados anteriormente. No así al hardware necesario para la conexión segura de la memoria SD con el sistema.

Descripción del sistema

Descripción funcional: El sistema debe realizar operaciones de lectura y escritura sobre la unidad de memoria SD con un formato de archivo accesible desde una PC con sistema operativo Windows o Linux.

Diagrama de bloques: El sistema está compuesto por un microprocesador (Atmega32), una memoria SD (nand), max232 y un A determinar.

Requerimientos y restricciones del sistema

Procesamiento y memoria: A determinar.
Tiempos de respuesta: A determinar.
Restricciones: los datos almacenados en la memoria SD deben tener un formato de archivo que pueda ser reconocido por un PC con Windows o Linux.

Diseño preliminar

Plataforma de hardware: Se utilizará el microprocesador Atmega 32 de ATMEL.
Arquitectura de software: Se implementará con un planificador por colas de funciones ya que de esta manera en el bucle.

Planificación

Entregas:

- Documentación del sistema.
- Informe final del proyecto.

Hitos intermedios:

- 25 de mayo: Hardware analizado y armado. Estudio de alternativas de formato de archivos (FAT16, FAT32, NTFS, etc). En caso de que la utilización de formatos de archivos sea inviable se grabaran los datos sin formato.
- 10 de junio: Avance del desarrollo del sistema.
- 22 de junio: Presentación final.

F. Documentación

Lista de los archives que se describen:

- file **fun_UART.c**
- file **fun_UART.h**
- file **main.c**
- file **SD_Fun.c**
- file **SD_Fun.h**
- file **SD_Interfaz.c**
- file **SD_Interfaz.h**
- file **SPI.c**
- file **SPI.h**

fun_UART.c

```
#include <stdio.h>
#include <stdlib.h>
#include <iom32.h>
#include "fun_UART.h"
#include "SPI.h"
```

Functions

- void **inicializarUART** (unsigned int baud)
- void **FunTX** (unsigned char data)
- unsigned char **FunRX** (void)
- void **StringTX** (unsigned char dato[])

Function Documentation

unsigned char FunRX (void)

Recepción de un dato por la UART

RETURNS:

el dato enviado por la UART

Esperamos a recibir un dato

void FunTX (unsigned char *data*)

Transmite por la UART

PARAMETERS:

data dato a transmitir

Wait for empty transmit buffer

void inicializarUART (unsigned int *baud*)

Inicializa la UART

PARAMETERS:

baud (fosc = 1MHz, baud = 12 ==> baud rate:4800, error 0.2%)

Set baud rate

Habilitamos recepción y transmisión

8 bits de datos, 1 bit de parada

void StringTX (unsigned char *dato*[])

Transmite un string por la UART

PARAMETERS:

dato[] el string a transmitir

Transmitimos cada caracter del string por la UART

fun_UART.h
Functions

- void **inicializarUART** (unsigned int baud)
- void **FunTX** (unsigned char data)
- unsigned char **FunRX** (void)
- void **StringTX** (unsigned char dato[])

Function Documentation

unsigned char FunRX (void)

Recepción de un dato por la UART

RETURNS:
el dato enviado por la UART

Esperamos a recibir un dato

void FunTX (unsigned char *data*)

Transmite por la UART

PARAMETERS:
data dato a transmitir

Wait for empty transmit buffer

void inicializarUART (unsigned int *baud*)

Inicializa la UART

PARAMETERS:
baud (fosc = 1MHz, baud = 12 ==> baud rate:4800, error 0.2%)

Set baud rate

Habilitamos recepción y transmisión

8 bits de datos, 1 bit de parada

void StringTX (unsigned char *dato*[])

Transmite un string por la UART

PARAMETERS:
dato[] el string a transmitir

Transmitimos cada caracter del string por la UART

main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <iom32.h>
#include <intrinsics.h>
#include "fun_UART.h"
#include "SPI.h"
#include "SD_Fun.h"
#include "SD_Interfaz.h"
```

Functions

- void **main** (void)

Function Documentation

void main (void)

Inicializamos el puertoB (SPI)

Inicializamos la UART con boud rate: 4800

Enviamos por la UART: Presione Iniciar para inicializar la tarjeta SD

Esperamos que el usuario presione una tecla

Inicializamos la tarjeta SD

Desactivamos el CRC

Enviamos por la UART: Tamaño de bloque por defecto (512 bytes).

Desplegamos el menu esperamos que el usuario ingrese 1(si desea escribir) o 2 (si desea leer) le pedimos la dirección a leer o escribir en caso que desee escribir esperamos los datos a escribir volvemos a desplegar el menu...

SD_Fun.c

```
#include <stdio.h>
#include <stdlib.h>
#include <iom32.h>
#include "fun_UART.h"
#include "SPI.h"
#include "SD_Fun.h"
```

Defines

- #define **CS** (0x0010)

Functions

- void **SD_Init** (void)
- char **SD_Init_Check** (void)
- void **CMD_Tx** (char CMD, unsigned int **ARG**, char CRC)
- char **Get_Resp** (void)
- void **SD_DesactivoCRC** (void)
- void **SD_RD_SGL_BLK** (unsigned int address)
- void **SD_WR_SGL_BLK** (unsigned int address, char data[])

Define Documentation

#define CS (0x0010)

Function Documentation

void CMD_Tx (char *CMD*, unsigned int *ARG*, char *CRC*)

Enviar los comandos a la tarjeta

PARAMETERS:

CMD Comando (1 byte)
arg argumento (4 bytes)
CRC checksum (1 byte)

Separamos los 4 bytes del argumento OJO!!!!!!!!

Enmascaramos el comando con 0x40, el bit 7 debe ser 0 y el bit 6 1

Enviamos por el SPI el comando, el argumento (4 bytes), el CRC)

char Get_Resp (void)

RETURNS:

Respuesta de tipo

void SD_DesactivoCRC (void)

Desactiva el CRC (checksum)

Baja CS para Enviarle los datos a la tarjeta

Envia el comando 59 (Con un 1 se habilita el CRC, con un 0 se desactiva), argumento 0 y CRC 95h

Obtenemos respuesta de tipo R1

Enviamos 0xFF por SPI

Subo CS para terminar el envio de datos a la tarjeta

void SD_Init (void)

Inicializa la tarjeta SD

Envia por la UART que comienza el proceso de inicializacion de la tarjeta

Activa el reloj con CS en alto

El CRC del CMD0 "10010101"

Bajo CS para Enviarle los datos a la tarjeta

Envia el comando 0 (resetea la memoria), argumento 0 y el CRC

Obtiene la respuesta tipo R1

Envia 0xFF luego de obtener R1 (esta en el manual)

Sube CS para terminar el envio de datos a la tarjeta

Analizamos la respuesta de tipo R1

Esperamos a que la tarjeta inicialice correctamente 10000 "ciclos"

Si luego de 10000 "ciclos" la tarjeta no inicio correctamente enviamos Mensaje de Error, sino que la tarjeta inicio correctamente

char SD_Init_Check (void)

RETURNS:

Respuesta de tipo R1

Bajamos CS para Enviarle los datos a la tarjeta

Envia el comando1 (activa el proceso de inicialización de la tarjeta SD) , los argumentos y el CRC

Obtiene una respuesta de tipo R1

Envia 0xFF luego de obtener R1

Subimos CS para terminar el envio de datos a la tarjeta

void SD_RD_SGL_BLK (unsigned int *address*)

PARAMETERS:

address Lee el bloque en esta direccion

Baja CS para Enviarle los datos a la tarjeta

Enviamos el comando 17 (lectura de un bloque de datos), address: la dirección del bloque a leer, CRC: 95h

Obtiene respuesta de tipo R1

Si la respuesta esperamos que llegue el data start token (0xFE) o Data Error

Enviamos por la UART que se comienza a leer

Recibimos los datos de la tarjeta

Enviamos por la UART que se termino de leer

Subimos CS para terminar el envio de datos a la tarjeta

void SD_WR_SGL_BLK (unsigned int *address*, char *data*[])

PARAMETERS:

address La dirección a escribir el bloque los datos a escribir

Bajo CS para Enviarle los datos a la tarjeta

Enviamos el comando 24 (Escribir un bloque de datos en la tarjeta SD), address: la dirección a guardar los datos, CRC 95h

Obtenemos respuesta de tipo R1

Si R1 es nulo

Transmite FFh por SPI

Transmite el Start block token

Manda el bloque de datos

Transmite el CRC (16 bits)

Esperamos que la tarjeta nos envia una respuesta del tipo XXX0xxx1

Transmitimos la respuesta por la UART

Si la respuesta xxx es 010 transmitimos por la UART que los datos fueron aceptados Si la respuesta xxx es 110 transmitimos por la UART que los datos no fueron aceptados por un error de escritura

Esperamos que la tarjeta deje de estar busy

Enviamos el comando 13 (Envia el status de la tarjeta, devuelve respuesta de tipo R2 (R1 + 1byte), address nulo, CRC 95h

Obtenemos respuesta de tipo R1 (primer byte respuesta R2)

Transmitimos FF por la UART

Obtenemos segundo byte respuesta R2

Si la respuesta es nula enviamos por la UART que la escritura fue correcta, sino enviamos Error

Si la respuesta R1 no era nula: Si el bit 6 es 1 enviamos Address error Si el bit 5 es 1 enviamos Parameter error; Subo CS para terminar el envio de datos a la tarjeta

SD_Fun.h

#include <stdio.h>
#include <stdlib.h>
#include <iom32.h>
#include "SPI.h"

Defines

- #define **CMD0** 0x00
- #define **CMD1** 0x01
- #define **CMD9** 0x09
- #define **CMD13** 0x0D
- #define **CMD16** 0x10
- #define **CMD17** 0x11
- #define **CMD24** 0x18
- #define **CMD59** 0x3B
- #define **Command** 0x40

Functions

- void **SD_Init** (void)
- char **SD_Init_Check** (void)
- void **CMD_Tx** (char CMD, unsigned int **ARG**, char CRC)
- char **Get_Resp** (void)
- void **SD_DesactivoCRC** (void)
- void **SetBlockSize** (unsigned int blockSize)
- void **SD_RD_SGL_BLK** (unsigned int address)
- void **SD_WR_SGL_BLK** (unsigned int address, char data[])

Define Documentation

#define CMD0 0x00
#define CMD1 0x01
#define CMD13 0x0D
#define CMD16 0x10
#define CMD17 0x11
#define CMD24 0x18
#define CMD59 0x3B
#define CMD9 0x09
#define Command 0x40

Function Documentation

void CMD_Tx (char *CMD*, unsigned int *ARG*, char *CRC*)

Enviar los comandos a la tarjeta

PARAMETERS:

CMD Comando (1 byte)
arg argumento (4 bytes)
CRC checksum (1 byte)

Separamos los 4 bytes del argumento OJO!!!!!!!!

Enmascaramos el comando con 0x40, el bit 7 debe ser 0 y el bit 6 1

Enviamos por el SPI el comando, el argumento (4 bytes), el CRC)

char Get_Resp (void)

RETURNS:

Respuesta de tipo R1

void SD_DesactivoCRC (void)

Desactiva el CRC

Baja CS para Enviarle los datos a la tarjeta

Envia el comando 59 (Con un 1 se habilita el CRC, con un 0 se desactiva), argumento 0 y CRC 95h

Obtenemos respuesta de tipo R1

Enviamos 0xFF por SPI

Subo CS para terminar el envio de datos a la tarjeta

void SD_Init (void)

Inicializa la tarjeta SD

Envia por la UART que comienza el proceso de inicializacion de la tarjeta

Activa el reloj con CS en alto

El CRC del CMD0 "10010101"

Bajo CS para Enviarle los datos a la tarjeta

Envia el comando 0 (resetea la memoria), argumento 0 y el CRC

Obtiene la respuesta tipo R1

Envia 0xFF luego de obtener R1 (esta en el manual)

Sube CS para terminar el envio de datos a la tarjeta

Analizamos la respuesta de tipo R1

Esperamos a que la tarjeta inicialice correctamente 10000 "ciclos"

Si luego de 10000 "ciclos" la tarjeta no inicio correctamente enviamos Mensaje de Error, sino que la tarjeta inicio correctamente

char SD_Init_Check (void)

RETURNS:

Respuesta de tipo R1

Bajamos CS para Enviarle los datos a la tarjeta

Envia el comando1 (activa el proceso de inicialización de la tarjeta SD) , los argumentos y el CRC

Obtiene una respuesta de tipo R1

Envia 0xFF luego de obtener R1

Subimos CS para terminar el envio de datos a la tarjeta

void SD_RD_SGL_BLK (unsigned int *address*)

PARAMETERS:

address Lee el bloque en esta direccion

Baja CS para Enviarle los datos a la tarjeta

Enviamos el comando 17 (lectura de un bloque de datos), address: la dirección del bloque a leer, CRC: 95h

Obtiene respuesta de tipo R1

Si la respuesta esperamos que llegue el data start token (0xFE) o Data Error

Enviamos por la UART que se comienza a leer

Recibimos los datos de la tarjeta

Enviamos por la UART que se termino de leer

Subimos CS para terminar el envio de datos a la tarjeta

void SD_WR_SGL_BLK (unsigned int *address*, char *data*[])

PARAMETERS:

address La dirección a escribir el bloque los datos a escribir

Bajo CS para Enviarle los datos a la tarjeta

Enviamos el comando 24 (Escribir un bloque de datos en la tarjeta SD), address: la dirección a guardar los datos, CRC 95h

Obtenemos respuesta de tipo R1

Si R1 es nulo

Transmite FFh por SPI

Transmite el Start block token

Manda el bloque de datos

Transmite el CRC (16 bits)

Esperamos que la tarjeta nos envia una respuesta del tipo XXX0xxx1

Transmitimos la respuesta por la UART

Si la respuesta xxx es 010 transmitimos por la UART que los datos fueron aceptados Si la respuesta xxx es 110 transmitimos por la UART que los datos no fueron aceptados por un error de escritura

Esperamos que la tarjeta deje de estar busy

Enviamos el comando 13 (Envia el status de la tarjeta, devuelve respuesta de tipo R2 (R1 + 1byte), address nulo, CRC 95h

Obtenemos respuesta de tipo R1 (primer byte respuesta R2)

Transmitimos FF por la UART

Obtenemos segundo byte respuesta R2

Si la respuesta es nula enviamos por la UART que la escritura fue correcta, sino enviamos Error

Si la respuesta R1 no era nula: Si el bit 6 es 1 enviamos Address error Si el bit 5 es 1 enviamos Parameter error

Subo CS para terminar el envio de datos a la tarjeta

void SetBlockSize (unsigned int *blockSize*)

SD_Interfaz.c

```
#include <stdio.h>
#include <stdlib.h>
#include <iom32.h>
#include <string.h>
#include <intrinsics.h>
#include "fun_UART.h"
#include "SPI.h"
#include "SD_Fun.h"
#include "SD_Interfaz.h"
```

Functions

- void **Menu** (void)
- void **desplegarMenu** (void)
- void **esperaDatos** (void)
- long int **armarDatos** (void)

Variables

- int long **ARG**

Function Documentation

long int armarDatos (void)

A partir de los datos almacenados en el arreglo str arma la dirección que se desea leer/escribir

void desplegarMenu (void)

Enviamos por la UART el menu: 1 - Leer 2 - Escribir Ingrese una opcion:

void esperaDatos (void)

Recibe y almacena los datos transmitidos por la UART hasta recibir una x!!!

void Menu (void)

Desplega el menu en pantalla
Espera que el usuario ingrese el comando de escritura o de lectura
Si el usuario ingresa un 1 (lectura)
Envia por la UART: Escriba la dirección para leer
Espera la dirección del bloque a leer
Modifica el formato de los datos ingresados
Envia el comando de lectura
Si el usuario ingresa un 2 (escritura)
Envia por la UART: Escriba la dirección para escribir
Espera la dirección a escribir
Modifica el formato de los datos ingresados
Envia por la UART: Escriba el dato a escribir:
Espera los datos
Enviamos el comando de escritura

Variable Documentation

int long ARG

SD_Interfaz.h

```
#include <stdio.h>
#include <stdlib.h>
#include <iom32.h>
#include "SPI.h"
#include "fun_UART.h"
#include "SD_Fun.h"
```

Functions

- void **Menu** (void)
- void **desplegarMenu** (void)
- void **esperaDatos** (void)
- long int **armarDatos** (void)
- void **procesarData** (void)

Function Documentation

long int armarDatos (void)

A partir de los datos almacenados en el arreglo str arma la dirección que se desea leer/escribir

void desplegarMenu (void)

Enviamos por la UART el menu: 1 - Leer 2 - Escribir Ingrese una opcion:

void esperaDatos (void)

Recibe y almacena los datos transmitidos por la UART hasta recibir una x!!!

void Menu (void)

Desplega el menu en pantalla
Espera que el usuario ingrese el comando de escritura o de lectura
Si el usuario ingresa un 1 (lectura)
Envia por la UART: Escriba la dirección para leer
Espera la dirección del bloque a leer
Modifica el formato de los datos ingresados
Envia el comando de lectura
Si el usuario ingresa un 2 (escritura)
Envia por la UART: Escriba la dirección para escribir
Espera la dirección a escribir
Modifica el formato de los datos ingresados
Envia por la UART: Escriba el dato a escribir:
Espera los datos
Enviamos el comando de escritura

void procesarData (void)

SPI.c

```
#include <stdio.h>
#include <stdlib.h>
#include <iom32.h>
#include <intrinsics.h>
#include "SPI.h"
```

Defines

- #define **CS** (0x0010)

Functions

- void **SPI_MasterInit** (void)
- void **SPI_MasterTx** (char cData)
- char **SPI_MasterRx** (void)
- void **SPI_MoverCK** (void)

Define Documentation

#define CS (0x0010)

Function Documentation

void SPI_MasterInit (void)

Inicialia SPI

Set MOSI, SS and SCK output in Regiter B, all others input

Enable SPI, Master, set clock rate fck/16

Asserting SS pin to '1'

char SPI_MasterRx (void)

Recibe por SPI

RETURNS:

el caracter recibido por SPI

void SPI_MasterTx (char *cData*)

Transmite por SPI

PARAMETERS:

cData parametro a transmitir por SPI

Comenzar la transmisión

Espera a que se complete la transmisión

void SPI_MoverCK (void)

Mueve el reloj

Comienza la transmisión

Espera a que se complete la transmisión

SPI.h
Functions

- void **SPI_MasterInit** (void)
- void **SPI_MasterTx** (char cData)
- char **SPI_MasterRx** (void)
- void **SPI_MoverCK** (void)

Function Documentation

void SPI_MasterInit (void)

Inicialia SPI

Set MOSI, SS and SCK output in Regiter B, all others input

Enable SPI, Master, set clock rate fck/16

Asserting SS pin to '1'

char SPI_MasterRx (void)

Recibe por SPI

RETURNS:

el caracter recibido por SPI

void SPI_MasterTx (char *cData*)

Transmite por SPI

PARAMETERS:

cData parametro a transmitir por SPI

Comenzar la transmisión

Espera a que se complete la transmisión

void SPI_MoverCK (void)

Mueve el reloj

Comienza la transmisión

Espera a que se complete la transmisión

G. Comentarios de la Planificación

La planificación fue correcta, pues a pesar de no haberla podido cumplir en forma, los tiempos se armaron de manera de dejar la última semana libre por si se presentaban dificultades. La principal dificultad fue debido a que algunos detalles sobre el funcionamiento de la tarjeta SD no estaban especificados. Además, se presentaron varios problemas con el hardware, entre ellos, problemas con la UART y el Dragon.

En total, las horas dedicadas al proyecto fueron 350.