Underwater Arduino Data Loggers

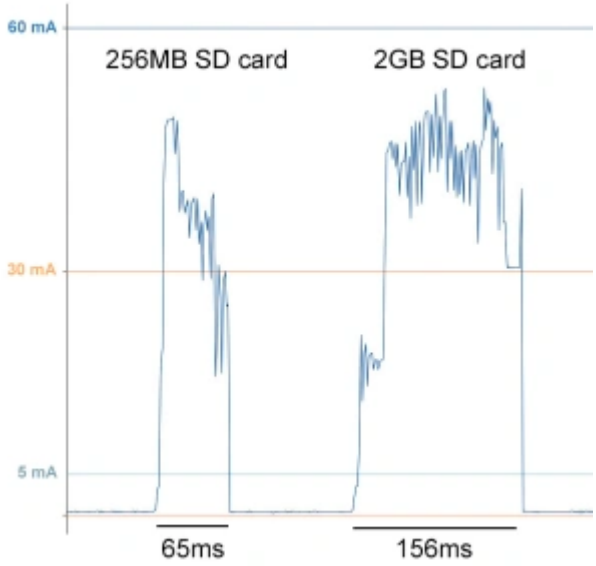
Cutting Power to Secure Digital Media cards for Low Current Data Logging

The tweaks I discuss in the power optimization post bring sleep current on a typical build into the 0.1-0.12mA range; leaving the sleeping µSD cards as the largest remaining power consumer on the Cave Pearl loggers. And those cards have been a burr under my saddle for quite some time now, as they are probably responsible for most (if not all..) of the power consumption irregularities that were showing up in some of the battery voltage logs.

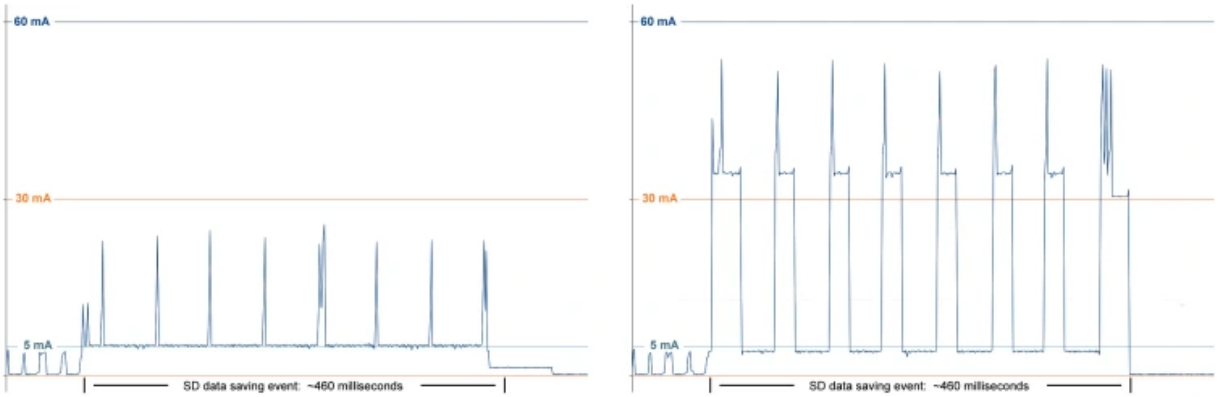
I already knew the various brands of secure digital media (SD) cards could have dramatically different sleep current, but a comment in the Arduino.cc forum by William Greiman (the author of the SdFat library for Arduino) made me look a little deeper at what they were doing the rest of the time:

"Performance of microSD cards is not easy to predict on Arduino. Arduino uses SPI to access the cards and the SD library has a single 512 byte cache. Modern SD cards are designed to be used on the high speed 4-bit SDIO bus with very large (16 KB or larger) multi-block writes and reads so they must emulate the single block access that Arduino libraries use. This can mean much internal data movement, erasing of flash, and rewriting of data."

This shows up clearly during data save events:



This composite graph compares logger current during identical sd.begin(), File Open & File Close sequences with two different Sandisk brand SD cards on the same datalogger. The 256MB card used less than 3mAs, while the 2GB card burned more than twice as much power during this initialization. Older low cost/size cards often perform better in SPI mode, which is simply an after-thought for high end cards, because it's required by the SD spec. Other file open/close artifacts can occur. Write methods are also a factor.



*These screen captures of IDE serial potter output show current drawn during a data writing event with 256mB (left) and 2GB (right) Sandisk SD cards. 4kB of CSV format ASCII data was saved, and the gaps between the writing spikes were caused by i2c coms while SDfat's cache was filled with data retrieved from the EEprom on the RTC module. It looks like the 2GB card has to do a great deal of shuffling to accomodate the 512byte blocks coming from the SDfat library. (**Note:** see our article in Sensors for more details about the the code behind this logging event)*

After seeing that I tested a variety of different SD cards, finding that save event power use increased by more than 5x over the range from old 64 & 128mb Nokias to newer 2 & 4Gb Sandisk cards. There is no guarantee that any given brand's controller will handle SPI coms gracefully, and newer cards from top manufactures often have bad SPI performance since they are are not expecting anyone to use high capacity cards with that bus.

It took me a good while to realize that I had fallen into yet another forest-for-the-trees situation, because even the worst offenders were only using ~30 mAs per save event, but all the cards were delivering similar sleep currents. A day has 86,400 seconds in it, so the best sleepers, coming in around 70µA, were still burning *six thousand milliamp seconds per day overall…*

That brought me back to the question of de-powering those SD cards. I had been discouraged from trying this *early* in the project by some of Grieman's other forum remarks where he suggested that there was nothing in the default SD library to support multiple shut downs & restarts safely. But over time I found that Nick Gammon, and several others had SDcard power control working with SdFat, and seeing the folks at OSBSS claim they had power cycled SD cards more than a hundred thousand times, convince me to give it a shot.

As I had no logic level P-channel fets lying around I went with a garden variety 2n2222 BJT, configured as a ground side switch with a 30k pulldown. Driving it to saturation using a 330Ω base resistor (assuming Hfe = 30) should give me enough wiggle room to handle spikes up to 150mA, though it will burn another 10mA to keep the BJT on for the full second I needed to wait before pulling the plug. Write latencies for SD cards can be quite large, and some cards have more than one stage of sleep, drawing around 1.0 ma for maybe a second before entering deep sleep. But with 6000 mAs/day on the other side of the scale, I could afford the extravagance.

The cross leakage stuff I'd seen on EEVblog convinced me that I needed to actively pull up all of the SPI pins after the ground was disconnected. I cobbled together a set of ON/OFF functions with pinmode commands, but it did not work reliably until I switched over to port manipulation (like they did at OSBSS), so the lines were all pulled simultaneously. I was already disabling peripherals like the ADC with the PRR register, but that was just to save a little runtime power. Now it was required because when SPI is active, it controls MISO,MOSI & SCLK. So you must shutdown the SPI interface before you can set those pins directly.

```
#include <LowPower.h>
#include <avr/power.h>
#include <SPI.h>
#include <SdFat.h>
SdFat sd;                          // Create the objects to talk to the SD card
SdFile file;
const byte slaveSelect = 10;   // sd card slave select on pin D10
#define SDpowerPin 9           // pin controlling the BJT on the ground line for the SD card
boolean SDcardOn = true;     // flag for error routines
byte  keep_SPCR;

void setup () {
keep_SPCR=SPCR;               // save the default SPCR register contents
// the following pullup steps are all optional:
// turn on internal pullups for three SPI lines to help 'some' SD cards go to sleep faster
pinMode(slaveSelect, OUTPUT);
digitalWrite(slaveSelect, HIGH); //pullup the CS pin on the SD card (but only if you don't already have a hardware pullup on your module)
pinMode(11, OUTPUT);
digitalWrite(11, HIGH); //pullup the MOSI pin on the SD card
pinMode(12,INPUT_PULLUP);//pullup the MISO pin on the SD card
pinMode(13, OUTPUT);
digitalWrite(13, LOW); //pull DOWN the 13scl pin on the SD card (IDLES LOW IN MODE0)
// NOTE: In Mode (0), the SPI interface holds the CLK line low when the bus is inactive, so DO NOT put a pullup on it.
// NOTE: when the SPI interface is active, digitalWrite() cannot effect MISO,MOSI,CS or CLK. . . }

void turnOnSDcard()
 {
pinMode(SDpowerPin, OUTPUT); digitalWrite(SDpowerPin, HIGH); //turn on the BJT on SD ground line
delay(6);                                  // let the card settle
// some cards will fail on power-up unless SS is pulled up  ( &  D0/MISO as well? )
DDRB = DDRB | (1<<DDB5) | (1<<DDB3) | (1<<DDB2); // set SCLK(D13), MOSI(D11) & SS(D10) as OUTPUT
// Note: | is an OR operation so  the other pins stay as they were.              (MISO stays as INPUT)
PORTB = PORTB & ~(1<<DDB5);  // disable pin 13 SCLK pull-up – leave pull-up in place on the other 3 lines
power_spi_enable();                  // enable the SPI clock
SPCR=keep_SPCR;                       // enable SPI peripheral
delay(10);  SDcardOn = true;       // just a flag // may need to extend this delay with some cards
}
```

```
void turnOffSDcard()
 {
delay(6);
SPCR = 0;                                // disable SPI
power_spi_disable();                     // disable SPI clock
DDRB &= ~((1<<DDB5) | (1<<DDB4) | (1<<DDB3) | (1<<DDB2));   // set All SPI pins to INPUT
PORTB |= ((1<<DDB5) | (1<<DDB4) | (1<<DDB3) | (1<<DDB2));    // set ALL SPI pins HIGH (~30k pullup)
// Note: you must disconnect the LED on pin 13 or you'll bleed current through the limit resistor
LowPower.powerDown(SLEEP_1S, ADC_OFF, BOD_OFF); // wait 1 sec for internal SDcard housekeeping
delay(6);
pinMode(SDpowerPin, OUTPUT); digitalWrite(SDpowerPin, LOW);  //turn off BJT controlling the ground line
delay(6); SDcardOn = false;
}
```

These two functions book-end any code that needs to write data to the SD cards:

```
turnOnSDcard();    flushEEpromBuffer();    turnOffSDcard();
```

That SD card data saving function starts by checking the main battery to make sure there is enough power to save the data without a brown-out, and then *re-initializes* the card with *sd.begin* before opening any files:

```
vBat = readBattery();   // This function shuts down the logger if the main battery is below 3.65V

if (!sd.begin(chipSelect, SPI_FULL_SPEED)) {
Serial.println(F("Could NOT initialize SD Card"));Serial.flush();
error();  // note: the error event includes:  if (SDcardOn) {turnOffSDcard();}    inside the function
}
delay(10);
file.open(FileName, O_WRITE | O_APPEND);  //see this post by Grieman

//...save your stuff...

file.close();
```

Looking at the datasheets for the Mic5205, or the Mcp1700, you see the **regulator dropouts can reach 300mV at 100mA+ currents** you could see during SD card initialization, so your input cutoff for a 3.3V system needs to be above 3.65V to handle the load. After the data is saved it is *critical that all open files are closed properly* before the turnOffSDcard function gets called, otherwise your data will be lost. The graphs tell me that a full one second delay before powering down the card is probably longer than it needs to be, but in data logger applications it's pays to err on the side of caution. According to Greiman:

"The standard says reliably removing power is not supported in SPI mode. It does suggest that you can remove power **one second after** the card goes not busy but does not guarantee this will work. You can't depend on isBusy() to power down a card. It only means the card can accept a command. It may still be programming flash or moving data for wear-leveling. You really need the one second delay after not busy."

Of course, it was pretty flakey the first few times I tried it. Half of the loggers worked, but the other half were restarting every time there was a data save event (killing off SD cards in the process...) This problem affected every logger built around the Rocket Scream Ultra, which has been one of my favorite small form factor boards. Closer examination of the two-penny clones that *were* working ok revealed that they had 10µF tantalum capacitors beside the voltage regulator rather than the little 1µFs beside the Ultra's MCP1700. So those cards were hitting the rail pretty hard when the SD ground line was re-connected, and this caused brief transients that were low enough to restart the processor on half of my units. Some add a small (33Ω) resistor in series to limit these inrush currents, but I found that adding 2-3 10µF (106) ceramics to buffer that spike got them all working ok, and for field deployment units I'll probably add more.

Lately I've been using these 60¢ SD adapters, and removing the bottom three 10k smds that these boards have on the SCLK, MOSI & MISO lines. (the other resistor keeps the 'RSV' pins from floating) Having a pullup on the clock line wasted power during mode o sleep as the clock idles low, but now that I'm cutting power rather than just sleeping the SD cards, I could leave those resistors in place...then I wouldn't need to pull up those lines in the code, (though I'd still have to pull SS...) Thing is, Grieman says SPI should not have pull-ups or pull-downs on MISO, MOSI or SCK. So I'll stick with the 328's internal 25k pulls for now, because doing it in code is reversible. But that does leave some pins floating during the boot process.

I set a several units running on the bookshelf, with a rapid six second sampling interval. A couple of weeks later they were all still going, with some of them seeing more than 30,000 SD card power cycles without error. Given that the loggers normally see less than one save event *per day*, I'm calling that a successful test. If you run into issues, the first thing to try is extending that delay after sd.begin() and adding a few more delays throughout your functions. If you look at the spec you find that SD cards are allowed to take huge amounts of time for *everything* from initialization, to file open/close. While I did not see that in cards I used for my tests, these latencies are 'officially' allowed to stretch well beyond 100ms.

With both pin-powering on the RTC, and ground line switching on the SD card, the loggers get down to between 0.03-4mA between samples, which should push my operating lifespan into multi-year territory. Or, if I'm really lucky, they'll make it through *one* winter-time deployment in Canada 🙂

I was also pleased to discover that the On/Off code seems to work on loggers that do not have the ground side switch installed *provided I do not try to re-initialize the cards with sd.begin*. SPI shutdown & line

pullup seems to cause the SD cards to enter sleep mode more quickly than they did before, and I have not seen any current leakage. So hopefully I won't have to maintain vastly different code versions for older non-switched loggers. (Update 2017-06-12: Further tests of SPI shutdown, *without* the BJT to disconnect power from the SD card have not been reliable. Some worked, some didn't. When I figure out why that is I will post an update)

Addendum 2017-06-06

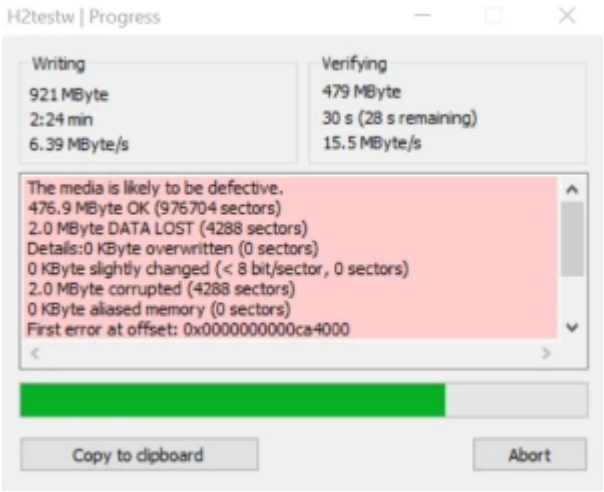
A commenter over at Dangerous Prototypes made a point about my use of the 2n2222 which is important enough that I should pass it on:

"I'm surprised he didn't check the 2N2222. Look at its data sheet, the V(CE) performance is not great. Take 0.3V at 100mA, then the SD card would have been actually running at 3.0V, right at the -10% VCC rating edge. I'm surprised the problems are not worse. Of course it would be extremely sensitive to VCC sag..."

The drop across the collector-emitter was something I had simply missed, and I still struggle to read those datasheet graphs properly. And I was so used to seeing card operating voltage specified between 2.7-3.6v, that I also missed the fact that *in SPI mode*, only 3.3v is officially supported. The net result is that I'm probably sailing closer to the wind here than I realized, and I'm going to call this technique "experimental" until I see real-world deployments saving more than a year of data safely. And if I stay with ground-side switching in future, I will start looking for a good logic level N-channel MOSFET, with low on resistance, to replace that BJT. The Supertex TN0702 looks like a good option with the promini's with 3.3v logic.

Addendum 2017-06-06

Just thought I should post a reminder to test your SD cards thoroughly before embarking on SD power shut down experiments. I use SD formatter v4.0 & H2testw. An occasional check with H2testw after deployment is also a good way to make sure that you are not damaging your cards over time...



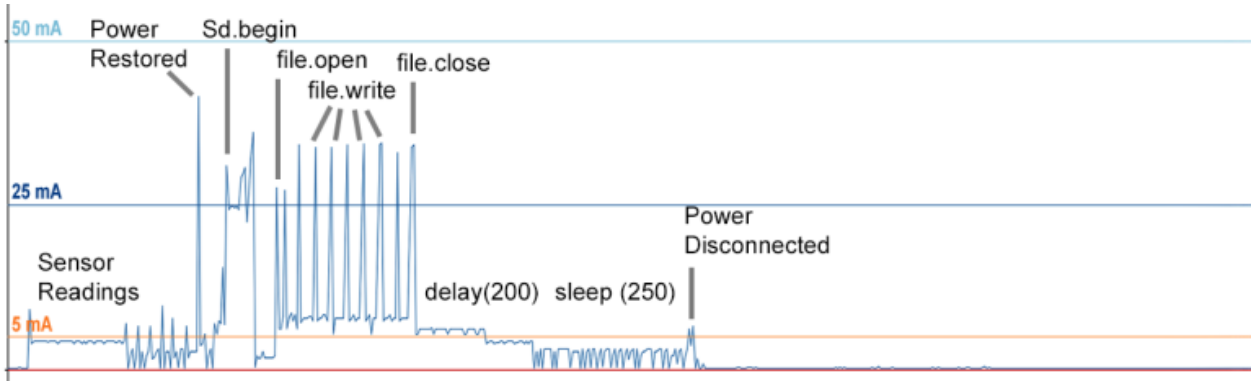
This card gets thrown straight into the garbage.

Addendum 2018-03-22

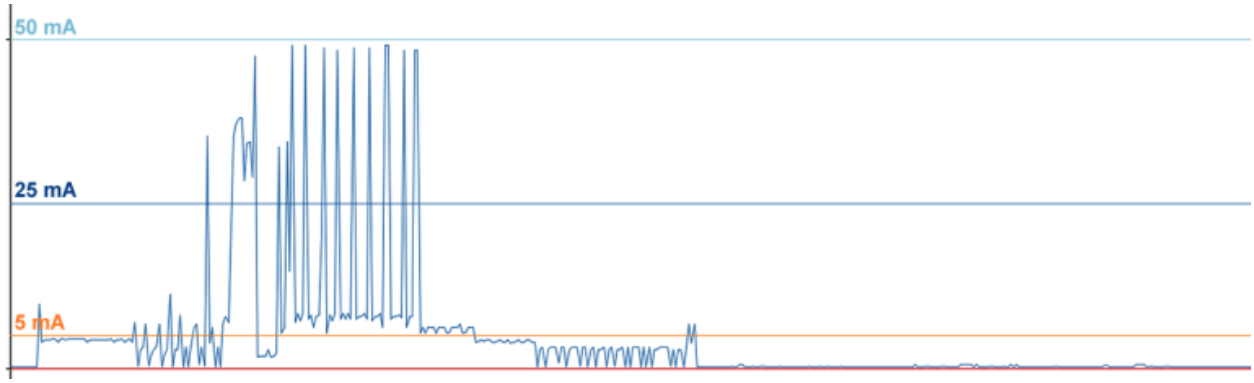
I've been doing more experimenting with de-powering the SD cards during sleep. It works great *most* of the time, but I was still seeing a few frustrating re-boots on some loggers when the SD power was restored. To get

to the bottom of this I set up a unit repeating a standard SD data-save cycle and looked at the current through a shunt resistor with my Arduino DAQ. The results made it pretty clear what was happening: (click image for larger versions)

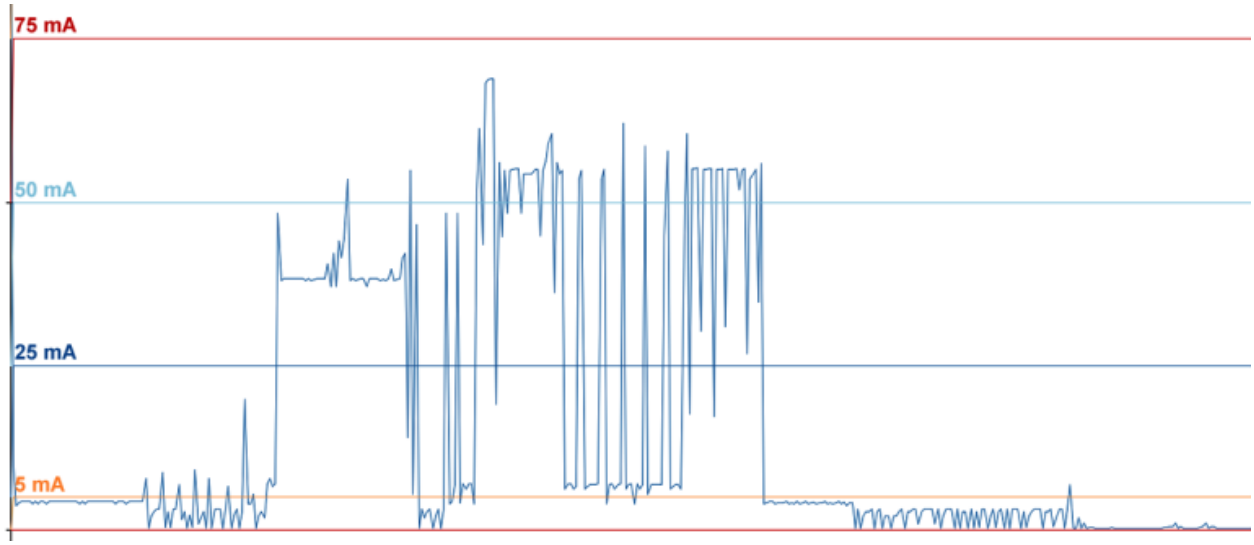
Kingmax 128mb (40mA peaks, 25mA sustained ~60ms, best performance of all cards tested)



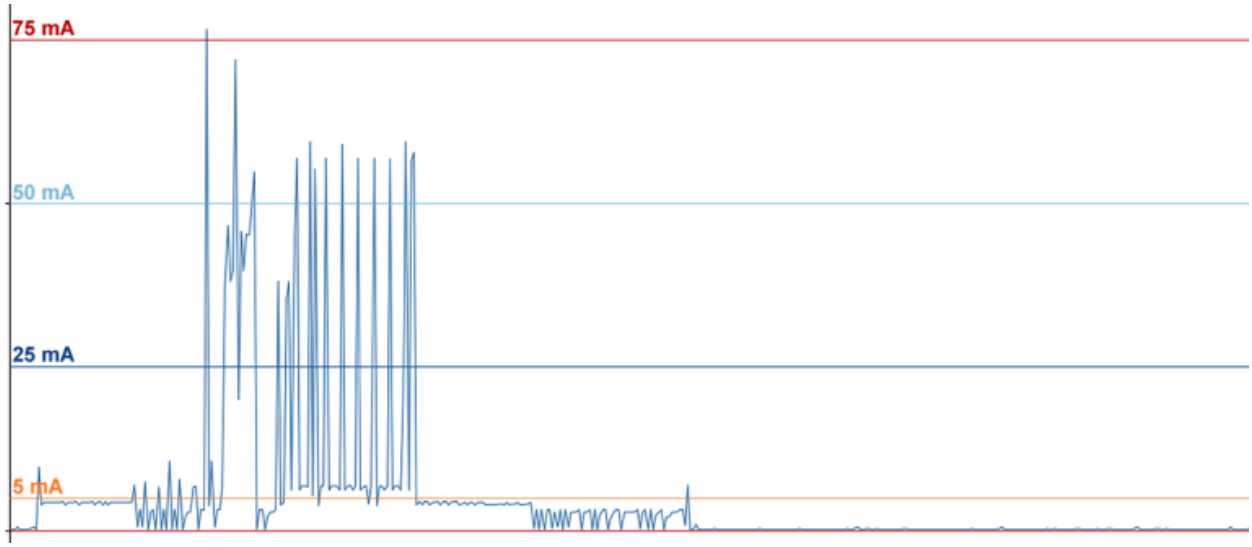
Nokia 256mb (50mA peaks, 35mA sustained ~40ms)



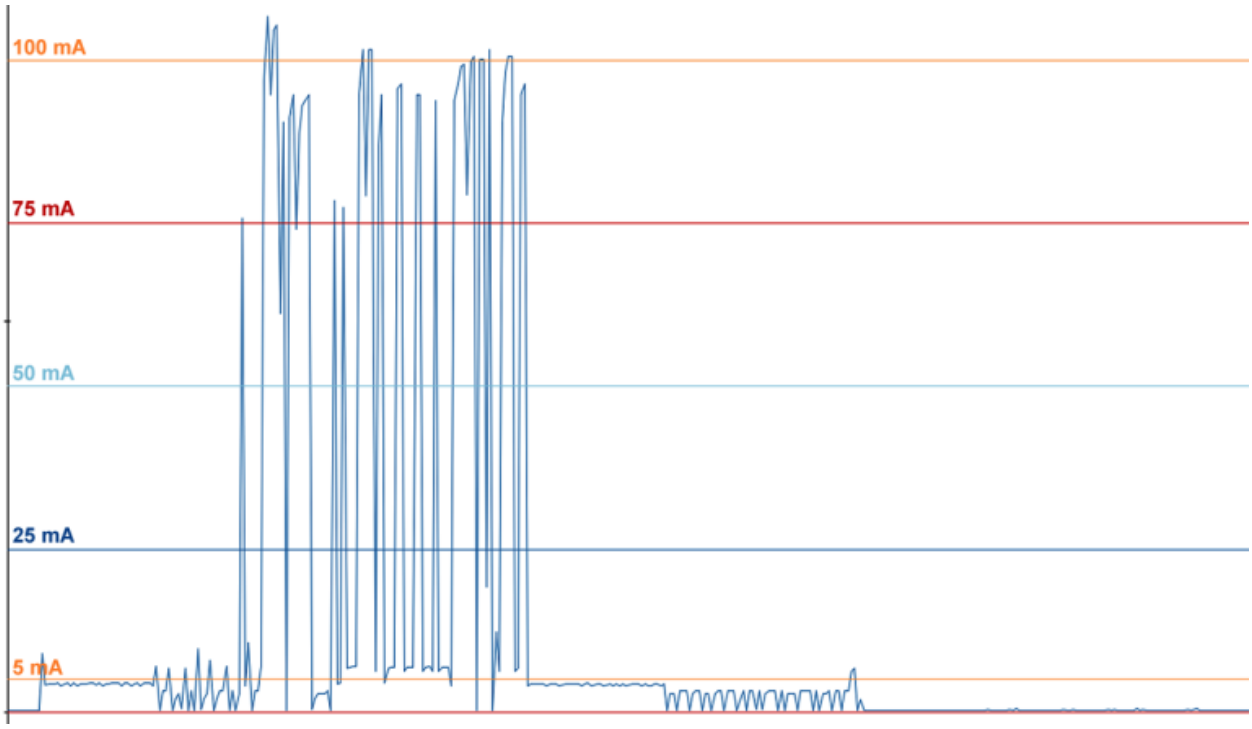
Nokia 128mb (65mA peak, 35mA sustained ~200ms and 55 mA sustained 2x 100ms)
[these artifacts occurred during every single save, and on the other Nokia 128mb cards tested]



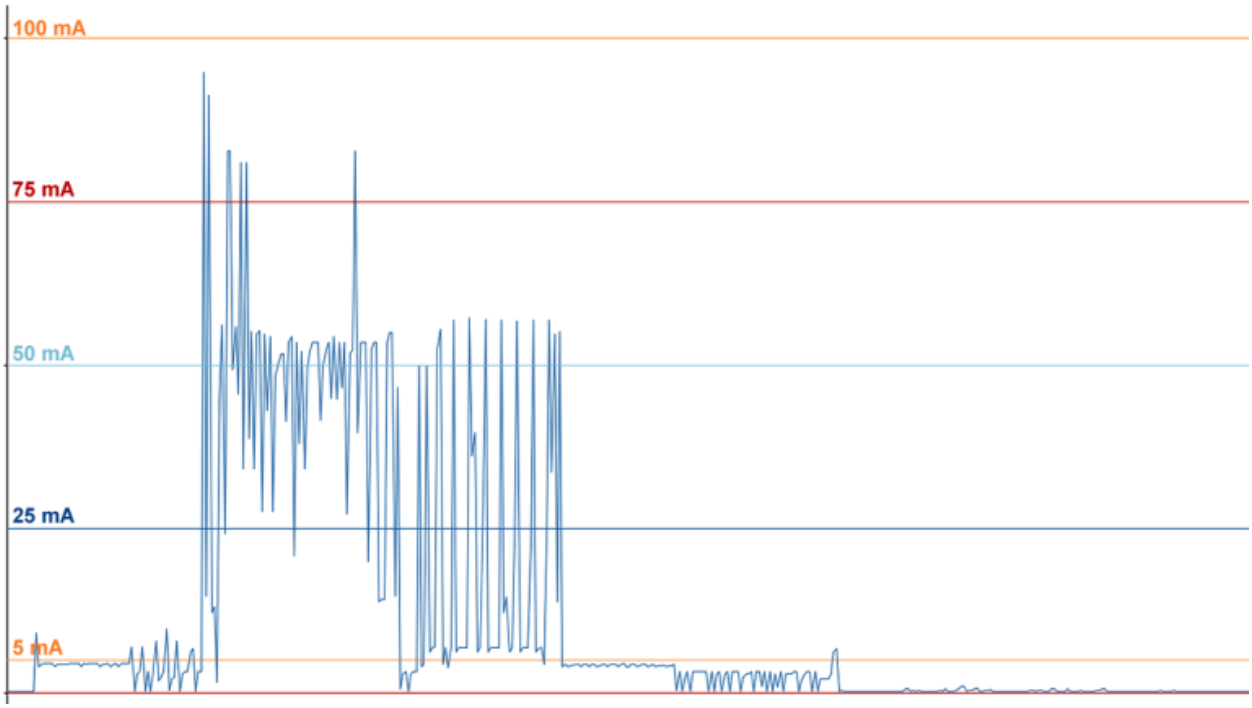
Sandisk 512mb (75mA peaks, 50mA sustained ~60ms)



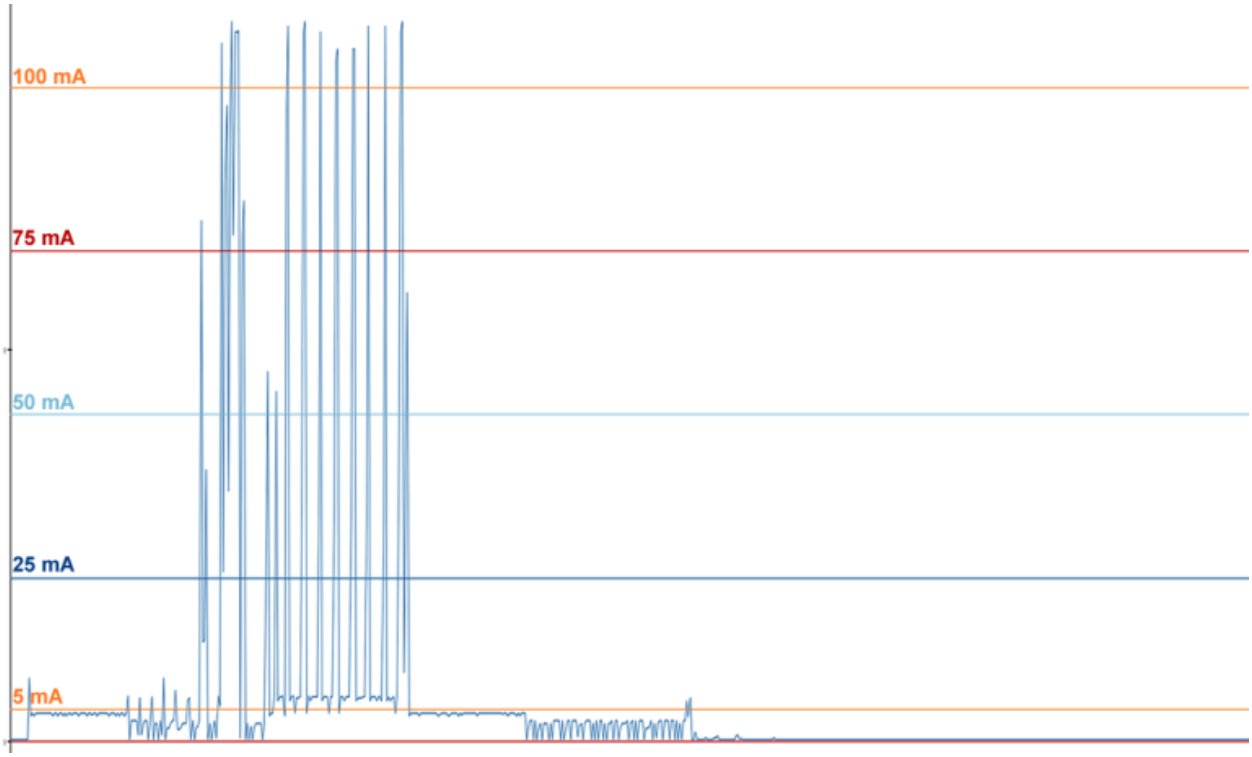
Sandisk TRANSflash 128mb (150mA peaks, 90mA sustained ~80ms)



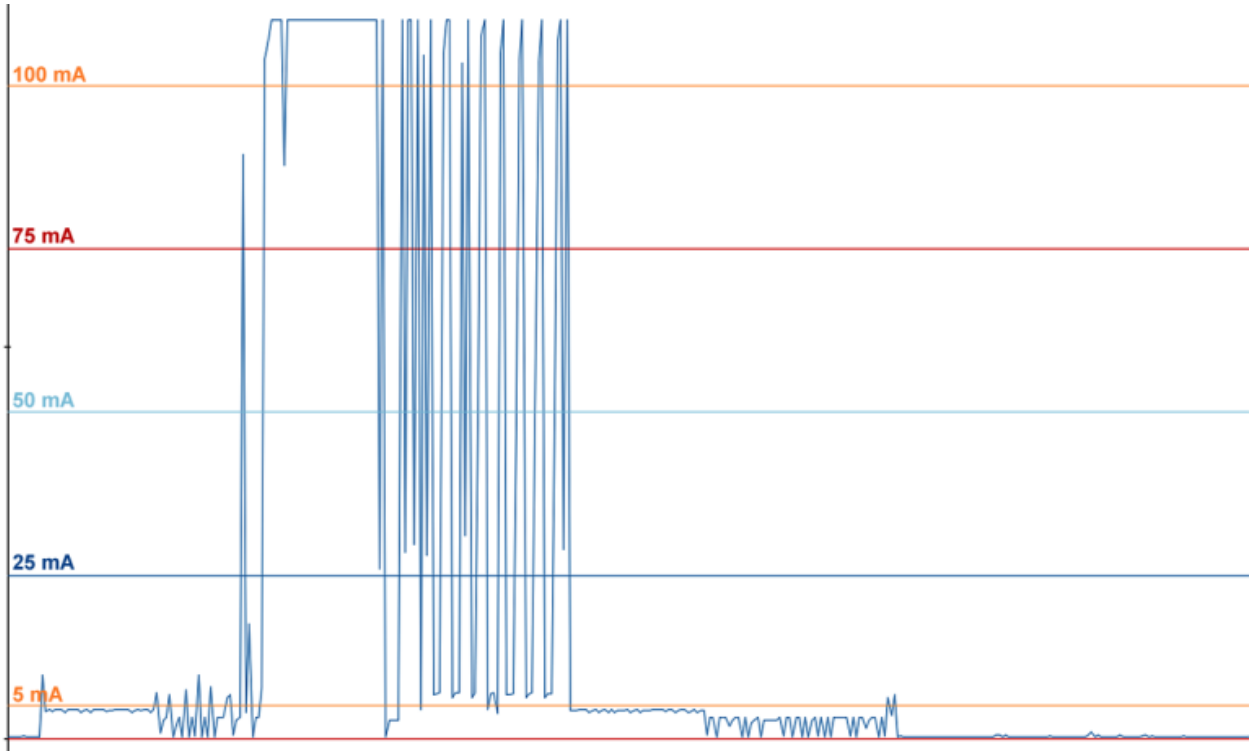
SanDisk 2Gb (90mA peak, 55mA sustained ~350ms)



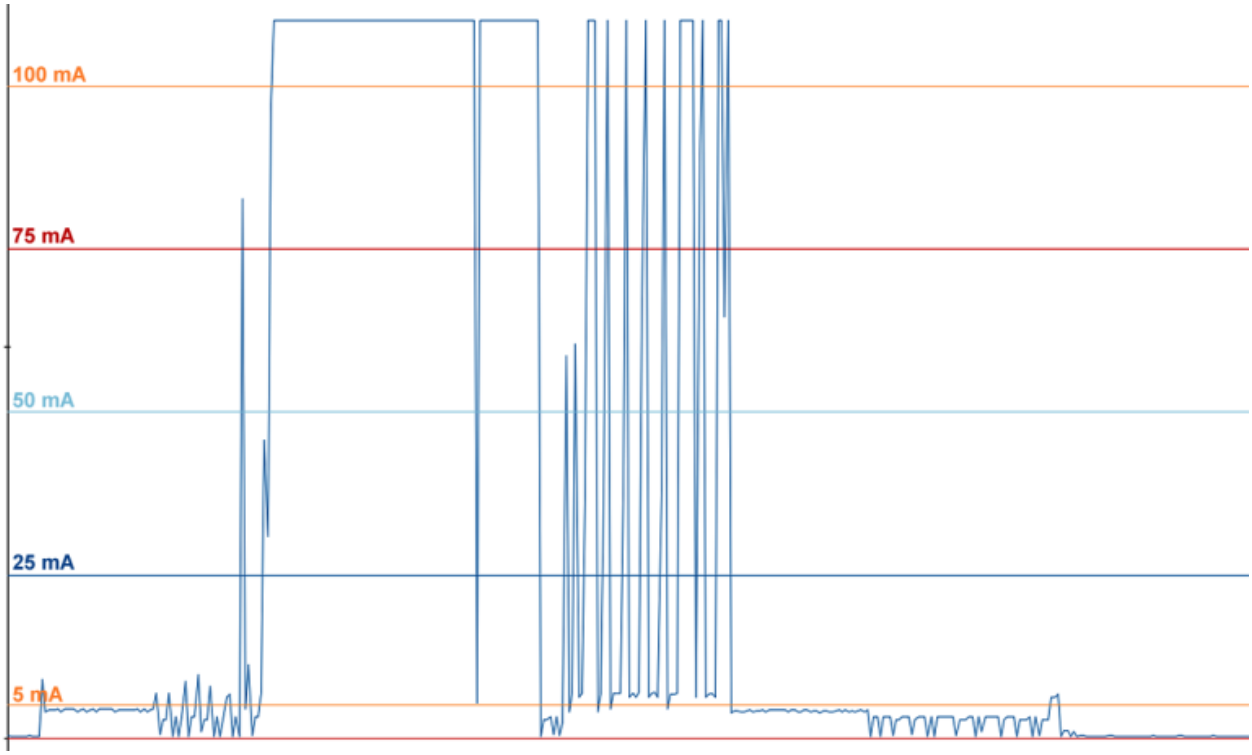
NONAME 1Gb (110mA peaks, 90mA sustained ~45ms)



SanDisk 32Gb Ultra (110mA peaks, >110mA sustained ~200ms)



MUVEmusic 1Gb(+3Gb) (>110mA peaks, *>110mA sustained ~400 ms!!!*)



All these tests were done with the same code, on the same logger – only the cards were changed. All of the cards shown above were successfully saving the data to the CSV file. I was using a fairly large 10 ohm shunt, with the internal 1.1v as Aref so those clipping plateaus at ~110 mA were a limit of my method. The SD spec says cards can actually draw up to 200mA during initialization events, and I suspect those last two get up to at least 150mA.

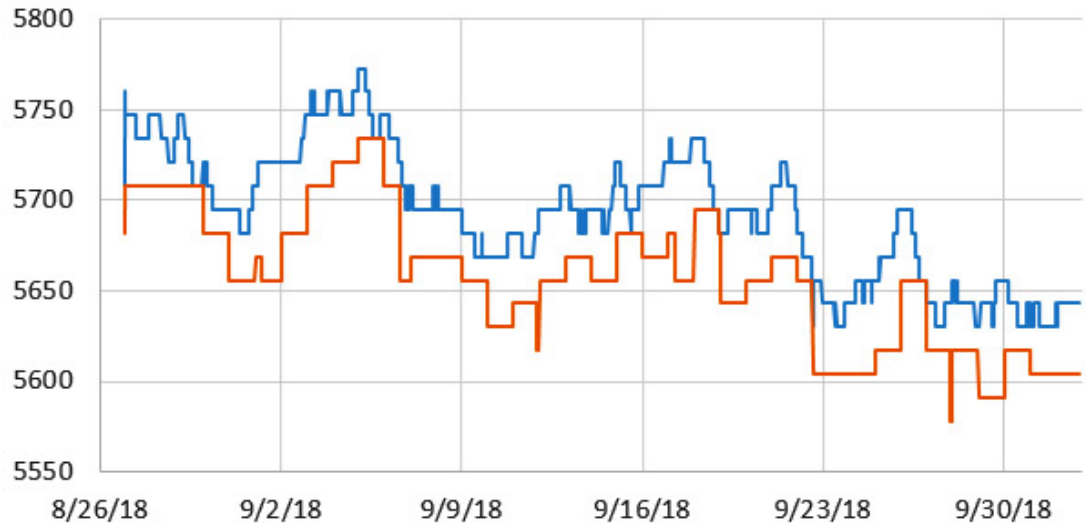
Given the huge difference between the peak currents, and the size of the sustained power loads, it's not surprising that some of the loggers were suffering from brown-out restarts. A few caps could buffer the short spikes, but those larger sustained loads were too much for the MCP1700's I'm using. Another thing that's important to note here is that (with the exception of the 32Gb) these cards were selected from a batch that I had already tested for low sleep currents. So it looks like I'll have to retest all cards that are destined for the low-power logger deployments that de-power the SD cards. Generally speaking it's still better to stick to the older 256mb cards, though some of those have strange housekeeping events at every data save. It's all just a reminder that SD memory is actually more complex than the Arduino since the card itself may contain a 32 bit arm core.

Addendum 2018-10-03

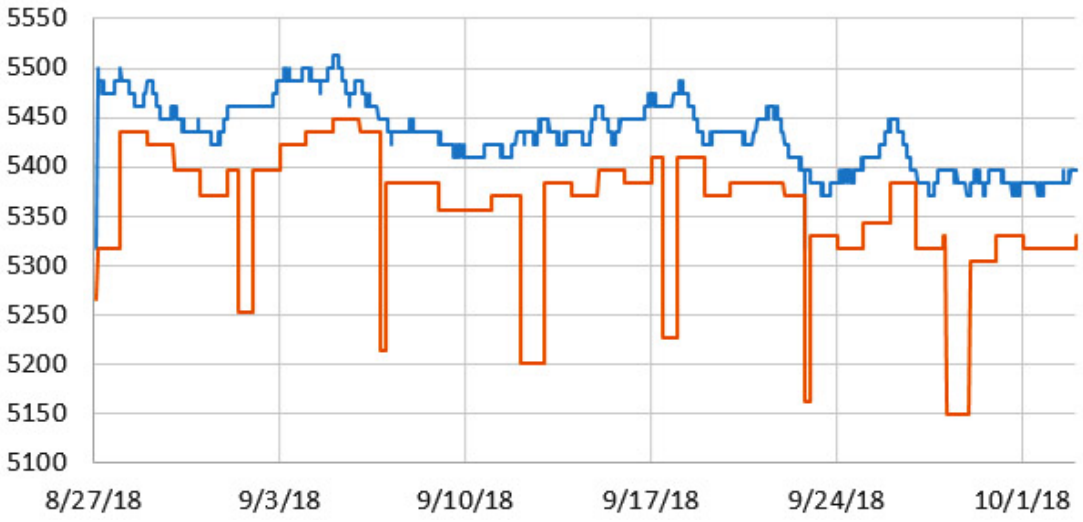
I've a new crop of identical drip loggers running through pre-deployment testing with the same code. Since these units were de-powering the SD cards between saves, I didn't worry too much about which cards I put in the units figuring that a few intermittent power loads would not affect the operating lifespan very much.

In the following records I read the battery voltage every 15 minutes (the blue line) and also track the "lowest" battery reading during logger events like SDcard saves (the orange lines) with small daily saves, and a big data transfer event about every 5 days. The 15 minute record is quite variable as the AA alkaline battery voltage responds to the ambient temperature in the room.

This first graph is from a logger with a Nokia 256mb SD card, and the lowest main battery readings hold within 50mv of the typical readings:



and this record is from a logger with the generic 1GB sd card shown in the earlier graphs:



The width of those drops is an artifact from the fact that "lowest reading per day" variable only gets reset at midnight. Since there's a 5-10ms cap stabilizing delay in my ADC reading function, and the SD code is blocking during the high-drain card initializations, it's likely that the maximum voltage drop on those cells was larger than the 250mv that showed up in this record. Any main battery reading that approaches the main regulators minimum input voltage sends the loggers into a controlled shutdown, so those dips could result in a significant amount of missed run-time by triggering the shutdown too early. If you can't get your hands on those old Nokia cards via eBay, I'd suggest you use lithium batteries to avoid this voltage droop issue when using larger, newer, micro SD cards. The tests I did with lithium cells showed virtually no SD writing drops no matter which card was in the unit, and this effect will no doubt be amplified when deployment temperatures approach zero degrees Celsius.

Addendum 2020-04-11

Wanted to add a note that on our recent builds with no regulators – running directly from 2x LITHIUM AA batteries – we have not yet been able to detect any voltage dip on the main battery during the big transfer of data stored in the eeprom buffer, out to the SD cards. And that's with no buffering capacitors other than those already on the ProMini module. This is notable because most of our MCP1700 regulated builds see a 200-300 mv drop on the main cells with Alkaline batteries. Also tested those same regulated builds with lithium batteries and the SD writing dip went away – so it's definitely the alkaline chemistry struggling to keep up with the speed of the pulse load generated by the cards.

Also, here's a link to someone testing a batch of new large-size cards. Some with current draws significantly larger than the old 256 & 512mb Nokia cards we use.

Addendum 2020-12-20 Ahhh the IEEE . . . Better late than never, Eh?

Optimising SD Saving Events to Maximise Battery Lifetime for Arduino™/Atmega328P Loggers

"The exact power consumption of an Arduino/SD card during saving events is analysed for the first time..."

Ummm…really? And with such pithy gems as buffering to SRAM to reduce SD writing events? Never would have though of that.

When you run power tests on several different SD cards you discovered the real problems: There is enormous variability between different brand/type cards wrt both the power and the time they need to initialize. And then theres what I call 'super housekeeping events' which are significantly longer than power-up initializations. These are hard to capture because they get triggered at different points depending on the cards internal code and how this interacts with the previous power and usage pattern: but the SD card essentially turns into a solid block of maximum power draw for several hundred microseconds. This is why we say our loggers can only sample at a max rate of 1Hz – because you never know when you are going to hit one of those blocking events and hang your logger. If you monitor your battery voltage closely you will see the unmistakable after-effect of these high drain events in the record periodically.

Also, some cards are ok with SPI access, and some cards absolutely are not cool with it. When you use a card that does not like SPI access it will still "work' but each save triggers massive internal memory juggling no matter how much data you actually write to the card. Again – essentially a solid wall of current. So far the best performers with our loggers seem to be older 256Mb to 1Gb Sandisk or Nokia cards because it was not unusual for that generation of cards to be accessed in SPI mode.

Even **our basic student logger** only pulls ~250µA while sleeping *with an SD card*. Clipping Vcc on the RTC typically gets that down to ~150µA and a *good* SD card will get you below 100µA for the entire logger. So I can't say they did a very good job on the power optimization before adding the mosfet. When you implement SD power switching a ProMini logger gets below 30µA, with some getting below 20µA if your sensors have low current sleep modes. At that point the main problem with your logger is that alkaline batteries 'age out' after a couple of years in service (due to pressure or thermal cycling?) and start leaking long before you've used even half their rated capacity. But I've no doubt someone will publish that 'for the first time' in another IEEE paper… in about 5 years.

And don't set all your logger pins to OUTPUT as the recommend in the paper: the power saved is negligible and INPUT mode protects against accidental shorts.

This entry was posted in Reducing power consumption on May 21, 2017 [https://thecavepearlproject.org/2017/05/21/switching-off-sd-cards-for-low-power-data-logging/] .

---

19 thoughts on "Cutting Power to Secure Digital Media cards for Low Current Data Logging"

zoomxx
May 22, 2017 at 2:35 am

Great post, it's long time that I am thinking about turning off the SD and have read about the problems about that. In this way you don't need anymore to remove resistors from the adapter.

---

edmallon Post author
May 22, 2017 at 9:35 am

I think I will keep removing the 10K pullup resistors on MOSI, MISO & SCLK from that adapter, and using the internal pullups. Not that I really want to go through the headaches, but if I ever need to use SPI sensors (which might use any of the four different SPI modes), staying with processor side pullups leaves the the flexibility to change the configuration in future.

---

moondrake
May 23, 2017 at 10:07 am

Great Post!

The ppl at OSBSS seem to require an older version of the SDlib to pull of the trick with shutting down reliably. I am doing the same with my own loggers now, but I would rather use the original library. Did you experience the same/ Could you comment on this?

---

edmallon Post author
May 23, 2017 at 1:11 pm

Thank you for reminding me about versions!. I checked my sd fat version at: #define SD_FAT_VERSION 20160719, which is still the current one at github, and I'm using that with IDE version 1.6.9. I'm not sure what the differences are between the older vs newer SDfat libs, but I saw many people mention in the forums that they failed to get SD power control working with the default SD library that comes with with the IDE. If OSBSS did get it working with the default lib, then that's the first time I've heard of it being done.

WRT early vs later versions, I suspect that it all hinges on getting the timing right no matter what version you use, and that's going to have as much to do with the SD card controller as anything on the Arduino side of things. According to Grieman, calling SD.begin at the wrong time always has the potential to smoke your card because it resets the controller. There are methods available to avoid problems like this in the SDIO spec, that can't be implemented with SPI mode access that Arduinos use. When in doubt, add longer delays to let the card settle down before doing the next step.

---

zoomxx
May 24, 2017 at 3:21 am

The default IDE library name is SD only, SDfat is another library. SD is derived from an older version of SDfat.

---

moondrake
May 24, 2017 at 6:28 am

Thanks for the reply. Interesting you got it to work with a recent version, because the OSBSS ppl insist it only works with < 20111205 (of the Grieman lib, not the really old SD lib that indeed had various problems). They offer a special download for this specific git checkout.

There are problem reports in the OSBSS forums of ppl who tried with newer versions (though it also depends on the type of SD card it seems). So I guess I will have to run some test with the new version and sprinkle delays in case of problems. And perhaps you are doing something that makes it work better than the OSBSS code.

I feel it is somewhat worrying the whole thing depends on timings so much.

---

edmallon Post author
May 24, 2017 at 10:33 am

Yep. And according to these folks, some cards require hundreds of ms between the moment they are powered up and the moment they are ready to communicate. Also, SD cards of all speeds and flavors, first wake up into the SPI mode, so my order here of powering the card before enabling SPI might be incorrect, though if those

delays are really that long I'm probably fine with the timing anyway. I'm also generally using older 256mb uSD cards, and I think the older cards are more forgiving than the newer ones.

I've also been wondering if I ought to set the bus to SPI_HALF_SPEED? Just for some safety margin?

One factor that is repeatedly mentioned in the datasheets is that the power supply MUST be stable, but I already know from the restart problem that the v-regs on my Arduino boards struggle to handle the hit. And that's not even mentioning all the other potential problems...

---

gerben123
May 26, 2017 at 1:06 pm

Shouldn't you turn off the BJT after you set the SPI pins to inputs with pull-up, instead of before?

Any cross leakage will only be a few ns, but if done often enough, it might eventually damage the SD card. (Not sure how strong those clamping diodes are inside a SD card)

---

edmallon Post author
May 27, 2017 at 9:19 am

I think you are right, especially if SCLK is being held low in mode0 on idle by the SPI peripheral. Once the main ground is gone, then that becomes the only grounded line – I will do some trials with the SPI shutdown being done first and see how those loggers behave.

Update 2017-05-30: My preliminary tests show that it works with SPI shutdown done before the BJT in the turnOffSDcard function. I've updated the code in the post to reflect that. More extensive tests underway now.

Gabriel
June 7, 2017 at 10:55 am

Hi Mr. Mallon,

We are geography master students from Montreal university currently conducting some field work in the Arctic close to Iqaluit, Nunavut.

We built some data loggers (Pro-mini AT mega328 3.3v, same cheap SD adapter presented on this page and rtc) and pressure sensor (MS-5803 5ba). We followed your tutorial and are really grateful for your amazing work. We use exactly the same wiring as you presented in that tutorial: https://edwardmallon.wordpress.com/2016/10/27/diy-arduino-promini-data-logger-2016-build-update/

We are now struggling with power consumption problems and the sketch to put the sd.card adapter on sleep mode. We first tried this sketch :

[———-pages of code removed here by Ed. M.———]

But we still have the same problem, when we turn LOW the port D3 or SDpowerPin, the data stops being written on the sd card..

Can you help us with that, do you have an idea of what could be the problem...

---

edmallon Post author
June 7, 2017 at 11:58 am

Hi Gabriel,
Sorry for deleting your code here, but it was very long and would have prevented people from seeing the rest of the comments. In future try posting your code to GitHub, or some other platform like that so you can share big lumps of code more easily. There is also expert advice available over at the Arduino.cc forum as well, and their moderators have helped me many times.

WRT the sd card saving, the most important part that you left out was that you did not re-initialize the SD card before trying to open the file:

```
if (!sd.begin(chipSelect, SPI_FULL_SPEED)) {
Serial.println(F("Could NOT initialize SD Card"));error();
//add your own error function
}
delay(10);
```

If you de-power the SD cards you must call sd.begin BEFORE you try to open files with

File dataFile = sd.open("datalog.txt", FILE_WRITE);

Also, you are powering & depowering the SD card for every single reading, which from the looks of your loop delay, means you capture data every 8 seconds? I know that the folks at OSBSS saved data on every read cycle but I would only ever power cycle the SD cards that frequently for testing. On an actual deployment I buffer data to the EEprom on the RTC board so that data only gets written to the SD cards once per day. I don't know how may power cycles the cards were designed for, but I'm sure 10,000 per day is too much.

————————–

I've tried to convey that depowering the SD cards is the power saving technique with the **highest potential risk**, but if you are going to do it I'd reccomend that you implement it in three stages:

1) first get your logger code working WITHOUT SD depowering at all, or SPI shutdown. Just let the cards sleep normally and see what kind of sleep current you get. Once you get below ~0.18 mA sleep current, then the logger will probably run for more than a year on 3AA batteries, and you might not even need to do the SD card shutdown at all. Adding a second parallel bank of batteries is also a simple solution with no risk to your data.

2) Once your logger is capturing data, and saving it properly, the next step is to add *ONLY the code* for SD depowering (but not the BJT disconnecting the ground line) and see if you can get that working without calling sd.begin in your data writing functions. In my tests the code above works fine in loggers without the BJT installed, but I am also sleeping the loggers with Gammon's sleep code (Sketch J) , because I did not know if the Rocketscream sleep library would try to turn the SPI peripheral back on when the unit woke up.

3) If the SPI shut-down & Card Re-initialization work with the ground line still connected, then the last step would be to solder the BJT into place, so that the SD card ground line is actually disconnected while the logger is sleeping. Note that I am using old 256mb Sandisk cards, which might actually handle this depowering better than the newer high density cards.

————————-

P.S. I also noticed that you were shutting down timer0. Personally, I never mess with timer0 – it gets used in many important functions that are beyond my current coding ability to deal with if they stop working properly. Also be sure to use lithium AA batteries, rather than alkaline batteries, as they have much better cold weather performance.

Pavel Peřina
June 7, 2017 at 5:29 pm

It's nice, I've tried to do datalogger this way last year, it's almost complete (I sort of lost intereset after building prototpe) and code is here: https://github.com/pavel-perina/weather_logger .

I remember that I've used some P-MOSFET and NPN? to disconnect SD-Card module (with voltage regulator) from batteries and I've tested power consumption and ability to provide something over 150mA in ON state. Then I used this to shutdown power to SD card module, wired everything and left it to run for day. Then I measured that batteries won't last week, build simple oscilloscope (arduino+INA219 chip sending measured current over serial line like crazy), used gnuplot to draw graph (here http://i.imgur.com/gfXQ0pZ.png) and it seemed like power was not shut down once sd card was initialized. To my surprise, current was somehow able to flow through pins on sleeping arduino, through SD card and voltage regulator to ground (if I remember correctly.) Then I've changed SPI pins to input, waited shortly, shut down power to sd card module with transistors, put arduino to low power sleep, problem solved. I believe only remaining problems are mechanical and I remember I wanted to replace mosfet by another one that can operate at lower voltage.
Now I decided to make it work again and to complete documentation. To my surprise you decide for the same approach
———–
By the way have you tried to solve how to shutdown logger forever, once battery voltage drops below some threshold

---

edmallon Post author
June 7, 2017 at 6:59 pm

With my sleep current being so low now I just to sleep with the brown out detection enabled when the main battery pack approaches the regulator's input limit. That usually gives me many months before the cells get low enough to risk leaking. Trish never wants to wait much more than a year for her data anyway, so we rarely run the batteries to that point unless their is a sensor failure that somehow pulls the logger down.

So I'm not implementing a full power shut down yet, but when I get there I'll use something like Kevin Darrah's kill power circuit, as that's a nice straight forward implementation. But there are plenty of others out there to choose from. The key is making the circuit really stable/ON for loggers being bounced around on an installation dive.

---

rocketscream
June 8, 2017 at 5:46 am

Hi Edward,
Great write up here (had the same struggle few years back while using SD card for logging). I personally would use serial flash for data logging due to it's robustness and ultra low power (1-2 uA during sleep). Paul Stoffregen's SerialFlash library would allow you to read/write files just exactly like how you did with the SD

card. I'm not sure about the size you need though but 32MB is pretty easily available at decent price. Thanks for good word on the Mini Ultra!!!

---

edmallon Post author
June 8, 2017 at 10:27 am

Thanks for the tip about that library, which looks like it would make the transition from SD to EEPROM much easier. I'm pretty sure I could fit the data in there if I switch over to C-structs, but I'm still wrapping my head around that. The choice to use SD cards was largely driven by field logistics, where simply swapping batteries & cards makes on-site servicing (1 hour or more into a cave) much easier. The trifecta here will be when I transition to Serial flash, structs & an optical modem with a transfer window so I can retrieve the data from the units while they are still in place. But I'm probably a few years from pulling that all together, as current push is for new sensor development, and that is one long laundry-list of projects…

With your power management library, the Mini Ultra has long been one of my two favorite compatibles, with the other being Felix's Moteino Mega for situations where I need the 1284P's ram for things like circular buffers, etc. The old guard seem to have a real dislike of the Arduino system, and they keep telling me to switch over to a raw AVR. But having the v-reg & caps in place, with the pins broken out, just makes the kind of loggers I'm assembling much easier. Like the eeprom change over, I will probably get to a custom PCB eventually, but if I do it will only be because I built on the work of others, and the wonderful variety of boards in the open source ecosystem.

Saddle
September 14, 2017 at 12:10 am

I've put large 8 pin SPI based serial rams on the board and only powered up the SD card once every two hours to do a large dump to them. The rams are cheap and the ones I have are 128k by 8 (1Mbit by 1) and when you run multiple of them, you only enable one at a time for the write/read, cutting down the average current significantly. I'm running on single low volt batteries so I've been thinking of slowly running up a cap in a charge pump on the board to handle the in rush for the SD card.

Depending on the quantity of data, could you forego the SD altogether with an array of serial rams and live with direct from board dumps?

---

edmallon Post author
September 14, 2017 at 11:36 am

I've been page-write buffering (32 characters at a time) ascii format data to I2C eeproms for ages, since they come with the RTC boards I've been using. But they are so slow (in comparison to SPI memory) that there is a point of diminishing returns with that kind of buffering, so I generally don't buffer more than a weeks worth of data before flushing it all to the SD card. I've also been looking at Fram, since it is so fast it saves power, but again only if your bus is fast enough.

But to really use ram I'd need to switch over to bit-banging, or structs, and both of those options change my generic "attach any sensor you want" loggers into one-of code builds for each sensor combination- and that's something I'm trying to avoid. I want all my logger components to be completely interchangeable and including the codebase. That's easy with the unlimited space of an SD card, and much harder with the constraints of an all ram solution.

Saddle
September 14, 2017 at 6:18 pm

Thanks for the reply. I appreciate it as I find it hard to speak to others about this type of design query.

I've been working on getting an accel and temp logger running off a single button cell for as long as possible. There are ones available that run off a single AA or AAA cell, but unfortunately they are too big. Space is the major hassle for me.

I agree about the constraints of the ram and the eeprom timing. I looked at eeproms but although the largest is twice the size available of what is available for rams, found the write delays too cumbersome coding wise being forced to wait on page writes. I have found the serial rams a good swap for the eeproms (almost the same code required) and other than pumping the data in, they have both a zero write time and when not being spoken to, virtually zero current. The fact they come with a Vbatt backup line makes them (to me) as good as any eeprom but without the higher current and time delays.

Have you considered sharing the code load around by adding another processor to the board to off-load the storage hassles? Doesn't have to be much of a processor. The trade-off being a comparison of the current required for it versus other schemes. I could see a one-shop stop interface (SPI, serial, whatever between the two) and have the second one handle storage for the first. The second being able to sit asleep whenever idling along should also help.

---

edmallon Post author

September 18, 2017 at 9:57 am

Given how cheap these processors are, dedicating a second one in a logger to handle data storage would be feasible. That's essentially what the Sparkfun OpenLog is, and all you have to do is print your data to the serial line, and it then writes that to the SD card. But for me it's been easier to crank up the I2C bus speed, and then put the main processor to sleep during eeprom write delays. That approach can be done with a couple of lines of code, and it's been working for me for years. With one logging event / 15 min as the environmental monitoring standard, sleep current is several orders of magnitude more important to operating lifespan than the power used during data saving events, and my SD cards are now disconnected during sleep.

Comments are closed.