# Interface Specifications for the Smart Media EEPROM

**Last Updated: September 2, 2003**

## 1 Overview

This document was written to explain the interface and protocol requirements for the SMEEPROM Smart Media Electronic Erasable Programmable Read Only Memory. Please report typos, inaccuracies, and especially unclear explanations to us at `spark@sparkfun.com`. Suggestions for improvements are welcome and greatly valued.

This documentation is meant for general users including students, hobbyists, and professionals who are familiar with micro-controllers (PIC, AVR, MCP) and microprocessors (Motorola 8051, Intel x86).

This document has three parts: Section 2 describes the RS232 requirements between the SMEEPROM and various external devices. Section **??** describes the hardware layout and physical dementions of the SMEEPROM device. and Section **??** gives some general applications and pseudo-code examples.

It's assumed that users are familiar with microcontrollers/microprocessors and the complexities involved with interfacing them to external devices.

### 1.1 Boat loads of memory

The entire point of creating the SMEEPROM was because we needed EEPROM type memory for our projects - and lots of it. While smart media was exteremly cheap ($16 for 64Mb at the time of writing), it was incredibly hard to find good interface hardware (the physical socket) and the software (example control code) for such memory. Luckily, students have been hacking away at Smart Media Cards for years.

Using the SSFDC forum (http://www.ssfdc.or.jp/), and the senior design project report by student Chris Morley (Morley Electronics) provided a wealth of information both to the physical and logical layers of the SSFDC (Smart Media) format.

## 2 Interface Specifications

This section describes the nature of the interface between the SMEEPROM and the controling device.

### 2.1 How To Talk

```
All transmissions occur at 9600bps with 1
  start bit, 1 stop bit, and no parity.
```

While there is only one protocol supported currently (RS232), there are plans for I2C and SPI interfaces in the near future. RS232 9600 baud was chosen for its uC and uP independance. If you've got a hardware UART, USART, EUART on your micro, or if you can fake it with software (we do!), then you can talk to the SMEEPROM.

You must tell the SMEEPROM what to do, and where to do it. An example conversation can be found `here` 1.

`FFFF` is the 4-bit command (see below). `AAA` is the 26-bit absolute address. `DDD` is the 8-bit data to be stored. The header byte #0 (`0xD4`) and the terminator byte #6 (`0x4A`) must be included around every transmission. The SMEEPROM uses these as error and transmission detection bytes. Byte 0 must be transmitted first followed by the rest of the conversation, ending with the terminator byte.

### 2.2 Storage Space

Smart Media memory is broken into various blocks, pages, bytes, and pointers. But you don't have to worry about any of that! The 26-bit address contained in the conversation is an absolute address. The controller takes care to split the transmitted address (your address) into the specific block, page, byte, and pointer where your data will reside. This makes sequential storage very easy.

Simply increment two 16-bit or one 32-bit counter and use it as the next available storage location.

Because of the nature of the memory (NAND Flash), there are different ways of writing to a given memory position. All Smart Media is NAND Flash. Flash means that it can be erased and written to many (100,000) times before it will begin to fail. The NAND part means that new data is ANDed together with the current data when a store command is initiated. If you've got 0x00 stored at location 0x1234 and you try to write a byte 0xAA to that spot, (0xAA & 0x00 = 0x00) nothing will be stored. A blank memory position is 0xFF. Anything ANDed with 0xFF is that thing (0xAA & 0xFF = 0xAA).

The only way to "'erase"' a position in memory (return it to 0xFF) is to do a block erase (32 pages x 512 bytes per page = 16k must be erased in one hitch). Because of this limitation, the onboard controller was designed to be able to handle single byte edits. It does this by copying the entire block to a temporary location, omitting the single byte that needs to be changed, and copying the entire block back to its original position. While this may seem drastic, single byte edits take no more than 190ms to complete.

There are things to consider when using the SMEEPROM for an application. You will need to be certain not to a "'write"' to an area that already has stored data. Both previously stored data and new data will be destroyed.

## 2.3 Command Overview

Here is a list of the different commands currently recognized by the SMEEPROM. The SMEEPROM will respond to each command with a response byte **0b.—-.1010** where —- is the given command. All commands must be included in the 4-bit command section of byte #1 of the conversation.

1. **Status - 0000** : The SMEEPROM will respond with a single byte (**0x0A**) if it is available for interviews. Otherwise, it is in the middle of an edit or block erase and can not (will not) be bothered. Execution time - 400us. Use this command to poll the SMEEPROM before initiating a conversation.

2. **Read - 0010** : The SMEEPROM will look up the byte located at the given address and return an acknowledge byte **0x2A** followed by the value of the retrieved byte. Response time - 400us. Each read address is stored locally for multi-read commands (see below). Before a multi-read command is initiated, a single normal read must be completed to locate the starting point for the multi-read.

3. **Write - 0110** : The SMEEPROM will store the provided byte to the given address and return one acknowledge byte **0x6A**. Execution time - 400us. If you attempt to write to a byte that already cotains data, both bytes will be anded together and stored. ie, all "'blank"' bytes are actually 0xFF. When anded together, your data is stored.

4. **Edit - 0100** : The SMEEPROM will store the provided byte to the given address and return one byte **0x4A** upon successful completion. Execution time - **180ms**. The reason for such an extereme difference in execution time lies in the technology behind Smart Media. Smart Media Cards are simply chunks of flash NAND memory embedded in a thin piece of plastic. It is impossible to change a single location in memory. The entire block in which the byte resides must be copied, erased, then copied back with the single new byte in place. While the controller takes complete care of the entire process, be aware that editing is a very intense process and will delay the controller in time sensitive applications. In other words, use the edit command for special purposes and use the write command sequentially for raw data storage. **Note:** The edit command uses the last available block in memory as scratch area. This area in memory should be considered un-usable since any data written contained in that location will be corrupt after the completion of an 'Edit' command.

5. **Block Erase - 1100** : The SMEEPROM will erase the block that contains the provided address and respond with a single byte (**0xCA**) upon completion. Execution time - 1.9ms. Use this command to format (restore to 0xFF values) an area of the card before a sequential write.

6. **Next Open Spot - 1000** : On boot up, the SMEEPROM will attempt to determine the next available free position and stores this address to memory. Upon receipt of the NOS command, the SMEEPROM will respond with five bytes. The first byte (**0x8A**) will be followed by four bytes that makeup the absolute address. For example (see Table 2).

   Where byte #1 is the most significant byte of the 26-bit absolute address. The NOS address is updated with every write command. That is to say, if there is a write to a random section of memory, the NOS address will start from that address.

7. **Multi-Write - 1110** : Used for sequential writes to memory. Only four bytes must be transmitted to the SMEEPROM - No address is transmitted. Using the NOS address, the SMEEPROM will write the provided data byte to the next available position in memory. Upon receipt of the Multi-Write command, the SMEEPROM will respond with a single byte (**0xEA**) upon completion. A multi-write conversation is shorter than a standard write conversation:

```
Byte#           0        1        2        3
Bit       76543210 76543210 76543210 76543210
Function  11010100 1110---- DDDDDDDD 01001010
```

   Where byte #0 and #3 are the header and terminator bytes respectively. Byte #1 is the command byte. And byte #2 is the data byte to be stored.

8. **Multi-Read - 1010** : Used for sequential reads from memory. Only three bytes must be transmitted to the SMEEPROM - No address or data bytes are transmitted. Using the last read address, the SMEEPROM will return the next data byte in memory. Upon receipt of the Multi-Read command, the SMEEPROM will respond with two bytes. The first byte (**0xAA**) is followed by the data byte that has been read from memory. A multi-read conversation is shorter than a standard read conversation:

```
Byte#           0        1        2
Bit       76543210 76543210 76543210
Function  11010100 1010---- 01001010
```

   Where byte #0 and #2 are the header and terminator bytes respectively. Byte #1 is the command byte. Before a multi-read command is initiated, a single normal read must be completed to locate the starting point for the multi-read. The multi-read address is automatically advanced with each issued command. The address is also advanced across all pages, blocks, and pointers automatically. A sequential read of all 64Mb of memory is possible.

## 3 Hardware Layout

This section describes the physical nature of the SMEEPROM. Everything is pretty self explanitory.

### 3.1 Physical Dimensions

The SMEEPROM is 1.66"' by 2.06"' with a maximum depth of 0.60"'. The Smart Media socket is located on the reverse side of the PCB. All Smart Media cards must be inserted with the metal contacts facing toward the PCB. There are four standard mounting holes of .130"' diameter.

### 3.2 Power Requirements

There is a 3.3V regulator that provides a regulated supply of power to the PIC controller and to the media card. Because of this regulation, a higher voltage supply (equal or greater than 5V) must be applied to the board for proper function. While the regulator may provide some protection, reversing the polarity of the power connections is a very bad idea. Double check your connections before power-on.

### 3.3 Screw Terminal Connections

There are two power connections and two communication connections. The positive voltage terminal must have atleast 5V applied. GND is power ground. RX is the incoming information to the SMEEPROM. The RX port should be connected to a TX port of the microcontroller. TX is the outgoing information from the SMEEPROM. The TX port should be connected to an RX port of the microcontroller. The SPI connections (SCL, SDI, and SDO) are currently unimplemented. All port pins should operate within 3-6V.

### 3.4 DI1 Indicator LED

The red indicator LED (silkscreen designator DI1) is used to communicate the state of the SMEEPROM. When the SMEEPROM is powered, the DI1 LED will blink until a successful command has been sent to and recognized by the SMEEPROM. While not active, the LED will be turned off. During any operation (read, write, edit, status, etc), the LED will turn on until that operation is complete.

### 3.5 JC1 ICSP Port

The JC1 Polarized header is used to program the PIC in-circuit. Any of the popular Olimex Programmers (PG1, PG2C, PG3B, and MCP) are capable of loading new firmware onto the SMEEPROM. We welcome and encourage firmware suggestions and revisions. Build, compile, and burn your own 'kernel' onto the SMEEPROM.

The current version of the SMEEPROM firmware can be found at www.sparkfun.com and all code is open sourced.

### 3.6 Schematics

The SMEEPROM schematics can be found at the appendix portion of this datasheet.

### 3.7 PCB Layout

The PCB is a straight forward two layer design. If you can't tell, we like to use the autorouter. If you need the footprints for your own design, feel free to email us spark@sparkfun.com.

## 4 Smart Media Card Considerations

This section tells you what you need to know to pick the right Smart Media Card for your specific application.

### 4.1 Warning!!

The SMEEPROM does low-level writes to the Smart Media Card. These means that after you have written to the card, the card will NO LONGER BE READABLE in digital cameras, in MP3 players, or anything else you may use your Smart Media Card for. This is because the CIS (Card Information Structure) is destroyed when you record data to the first few blocks of memory. While this is not a permanent alteration, it can be difficult to recover the card. More info on card recovery can be found here: http://www.digit-life.com/articles/smcrestore/.

### 4.2 Voltage Type

Do NOT use the old (pre-2000) version of the Smart Media Card. These cards operate at 5V and will operate with finnicky results. The 5V cards can be identified by a '5V' white marking on the card and having a notch on the top left corner of the card when the metal pads are showing. 3V cards have a notch on the upper right-hand corner.

### 4.3 Card Size

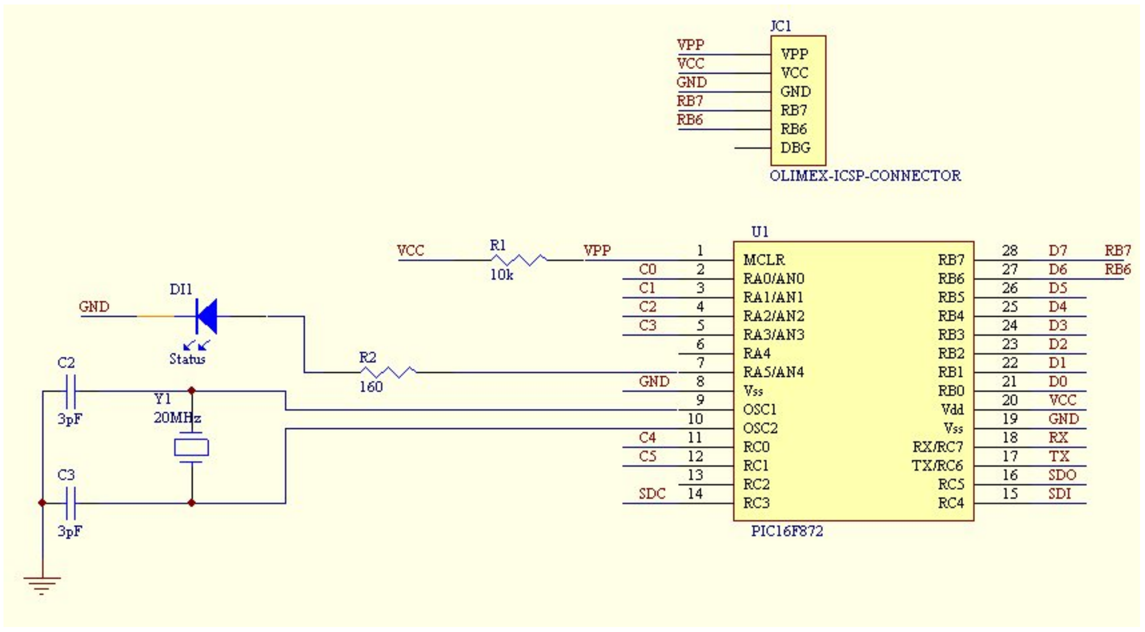
It is very unlikely that we will ever see 256MB Smart Media Cards because the defined standards do not allow for anything past 128MB. The SMEEPROM currently supports 8MB, 16MB, 32MB, 64MB, and 128MB cards.

### 4.4 Card Installation

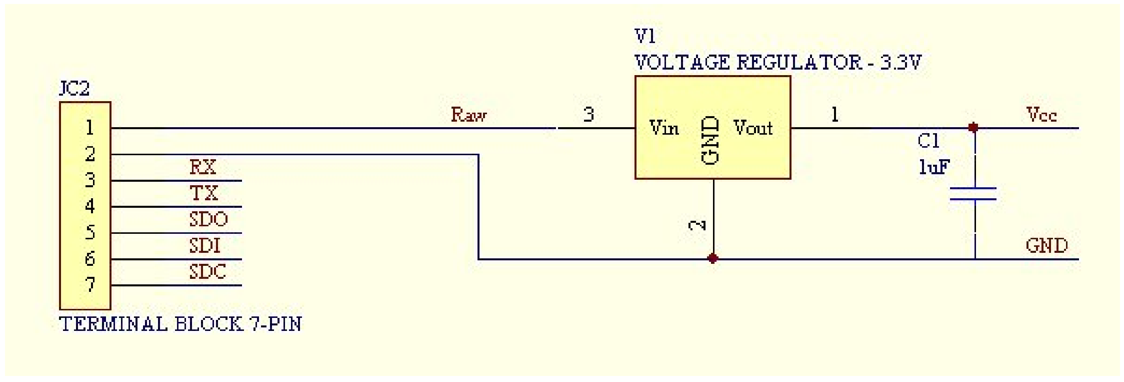
Simply insert a Smart Media Card into the socket with the metal pads towards the PCB. Insert all cards before power-up. Removal of a card after power-up is not a problem. However, re-insertion should only be done after a power-down so that the onboard controller can re-map the available memory locations.

## 5 Schematics

### 5.1 Main



## 5.2 Power Regulation



## 5.3 SMC Socket

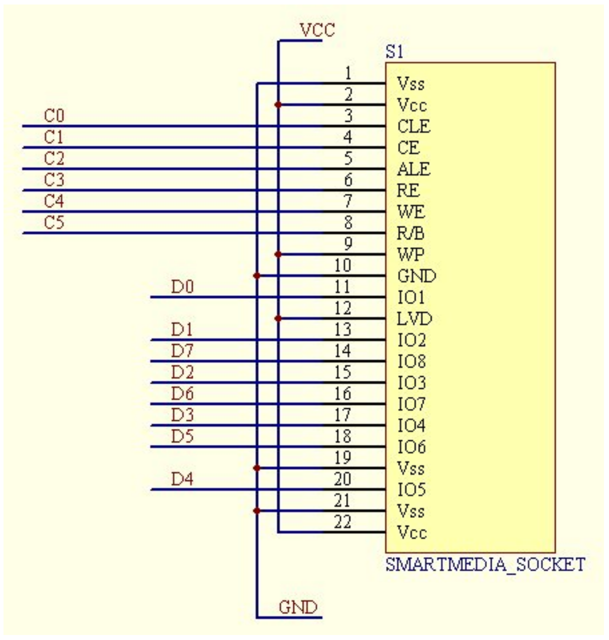

VCC
S1
1 Vss
2 Vcc
3 CLE
4 CE
5 ALE
6 RE
7 WE
8 R/B
9 WP
10 GND
11 IO1
12 LVD
13 IO2
14 IO8
15 IO3
16 IO7
17 IO4
18 IO6
19 Vss
20 IO5
21 Vss
22 Vcc
C0
C1
C2
C3
C4
C5
D0
D1
D7
D2
D6
D3
D5
D4
SMARTMEDIA_SOCKET
GND

Table 1: An example conversation.

| Byte# | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| Bit | 76543210 | 76543210 | 76543210 | 76543210 | 76543210 | 76543210 | 76543210 |
| Function | 11010100 | FFFF–AA | AAAAAAAA | AAAAAAAA | AAAAAAAA | DDDDDDDD | 01001010 |

Table 2: Response from the Next Open Spot command.

| Byte# | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Bit | 76543210 | 76543210 | 76543210 | 76543210 | 76543210 |
| Function | 10001010 | ——AA | AAAAAAAA | AAAAAAAA | AAAAAAAA |