**Device with GPS pps flash**

**for setting optical time stamps**

**in astronomical videos from digital cameras**

Björn Kattentidt - IOTA/ES - bjoern@kattentidt-astro.de

**Motivation:**

Previous analogue cameras could be stamped with time information in the video data stream using VTI (video time inserter). Modern digital cameras only allow this with complex programming. This has resulted in many very good homemade camera systems, but at considerable expense. With state-of-the-art ‘smart telescopes’, the problem becomes clear that it will be even more difficult to meet the special requirements for time stamps with millisecond accuracy in videos.

A device that can be attached to any optical system and, when switched on, uses NMEA data after a few minutes to meet the requirements for evaluating time stamps in star occultation videos will be of interest. Such a device is presented here.

The system is published with open source code [1].

The device is intended for DIY construction. However, the author is happy to provide assistance with sourcing parts and assembling the finished device.  
This publication is intended to help you build the device yourself.

**Features at a glance:**

* pps flash (40 ms) via LED on clamp with cable (approx. 1 m) to control unit
* Brightness adjustable via rotary potentiometer with integrated switch
* Time flash distance adjustable over a wide range with two BCD encoding wheels
* Various modes preset in the upper number range, up to 100 modes possible
* Special functions: No settings required, GPS is used for GPS-synchronised flash patterns. If the GPS signal is invalid, the internal Arduino LED changes every second and the LED on the clip flashes every second at pps.
* 3.2 V LiFePo4 battery with 600 mAh, consumption: typically 80 mA
* Operating time approx. 7 hours
* ast 0.5 to 1 hour: ‘low batt.’ indicator flashes rapidly
* Deep discharge protection: device switches off, residual current 2mA, all LEDs off
* 2 hours charging via USB-C, integrated charger.  
  Lower window: Red light ‘Charging’, orange light ‘Full’
* Arduino NANO with open source code via DC-DC converter (3.2 V to 5.0 V)
* GPS module ATGM336H (3.2 V) GPS+BD  
  Warm start 10-25 seconds (author), cold start 35 seconds (manufacturer),   
  up to several minutes in case of poor reception  
  Middle window: pps GPS ready indicated by inverted flashing LED
* Step-by-step instructions for DIY construction on github.com [1]
* Secondary solution for timestamps in occultation videos if the primary timing system suddenly fails for technical reasons
* Eyelet for hanging with correct position of the GPS antenna
* Strap for hanging and securing the LED cable

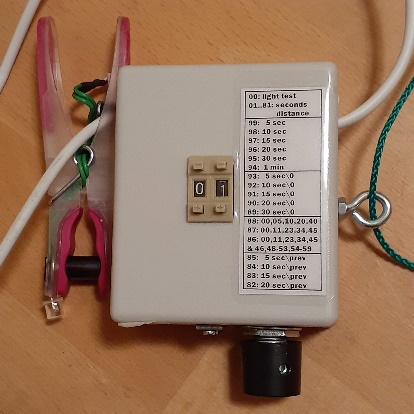
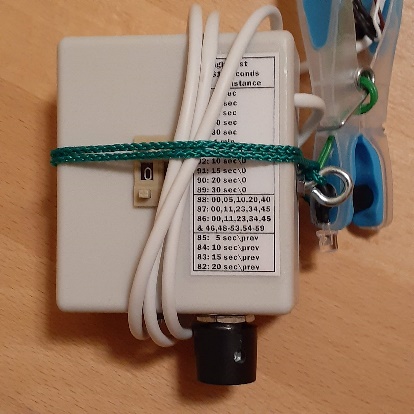
  

Figure 1: GPS pps-flasher, finished device

**Functionality:**

A flashing LED is attached to a clamp. This is connected to the control unit by a cable to reduce the mechanical load on the optics and enable wobble-free operation.

The clamp is attached to the dew cap of the optics with the LED facing inwards in the beam path. If the distance to the edge of the beam path is large, the clamp width can be modified by extending the black cylinder.

The code is optimised so that the flashes appear less than 100 µs after the pps signal (peak per second) from the GPS. The loop time is less than 0.25 ms. The light signal is therefore delayed by significantly less than 1 ms. Our own measurements using a DVTI yielded a maximum of 2 ms. However, this has not yet been reliably verified and still needs to be confirmed using EXTA!

Except for #00, the flash always has the same duration of 40 ms.

In mode #00, the LED lights up for a long time. This allows the brightness to be adjusted using the switching potentiometer. The faint glow can just about be seen in the video. PyOTE [2] can even evaluate markings when the flashing is no longer visible.

The special functions (#82-#99), on the other hand, use NMEA data, including for the second, while the time trigger is based on the significantly more accurate pps signal.

Only with valid NMEA data will the special functions work as expected and the upper status LED on the Arduino NANO go out! If the NMEA data is invalid or missing, the upper status LED will flash slowly (changing every second).

When the battery is fully charged, the device will flash for approx. 7 hours, with the status LED flashing rapidly for the last half hour to hour. The deep discharge protection then puts the device into sleep mode with a residual current of 2 mA. However, the device must still be switched off completely with the switch to protect the battery! The battery has a capacity of 600 mAh with a maximum consumption of 80 mA. The battery is charged via the USB-C port. A charger is integrated for this purpose. The control unit can be charged via the USB port when switched off or during operation. However, it is better not to charge during operation to protect the GPS module, which is then connected to the battery at increased voltage. The charging time is approximately two hours.

Deep discharge protection is implemented. At low voltage (3.1 V), the upper status LED flashes rapidly (4 hertz). At the end-of-discharge voltage (2.8 V), the device switches to sleep mode. When deep discharge protection is activated, all LEDs go out. However, a residual current still flows, causing the battery to be deeply discharged after another 2 days. Therefore, the device must always be switched off using the switch, as a single deep discharge can destroy the LiFePo4 battery.   
The device can be reactivated by switching it off using the rotary knob and then switching it back on again. If it switches off again immediately, wait a few seconds before switching it on again.

**Normal use:**

Hang the device from the eyelet or set it up with the eyelet facing upwards so that the GPS has optimal reception.

Clip the bracket onto the dew cap with the LED pointing into the optical path.

Switch on by turning the knob to the right.

Initially, the 1pps LED is permanently lit and the status LED flashes every second because no pps signal and no NMEA data have been detected yet. When the status LED is off and the 1pps signal briefly goes out very short every second, the device is ready for operation.  
This will happen within a minute in the open air, but sometimes it takes several minutes.

If the status LED flashes slowly, no valid NMEA data is available yet. This means that the special functions from #99 down to #82 cannot be used. However, the exact 1pps signal can be used with functions #01 to #81 if the 1pps LED on the device is already flashing. If the upper status LED flashes quickly, this means that the battery will soon be empty and the device will then switch to sleep mode.

The exact electrical pps signal from the GPS module is used for the LED on the clip.

First set the encoding wheels to #00 and adjust the required brightness on the potentiometer.

Then set the desired mode. You can switch to any other mode at any time using the two encoding wheels. After one second, the signal will be correct.

The device can now be used to set millisecond-accurate timestamps visually.

After use, switch off the device again using the rotary knob!

**How it works:**

A red LED flashes briefly at the tip of the clamp every second, synchronized with GPS. If the number is set to #11, for example, the LED flashes every eleventh second. This allows any cyclical repetition to be set and, with a little practice, the start can also be preset with the aid of a very accurate clock by quickly increasing the number starting at #01.

To clarify: An internal counter counts up from zero after a flash until it reaches the set number. If you set this number higher after a flash at a specific second, you have set the sequence to the beginning.

Until the set number is exceeded by the counter, you can continue to adjust the encoding wheels. With a little skill and thought, such as setting the tens first, there is enough time to complete the settings.

This gives you a lot of freedom, but this type of setting requires skill and you still need an accurate time (Android: e.g., AtomUhr app). If you start and stop the video with a robot app (e.g., MacroDroid) when using a “smart telescope” with a cell phone, the start and end of the video are accurate to the second, which makes it much easier to determine the minute.

For significantly easier use without additional apps, special functions are implemented from #99 downwards (see list). These modes use NMEA data, which means that the flash always lights up exactly at the set sequence according to the GPS second!

The special functions and longer flash intervals are only suitable for cameras that can be relied upon for time synchronization, such as ASI cameras, while they are less suitable for QHY cameras, which appear to have firmware that causes the actual exposure times to fluctuate and unpredictable pauses to occur (author's own experience). However, for exposure times longer than 0.1 seconds, this is not very relevant for accurate timestamps.

If you are in a hurry, you must use a manually adjustable flash sequence and synchronize the start yourself with an accurate reference clock (Android: e.g., AtomUhr APP).

**List of available modes (expandable)**

#00: Only for the purpose of adjusting the brightness of the LED on the clip.  
The LED lights up for 0.8 seconds after each pps.

#01 ... #81: The LED flashes at the set interval in seconds. It remains off in between. A specific setting can be achieved as described in the section “How it works”.

|  |  |  |  |
| --- | --- | --- | --- |
| BCD | flashes at specified GPS second | LUT | (Kode) |
| Second mask fully synchronized | | | | |
| #99 | 00, 05, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55 | LUT\_05s | FF |
| #98 | 00, 10, 20, 30, 40, 50 | LUT\_10s | FE |
| #97 | 00, 15, 30, 45 | LUT\_15s | FD |
| #96 | 00, 20, 40 | LUT\_20s | FC |
| #95 | 00, 30 | LUT\_30s | FB |
| #94 | 00 |  | FA |
| Second mask with omission at zero crossing (minute change) | | | | |
| #93 | 05, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55 | LUT\_05s0 | EF |
| #92 | 10, 20, 30, 40, 50 | LUT\_10s0 | EE |
| #91 | 15, 30, 45 | LUT\_15s0 | ED |
| #90 | 20, 40 | LUT\_20s0 | EC |
| #89 | 30 | LUT\_30s0 | EB |
| Second mask time-coded | | | | |
| #88 | 00, 05, 10, 20, 40 | LUT\_DF | DF |
| #87 | 00, 11, 23, 34, 45 (similar Pseudo-Barker Kode) | LUT\_DE | DE |
| #86 | 00, 11, 23, 34, 45; Minute binary: 48-53 und 54-59,  LSB starting, parity bit 46 | LUT\_DD | DD |
| Second mask without lead-in event before the minute change | | | | |
| #85 | 00, 05, 10, 15, 20, 25, 30, 35, 40, 45, 50 | LUT\_05sv0 | CF |
| #84 | 00, 10, 20, 30, 40 | LUT\_10sv0 | CE |
| #83 | 00, 15, 30 | LUT\_15sv0 | CD |
| #82 | 00, 20 | LUT\_20sv0 | CC |
|  | | | | |
| #01 … #81 | Second interval, starting with any second |  |  |

Table 1: Modes

Sequences #94 and #99 enable GPS-synchronized flashes at equidistant intervals.

Sequences #89 to #93 do not have a flash at GPS second 0 for classifying the flashes relative to the minute.

Sequences #88 and #87 allow reliable time data to be determined and checked with PyMovie/PyOTE [2] if the minute of the start or end of the video is known.

Sequence #86 is the same as #87 but with additional integrated encoding of the minute with repetition and parity bit at fixed positions, similar to the method used in the DCF77 signal. This eliminates the need to note the minute.

Example:  
Even parity: Total number of ones with parity is even   
1 0 0 0 1 1 binary = 35 decimal: 35th minute  
MSB x x x x LSB, LSB comes first in time  
The parity bit is therefore calculated for even parity so that the total number of ones is even.

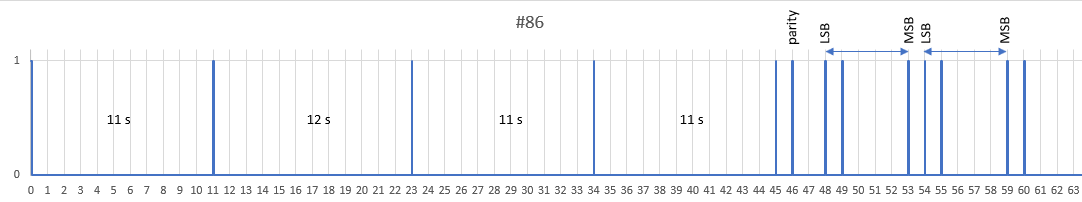


Figure 2: Mode #86, minute 35

Sequences #82 to #85 have this missing flash at the end of the minute sequence. This makes it easier to determine the start of the minute.

The advantage of the special functions is that the light curve is less disturbed by the flashes than if there were a flash every second.   
Any other encodings can be implemented.

The disadvantage of the special functions is that the reception of the NMEA data must be valid, which can take longer or may be interrupted if reception is poor.

If only the pps signal is used in modes #01 to #81, it is sufficient for the 1pps-GPS LED to flash.

**Four useful applications:**

#01:   
LED flashes every second. Basic function. Use when a precise second signal is required very quickly, or when images are lost at high frame rates, or when the camera's image times jitter. The minute change and minute must be determined differently, e.g., by noting the minute and the start of the video automatically at a specific GPS second.

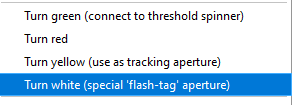
#85:  
LED flashes every 5 seconds except in the 55th second. If the flashes are clearly visible, the minute change can be easily evaluated. Quick replacement solution in case of primary system failure. The minute must be noted.

#88:  
LED flashes at 00, 05, 10, 20, and 40 seconds. With few flashes, videos longer than 2 minutes can be reliably recognized when the minute changes. The minute must be noted.

#86  
With flashes at 00, 11, 23, 34, and 45 seconds at unequal intervals according to the pseudo-Barker code, the change of minutes is clear after less than a minute. The minute can be determined in the 6 bits of the 48-53rd and 54-59th seconds. Can be used if you do not want to note the minute, but the second is more difficult to read. Parity bit for additional security at 46th second. Safest method in case of risk from passing clouds. In this case, it is better to record several minutes of video.

**Video evaluation:**

The flashes are currently 40 ms long (25 fps), so that the brightness of the flash set with the rotary knob remains the same across all exposure times on one optic. The flashes can be set so low that they are almost invisible, do not affect the evaluation, but can still be reliably detected with PyMovie (white aperture) and PyOTE [2]. At maximum brightness, the flash is clearly visible. This is hardly dependent on the size of the telescope!

PyOTE: 

**Electrical overview:**

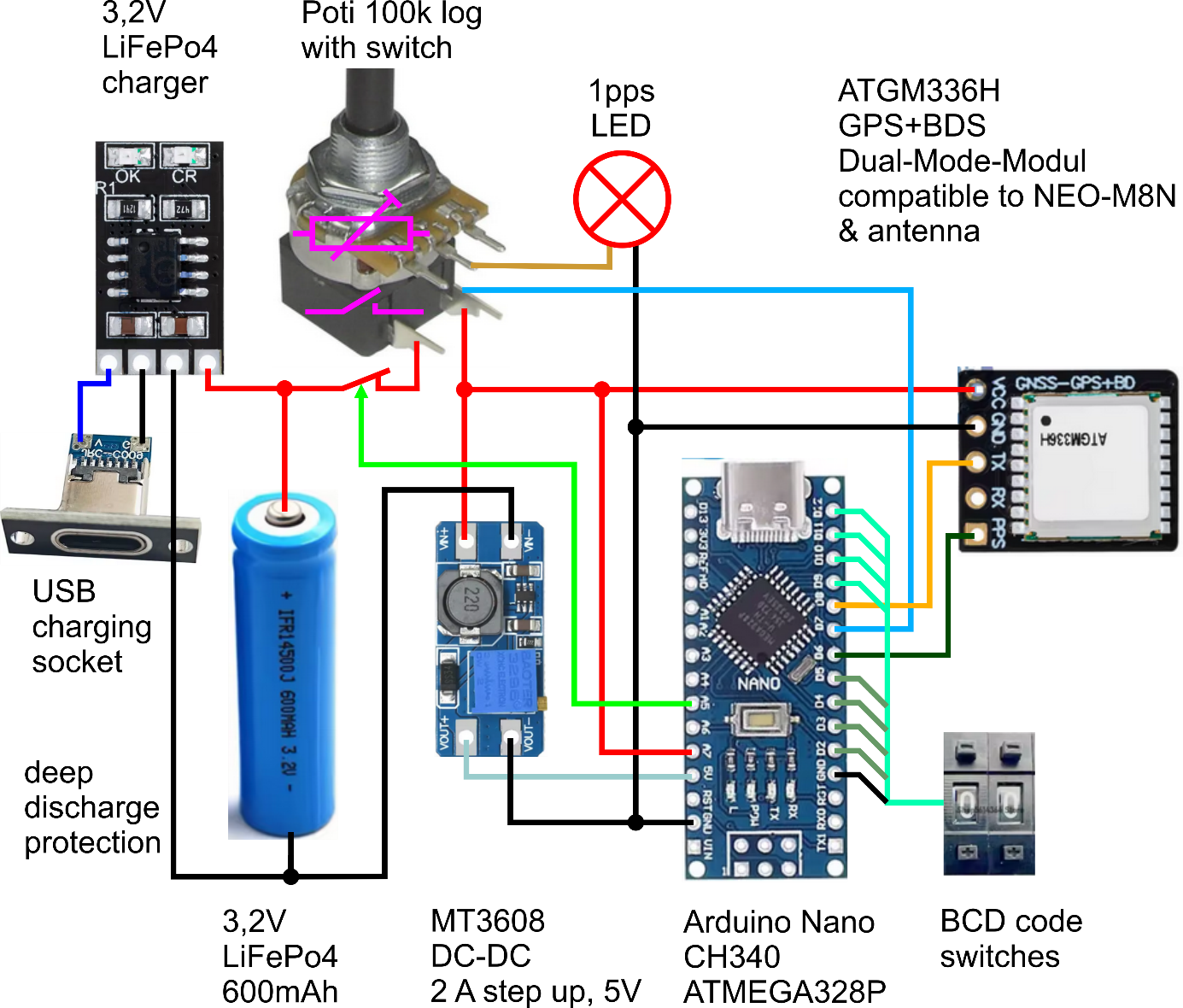


Figure 3: Electrical connection illustrated

**DIY instructions**

**Connection circuit board:**

Connection circuit board schematic (1PPS\_Teiler\_v5\_pro.sch)  
This circuit board is mainly used to establish connections between devices with plug connections. This makes it easy to exchange, test, or replace the modular components of the only partially soldered device shown in the event of a defect. Those who prefer secure electrical connections can solder more or all of the connections. However, plug connections with pin headers and socket headers are sufficient.

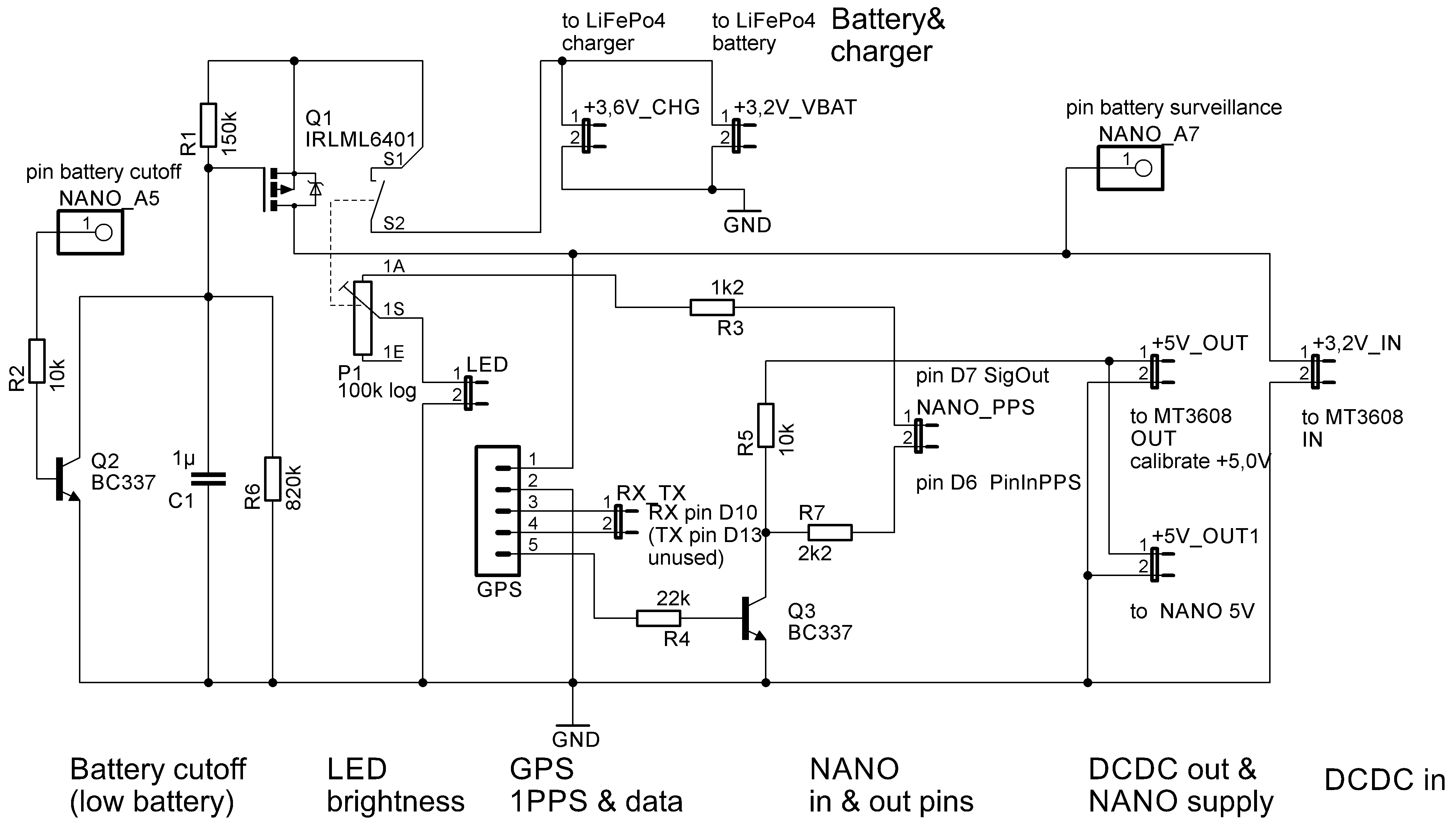


Figure 4: Circuit diagram of the connection circuit board with EAGLE [3]

Note: Resistor R7 must be soldered into the cable, not onto the board for the 1PPS\_Teiler\_v5\_pro circuit board! The design has been changed accordingly for the 1PPS\_Teiler\_v6\_pro, but this version is not yet available.

Parts list (1PPS\_Teiler\_v5\_pro.sch)

|  |  |
| --- | --- |
| Part | Value |
| C1 | 1µ ceramic |
| P1 | 100k log. |
| Q1 | IRLML6401 or p-channel MOSFET with RDS (< 1 ohm) and UDSthd (< 1 V) |
| Q2 | BC337-40 or similar |
| Q3 | BC337-40 or similar |
| R1 | 180k |
| R2 | 10k |
| R3 | 1k2 |
| R4 | 22k |
| R5 | 10k |
| R6 | 820k |

Table 2: Parts list

Printed circuit board: 1PPS\_Teiler\_v5\_pro.brd

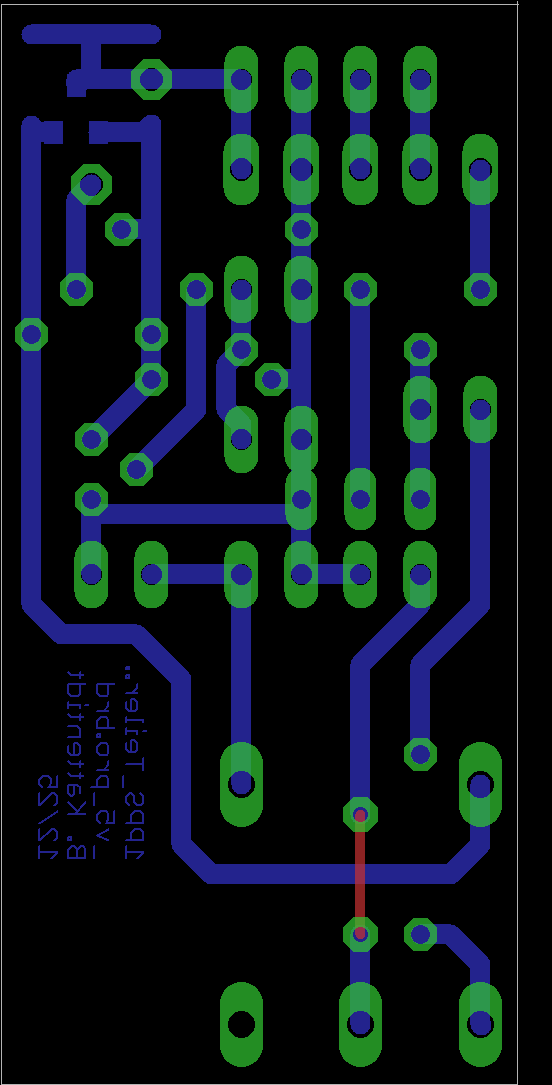
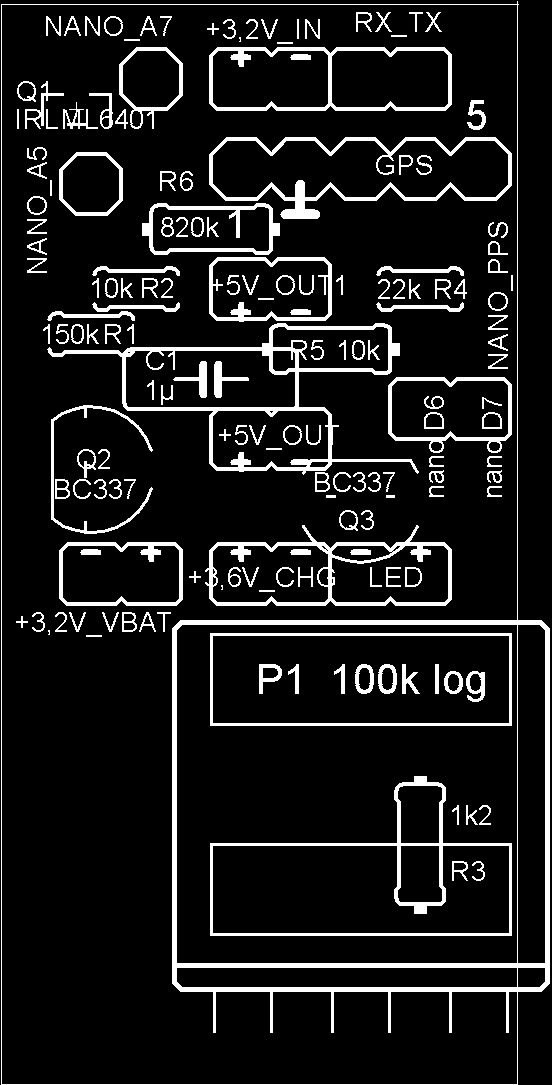
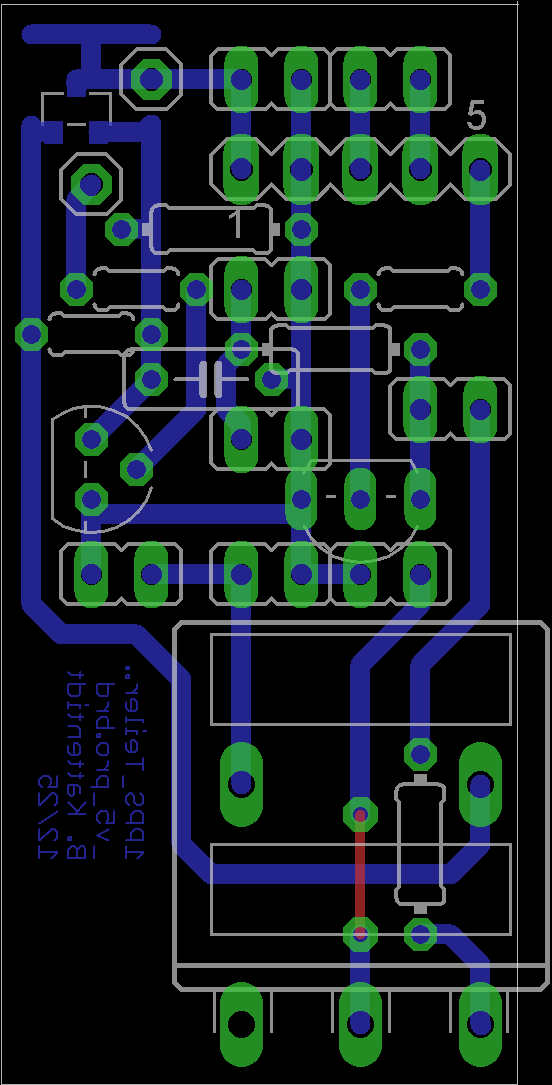
  

Figure 5: Printed circuit board – conductor side, assembly, combined (EAGLE) [3]

**Programming:**

In the open-source program code [1], values that should be observed are commented with “// SET: ...”. In addition, reference is made to comments in the code.

The Arduino IDE [4] was used for programming.

The following external libraries were integrated into the code: #include  
<AltSoftSerial.h> [5], very stable serial interface for GPS (timer-based)  
<TinyGPS++.h> [6], parsing of NMEA GPS data (time, status)  
AltSoftSerial is pin-fixed (Arduino Nano): RX = Pin D8  
Not compatible with boards such as ESP32, ESP8266, or Leonardo without modification.

All other functions in the code come from the Arduino standard core.

**Signal flow:**

The Arduino NANO reads the digital signal from ATGM336H pin 5 (PPS) via its pin D6 (code: PinInPPS) after voltage adaptation on the circuit board. This pps signal is provided for time-synchronized flashes of the LED on the clamp via Arduino NANO pin D7 (code: SigOut). The NMEA data is read via Arduino NANO pin D8 (code: AltSoftSerial) for accurate classification in the special functions. This evaluates the NMEA data for seconds and minutes and uses it for coding in upper modes (#99 downwards).   
The left encoding wheel is connected to D2 ... D5 and GND, the right one to D9 ... D12.

Attention: A 2200 ohm resistor must be soldered into both the line to D8 and from the (left) BCD encoding wheel to the GND pin! This is done at GND to protect port D9 and at D8 to ensure that the NMEA data is read reliably.

The voltage for the deep discharge protection is read at A7 and the shutdown is performed via A5. The Arduino NANO switches off the entire power supply via a circuit with two transistors, except for a small residual current.

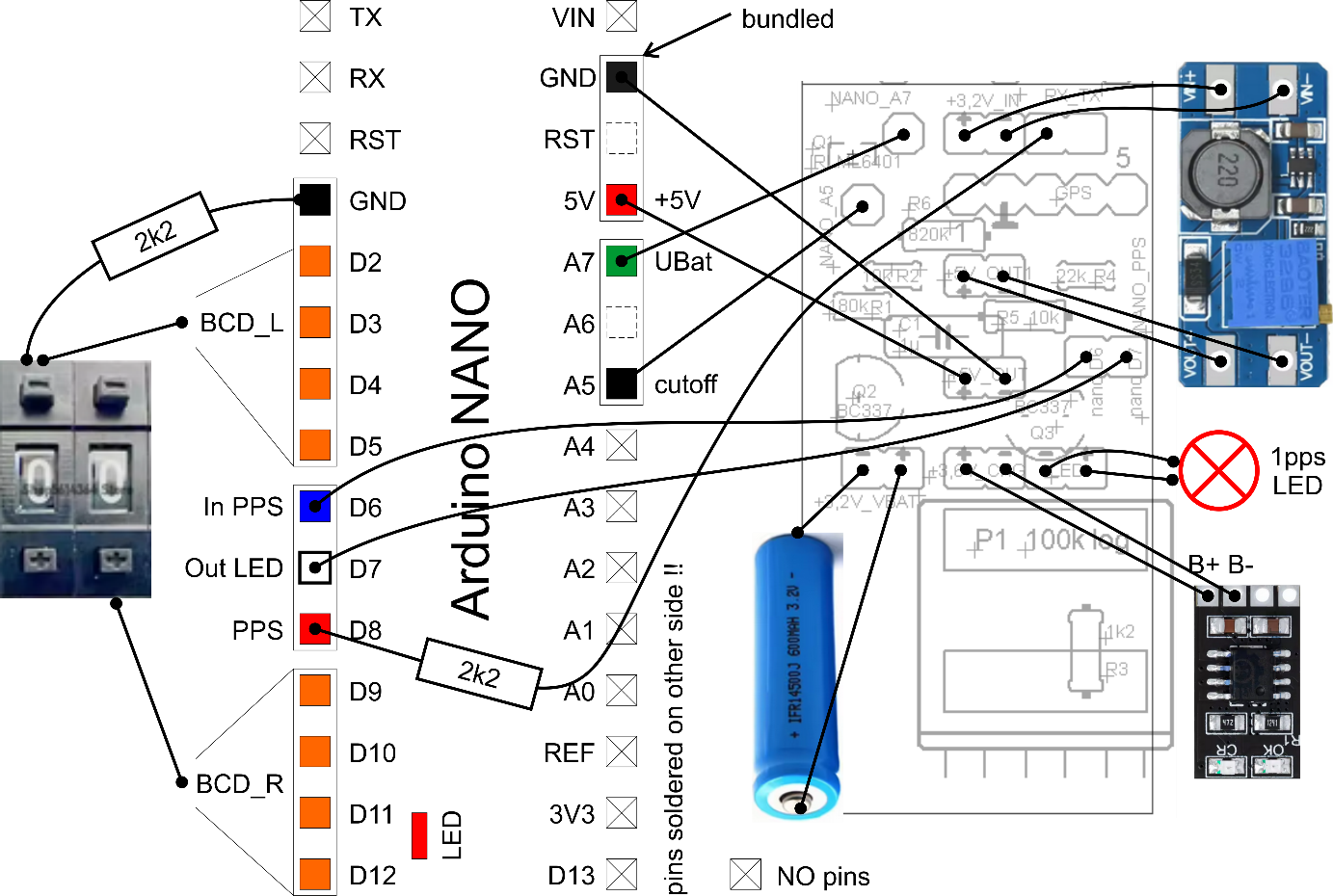


Figure 6: Electrical connections

**Assembly:**

Experience with electronic soldering is required!  
Mechanical dexterity is also very advantageous.

Important: When soldering, especially the wires, never touch the components during soldering, but fix them in place before soldering (clamps, masking tape). “Cold solder joints” will lead to premature failure or malfunction of the finished device. Only solder as hot as necessary. The solder should emit almost no smoke and soldering should be completed within a few seconds. The solder must flow and have a shiny surface when cooled. Do not move the solder joint until the solder is set!



Figure 7: Soldering cables and pins

These plug connections must be made; one has a resistance of 2200 ohms:

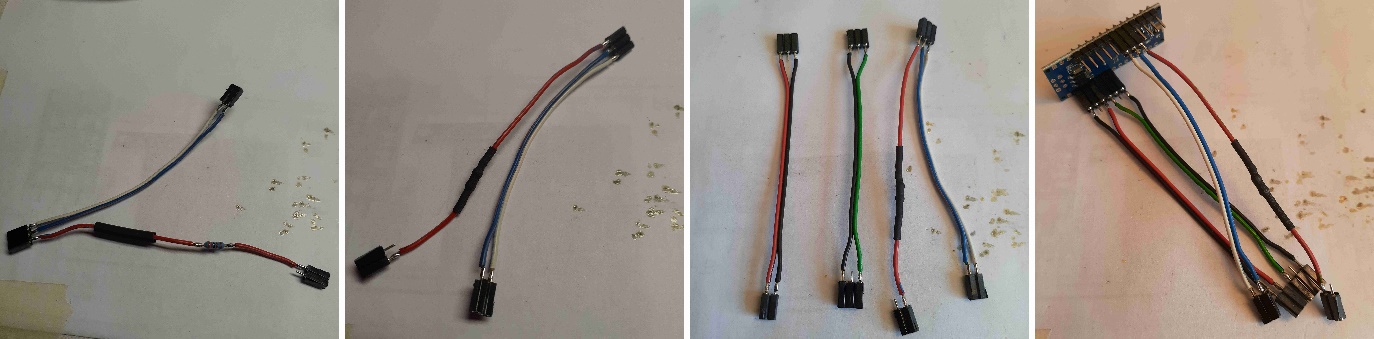


Figure 8: Required plug connections

The cutting lengths (mm) for the flexible cables are suggested in the diagram.  
Shorter rather than longer dimensions should be used.

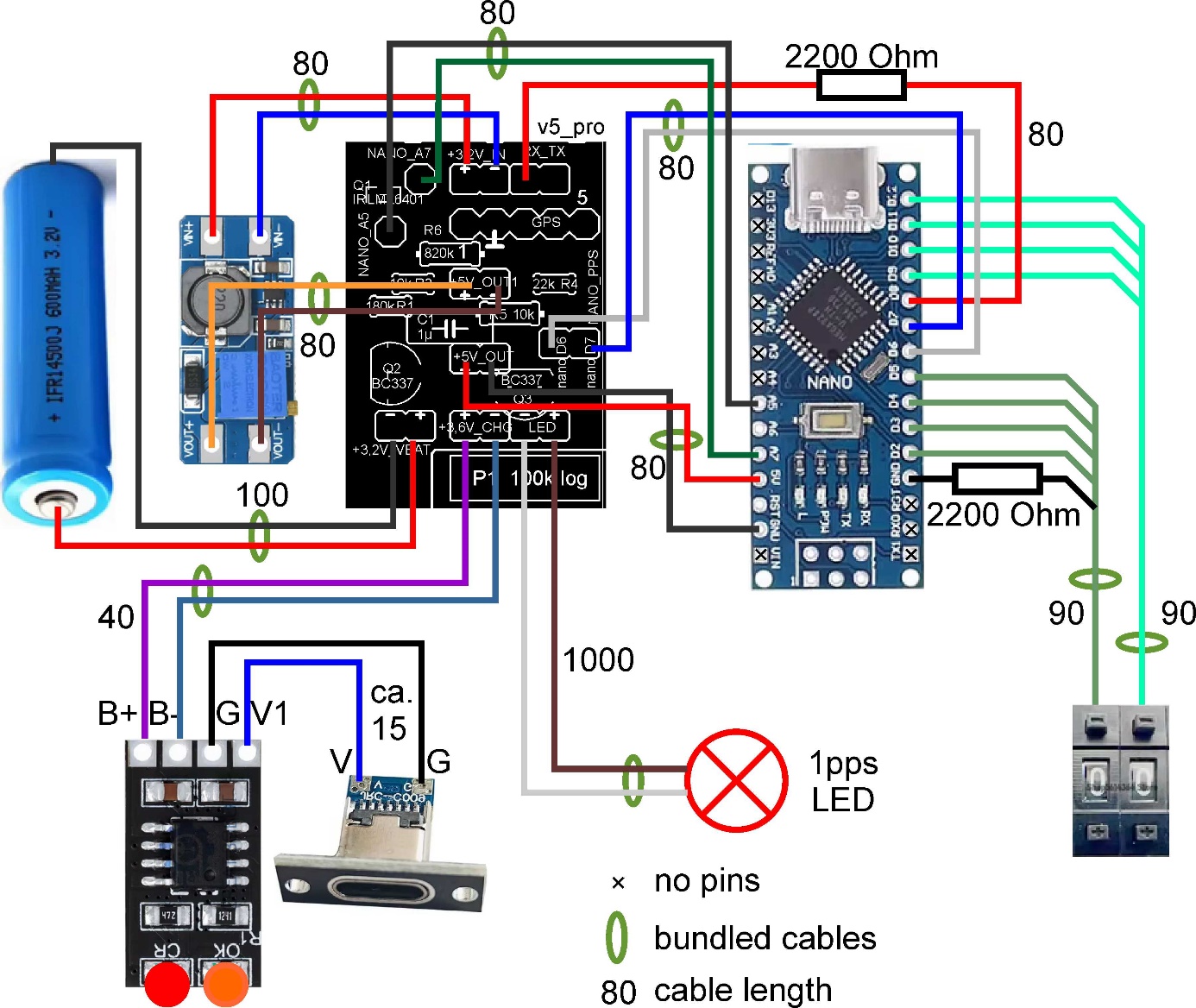


Figure 9: Wire lengths and types illustrated

Colors used for visualization may differ from actual wire colors!

**Printed circuit board:**

A recess must be cut into the printed circuit board for one corner.

Solder in the following order: bridge on top, resistors (lying down, then standing up), transistors, one SMD MOSFET on the conductor side, pin headers (with sockets attached as lateral spacers), capacitor, and finally the rotary switch potentiometer and long pin header for GPS. Shorten the shaft of the potentiometer to 10 mm. It is best to secure the SMD component with double-sided adhesive tape before soldering.

After soldering, shorten any protruding wires on the conductor side to the solder joint. Do the same for the potentiometer. Then sand at the potentiometer very flat to a residual height of 1 mm using 320-grit sandpaper. This work is easier with an orbital sander. This is necessary so that the potentiometer fits into the housing. Remove all metal dust thoroughly!

The “RX” pin is not used and remains free.  
All other pin pins are connected to wires.

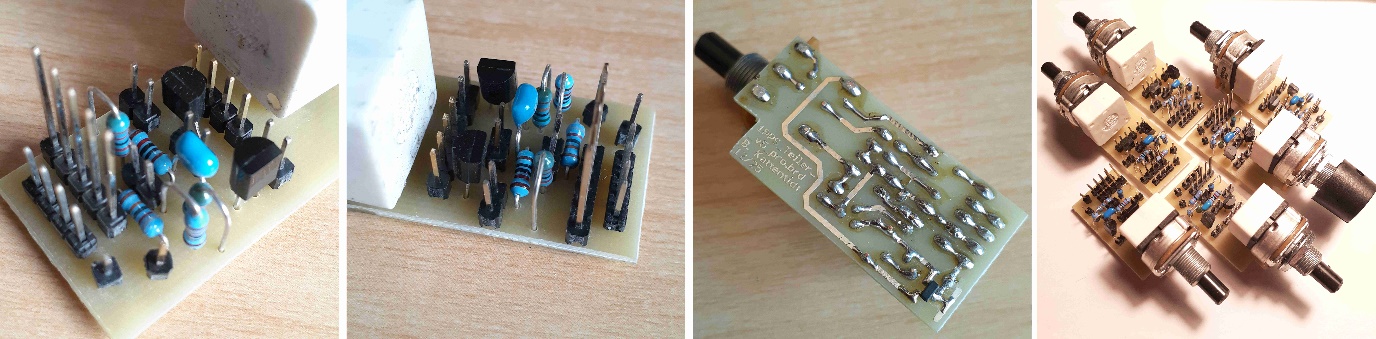


Figure 10: Circuit board for connecting the devices

**Encoding wheels and DC-DC converter:**

Solder the wires to the DC-DC converter and the encoding wheels. Solder the encoding wheels individually. Place the encoding wheels on top of each other and then connect the ground (black here) with a wire bridge. The ground wire should be soldered to the left digit. Otherwise, this can be changed in the Arduino code (LCcodeswiOrder).

Caution: A 2200 ohm resistor must be soldered into the common ground wire to protect the Arduino Nano ports. Not shown in the picture.

Carefully cut off the four outer protruding pins on the side of the housing so that they are flush. Be careful not to damage the four plastic rivets that hold the housing together.

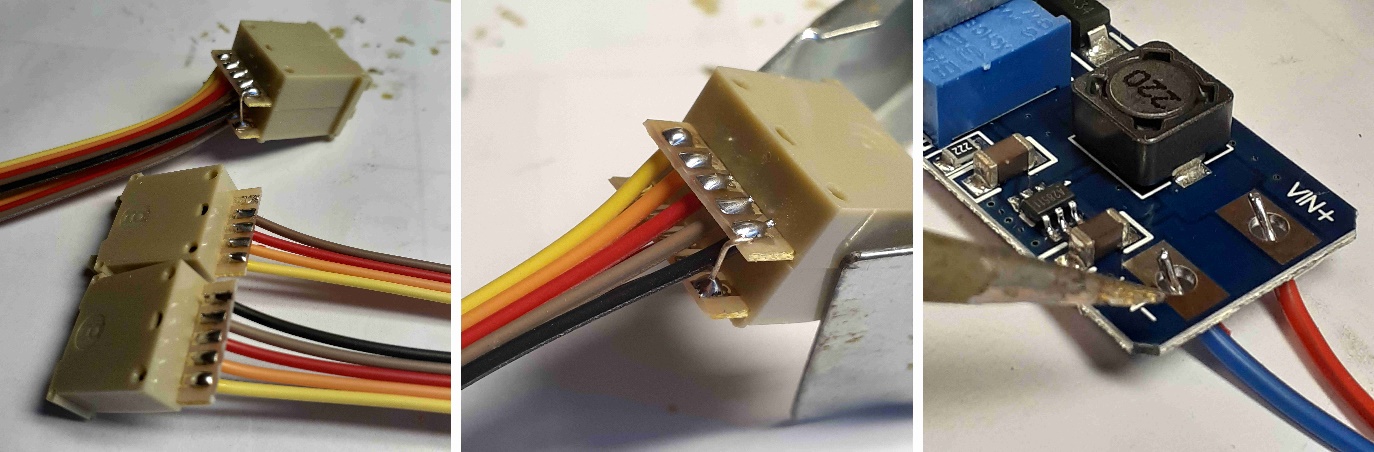


Figure 11: Bundle cable on encoding wheels and DC-DC converter

**Arduino NANO:**

Solder the pin headers on the side opposite the reset button and LEDs! Only solder the pins you need: 6 ports from GND to A5 and 12 ports from GND to D12. Pay special attention to the correct position of the pin headers! A breadboard makes it much easier to solder vertical pin headers.

Cover the “POW” and ‘TX’ LEDs with opaque material (insulating tape, heat-shrink tubing, “liquid rubber”). Their light interferes with the display of the “L” LED, which later indicates a low battery or invalid NMEA data.

The RST and A6 pins remain free for later use.

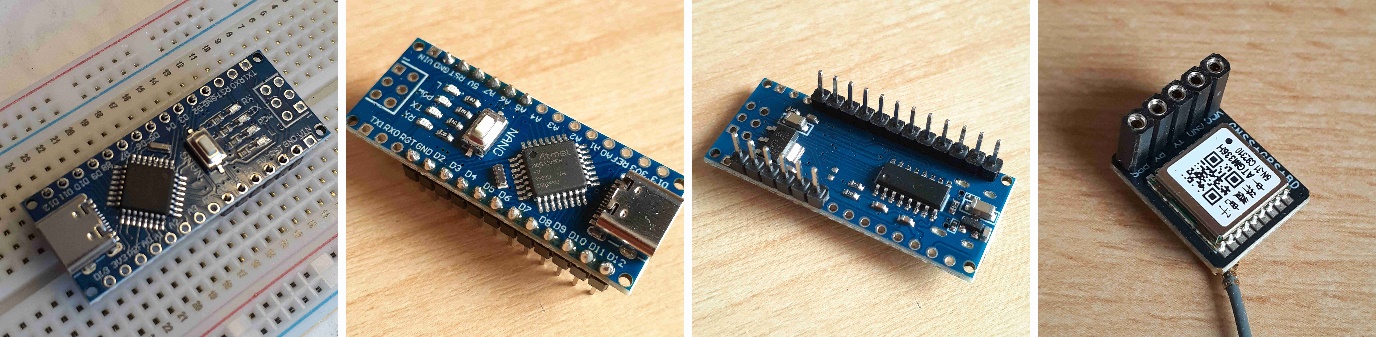


Figure 12: Pin and socket strips on the Arduino NANO and the GPS module

**GPS module:**

Solder the socket strip (5-pin) on the side with the large square IC opposite the side with the antenna connection.

The antenna cable must be pressed on very carefully with strong pressure and without tilting. It clicks into place and can then be rotated.   
Caution: The antenna connection must not touch the silver cylindrical component after rotation, as this short circuit will at least cause the GPS to malfunction.   
Lay the antenna cable without kinks! The antenna fits between the potentiometer and the housing. The side with the green circuit board and the metal box must be pointed towards the sky for optimal reception.

**Charger:**

The integrated charger is powered via the USB-C port and is directly connected to the battery. A red LED indicates that charging is in progress, while an orange LED indicates that the battery is fully charged. Charging takes approximately 2 hours.

Note: To ensure that the LED of the charging module is located in a specific position, it is recommended that the cables to the USB port be made of wire.

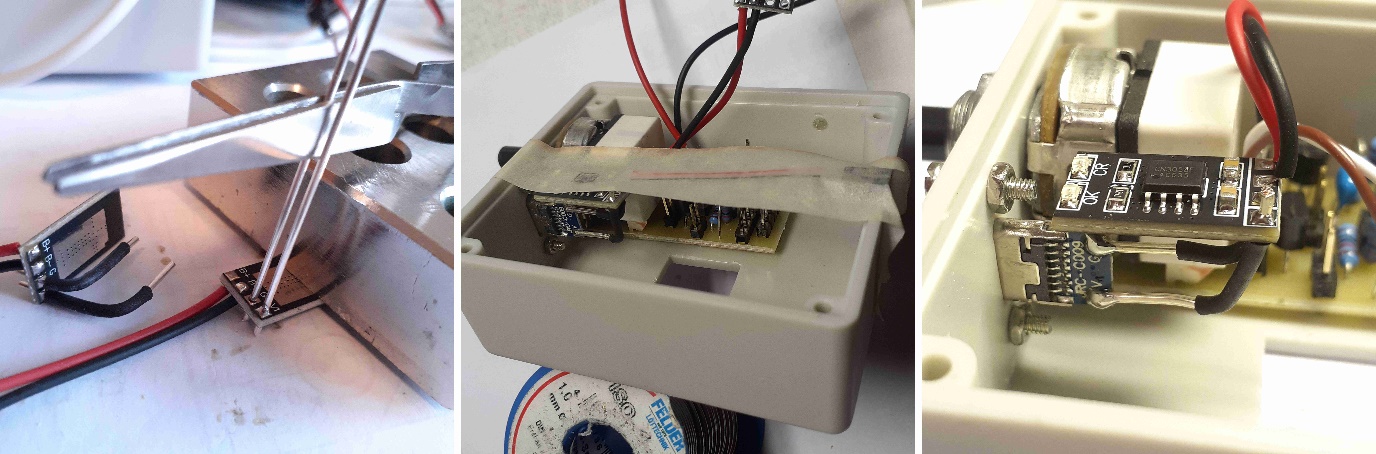


Figure 13: Charger for the LiFePo4 battery

**Housing:**

Here are some helpful tips for the mechanical work. It is best to have a small workshop and basic tools (files, saws, milling cutters, chisels, sandpaper, table drill, small hand drill, vise, workbench). The fewer tools you have, the more skilled you need to be in using them!

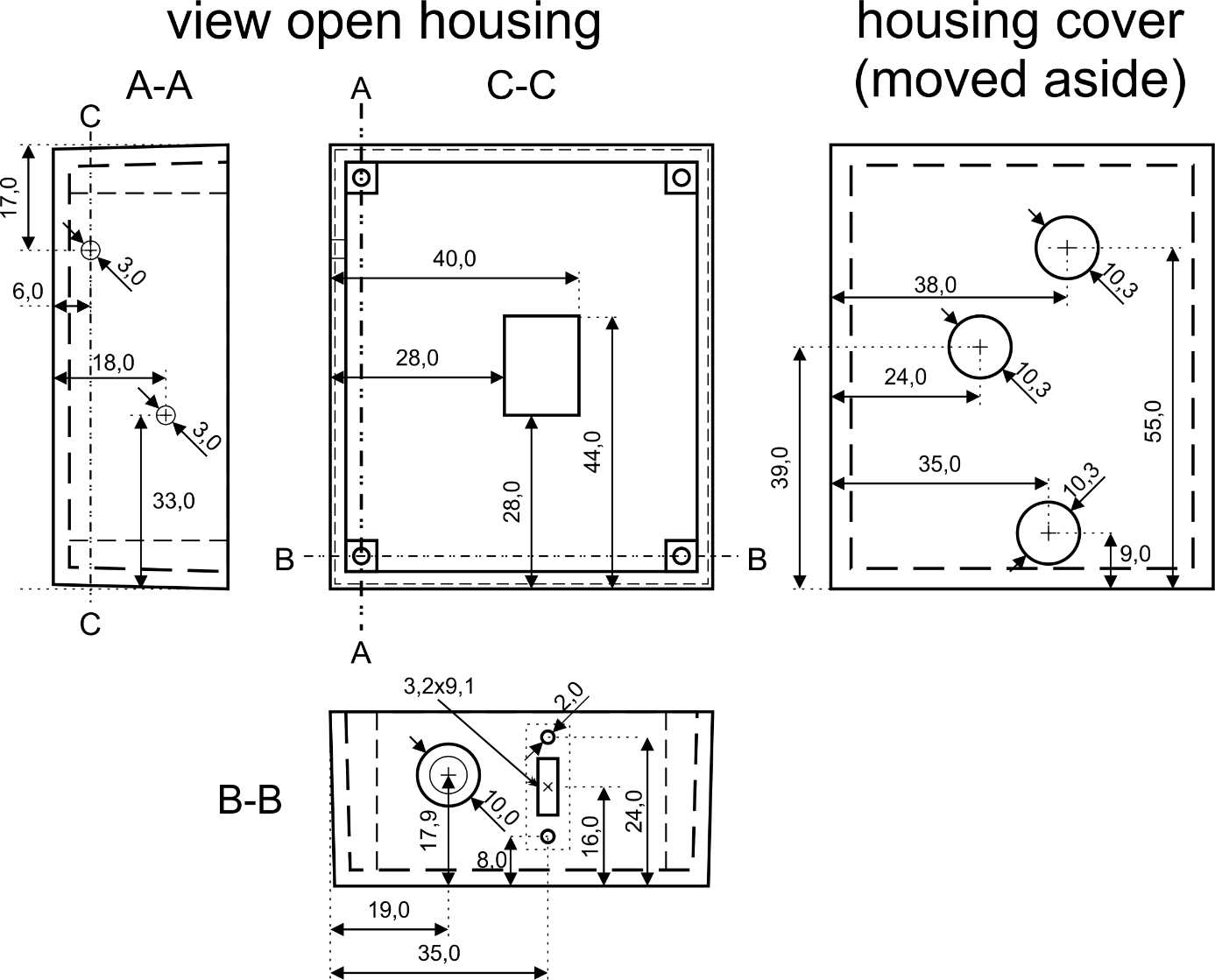


Figure 14: Housing dimension drawing

The rectangular opening for the **encoding wheels** must be positioned very precisely (0.5 mm) and designed as a press fit. Otherwise, the coding wheels must be fixed inside with hot glue.

Mark out the future recess. Cover the surface with adhesive tape so that the markings do not damage the housing. First, mill the recesses undersize on a bench drill by guiding the workpiece along a guide block. Mill the holes end milling step by step and then side mill them using the guide.



Figure 15: Making the opening in the housing for the encoding wheels

The fine work is done with a rectangular key file up to the clearance dimension. Always check with calipers to ensure a press fit for the counting wheels on the long side!

Make the slot for the **USB charging port** in the same way.   
Use a 3 mm round file to chamfer from the outside towards the screw holes so that the USB component fits flush against the housing. Position the USB component and drill the 2 mm holes accordingly. Use a screw for the second hole.

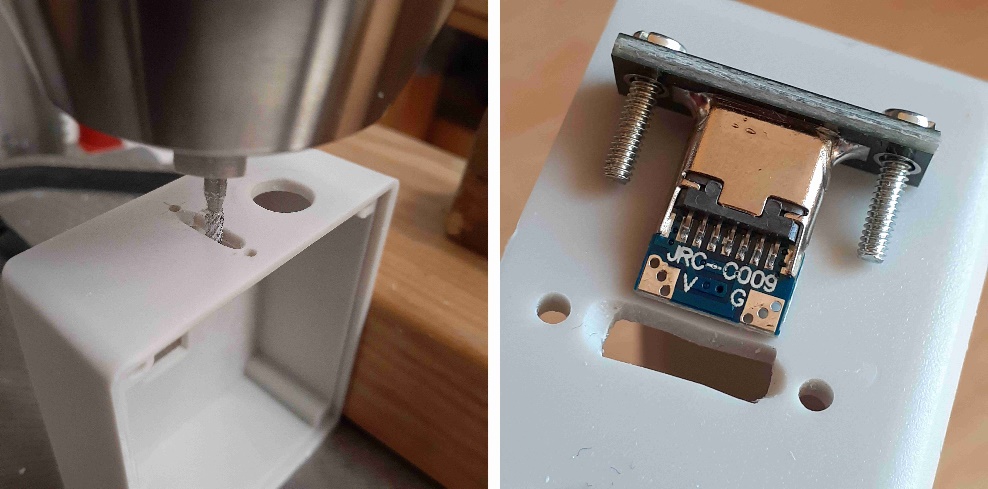


Figure 16: Making the opening in the housing for the USB-C charging port

The **switch rotary potentiometer** requires 0.1 mm accuracy in height, otherwise the cover will not fit on it later. If it does not fit in the end, make a flat milling or grinding on the inside of the cover with a precision hand grinder (D ~15 mm) or enlarge the hole with a round file.

The **battery holder** does not fit completely into the side of the housing. Use a chisel to remove the lower part of the two columns in the housing so that the battery holder fits completely into the side. It is not recommended to modify the battery holder to make it fit.

The positions for the LED lenses in the **housing cover** may need to be adjusted by a few millimeters compared to the dimensional drawing. Therefore, only do this work shortly before completion. Mark the 3 holes and drill them to 10 mm with a step drill. Then use a 10 mm round file to create a press fit for the lenses. Caution: The lens must be shortened flush with the inside of the cover for the GPS and Arduino NANO. Otherwise, pressure could be exerted on the GPS module and the reset button could be pressed on the Arduino. You should also consider flattening the inside of the cover at the location of the Arduino NANO's reset button.

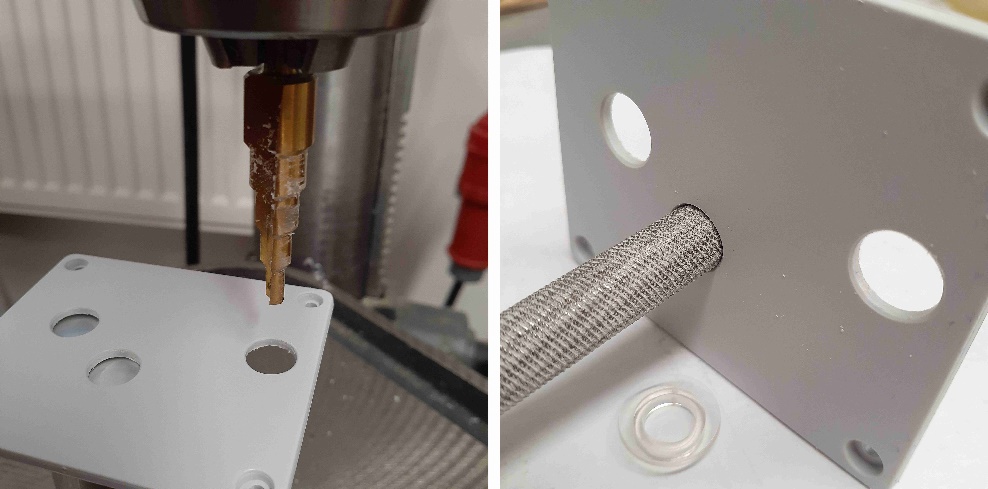


Figure 17: Making the opening in the housing cover for the “lenses”

If you do not want to use the LED lenses, you can also proceed as follows:   
Drill holes (5 mm) in the housing cover at the positions of the LEDs, deburr on both sides, and seal with clear hot glue. This creates slightly opaque viewing windows.   
The best way to do this is to use two pieces of Teflon (0.5 mm thick), place one under the cover on a metal table, and press the cover firmly against the table. Then put a suitable amount of hot glue into the hole, quickly place the second piece of Teflon on top, and press firmly with a finger in a glove (hot!) for about 20 seconds. After another minute, carefully remove the Teflon. There should now be no excess material on the outside; if there is, carefully smooth it down with a chisel on the outside and, if necessary, on the inside. If you are not satisfied with the result, you can drill out the holes and try again or enlarge the holes.

**Final assembly:**

Solder the 6- and 12-pin male headers to the Ardiono NANO as described and the 5-pin female header to the GPS module.

Solder plug connections to cables if they are not already equipped with female connectors.   
ATTENTION! With the 1PPS\_Teiler\_v5\_pro.brd circuit board, a 2200 ohm resistor must be soldered into the cable for the GPS data (TX in the middle of the GPS to NANO D8), otherwise the data will not be recognized by the Arduino NANO! With the 1PPS\_Teiler\_v6\_pro.brd, the resistor is applied to the circuit board.

Screw the USB charging socket onto the housing. Hold the lower nut with needle-nose pliers. The circuit board of the USB charging socket fits in between.   
Solder the USB charger (G, V1) to the USB connection (G, V) with short, sturdy wires, ensuring correct polarity. The position of the LEDs on the USB charger should be under the corresponding lens in the cover!   
Solder the plug sockets to the USB charger (B+, B-) cables. Connect the USB port to any power source and check that the red charging LED is lit.

Set the DC-DC converter to 5.0 V.

Feed the cable with the LED through the clamp into the housing and secure it inside with cable ties or hot glue. Solder the socket to the strands.

Solder the sockets to the cables from the battery holder.

Screw the switch rotary potentiometer into the housing and attach the rotary knob.

Attach the eyelet.

Thread the encoding wheel wires through the hole and press in the encoding wheels, securing them inside with hot glue if necessary.

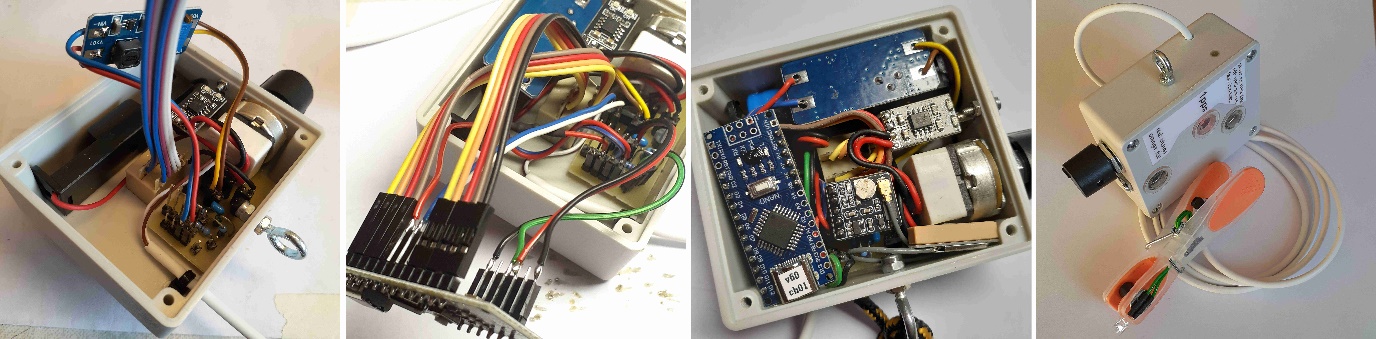


Figure 18: Positioning of the individual devices in the housing

Connect all plug connections correctly as follows: (port name/printed circuit board)  
From the charger (B+, B-) to the printed circuit board (+3.6V\_CHG)  
LED cable on the clip to the circuit board (LED)  
from the battery holder to the circuit board (+3.2V\_VBAT)  
from the DC-DC converter VOUT to the circuit board (+5V\_OUT)  
from the DC-DC converter VIN (+ and - bundled) to the circuit board (+3.2V\_IN)  
From Arduino NANO (D6) to circuit board (NANO D6), code: PinInPPS, PPS input  
From Arduino NANO (D7) to circuit board (NANO D7), code: SigOut, LED clip  
GPS data line from Arduino NANO (D8) to circuit board (RX, middle pin on the module)  
from Arduino NANO (A5) to circuit board (NANO\_A5), code: cutoffPinOut, deep discharge protection  
from Arduino NANO (A7) to circuit board (NANO\_A7), code: UBatanaIn, U battery input  
from Arduino NANO (GND, 5V) to circuit board (+5V\_OUT1)

Connectors from the encoding wheels to the Arduino NANO: GND, D2...D5 from the left digit, D9...D12 from the right digit, and finally the GPS module with antenna (GPS) to the five long pins. Place the antenna in the free space next to the potentiometer and lay the cable carefully.

Then insert the LiFePo4 battery into the battery holder. Switch on for electrical testing and check the function of the device in modes #00 and #01 outdoors.

It is helpful to use a 3.2 V regulated power supply with a current limit of 150 mA instead of the battery at first. This prevents anything from burning out. The current consumption must not exceed 100 mA. Always ensure correct polarity!

Gradually shape the cables so that everything fits into the housing and there are no unwanted contacts or high forces on the pins. Lay the cables as deep as possible in the housing, otherwise the space in the cover area will be too tight. The cables of the coding switches can be routed under the Arduino NANO.

The GPS module should be able to be plugged in without applying excessive force against the wires. To do this, bend the wires on the connectors under the module at a right angle in the appropriate direction before plugging them in. Space for all the wires is limited.

Finally, nothing should protrude over the edge of the housing, then the cover will fit without pressure. This can be easily checked by looking over the edges or using a ruler. If this is not possible due to inaccuracies, the cover can be sanded down in the appropriate places.   
Put the cover on. It should fit flush with light pressure. If this is not possible, make sure that the Arduino NANO and the DC-DC converter are not pinched.   
Screw the cover tight.

After another successful function test, especially with one of the special functions that require GPS data, e.g. #99, the device is ready for use!

When reopening the housing and removing the components, care must be taken to ensure that the wires do not slip off the pins and that the connections are noted beforehand (photo) so that they can be reconnected correctly.   
Too much movement on soldered wires can quickly lead to them breaking!

**Parts list:**

LED 3000 mcd with clip, complete, cable ties  
Housing, 3 lenses (machined)  
DC-DC converter 3.2 to 5.0 V  
Arduino NANO programmed, tested  
GPS module NEO-M8N GPS BDS dual-mode module ATGM336H, 5N1-31 with antenna  
LiFePo4 rechargeable battery, battery holder  
LiFePo4 charger, 1 mm wire, heat-shrink tubing  
Connecting circuit board  
Assembly parts, rotary potentiometer Pins aligned, shaft shortened by 10 mm, rotary knob  
Wire bridge  
USB connector, 2 pcs M2x8, 2 nuts  
BCD encoding wheel, 2 pcs   
Round eyelet 3 mm with 2 nuts (one self-locking)  
Liquid rubber  
Plug strips, socket strips:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Quantity | 2 | 3 | 4 | 5 | 6 | 12 |
| Pin | 7 | 1 | 2 | 1 long | 1 | 1 |
| Socket | 8 | 4 |  |  |  |  |

Table 3: Required connector strips, socket strips

All cables (mm):

|  |  |
| --- | --- |
| red-black: 100 (+3.2V\_VBAT) | red-black: 80 (+3.6V\_CHG) |
| brown-yellow: 80 (+5V\_OUT) | red-blue: 80 (+3.2V\_IN) |
| white: 80 (D6) | blue: 80 (D7) |
| red: 80 (D8) | red-black: 80 (+5V OUT1) |
| green: 80 (A7) | black: 80 (A5) |
| 4 + 5 bundles for BCD encoder wheels | 2-pin cable to LED on clip: 1000 |

Table 4: Cables

**Summary:**

It has been shown that a simple idea can be used to create a cost-effective method for accurate time stamps in astronomical videos in the range of less than a millisecond for any optics. The process of building your own device has been shown in detail [1]. The source code is available [1]. The author provides support in case of problems with parts procurement (suppliers) and has several devices for his own use, which can be borrowed for testing.

**Sources:**

[1] open-source code 1pps flasher  
<https://github.com/leontodon1/1pps_flasher>

[2] PyMovie, PyOTE  
<https://occultations.org/documents/Windows-PyMovie-PyOTE-Anaconda3-installation%202020%20updated%2012-2020%20.pdf>   
<https://pypi.org/project/pymovie/>

[3] EAGLE, <https://en.eagle.cool/download>

[4] Arduino IDE 2.3.7, Programming Arduino NANO  
<https://www.arduino.cc/en/software/>

[5] AltSoftSerial.h: open-source, Paul Stoffregen,   
<https://github.com/PaulStoffregen/AltSoftSerial/blob/master/AltSoftSerial.h>

[6] TinyGPS++.h: open-source, Mikal Hart   
<https://github.com/mikalhart/TinyGPSPlus>

Translation with [deeple.com]   
<https://www.deepl.com/de/translator>)

**Version:**

Arduino: v61  
PCB: 1PPS\_Teiler\_v5\_pro  
Date: 2026/02

**List of figures:**

|  |  |
| --- | --- |
| Figure | Description |
| Figure 1: | GPS pps speed camera, finished device |
| Figure 2: | Mode #86, minute 35 |
| Figure 3: | Electrical connections illustrated |
| Figure 4: | Circuit diagram of the connection circuit board with EAGLE |
| Figure 5: | Circuit board conductor side, assembly, combined (EAGLE) |
| Figure 6: | Electrical connections |
| Figure 7: | Soldering cables and pins |
| Figure 8: | Required plug connections |
| Figure 9: | Cable lengths and types illustrated |
| Figure 10: | Circuit board for connecting the devices |
| Figure 11: | Bundle cable on encoding wheels and DC-DC converter |
| Figure 12: | Pin and socket strips on the Arduino NANO and the GPS module |
| Figure 13: | Charger for the LiFePo4 battery |
| Figure 14: | Housing dimension drawing |
| Figure 15: | Making the opening in the housing for the encoding wheels |
| Figure 16: | Making the opening in the housing for the USB-C charging port |
| Figure 17: | Making the opening in the housing cover for the “lenses” |
| Figure 18: | Placing the individual devices in the housing |

**Glossar:**

|  |  |
| --- | --- |
| Abbreviation | Description |
| Arduino NANO | Microcontroller type |
| BCD | Binary Coded Decimals, 0 to 10 encoded in binary on 4 bits |
| DC-DC | Direct Current, DC converter |
| EXTA | Exposure Timing Analyzer |
| fps | Frames per second, images per second |
| GPS | Global Positioning System |
| LED | Light Emitting Diode |
| LiFePo4 | Lithium iron phosphate battery type |
| LSB | Least Significant Bit |
| LUT | Look Up Table |
| MSB | Most Significant Bit |
| NMEA | National Marine Electronics Association, standard for GPS data |
| open source | Open, freely available, may be modified |
| pps, 1pps | Peak per second, signal per second |
| USB-C | Universal Serial Bus, Type C, used here only for power supply |
| VTI | Video Time Inserter, device for generating time stamps |