AppleSqueezer GS API

This page offers valuable insights for programmers seeking to unleash the full potential of the AppleSqueezer GS. It provides comprehensive instructions on harnessing the device's complete memory capacity and utilizing the SD card functionality.

240MB RAM memory access

The AppleSqueezer comes equipped with 256MB of DRAM memory, of which 16MB is accessible directly by the 65C816 CPU (its maximum addressable range). In practice, it has access to a bit less, as part of this address space is used by the ROMs and other details, and so only 13MB is actually accessible through GS/OS.

The remaining 240MB, typically unused, can be accessed by programs through specific addresses outlined below. One such program that makes use of this is the 32MB RAM Disk driver.

To access this range, **first set the address** in the DRAM memory space by writing bytes to the `DRAM_ADDRESS_XXX` locations as indicated below.

```
#define DRAM_ADDRESS_MAIN_BANK  0xE30000L
#define DRAM_ADDRESS_LOW        0xE30002L
#define DRAM_ADDRESS_HIGH       0xE30004L
#define DRAM_ADDRESS_BANK       0xE30006L

#define DRAM_ACCESS             0xE30008L
```

The full address is as follows: `{ MAIN_BANK, BANK, HIGH, LOW }`, of which `DRAM_ADDRESS_MAIN_BANK` can span from `0` to `0e` (including both), for a total of **15** main banks. Each main bank's address range is $3 * 8 = 24 bits$. $2^{24} = 16MB$ for each main bank, so $15 * 16MB = 240MB$.

After setting the address, you can then write or read bytes to and from `DRAM_ACCESS`, which will auto-increment after each read or write. So, in most cases, you will set the address once and then read/write your bytes consecutively from/to `DRAM_ACCESS`.

SD card access

The information about the SD card is subject to change until the core involved is released publicly.

Newer, purple versions of the AppleSqueezer have an SD card slot, which is not currently used by the hardware or software, but being developed. A special core version is required to access it. When this core version is installed, you can use the information below to access the SD card. This allows you to read or write sectors of 512 bytes to the SD card, as described in many resources online, such as:

- SD and SDIO
- https://www.convict.lu/pdf/ProdManualSDCardv1.9.pdf
- https://community.nxp.com/pwmxy87654/attachments/pwmxy87654/imx-processors%40tkb/3706/1/Part_1_Physical_Layer_Specification_Ver3.01_Final_100218.pdf

To use this in a meaningful way, you will need to use a file system to access this raw data. To do this, you can use existing libraries, such as this one:

- http://elm-chan.org/docs/fat_e.html
- FatFs - Generic FAT Filesystem Module
- Petit FAT File System Module

The last library, Petit FAT, was successfully compiled using ORCA/C on the IIGS, and it works very well. A special disk image can be requested with code that reads and writes sample files from/to the SD card. To use the Petit FAT library, you need to implement the `disk_readp` and `disk_writep` functions for reading and writing a sector. This is done using the **CMD17** (read block) and **CMD24** (write block) SD card commands, which can be accessed as follows:

```
#define SD_ADDRESS_SET_MSB    0xE40000L
#define SD_ADDRESS_SET_MSB_1  0xE40002L
#define SD_ADDRESS_SET_MSB_2  0xE40004L
#define SD_ADDRESS_SET_MSB_3  0xE40006L
```

```
#define SD_START_READ          0xE40008L // starts reading
#define SD_START_WRITE         0xE4000cL // starts writing

#define SD_ACCESS              0xE4000aL
```

Start by setting the sector number of the SD card by using the `SD_ADDRESS_SET_XXX` addresses, 8 bits each. The sector number is formed as follows: `{ SET_MSB, SET_MSB_1, SET_MSB_2, SET_MSB_3 }`, for a total of 32 bits (FAT32). Then write a `1` to `SD_START_READ` or `SD_START_WRITE`, respectively, and after that, read or write **exactly** 512 bytes to `SD_ACCESS`. In case of a write block, the CRC bytes will be automatically calculated and added at the end by the hardware.

Here's an implentation of the disk_writep and disk_readp functions for the PetitFAT library that works with the AppleSqueezer SD card:

```
/*-------------------------------------------------------------------
/* Initialize Disk Drive
/*-------------------------------------------------------------------

DSTATUS disk_initialize (void) {
    DSTATUS stat = 0;

    // Put your code here

    return stat;
}

void skip(int count) {
    for (int i = 0; i < count; i++) {
        BYTE byte = *((BYTE *) SD_ACCESS);
    }
}


/*-------------------------------------------------------------------
/* Read Partial Sector
/*-------------------------------------------------------------------

DRESULT disk_readp (
    BYTE* buff,        /* Pointer to the destination object */
    DWORD sector,    /* Sector number (LBA) */
```

```
    UINT offset,    /* Offset in the sector */
    UINT count        /* Byte count (bit15:destination) */
) {
    DRESULT res;
    int bytesRead = 0;

    // set address
    *((BYTE *) SD_ADDRESS_SET_MSB) = sector >> 24;
    *((BYTE *) SD_ADDRESS_SET_MSB_1) = (sector >> 16) & 0xff;
    *((BYTE *) SD_ADDRESS_SET_MSB_2) = (sector >> 8) & 0xff;
    *((BYTE *) SD_ADDRESS_SET_MSB_3) = sector & 0xff;

    // start reading
    *((BYTE *) SD_START_READ) = 0x01; // dummy data

  // simple implementation reading 512 bytes, commented out because th
  // speeds it up.
    // read max. 512 bytes
    //for (int i = 0; i < 512; i++) {
    //    BYTE byte = *((BYTE *) SD_ACCESS);
    //    if (i >= offset && bytesRead < count) {
    //        *buff++ = byte;
    //        bytesRead++;
    //    }
    //}

    int bc = 512 - offset - count;
    if (offset) skip(offset);

  // write bytes in blocks of 8, to reduce the amount of instructions
    int count8 = count / 8;
    if (count8) {
        do {
            *buff++ = *((BYTE *) SD_ACCESS);
            *buff++ = *((BYTE *) SD_ACCESS);
            *buff++ = *((BYTE *) SD_ACCESS);
            *buff++ = *((BYTE *) SD_ACCESS);
            *buff++ = *((BYTE *) SD_ACCESS);
            *buff++ = *((BYTE *) SD_ACCESS);
            *buff++ = *((BYTE *) SD_ACCESS);
            *buff++ = *((BYTE *) SD_ACCESS);

            count -= 8;
        } while (--count8);
    }

    if (count) {
```

```
            do {
                *buff++ = *((BYTE *) SD_ACCESS);
            } while (--count);
        }
        skip(bc);

        res = RES_OK;

        return res;
}


/*-----------------------------------------------------------------------
/* Write Partial Sector
/*-----------------------------------------------------------------------

DRESULT disk_writep (
        BYTE* buff,        /* Pointer to the data to be written, NULL:Init
        DWORD sc        /* Sector number (LBA) or Number of bytes to send
) {
        DRESULT res = RES_ERROR;
        static UINT wc;
        UINT bc, tmr;

        if (!buff) {
            if (sc) {
                // Initiate write process
                // set address
                *((BYTE *) SD_ADDRESS_SET_MSB) = sc >> 24;
                *((BYTE *) SD_ADDRESS_SET_MSB_1) = (sc >> 16) & 0xff;
                *((BYTE *) SD_ADDRESS_SET_MSB_2) = (sc >> 8) & 0xff;
                *((BYTE *) SD_ADDRESS_SET_MSB_3) = sc & 0xff;

                // start writing
                *((BYTE *) SD_START_WRITE) = 0x01; // dummy data

                wc = 512; /* Set byte counter */
                res = RES_OK;

            } else {
                // Finalize write process
                while (wc--) *((BYTE *) SD_ACCESS) = 0;
                res = RES_OK;
            }
        } else {
            bc = (UINT)sc;
            while (bc && wc) {        /* Send data bytes to the card */
```

```
            // Send data to the disk
            *((BYTE *) SD_ACCESS) = *buff++;
            wc--; bc--;
        }
        res = RES_OK;
    }

    return res;
}
```

Determine if AppleSqueezer is installed

To check if an AppleSqueezer is installed on your system, use the following code:

```
#define FL_IDLE    0xe2000aL
#define FL_VERSION 0xe2000cL

int isAppleSqueezer(void) {
    return *((char *) FL_IDLE) == 0x01;
}
```

To get the core version of the AppleSqueezer:

```
int getCoreVersion(void) {
    if (isAppleSqueezer()) {
        return *((char *) FL_VERSION);
    } else {
        return -1;
    }
}
```

Changing the speed of the AppleSqueezer

The speed of the AppleSqueezer is the only setting that can be changed instantly, without requiring a reboot. It can be done by writing to the SET_SPEED register, see below:

```
#define SET_SPEED 0xe50000L


#define speedTotal 5

/** These 5 settings correspond to the 5 speed options in
the AppleSqueezer control panel. 255 is the fastest setting,
corresponding to 14MHz. 223 is the slowest setting,
corresponding to 3MHz */
int speedOptions[speedTotal] = { 255, 251, 247, 239, 223 };

...
// set speed instantly (so it doesn't require a reboot)
*((char *) SET_SPEED) = speed;
...
```

Note that the speed setting is reset when the system is (hard-) rebooted. Also, it's possible to set the speed to other values than the ones listed in `speedOptions` above. Setting it lower than 223 may make the system unstable.

Changing the speed in this way doesn't disable acceleration completely. To disable acceleration, the flash settings need to be changed, which is a more complex procedure. Also, it's important to realize that parts of the system may still run somewhat faster than expected at a given speed (when changing it using the procedure above), since parts of the acceleration functionalities remain active.