MACHINERY AND TECHNICAL SCHOOL FAUST VRANČIĆ

Inexpensive electronics in space like environments

Testing various sensors in space like environments using a high altitude balloon

Student: Igor Subotičanec Mentors: Dajgoro Labinac, Lovro Dujnić

5/14/2014

This document aims to describe the process of designing a probe which is launched into the stratosphere, and the analysis of the recovered data. It also discusses design improvements which are derived from the data analysis.

Table of Contents

Introduction

The project is a part of the Teamstellar's competition for high schools in Croatia. This project is being performed by team Faust, from the machinery and technical school Faust Vrančić in Zagreb.

In this project we tested low cost electronic components aimed to be used in a space like environment. The circuit that was used for the experiment was enclosed in the form of a probe and was launched with a meteorological balloon that climbed up to 30km, where the air temperature and pressure is similar to those found in space. The circuit was composed of a set of inexpensive electronic sensors and a data logger. The plan was to set up the data logger for recording, and upon recovery analyze the data, and compare to expected values and datasheet specifications, and see what kind of effects occurred on the various electronic components.

The probe consisted of low cost MEMS sensors (accelerometer, gyroscope, magnetometer, thermometer, and barometer), a homebrew Geiger counter module and the data logger circuit which recorded the data on a SD card.

The following major components were used in this project:

- Microcontroller - AtMega328
- Data storage - SD card 4GB
- Accelerometer/gyroscope - MPU6050
- Magnetometer - MAG3110
- Barometer/thermometer - BMP085
- Geiger tube - SBM20 (СБМ20)
- Battery - Li-Ion 18650 3.7V
- Voltage regulator - MIC29300
- Battery management - 18650 cell protection module (eBay)

- Camera module - Not used, we didn't have enough time to implement that.
- Air quality sensor - Not used, we discovered that the heater element would use too much power, and that it would not perform well in low pressure conditions.

Project start

The idea was very simple, build a data logger circuit, and attach some sensors, and send it into the stratosphere. We started browsing datasheets of various components, and eventually ordered them online.

Microcontroller selection

Since we didn't have much time, we decided to use the Arduino bootloader and Arduino libraries, because by the time we would set up our own system in a properly working order, we would miss the deadline. So using the Arduino software platform meant that we have to use a Arduino compatible MCU. We figured that we could ask Atmel to send us some free samples of various MCUs and test them and see how it goes. So that is what we did, we sent a request to Atmel, and in the request we had to specify a valid UPS shipping account which we didn't have. So for students there was another email address to which you could ask for a shipping account. After sending a bunch of emails, and waiting for weeks we never got any reply from them. So looks like if you are a student and you wish to order a few MCUs from Atmel, you are going to have a lot of fun playing ping pong with the Atmel sample centre.

After that failed sample request, we decided to just go to the local electronic store and buy the only Arduino compatible MCU that they had, and AtMega328, TQFP package which turned out to be a pain to work with, since we could only etch homebrew PCBs which were low quality.

Voltage regulator, inverters and the battery quest

Once we had the MCU sorted out, we needed a power source. Since the data needs to be stored on a SD-card, and most of the sensors run on 3.3V, the logical conclusion was that our entire circuit should run on 3.3V. Also, since the mass limit was 150 g, and the project had to run for few hours, we needed a single battery cell that could provide enough power. Having a very limited budget, we ordered a Li-Ion 18650 battery. Along with that, we had to order also a battery control module that would prevent the battery from discharging totally, keeping it alive. So the 18650 battery is a 3.7V battery, and we need 3.3V. That leaves only 0.4V of space to regulate the voltage. Most of the popular 3.3V regulators accept a 5V input to generate 3.3V, like the LM1117-3. That wouldn't work for us, so we needed to find a regulator that would have a much lower voltage threshold. After a bit of research we found out that most of the low cost 3.3V regulators were in TO-92 packages only for small currents, which wasn't enough. Luckily after an evening spent browsing various datasheets we stumbled upon the MIC29300 which was in a TO-220 package and could provide up to 3A of current. We found the regulator on eBay, and quickly ordered it. The pin configuration is the same as in the 7805 regulator.

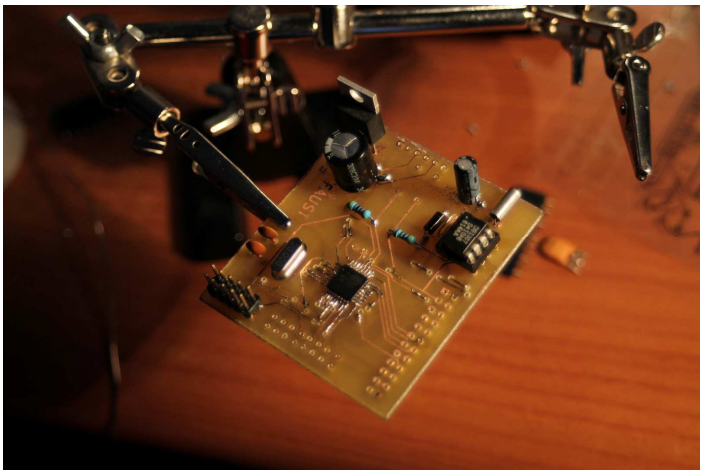
Since we like ordering components from eBay due to the low prices, we also had to wait for the unreliable shipping from China, which resulted with the battery arriving at the last moment. Luckily we had a Li-Ion battery from a broken cell phone that we could use for testing.

The Li-Ion batteries are not the best batteries for a space like environment, but since our budget was so low, we couldn't afford anything else. Another team that built another project which was meant to launch with the same balloon as ours, and with which we collaborated used non rechargeable batteries bought locally. They tried out every store, every type of battery, and in the end it turned out that no matter how that battery was fancy and expensive, their circuit would never work more than half an hour before the battery would completely die. That was a mystery, since in theory all of those batteries should have lasted for hours and hours, and not half an hour. In the end we concluded that all of the local stores sold old batteries that were in the store for years, and meanwhile their chemical composition degraded, making them useless. So it turns out that a new cheap Chinese battery is much reliable than a fancy expensive battery bought in the local store. Seems like our low budget actually saved us from that nonsense.

Once we got the battery and voltage regulator working, we still needed a 5V supply for the Geiger counter module, and the solution was to get a cheap miniature 5V inverter module from eBay. When we got it, we saw that it was tiny, so we were a bit sceptical if the thing would even work, let alone power a HV inverter module. It turns out that it actually works, and it was able to power the HV inverter module for the Geiger tube, and also the preamp circuit.

PCB design & issues

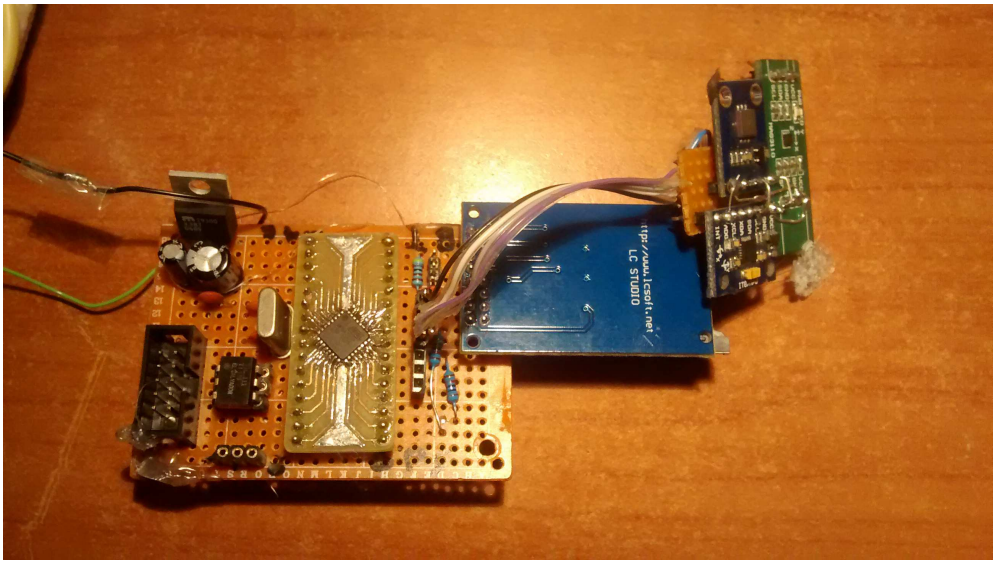
Once we figured out which components we would use, we started designing our PCB in KiCad. Once we had the board design done, we etched a prototype, which didn't look very good. That board wasn't that bad, a couple of jumper wires, and it was good as any, that is at least what we thought.



First prototype

We soldered all of the components to the board, examined all of the solder joints, and plugged the board to the BusPirate module to load the boot loader into the MCU. That is where it all went downhill. On the first try, for not bothering with the batteries, we just plugged the whole circuit to a standard wall AC adapter. That adapted had a ground lead that grounded the negative pole of the adapter, so that it wouldn't have any parasitic voltages that came through the transformer over to the secondary winding. So once we powered the board, we plugged the BusPirate, only to find out that nothing happened. We checked the crystal oscillator, no clock. The MCU was completely dead with no signs of life whatsoever. Ok, so we de-soldered that and soldered a new one. Once again, nothing, no signs of life whatsoever. So after that failure we decided to scrap that board, since it was defective. So having no time to etch a new one, we decided to just etch a small adapter PCB, and solder the thing on a prefboard, because it looked like that we would encounter a lot of trouble otherwise. Meanwhile when looking for something I noticed that the alligator clip that was supposed to ground the AC adapter fell off, leaving the AC adapter floating very faintly on 100V of parasitic AC voltage that leaked over the transformer. When connecting our circuit to some other grounded device, there was a 100V difference between the data pins, which is deadly for modern CMOS chips, and that is how we rendered perfectly working MCUs into small plastic bricks.

Once the new board was completed we connected the BusPirate once again, and it looked like it would work, but then it all just stopped, and once again, no sign of life. In that moment I noticed that one of the wires going close to the supply alligator clips had a small dark dot. Upon closer inspection we could see that the wire had a small hole in the insulation, probably caused by some other experiment, and since the insulation was damaged a small fibre of copper wire was exposed, and that small fibre touched the alligator clip of the adapter instantly killing the MCU. At that point we were getting a bit desperate with that evil AC adapter which seemed that its only goal was to kill our MCUs. Also, the switching noise coming from that adapter was horrifying, so that might also be a problem. In the end we just decided that the best solution was to use a cell phone battery, and not worry about it. Once we replaced the MCU for the fourth time it finally worked, and the BusPirate was able to load the Arduino bootloader in the AtMega328!



Final board

BusPirate

Not having a decent budget, and in the need of a way to use the ICSP programmer we got a inexpensive USB AVR programmer over eBay. When looking at tutorials for it on the web, we would always see the note saying that the programmer might be defective, and indeed our programmer was defective too. The solution was to use the BusPirate module that we already had, but the problem with that is that it is slow, so slow that loading the Arduino bootloader takes over 40 minutes! So that is how much we had to wait for every time we tried to load the bootloader.

SC card tests & issues

The first step in developing the firmware for the MCU was to test the most problematic part of it, the SD card library, which is the most complex part of the firmware. And as usually it was a pain to get it working. For start we just loaded the Arduino SD example, and it worked fine, but once we implemented that in our project, it just failed. Most of the times the MCU would just jam, and not write anything to the card, but after a bit of poking we got it to write something, but that something was corrupted, and that is not what we wanted. After being a bit desperate, we posted on the EEVBLOG forum, asking for help, and the main response was "improper memory allocation". Well, after testing all sorts of code, no type of memory allocation seemed to work. As time went by, we got a bit desperate about that not working, so we tried some nonsense desperate things, just to see what would happen. The first test was to print out the string that was supposed to be written into the card before and after the write function, and surprise! Before writing the string was intact, and after it was not. That meant that the stack was in some messy state, overwriting data that it shouldn't. One solution was to pass the argument over a global variable, and that helped, the data was written correctly, but not for long. After some more poking, the data was corrupted once again. Even the functions that were posted on the forum didn't work. It all looked like that when calling the write function somebody would came with a mixer, and mix up the ram of the MCU a bit, leaving a fine soup of bits that once were a coherent string of data. So what could cause that? We know that the Arduino has a bootloader which has some extra stuff sitting behind the sketch, so there must be something going on there. What could that be?

Since we were collaborating with team Tesla, and we agreed to write the firmware for their Arduino controller as well, part of our firmware would be used for their data logger circuit which was similar to the one we had. The other team had their hardware completed a before we had ours, so we first started writing their firmware which had to include a byte buffer that should be as large as possible. So in order to determinate how large it can be, we just looked at the memory usage, and found out that we could have it occupy 256 bytes of ram.

So what does a 256 byte buffer have to do with the write function of the SD library. It wasn't directly accessed, the buffer was there only to do some calculations before any data was written to the card. And yes, the buffer was the problem. Looks like that if you run your Arduino without a reserve of free memory odd things start to happen. The solution was that we only had to reduce the

buffer size down to 128 bytes, and it all started to work as it should. So that is where we spent an entire week of nerve-wracking moments.

Not willing to risk any bugs that might show up in mid flight, we added a watchdog timer that was supposed to check if the firmware is running. In the case that there was a problem, and the MCU jammed, the watchdog timer would expire, and reboot the MCU.

Fuses, fuses, fuses

When migrating the code from the other teams hardware to ours we encountered yet another problem. The MCU could be accessed by the BusPirate, but the bootloader wouldn't even start. What now?!

The Arduino is designed to run on 5V, and not on 3.3V, so as part of a design decision, the engineers that designed the Arduino decided to enable the brownout detector, and set it to 4V. After a bit of thought we came to the conclusion that the brownout detector was the cause of our problems, and that we should disable it. After a quick google search on how to get an Arduino working on 3.3V we found not very much useful information, but after looking for the brownout detector specifically we were able to find new fuse settings which would disable it. So we changed the fuse settings, but something didn't go right. So we had to reload the entire bootloader, which took almost an hour. Once that was in place, the MCU finally decided to work, and we finally got our test file on the SD card!

Geiger module construction

One of the sensor modules was the Geiger counter module which was supposed to measure the radiation coming from space. Existing modules were very expensive, some even reaching the price of 100$, which was way over what our budget could support, and the only way we would have one is that if we built it ourselves. The biggest problem is the HV inverter module, which has to convert 5V into 500V DC. In the first version we used inexpensive laptop LCD CCFL inverter modules which could provide up to 1KV. We discovered that in order for them to work we need to have a load that matched the impedance of the CCFL tubes, otherwise the controller chip on the module would shut down the inverter. The module didn't seem very stable, sometimes the required impendence would vary due to unknown reasons, and in the end we just concluded that the LCD inverter module was just too unstable for our proposes, so we discarded it. Poking through the drawers we found an old PC CCFL inverter which consisted of only two transistors, and a bunch of passive components, no controller circuit whatsoever, which meant that the inverter didn't care about the load impedance at all.

The Geiger tube itself is the SBM-20, an old Soviet tube that is the part of the surplus of components left from the cold war, we got that tube from eBay as well, even before this project started. The tube is designed to be powered with a constant DC voltage, and when an ionizing particle ionizes the gas inside the tube, the tube shorts out for few microseconds, but that signal is faint, and it needs amplification, which means that we need a preamplifier circuit. Upon trying out the first version of the preamplifier, the only thing that we got was noise from the inverter module that was just next to the tube, and no matter how much we tried that noise was just too problematic to shield. The solution was to modify the preamplifier circuit to attenuate the noise, but not the signal from the Geiger tube. After some poking, we found a solution which included bypass capacitors, and a dirty trick. The trick was to run the tube directly to the base of the first transistor, which is not such a good idea, because if something is not connected properly, the 500V from the tube would kill the transistor immediately, which happened a few times. But after a bit of work, we managed to build without having any more issues with high voltage killing the input transistor. While developing the preamplifier circuit we noticed that the signal of the Geiger tube is not constant, and that it has multiple ripples. Having multiple ripples meant that if we were to feed that signal directly to a digital counter module, such as a MCU, we would get a lot of false readings. The solution was to design a circuit that would block the ripples, and just give a straight logical signal. The circuit that could perform such a task is a monostable multivibrator, which can easily be implemented using a 555 timer circuit. The time interval has to be a couple of microseconds, depending on the tube. Note that when using the 555 timer in that configuration the output signal is inverted. For testing the entire circuit we used a few low level radiation sources. The simpler sources consisted out of Uranium glass marbles, and Thorium welding rods, which are a low gamma radiation source. We also had a sample of Americium 241 which is an alpha source, but it also emits gamma rays, and it is a much stronger source than the previous two. Handling Americium 241 with bare hands is not recommended. Do use precautions when handling with radioactive materials, such as gloves and other protective materials.

Geiger module in vacuum

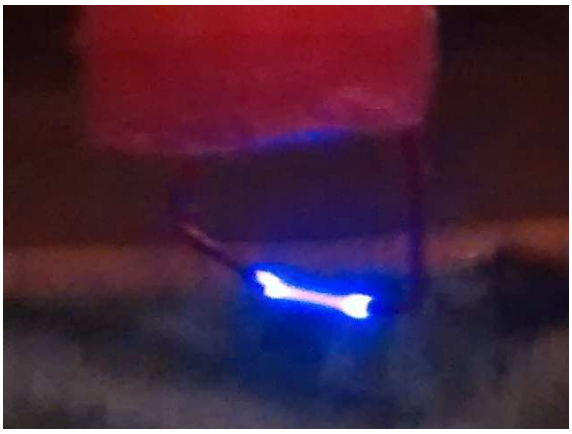
A benefit of collaborating with another team was that we were able to use the vacuum chamber which they obtained to test our circuit in low pressure conditions. The Arduino itself doesn't mind being in a vacuum, but the Geiger module does. Having a high AC voltage in low pressure conditions is a big problem since the voltage required for plasma discharges decreases drastically.

HV vacuum tests

Upon testing the Geiger counter module in low pressure conditions we quickly discovered that the HV would instantly arc through when the pressure would fall under 10 milliBars. We first tried to isolate the HV part with hot glue, but that didn't work very well, since there were pockets of air left that would leak through, and cause the circuit to arc. Then we tried using a PCB isolation

spray, but that didn't help either. But that was odd, if we isolated all of the traces, how could it arc? Since the HV is a generated by a high frequency inverter module, the thin isolation layer was acting as a dielectric material, and the low pressure air acted as a conductor, turning the thing into a capacitor, and allowing it arc once again.

Not knowing what the physical proprieties of low pressure plasma were, we decided to do some experiments, and see what affects plasma generation. We made a quick setup in the vacuum chamber with two electrodes. The first test was 300V DC, and see when it would arc. We turned on the vacuum pump and nothing, it didn't arc at all. So then we hooked a flyback transformer to an audio amplifier, and used the True RTA PC tone generator software to generate various waveforms and frequencies. We determined that all the frequencies on the kilohertz range and above were able to cause plasma discharges in the low pressure conditions. The lower the pressure the plasma becomes more unfocused, and it starts looking more like a glowing cloud, rather than a spark.



Above 10mBar

Under 10mBar

Pointed end aiming at a copper plate

We could not let the circuit arcing because the plasma would ran all over the preamplifier circuit and induce false signals, making the entire circuit useless. So we had to find a way to prevent the plasma from forming.

Polyurethane foam

Somebody suggested using polyurethane foam which when hardened contains a lot of air bubbles, and it is light. So we got a can of it, and started experimenting. The results were promising, but not good enough. The foam would work for a few minutes, but the air would eventually leak, and it would arc once again. After more experimenting we managed to fill all of the remaining gaps, and have it working stable in the vacuum chamber for an extended period of time.

Polyurethane foam enclosing the Geiger module

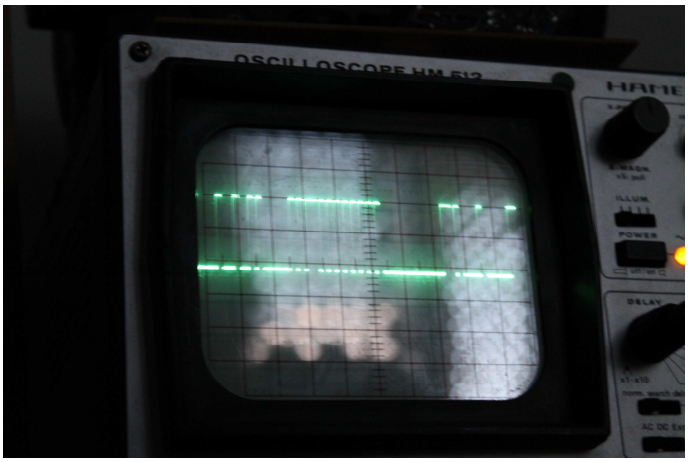
Optocoupler circuit

The Geiger counter module runs on 5V, while the rest of the circuit runs on 3.3V so a voltage level converter was necessary. Also after seeing the effects of low air pressure on the Geiger counter module we didn't want to risk a having high voltage running down the data line caused by a plasma discharge, so we decided to use a optocoupler circuit. The optocoupler module consists of a led diode, and a phototransistor packed in a single plastic chip package. When the led diode emits light, the phototransistor is forward biased, and current can flow from the collector to the emitter. The first thing to watch for is the current flowing thought the led segment, if the current is too low, the light intensity will be low, having the photo transistor limiting the amount of current, and as a side effect, having invalid output levels. Adding too much current can damage the led, so an adequate

resistor has to be added in series, the value depends on the model, in our case a 200 ohm resistor was enough. On the other side, there is also a resistor, that connects the positive voltage supply to the collector of the phototransistor forming a output that can provide valid logic voltage levels. If that resistor is too high then the maximum bandwidth decreases due to parasitic capacitance in the phototransistor, which as result, decreases the rise and fall time of the entire circuit (slew rate).

I2C nonsense

The last thing left to implement in the firmware were the I2C sensors. That looked like a fairly easy task, since all of the sensors used were popular in the Arduino community, and they all had some sort of library written for them. Not wishing for noise, and speed problems we decided to use 2k pull-up resistors on the I2C bus, and having the bus no longer than 20cm. The first test didn't go very well, whenever the MCU would access the I2C port the MCU would instantly reboot. So maybe there was something wrong with the code, but it all looked fine. Investigating with an oscilloscope we found out that the lines would go low for a clock period, and go high again while the MCU was rebooting. We suspected that there was a problem with the power supply, so we checked for noise in the power supply, and also for voltage drops when accessing the I2C line, but nothing out of the ordinary, we even added extra bypass caps, but that didn't help. Not wishing to lose any more time with that nonsense, we decided to bitbang the I2C protocol. We chose another port for that, and we first tested with no devices attached, and it worked, data was coming out, and the MCU was not restarting. Being pleased by the result we rearranged the circuit so that the I2C lines connected to those two pins, but when testing the circuit, once again, we encountered the same issue! We switched the pins once again, and the I2C signal was present on the new pins. So what was going on there? Why when driving the I2C bus the MCU would restart, even if we just made it go low? As a test, we increased the pull-up resistors to 8k, and it worked, then we also tried switching back to the hardware I2C peripheral, and it worked. So having a 2k resistor was the issue. But that doesn't make much sense, the current going through that resistor when pulling it low was only about 1mA, which is far inside the supported range. So that mystery was left unsolved.



The I2C data line is finally working correctly

Real time clock

The original plan was to have a real time clock chip in the circuit, but the board that we had ran out of real estate, so we had no room left for a RTC chip. An easy solution was to just use the millis() function that is provided by the Arduino library, and just convert that into seconds. The only drawback of that method seemed to be that we should know the exact time when the circuit is started. Having the firmware almost completed, and the only thing missing was a clock to keep track of time for the data logs. We used the millis() function to calculate the time, and then just write the elapsed time form when the device was last activated. It all worked fine, except for the magnetometer module, in that case it would always write 0, even having the same variable, and same code writing for all the other sensors, the file in the SD card would always say 0 for the magnetometer data, but that wasn't such of a problem, because all of the files were updated simultaneously, so we should be able to read the time from the other files.

Low battery issues

While testing the circuit we discovered that if the battery was depleted while operating, the SD card would get corrupted. We guess that the reason is that when the voltage is falling, the MCU can still work, but that the voltage levels, and the SD controller becomes unstable, causing a corrupted entry in the FAT table, which would corrupt the entire SD card. Usually this problem would be solved by having the brownout detector set to deactivate the entire MCU, but since we already disabled it, we had to come up with a new way to disable the MCU when the battery was running out of power. We calculated how much time the circuit could run, and for safety reasons we halved that time, and made a function that would block the circuit after 4 hours of operation, which is more that what is necessary for the trip.

Overheating issues

In the last days, right before launch, we got asked how do we keep the circuit from overheating. Overheating? But it is -60C at those altitudes. The problem is that the air is so thin, that it conducts head poorly, and it is not able to cool the components. The only way that components can cool in a vacuum like environment is through infrared emission. We quickly calculated the power dissipation on the voltage regulator, and we concluded that the power dissipation was minimal, and no heat sink was required. The problem was with the other team, which had a 7805 dissipating almost a watt of heat with no heat sink, which is not usually a problem. But when we calculated how much could the temperature be dissipated through infrared emission we got the figure of 350C. 350C would be way over the limit of the 7805 regulator, which would trigger the safety mechanism that is integrated, causing it to shut down. So we quickly tested the scenario with a resistor load, and a cheap IR thermometer. The test showed that the leads to the board were enough to keep the temperature under 90C which was getting close to the limit, but not enough to trigger any safety mechanisms.

Housing

The housing of the project had to be light and solid, so Styrofoam was a obvious way to go. The first thing we tested is to see how Styrofoam performs in a vacuum. The test showed that the air can exit the Styrofoam making it expand, and when returning it to ambient pressure it would shrink down. The Styrofoam that we used for testing was crispy and it looked porous, so we got new Styrofoam spheres form the local hobby store, which looked much more solid, and less porous. We couldn't test them, since they didn't fit in the vacuum chamber.

Working all nights, with the full moon keeping us company

Too much glue, too much foam, too much mass

The last step was to glue everything in place, and seal the housing. Once we had it all glued we weighed the probe, and we had over 50g of extra weigh. For gluing we used epoxy glue and the hot glue gun, which turned out to be big sources of weigh. When we removed all of the components, leaving only the empty Styrofoam shell with traces of glue, we measured that the glue itself had 8g of mass. Also the extra foam on the Geiger module added extra weight, so we had to cut it a bit, risking the thing to arc once again.

The enclosure itself must not be pressurised because that might cause it to explode once it reaches low pressure, so the housing must have venting holes.

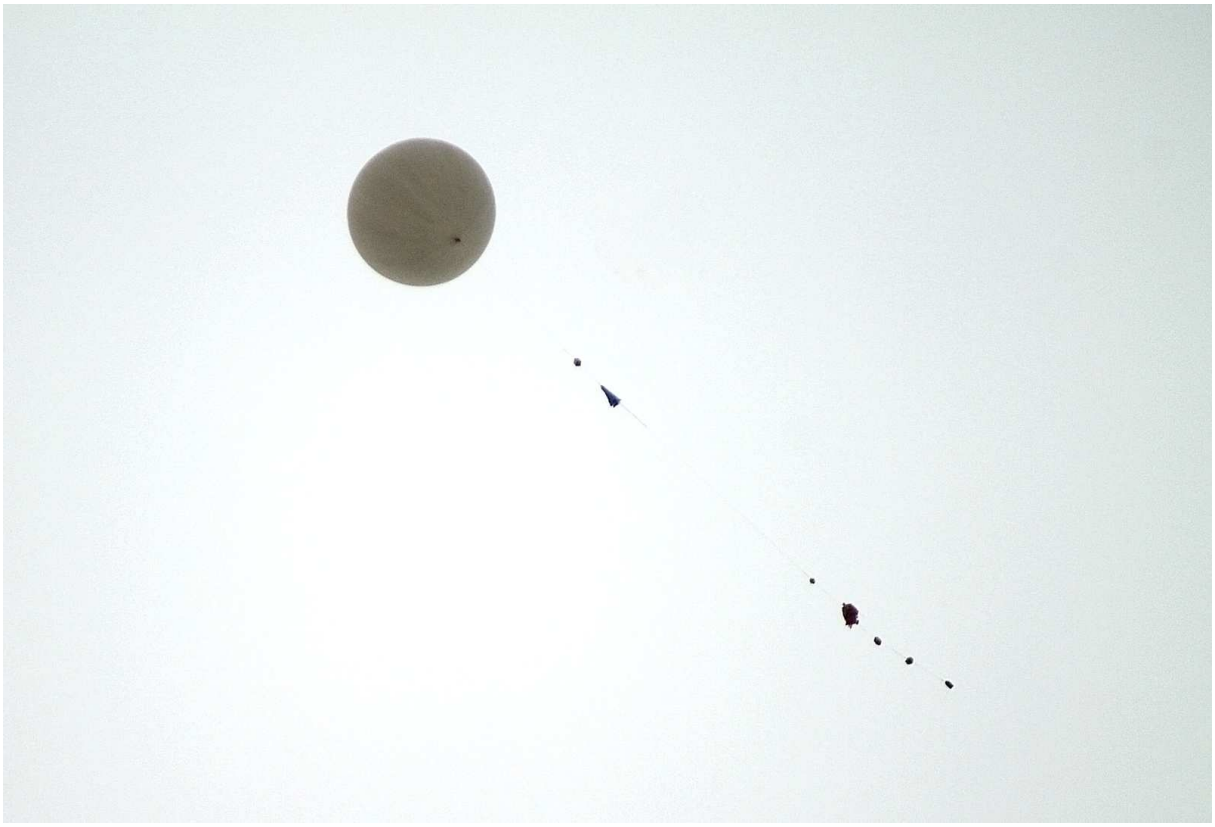
Another problem is rain. If it starts to rain, the rain can enter the venting holes, and cause a short circuit on the circuit board, so the venting holes have to be designed with care.

At the end of the flight the experiment will be falling at the speed of ~30km/h which is like having the probe being hit by a tram. The circuit boards have to be placed in such a way that even if they detach form the mounting place, a short circuit shouldn't be able to occur.

Data recovery

The balloon was launched from Zagreb, 23.4.2014 and the entire flight lasted for one hour and 50 minutes. The balloon landed somewhere in Zagorje (north of Zagreb) and it was recovered within minutes. A few days after the recovery, the probe was returned to us, and we were able to extract the SD card and read the data. The fall was not very soft, it fell with the speed of approximately 30km/h.

The modules were lightly glued in the interior of the probe, and upon impact everything unglued, and it turned into a big mess of wires.



The experiments have finally launched

Data analysis

Upon plotting transferring the data in a spreadsheet, and plotting it on charts, we discovered some interesting events. Let's analyze the data of each sensor separately.

Barometer

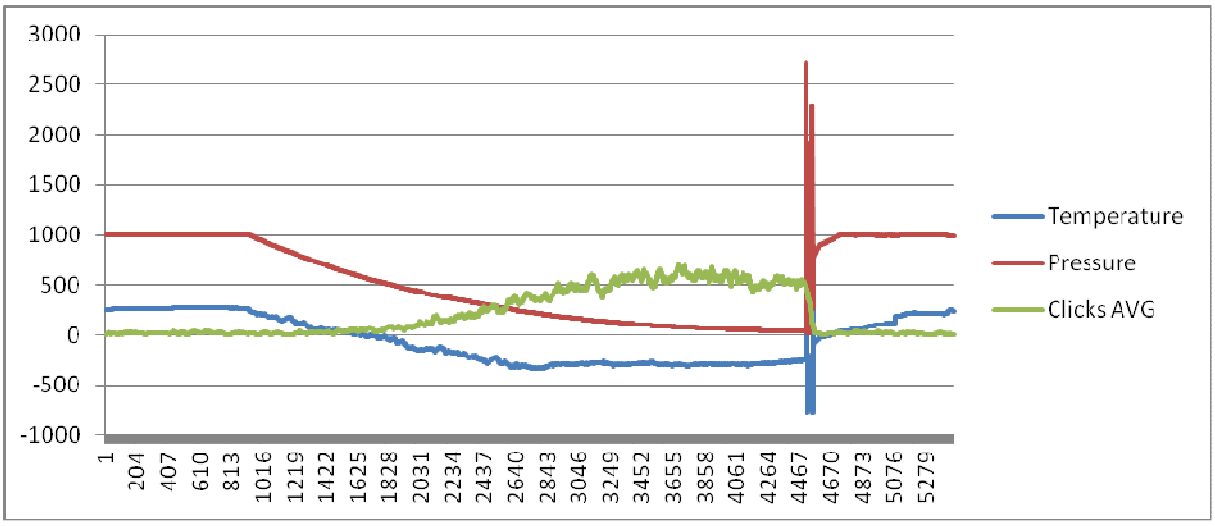
The chart of the barometer sensor clearly displays the pressure falling from 1002hPA down to 37hPa (~23km altitude) and at that point something happened. The chart shows two pressure spikes going up to 2724hPa. The thermometer sensor also has unusual readings at that point. The probable cause is the Geiger counter module. While shaving off the extra foam insulation that we added, we probably weakened it enough so that it leaked once again, turning the probe into a plasma ball. The interference probably interfered with all of the sensors, and eventually disabled the microcontroller.
The readings are incoherent and missing for a while, and they once again become coherent when it re-entered the dense atmosphere at the pressure of 781hPa. From that point on, the data seems to be correct.

Thermometer

The thermometer sensor is in the same module as the barometer, and it had similar issues as the barometer. When the two spikes occurred, the temperature reading was -76.5C.

Geiger counter

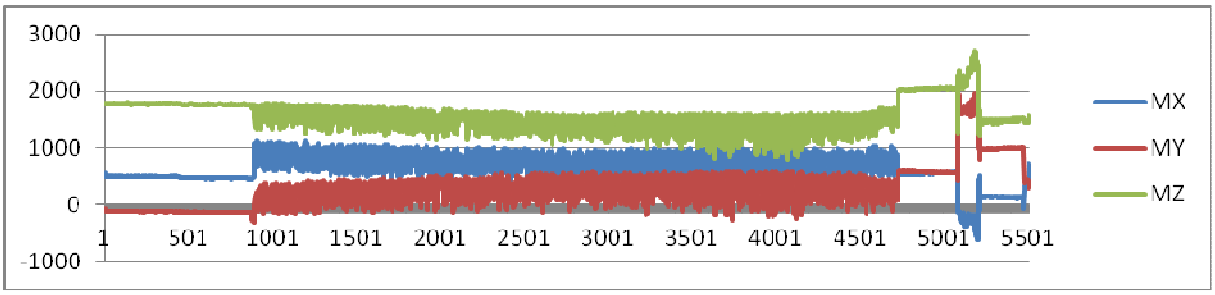
The Geiger module ridings look as expected, except from the already mention part where the MCU was temporarily disabled.



Readings from the experiment

Magnetometer

The magnetometer module also performed as expected, having the same issue when the MCU was temporarily disabled. The readings show the earth's magnetic field straight and direction . After launching the balloon the magnetometer indicates that the probe was constantly rotating in various directions, as the magnetic field measurements kept constantly shifting. The total magnetic intensity seemed to match the expected values. When calculating the total magnetic intensity there was some noise in the reading, which was probably caused by other elements interfering with the magnetic field in the probe as it was rotating and by the sensor itself not being perfect. The reading show that the magnetic field was gradually decreasing as the altitude increased.
The readings were once again incoherent when the strike occurred, and it shows a static magnetic field which intensity is greater of the earth's magnetic field, and it always had the same direction. That could be caused by the current of the plasma discharge, or it could just be that the data stream was corrupted. Upon landing the magnetic field returned to the nominal value, and after a while it showed some signs of rotation, which was probably caused by the recovery team picking up the probe.



Magnetometer readings

Accelerometer

The accelerometer sensor had unexpected readings constantly. We are not sure what the exact cause of that was. When testing the module prior to launch, it seemed as if it was functioning normally, but even the readings while it was still on the ground look unusual. The readings show some random spikes which form a low frequency saw like rising signal. When testing the circuit after the recovery, the problem persisted. That might indicate a problem with the accelerometer module or some other bug.

Future design improvements

As the project came to an end, we learned a lot of new things, like how to properly build probes that must endure space like environments. If we were to build another probe, we would take a slightly different approach and do some things different, making it more stable and secure.

The first and biggest change would be the design of the Geiger counter module. As we learned only HV AC currents are able to generate a significant plasma discharge, so to better address the issue the best solution would be to completely separate the HV power supply from the Geiger counter module. The HV inverter and the HV rectifier should be on a separate board tightly encased in some sort of resin or foam which would keep air bubbles permanently trapped. Also if possible, it should be enclosed in a metallic case, which would act as a faraday cage, trapping the interference inside, and in the case of a plasma discharge, the plasma would short out with the chassis, and not with some random component on the circuit board. Since HV DC currents don't seem to be so problematic, they could be lead with regular wires to the 4.7Meg resistor that is used to limit the current on the Geiger tube. The rest of the circuit, which include the Geiger tube, preamp, and monostable circuit should be on the separate board, requiring only light insulation, which could be some thin insulation layer or light foam.
For safety reasons the optocoupler circuit would be also recommended, as we already built in our probe.

The next design change would be to insulate all of the exposed connectors and wires, so if something detaches from the chassis, it won't cause a short circuit. The easiest was to do that is to enclose each module in some sort of plastic bag, or tape it down with electric tape, or any other insulating material, so even if all of the components detach, they won't short.
For better reinforcing the modules, more glue is suggested, also adding some sort of filling material, such as crumpled paper, light foam or sponge. The filling material should fill big air gaps, so when the probe hits the ground, the various modules don't go smashing each other. Also be sure that the SD card is held in place securely, because the force of the impact could make it fly out of the SD card holder. Interesting fact, after the trip the Styrofoam sphere that was initially 20xm in diameter shrieked by 1cm. That was caused by the air leaking from the Styrofoam pores, and not being able to fully return once it re-entered the dense atmosphere. So keep in mind that if you have a completely full styrofoam hosing, it might exert some pressure on the components if there is not enough free room to shrink.

The thermometer module should be exposed to the exterior, but in our probe we just mounted it in the inside, and punched a hole through the Styrofoam shell, which probably lead to inaccurate temperature readings. Also don't mount the thermometer in the line of sight of heat emitters, since infrared radiation could be received from them, offsetting the reading.
The best way to mount the battery would be to have it near to a heat source in such a way that it is always kept at some reasonable temperature.

The exterior Styrofoam colour can also be a factor, depending on what you are trying to achieve, changing the colour of the Styrofoam could lead to heating the entire probe. If it is black it will absorb sun rays, making the probe warmer, and leaving it white will reflect sun rays, not heating it much.
A spherical exterior seems as a good choice, since it will always have the same air drag regardless the rotation, making the flight balloon more stable.

Conclusion

Almost none of the components that we used were not rated for the conditions we exposed them to. The microcontroller was only rated from 0 to 70C as most of the other IC-s, and the barometer sensor was only rated form 300 to 1100hPa, and we clearly went far under the 300hPa limit. Regardless that limit, the pressure readings kept being precise all the way, until the incident, so it seems that this sensor can be indeed used to get decent readings down to vacuum like conditions. We are not sure about what long term effects could that have to the sensor, as it only experienced a short period of such conditions. The thermometer part didn't go out of the specified range of -40C to 85C, but it got very close to it, reading -33C.

Since we ran out of free space, we decided not to include the real time clock, which as it turns out, might not be the best design choice. Relying on the MCU timer is not the best solution since a failure of the MCU would render the time invalid. Having a RTC chip next to the MCU will assure that the time is always correct, even if the MCU is temporarily disabled, or if the brownout detector is triggered.

The Arduino bootloader doesn't seem to be the most stable platform to develop. It doesn't seems to work fine with a complex programming, since that uses a lot of the available memory, leaving it with scarce resources. The Arduino platform is aimed to be used by beginners and quick prototyping, since it offers a vast number of add-on shields and libraries. When developing complex projects and commercial products, I would recommend using a platform that is custom built to suit the required task. There are a lot of microcontrollers available on the market, each with its own hardware specifications, so when designing something, you can always find a mcu which will have the required hardware, so that you can most efficiently design your circuit. I would also recommend writing your software from scratch, because when using some software which is not guaranteed to be working finding bugs might be a difficult task since you don't know the inner workings of that code. Another advantage of writing your own code is that you can optimise it to achieve better performance for your device, lowering hardware requirements and total cost.

Using SD cards might not be always trivial, so other types of memory storage might often be more attractive. The biggest disadvantage of SD cards in embedded system is the file system. Not only that it is complex to interface with it, but it also creates more points of failure. In the case where the circuit is exposed to unknown conditions, and if something corrupts the FAT table, the entre file

system could be corrupted leading to total data loss. Extracting data from a corrupted SD card is not trivial, and it can be very expensive. If your device doesn't have to store more than one megabyte of data, consider using EEPROM memories, since they are much easier to implement, and even if there is some data corruption it will stay localised, keeping all of the other data intact.

As a final conclusion we can only say that inexpensive hardware can indeed be used for tasks for which it wasn't designed, but extensive testing is required to assure that. Also be aware that there are a lot of unpredictable things that can go wrong, so try to address all issues that could occur. Using old components can also be an option in low budget projects, such as this one. The optocoupler IC that we used was from an old machine dating back to the 1980' which was scrapped a long time ago, and yet, it still works as intended. Most of the passive components were also reused from old projects, and old component stocks. But when using such components make sure that they still perform as intended, since they can often be partially damaged.
And as a final advice, keep it simple, but don't exaggerate, since there might be unexpected issues which might tip of the stability of the circuit.