The M25P80 Serial Memory Device

Posted on **April 10, 2012**

This posting will bring together two threads; Non-Volatile Options for parameter storage and the Abstraction of hardware through the use of modular “C” programming techniques.

At the conclusion of the non-volatile options posting, all proposals save one had been rejected. The sole remaining avenue was to store parameter data the the M25P80 SPI Flash Memory device installed in the MMB32 development board. As stated in the that posting, this device is not at all ideal for the task. So let’s take this opportunity to take a closer look at the requirements compared to the device and determine what makes it problematic.

Capacity

The capacity requirements for parameter storage are very modest. Typically a few hundred or at most a few thousand bytes is more than enough for most applications.

(+) The M25P80 exceeds this need by a large margin with a capacity of 1,048,576 bytes, commonly referred to as one mega-byte. This capacity is so large that it opens up the possibility of other potential uses for the device.

Reading

The device should allow arbitrary bytes to be read from the media.

(+) The M25P80 meets this requirement.

Writing

The device should allow arbitrary bytes to be written to the media.

(–) The M25P80 does not allow the writing of data! Instead data must be written using a two step process in which the media is erased to the value 0xFF and then data is “laid down” using the bit-

wise AND of the current media value and the new data. If data is written without an erase, the result is the bit-wise AND of the old and new data.

Granularity

The device should allow operations to take place on arbitrary locations without concern for alignment or other requirements.

(–) The M25P80 does not allow arbitrary locations. Instead:

- Erase commands operate on either sectors of 65.535 bytes or the entire device.
- The bit-wise AND operation may start on any byte, but a sequence of these bytes may not cross a page (256 byte) boundary.

Speed

The device should perform all commands at a reasonable speed compared with the expected data update rate and general user interface requirements (< 100 milliseconds).

(–) The M25P80 execution times may be summarized as:

- Read: quick; as fast as the SPI interface. (< 1 microsecond per byte)
- Bit-wise AND: modest; as slow as 5 milliseconds, 0.8 milliseconds typical.
- Erase: very very slow!; up to 20 *seconds* and only 0.6 *seconds* typical.

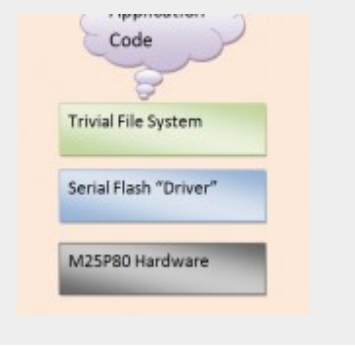
Endurance

The device should allow 100,000 write cycles.

(+) The M25P80 is rated for at least an endurance of 100,000 erase/write cycles and a data retention specification of 20 years at 55°C.

Bridging the Gap

At this point, the next step is to use modular programming and abstraction to bridge the gap between the requirements and the actual device. While the gap is rather larger large, potential “deal breakers” of inadequate capacity or endurance are not issues. Instead, the major areas needing attention are the bit-wise AND “writing” of data, erase granularity, slow erase cycles, and write page boundaries.



— Figure 1

The semantic gap between the application and the hardware is large enough that it cannot be easily bridged by a single abstraction. That's why this design uses two: A low level, device driver abstraction and a high level but very simple, file system that presents applications with an easy to use interface to the the storage device. This is illustrated in figure 1.

The low level driver is a fairly simple piece of code that abstracts out the SPI interface, the M25P80 command set and reworks them into a small set of callable "C" functions. This layer does deal with one issue though; the function that writes a data block automatically handles the cases where the block crosses page boundaries and performs the correct sequence of device commands. Thus at least that gap is closed. The function prototypes are listed below:

The balance of the gap is left to higher layers of software to deal with. These will be examined in an upcoming post on the Trivial File System.

As always, comments are invited and welcome!

Peter Camilleri (aka Squidly Jones)

This entry was posted in **Embedded Development** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=784]** .