MMC/SD FAT 16 mit AVR und C

Die Bibliothek von Roland Riegel enthält einen kompletten FAT16 Zugriff auf MMC/SD Karten über den SPI eines µC.

Anforderungen:

- SPI oder 5 freie PINs am Controller
- Für den einfach Schreib/Lesezugriff mindestens 1 kB Ram, besser 2 kB
- Ca. 14 kB freier Flash-speicher

Nötig sind folgende Dateien:

- sd_raw.c
- sd_raw.h
- sd_raw_config.h
- sd-reader_config.h
- partition.c
- partition.h
- partition_config.h
- fat16.c
- fat16.h
- fat16_config.h

Die Bibliothek kümmert sich damit um den Lowlevel und den FAT16 Zugriff. Dies vereinfacht den Umgang mit MMC/SD-Karten erheblich.

Wenn man Daten schreibt, sollte man wenn möglich 512 Byte auf einmal schreiben. Da die Karte immer nur komplette Sektoren schreiben kann. Dies schützt die Karte vor Beschädigungen durch wiederholtes Schreiben.

Initialisieren

MMC/SD Karten

Als erstes muss die MMC/SD Karte initialisiert werden.

```
sd_raw_init();
```

Diese Funktion liefert 0 zurück wenn die Karte initialisiert wurde.

Partition wählen

Als nächstes wird die Partition gewählt.

```
struct partition_struct* partition;
partition = partition_open(sd_raw_read,
                           sd_raw_read_interval,
                           sd_raw_write,
                           sd_raw_write_interval,
                           0
                           );
```

Dieser Code öffnet die erste Partition eines Datenträgers mit Schreib und Leserechten. Die Variable partition ist leer wenn es nicht ging.
Für einen Datenträger ohne MBR, auch „Superfloppy“ muss folgender Struct angelegt werden.

```
struct partition_struct* partition;

partition = partition_open(sd_raw_read,
                           sd_raw_read_interval,
                           sd_raw_write,
                           sd_raw_write_interval,
                           -1
                           );
```

Ist auch dieser leer gibt es keine Partitionen.

Dateisystem öffnen

Wenn die Partition gewählt wurde, muss das Dateisystem geöffnet werden.

```
struct fat16_fs_struct* fs = fat16_open(partition);
```

Die Variable fs ist leer wenn kein Dateisystem gefunden wurde.

Wurzelverzeichnis öffnen

Wenn das Dateisystem gewählt wurde, muss das Wurzelverzeichnis geöffnet werden. Wenn man ein Verzeichnis öffnen möchte, muss man zuerst ein Struct fat16_dir_entry_struct anlegen. Im Beispiel heißt dies directory. Dieses Struct wird dann mit dem Dateisystem der Funktion fat16_get_dir_entry_of_path übergeben. Ein geöffnetes Verzeichnis ist vom Typ fat16_dir_struct. Dies wird erzeugt aus fat16_dir_entry_struct.

```
struct fat16_dir_entry_struct directory;
fat16_get_dir_entry_of_path(fs, "/", &directory);

struct fat16_dir_struct* dd = fat16_open_dir(fs, &directory);
```

Wenn alles geklappt hat ist dd nicht leer.

Mit Dateien umgehen

Verzeichnis wechseln

Um ein Verzeichnis zu wechseln ist folgender Programmcode nötig

```
struct fat16_dir_entry_struct subdir_entry;
if(find_file_in_dir(fs, dd, Verzeichnis, &subdir_entry))
{
  struct fat16_dir_struct* dd_new;
  dd_new = fat16_open_dir(fs, &subdir_entry);
  if(dd_new)
  {
    fat16_close_dir(dd);
    dd = dd_new;
  }
}
```

Dieses Beispiel ist ähnlich wie Wurzelverzeichnis öffnen. Nur das hier als Pfad der Name des Verzeichnisses gewählt werden muss. Außerdem muss überprüft werden ob das Verzeichnis Existiert und ob es geöffnet wurde, da sonst das momentane Verzeichnis geschlossen wird und man nicht mehr zurückkommt.

Dateien im Verzeichnis auflisten

Um die Dateien eines Verzeichnisses aufzulisten, muss man die Informationen aus der Verzeichnissdatei lesen.

```
struct fat16_dir_entry_struct dir_entry;
while(fat16_read_dir(dd, &dir_entry))
{
  uart_puts(dir_entry.long_name);
}
```

`dir_entry.long_name` enthält nach jedem Aufruf von `fat16_read_dir(dd, &dir_entry)`den nächsten Eintrag der Tabelle. Im Beispiel wird dieser Eintrag auf den UART ausgegeben.

Dateien öffnen und lesen

Als ersten muss man die Datei öffnen.

```
struct fat16_file_struct* fd;
fd = open_file_in_dir(fs, dd, Dateiname);
```

Dateiname gibt den Namen der Datei an die geöffnet werden soll. fd ist leer wenn das Öffnen nicht geklappt hat.

Um aus dieser Datei Daten zu lesen muss folgende Funktion aufgerufen werden.

```
uint8_t buffer[8];
fat16_read_file(fd, buffer, sizeof(buffer));
```

In diesen Fall werden 8 Byte Daten in den Buffer geschrieben. Der Offset zum lesen wird dann automatisch hoch gezählt. Wenn man die Funktion noch mal aufruft werden die nächsten 8 Byte gelesen. Der Buffer kann natürlich größer sein, verbraucht dann aber auch mehr RAM.

Datei löschen

Dieser Programmteil dient dem löschen von Programmen.

```
struct fat16_dir_entry_struct file_entry;
if(find_file_in_dir(fs, dd, Dateiname, &file_entry))
  fat16_delete_file(fs, &file_entry);
```

Die Datei muss ähnlich wie im Beispiel **Verzeichnis wec**hseln in eine fat16_dir_entry_struct abgelegt werden. Mit fat16_delete_file(fs, &file_entry) wird die Datei dann gelöscht. Die Funktion gibt 1 zurück wenn die Datei erfolgreich gelöscht wurde.

Datei anlegen

Um eine Datei anzulegen muss erst wieder ein fat16_dir_entry_struct anlegen.

```
struct fat16_dir_entry_struct file_entry;
fat16_create_file(dd, Dateiname, &file_entry);
```

Mit fat16_create_file wird dann die Datei im offenen Verzeichniss dd mit den Dateinamen angelegt.

Verzeichnis anlegen

Um eine Verzeichnis anzulegen muss erst wieder ein fat16_dir_entry_struct anlegen.

```
struct fat16_dir_entry_struct file_entry;
fat16_create_dir(dd, command, &dir_entry);
```

Dies geschieht genauso wie im Beispiel **Datei anlegen** nur das diesmal fat16_create_dir benutzt wird.

Daten in Datei schreiben

Man muss eine Datei öffnen.

```
struct fat16_file_struct* fd;
fd = open_file_in_dir(fs, dd, Dateiname);
```

Dateiname gibt den Namen der Datei an die geöffnet werden soll. fd ist leer wenn das Öffnen nicht geklappt hat.

Der Funktion zum schreiben müssen dann folgende Parameter übergeben werden. Der Struct der geöffneten Datei, einen Charbuffer mit den Daten und die Länge des aus den Charbuffers zu schreibenden Daten.

```
fat16_write_file(fd, (uint8_t*) buffer, data_len);
```

Die Funktion gibt als Wert die Anzahl der geschriebenen Daten zurück. Wenn dies ungleich data_len ist, hat das Schreiben nicht funktioniert.

Datei schließen

Nach den Abschluss des Schreiben oder Lesen sollte man die Datei schließen.

```
fat16_close_file(fd);
```

Dateisystem synchronisieren

Ein wichtiger Schritt nachdem man etwas am Dateisystem verändert hat, ist das Synchronisieren.

```
sd_raw_sync();
```

Diese Funktion gibt einen Wert ungleich 0 zurück, wenn das synchronisiere funktioniert hat.

Verzeichnis schließen

```
fat16_close_file(dd);
```

Dateisystem schließen

```
fat16_close(fs);
```

Partition schließen

```
partition_close(partition);
```

Anmerkungen:

```
uint8_t find_file_in_dir(struct fat16_fs_struct* fs, struct
fat16_dir_struct* dd, const char* name, struct
fat16_dir_entry_struct* dir_entry)
{
    while(fat16_read_dir(dd, dir_entry))
    {
        if(strcmp(dir_entry->long_name, name) == 0)
        {
            fat16_reset_dir(dd);
            return 1;
        }
    }

    return 0;
}

struct fat16_file_struct* open_file_in_dir(struct
fat16_fs_struct* fs, struct fat16_dir_struct* dd, const char*
name)
{
    struct fat16_dir_entry_struct file_entry;
    if(!find_file_in_dir(fs, dd, name, &file_entry))
        return 0;

    return fat16_open_file(fs, &file_entry);
}
```

Dies müsste noch in die fat16.c integriert werden, damit der Dateizugriff über fat16 aus einer Datei kommt.