Drifters

The Drifter PCB is designed for five peripherals

1) Ublox gps communication via i2c
2) Iridium Rockblock communication via UART2
3) Xbee communication via UART1
4) DSM receiver via UART3 to RS485 data converter
5) Adafruit Memory card reader via SPI

The program repeats the following set of procedures every time it receivers pulse per second from the Ublox gps module at the beginning of every second

1) Print GPS coordinates and time to memory card
2) Ask the DSM for the time of last pulse and save it to memory card

The pulse per second from the gps is also sent to the DSM with which it uses as reference to keep track of its own time.

Error correction
On the printed pcbs I did not connect the data conversion RS485(as opposed to the time conversion RS485) to the power line, and we had to manually connect it, I have corrected this error in the PCB files afterwards, but the printed drifter pcbs(green) will have this errors.

Future Vision.
The xbee should be tossed and an RTC should be added to the PCB along with some transistors. The uc should be programmed to set all peripherals including itself to minimum power consumption along with a wake up alarm through the RTC. When the alarms goes off the drifter will start doing its normal functions like pinging etc., Along with that since the uc is fully event driven, the uc should be put to sleep when not processing any events. If all the above mentioned goals are accomplished, I predict that the device will have enough power to sustain off a 72Wh battery source for more than a week with the DSM pinger, perhaps a month without the pinger. Perhaps add an LED display and buttons to set the time settings, then we can avoid connecting the device to the computer every time we need to set the time settings.

Submersible

The Submersible PCB is designed for eight peripherals

1) Ublox gps communication via UART1
2) Iridium Rockblock communication via UART2
3) Xbee communication via UART1
4) DSM receiver via UART3 to RS485 data converter
5) Adafruit Memory card reader via SPI
6) Dead on Sparkfun RTC DS3234 via SPI
7) Big Easy Driver via I/O lines
8) OpenROV Pressure sensor via i2c
9) Coil Gun Charge and Fire via I/O lines.

Some of the peripherals(Memcard, Pressure Sensor, GPS, RS485 & DSM) also have nested BJT switches for the uc to turn off when not in use, however these have not been tested and on hindsight maybe I should have used MOSFETs.

The iridium and big easy driver have sleep lines to reduce power consumption and the RTC draws very little current on active state.

There are also lines to connect the opensegment LED display and other lines to connect 4 different buttons to program the time settings on the uc. Again these have not been tested either.

In terms of software, to maintain state the Flash memory and external RTC are leveraged to give the uc permanent state in unexpected times of temporary power failure or program crashing.

RTC stuff
The external RTC is used to synchronize the internal RTC of the uc during initialization, ad the internal RTC gets reset every time the uc is reset. 3 libraries have been put together to help with programming and synchronizing these RTC's. internal_rtc_clock, external_rtc_clock, DateNTime.

Error correction
The pressure sensor was not working at the time so instead of going down to a certain depth the uc is programmed to make a set number of motor rotation, this has to be changed once the pressure sensor problem has been fixed. There is an untested method called depth() which may be used to help with that goal.

The uc has 6 state, they are straightforward and can be understood by gleaning over the program.

Finally, every time the state is changed or the motor is rotated the corresponding information is recorded on to the non volatile memory and reloaded back into the program during initialization in times of abrupt uc restart. Not to mention, every time the uc is programmed all un programmed slots of the non volatile memory are set to 1, meaning any read byte from the non volatile memory will read 255. All this is accounted for in the program.

Future vision
As mentioned in the drifter section, the power consumption should be minimized by using better switches, RTC alarms and sending the uc and the other peripherals to sleep when not in use. Perhaps add the LED display to program time settings on the fly. If adding the LED display is too cumbersome, perhaps we can use the iridium messages to program it.

I have not documented the pin out of the DSM, which can give problems to future developers, but its a simple test using a multimeter connectivity tester on the two ends of the bus going between the uc and the DSM.

The documentation of the drifter PCB in terms of pin out is also very poor, I do apologize, the schematic will have to referenced along with the leads on the PCB.