Non-Volatile Options

Posted on **April 7, 2012**

Don't worry. I am not about to branch into offering investment advice; Especially not the dubious world of the derivatives marketplace. I am instead looking at one of the unresolved issues in my Touch Screen Calibration series; Namely, where do the parameters go when we've gathered them? They need to be stored in some sort of non-volatile storage. What are the ideal characteristics of this storage and what do I have on hand in my Mikroe MMB32 development board? Ideally, the storage:

- is non-volatile so that data is not lost when power is removed.
- is permanently attached so that system parameters are always available.
- supports atomic update so that parameters cannot be half written.
- is simple to program so that a lot of program storage is not wasted.

Plan A

Many microcontrollers have on chip non-volatile resources. A wide selection of PIC chips have on board EEPROM that is an excellent choice for this sort of parameter data. Regrettably the PIC32MX460F512L used in my development board has no on chip EEPROM storage, so this easy option is not available. Still, the part has flash memory that could be pressed into service as EEPROM emulation. I did not look into this option too deeply at this time due to the complexity of the software needed and some doubts as to how it would affect real-time performance by turning off interrupts for long periods of time. I may revisit this at a later time.

Plan B

If your microcontroller lacks non-volatile storage, an easy, inexpensive option is to utilize a low cost serial EEPROM device. They are is easy to use and come in I2C bus and SPI bus versions. This is a great idea, except that Mikroe didn't put one on the development board I'm using. I could easily add an EEPROM but that would be cheating! A sub-variant of this plan are all the modern FRAM and MRAM non-volatile serial RAM devices that are on the market. These offer greater write speed and higher write endurance than EEPROM at a premium price but are still cheating! I wanted to only use resources already on board!

Plan C

Another approach is to use a battery backed up calendar/clock with ram. Indeed the PIC32MX460F512L chip has an RTCC module and a plethora of power savings modes. The spec sheet has over 60 line items detailing power consumption so this is not a simple matter. In fact getting the chip into a low power mode is very complex and difficult. Further the PIC32 is not a member of Microchip's XLP family of low power chips designed for battery powered operation, so the low power modes are not that low. Finally the back-light control circuit for the MMB32 display consumes about 5ma of current even when the display is off. This would drain any battery dead in short order so this option won't work either.

Plan D

OK, if the RTCC in the PIC is no help to us, we can add an external RTCC/RAM module. There are many to choose from. Mikroe makes a very nice RTC PROTO Board that has it's own battery, RTCC and 240 bytes of non-volatile RAM. Perfect, and since it is separate from the PIC32, life is a lot easier code wise since the switch-over to low power mode is handled by the RTCC boards hardware and I shouldn't have to worry about power thief circuits. This too is cheating, but I purchased one of the Mikroe boards so I will be revisiting this option again at a later time.

Plan E

OK, so enough of what the MMB32 does not have. What about some non-volatile resources that it does have? Well there are two. The first is an SD memory card slot. This allows access to up to 2 giga-bytes of storage, which is staggering. The downsides are that this is removable storage, so the user can easily remove the memory device with all the essential parameters in it. In order to utilize an SD card, a low level driver is required, and a FAT32 file system to organize the data on that card. That is a lot off complex code that will be difficult to use and will devour resources. Now if a large file system is already needed, the incremental expense is minor to store parameters there. However if you were not already planning in that direction, it is a huge waste of flash memory.

Plan F

OK we are at the end, and still no closer to our answer. What's left on our MMB32? Well there is a M25P80 SPI flash memory. It is NOT an EEPROM, it's not all that flexible, and it's not an easy chip to use well. Why do I say that? Yes, the full megabyte capacity is great and the SPI interface is simple and straight forward, but while data can be written (well it's not a write so much as a bit-wise AND operation) one byte at a time, the device needs to be erased before any space can be overwritten. The bad news? The smallest unit that can be erased is an enormous 64K bytes and the erase time is up to 3 seconds! There is a command to erase the whole device but that can take up to 20 seconds! To use this device we need to create for it a file system that is small, simple and works well on this very primitive storage system. That file system is called the Trivial File System and is the subject of a my next posting on this topic.

Peter Camilleri (aka Squidly Jones)

SHARE THIS:

Print LinkedIn Facebook Twitter

LIKE THIS:

Like

Be the first to like this.

This entry was posted in **Embedded Development** by **Peter Camilleri**. Bookmark the **permalink [http://teuthida-technologies.com/?p=606]** .