ETH zürich

FACT
First G-APD Cherenkov Telescope

Technical Note

The FACT GPS Module

Max Ludwig Knötig

October 2013

For some time now the collaboration had ideas that required to know very precicely at what time the triggered showers occur. In particular it became clear that the Crab pulsar spectrum has no apparent cutoff below 400GeV. Furthermore, in order to study the reconstruction, it would be nice to have a way of cross calibrating with MAGIC showers. The FACT GPS module is the solution to the accurate timing problem. This document starts with an introduction about the Physics motivation. Then I present the system from the hardware side. In the third section, the custom firmware is explained, followed in the fourth section by a description of the ethernet server commands. Last, in the fifth section, I give the lab tests and their results.

Contents

1 Introduction

1.1 Physics Motivation

During the design of the FACT electronics it was decided a very precise timing not being necessary. It was not evident that pulsars emit such high energy gamma-rays that FACT could observe them. In fact, most Pulsars are believed to have a cutoff at some 10GeV. The findings from MAGIC[1] and VERITAS[2] however changed the picture. Their results indicate that pulsed signal above FACT threshold might exist, and since we need to take a large amount of Crab data for calibration purposes, these data would be an ideal sample to look for pulsed signals. Also, it became clear that in order to cross calibrate the FACT telescope with the MAGIC telescopes, we would need a way of identifying the simultaneously seen showers. On the 11. September Adrian wrote an email with his view of the situation and a call for action to build a system that allows us to measure the absolute timing to μs precision.

1.2 Timing Problem Solution

The initial idea:

> Using a GPS to get precise timing for FACT
> ...
> While there is no precise absolut timing signal, the timers on the FAD boards deliver good relative timing information. It seems possibile to synchronize these to an absolut time: There exists a connector for an external trigger signal. If a precise clock (e.g. GPS) can be connected there, this can be (ab)used for the needed synchronization.
> Commercial GPS receivers typically deliver following signals: a) PPS pulse per second: a pulse synchronized to UTC seconds with a precision of 200picoseconds and duration of >10msec (typical pulse length 100msec) b) an ASCII string containing detailed information about the last PPS over RS232 or USB
> One is tempted to use the PPS for the external trigger, and feed the RS232 to the DAQ computer. Unfortunately, due to the ethernet based readout we cannot exclude delays between camera and computer on the level of several seconds. Therefore, an information with longer intervalls is needed to ensure correlation between FAD and computer. The duration of the Signal is no problem, since the FTM is looking for rising edges only; nevertheless the pulse must probably be modified, since the FTM needs NIM logic.
> ...
> Looking around: GPS with precise PPS can be bought for less than 50 Euro.
> ...

> Possible implementations:
> ...
> c) Buy an inexpensive GPS with PPS. In addition, a FPGA/ microcontroller/Arduino/.... is reading the ASCII string. In case the ASCII string indicates a time XX;XX:59, veto the next PPS. ...
>
> Adrian Biland, Email 11.Sep 2013 fact-online mailinglist

In the following I would like to present this idea worked out in detail.

2 Hardware

2.1 Box Outside

This is the resulting FACT GPS module box:

Figure 2.1: The FACT GPS module

The box is a plastic 150mmx80mmx80mm box to house the electronics. On the outside there are three status LEDs and four connectors for power, ethernet, the antenna and for the trigger.

2.1.1 Connections

Four connectors sit the sides of the box, two on each small side. They look like in Fig. 2.2. Each connector and LED has a label attached. The connectors are used for connecting the power to the arduino with a standard 12V DC power supply. The power connector and the ethernet connector are mounted on the Arduino Ethernet. The SMA GPS antenna and the LEMO trigger signal connectors are feedthroughs.

Figure 2.2: The FACT GPS module sides

2.1.2 LEDs

The LEDs, located on the same side as the antenna and trigger feedthroughs, can be used for checking the system status:

- The left LED is the Arduino LED. It is ON when the Arduino is switched on. It is BLINKING with the rate/2 that the Arduino parses (if at all) the NEMA message from the GPS chip (typically 8Hz/2)
- The middle LED is the Venus GPS LED. It is ON when the GPS chip is on. It is BLINKING when the chip has a GPS lock with a rate of the PPS/2 = 0.5Hz
- The right LED is the Veto LED. It is ON when the VETO is active, i.e. no triggers are sent. It is OFF when the Veto is inactive. When in veto_60 mode it is ON for one second during the veto around the 59 - 00 GPS UTC second.

2.2 Box inside

The opened box has the electronics attached to the top. It looks like in Fig. 2.3 Attached to the Arduino one can see the USB2Serial module used to program the microcontroller.

2.2.1 Overview Schematic

The schematic is shown in Fig. 2.4 It basically consists of the Sparkfun Venus GPS module, a TI CD74AC00 NAND gate, and some basic electronics components. Everything is povered via the 5V Arduino breakout. The 5V are converted via two diodes down to about 3.3V in order to power the Venus GPS chip. When no current is flowing, the voltage inbetween the 5V Arduino pin and the GPS chip would be 5V, if not for the resistor R1.

The Arduino talks to the GPS chip via its serial communication pins RX and TX, where its NEMA message is parsed. The PPS from the GPS chip is fed to the NAND gate. Whenever

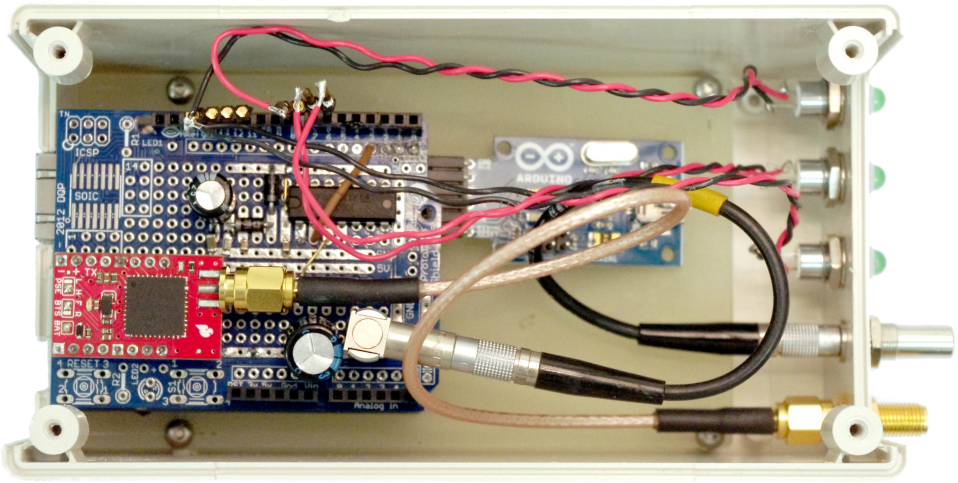


Figure 2.3: The FACT GPS module opened

the Arduino reads seconds = 59, the active low VETO is enabled, and the PPS signals are vetoed. In this way it is later possible to convert the TTL signal easily into NIM signals.

The last part of the board is used to convert the active low TTL signals into NIM pulses. It uses the fact that the coax cable as viewed from the NAND gate has an impedance of 50Ω, that can be used in a voltage divider with resistor R2 in order to get the signal amplitude down. The capacitor C4 is used to AC couple the signal to the output. This way, the active signal becomes negative. The resistor R3 is chosen such in order to make the AC coupling constant large enough for the 4ms PPS signals. A small capacitor C5 is used to make the rising edges fast (<5ns).

Overall the following components were installed:

- 150mmx80mmx80mm plastic box
- Three green LEDs with feedthroughs
- One SMA cable and feedthrough for the antenna
- One active GPS SMA antenna, 2m, magnetic in order to stick to roofs
- One Lemo connector, one Lemo cable and one Lemo feedthrough for the trigger signals
- One Arduino Ethernet
- One Arduino Stackable Protoshield V5
- The Sparkfun Venus GPS module with SMA connector

- One USB2Serial board in order to program the Arduino
- One Ti CD74AC00 fast NAND gate
- Various electronics components according to the schematic Fig. 2.4 from the workshop
- One 12V DC powersupply
- Cables to connect to the powernetwork and the ethernet
- One Lemo cable plus a Lemo to BNC adapter to connect the box t'o the external trigger-in

2.2.2 Arduino Ethernet

Digital I/O pins TX and RX on the Arduino Ethernet are used for the serial communication with the Venus GPS chip. These are also used in order to program the Arduino via the USB2Serial adapter. The serial line can, however, only be used by one device at a time and thus the GPS module has to be unplugged, in order to program the Arduino.

Digital I/O 2 is used to recieve the message NAV from the GPS chip. This pin changes states, whenever the GPS chip sends out a PPS signal while having a GPS lock on. It is therefore used to steer the GPS Led. Digital I/O 3 is used in order to veto the PPS on on the NAND gate. Digital I/Os 6,7,8 are used to steer the Veto, GPS, and Arduino LEDs.

The +3V pin from the Arduino does not supply enough current for the GPS chip ($\sim$90mA), the antenna ($\sim$50mA), and the LEDs ($\sim$30mA each) together. Therefore the GPS chip and antenna are attached to the +5V power line with an adequate Diode converter. The LEDs are powered directly via the digital I/O pins. The NAND gate draws negligible current.

2.2.3 Protoshield

On the protoshield all the components are handplaced and soldered. The top layer is used for the various SMD capacitors, resistors, etc. The bottom layer is used for routing with soldered wires.

Arduino Stackable Headers connect the protoshield to the Arduino and are used from above in order to connect the LEDs. In Fig. 2.5 I show how the Arduino, the GPS module and the shield fit together.

2.2.4 Venus GPS

The Sparkfun GPS module is connected to the protoshield via normal pin headers. This way the GPS module can be removed, reinstalled, and replaced in an easy way. It is the small red module in Fig. 2.3.

2.2.5 USB2SERIAL

In order to program the Arduino Ethernet a serial connection is needed. The GPS module uses the same communication path and has to be unplugged prior to programming the Arduino. Then the USB2Seial adapter can be plugged into the serial connection pins of the Arduino.

Together with a USB cable, the serial connection to and power supply from a computer can be established. The USB2Serial module is unplugged in the standard configuration. I delivered it to Watz, who took it to LP. In case we need to program the firmware of the Arduino, someone has to physically open the box (no USB feedthrough), unplug the GPS module, and plug the module with USB cable in. The module is shown in Fig. 2.6

2.3 Signals

Schematically the idea of the FACT GPS module is shown in Fig. 2.7. Every UTC second the GPS sends a LVTTL signal of 4ms length called Pulse Per Second (PPS). At the same time, the serial message from the GPS is parsed and interpreted on the Arduino. Then, when reading UTC second = 59 the Arduino activates the Veto. The Veto is active low, because later downstream, the signal has to be transformed from a TTL to a NIM pulse. This is done simply by a reasonable AC coupling together with a voltage divider.

After reading UTC seconds != 59 the Arduino stops the Veto and the PPS signals are let through.

In the real system, the trigger signal is fed through a ∼40m cable that goes to the FACT camera. Therefore a delay of the trigger with respect to the real UTC second of about ∼200ns is introduced. This delay however is small.

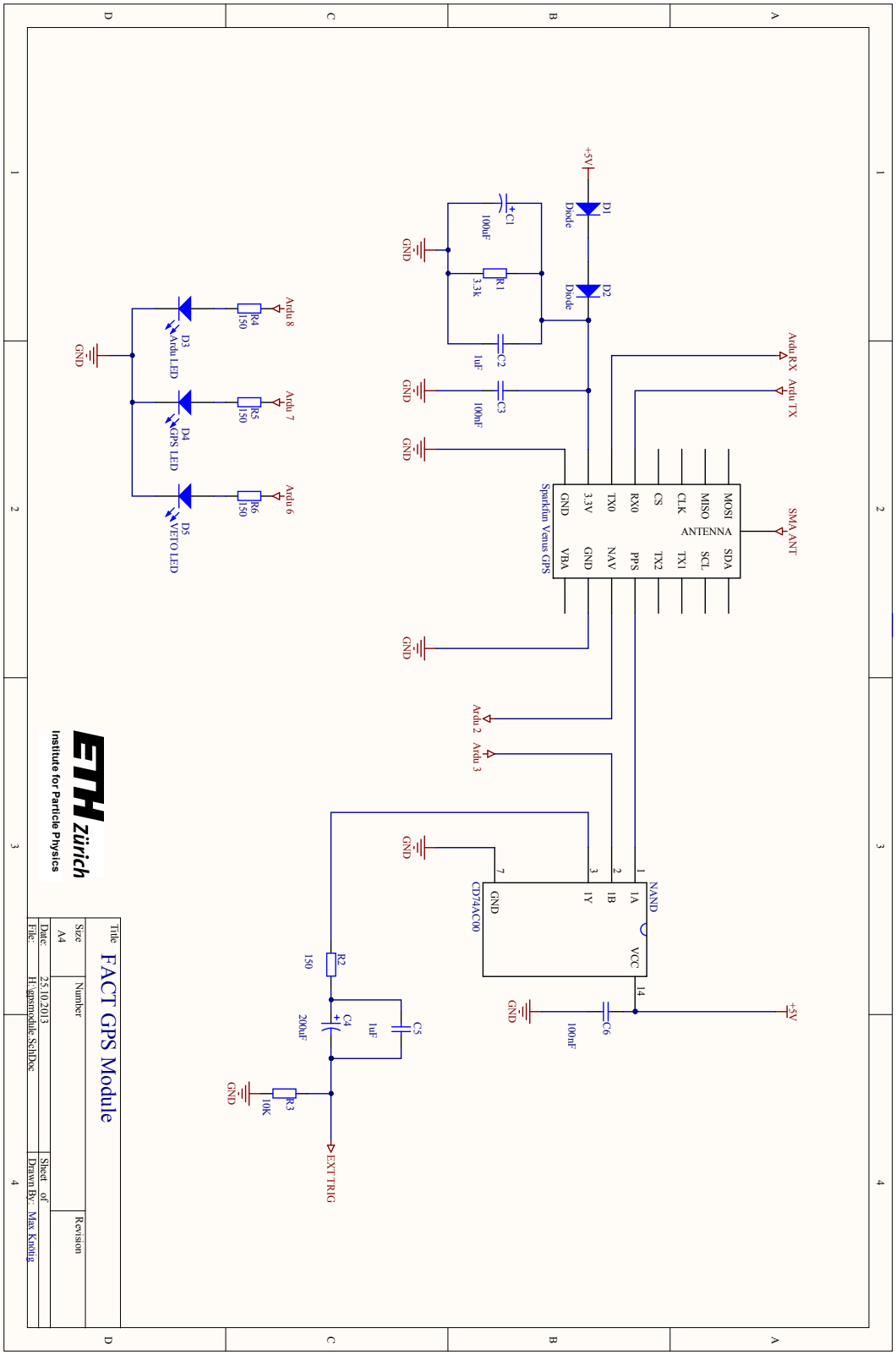


Figure 2.4: The FACT GPS module schematic

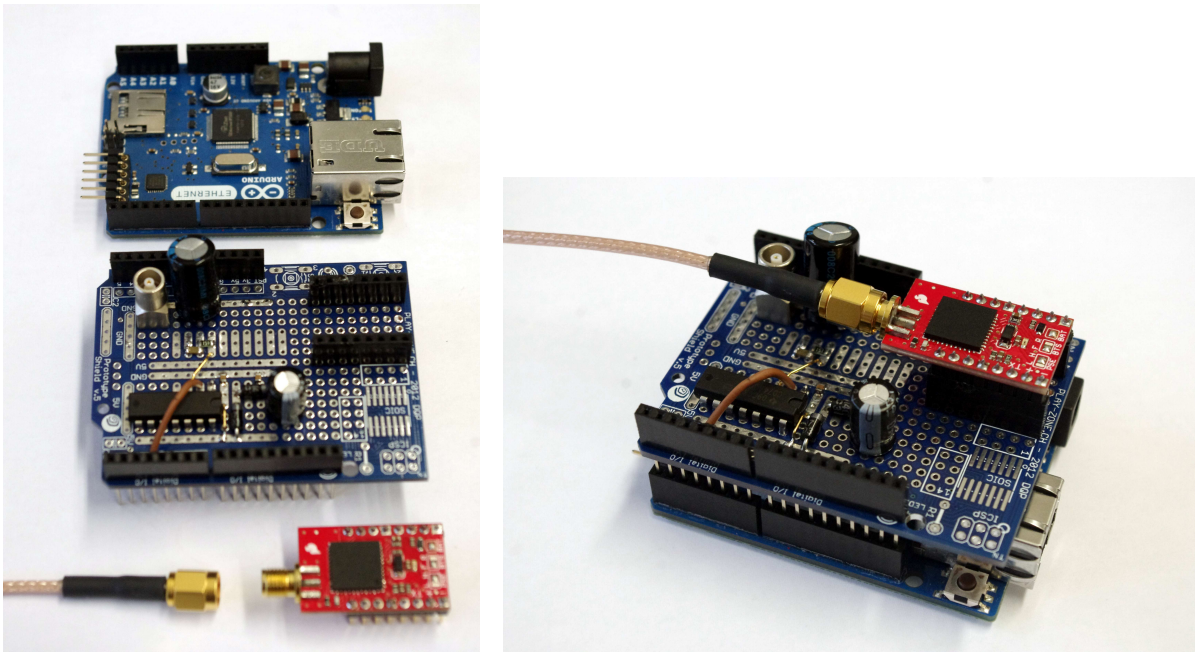


Figure 2.5: Left: The Arduino + Shield + GPS chip + SMA antenna. Right: Assembled

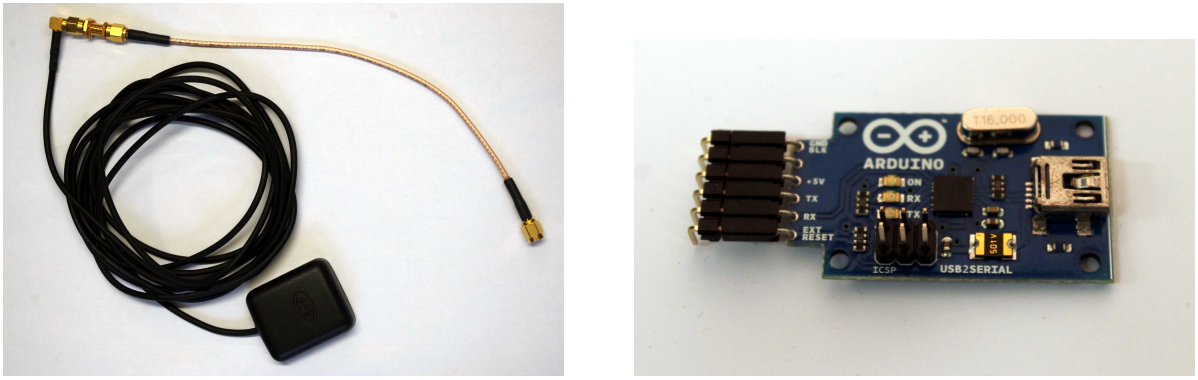


Figure 2.6: Left: GPS Antenna. Right: USB2Serial module

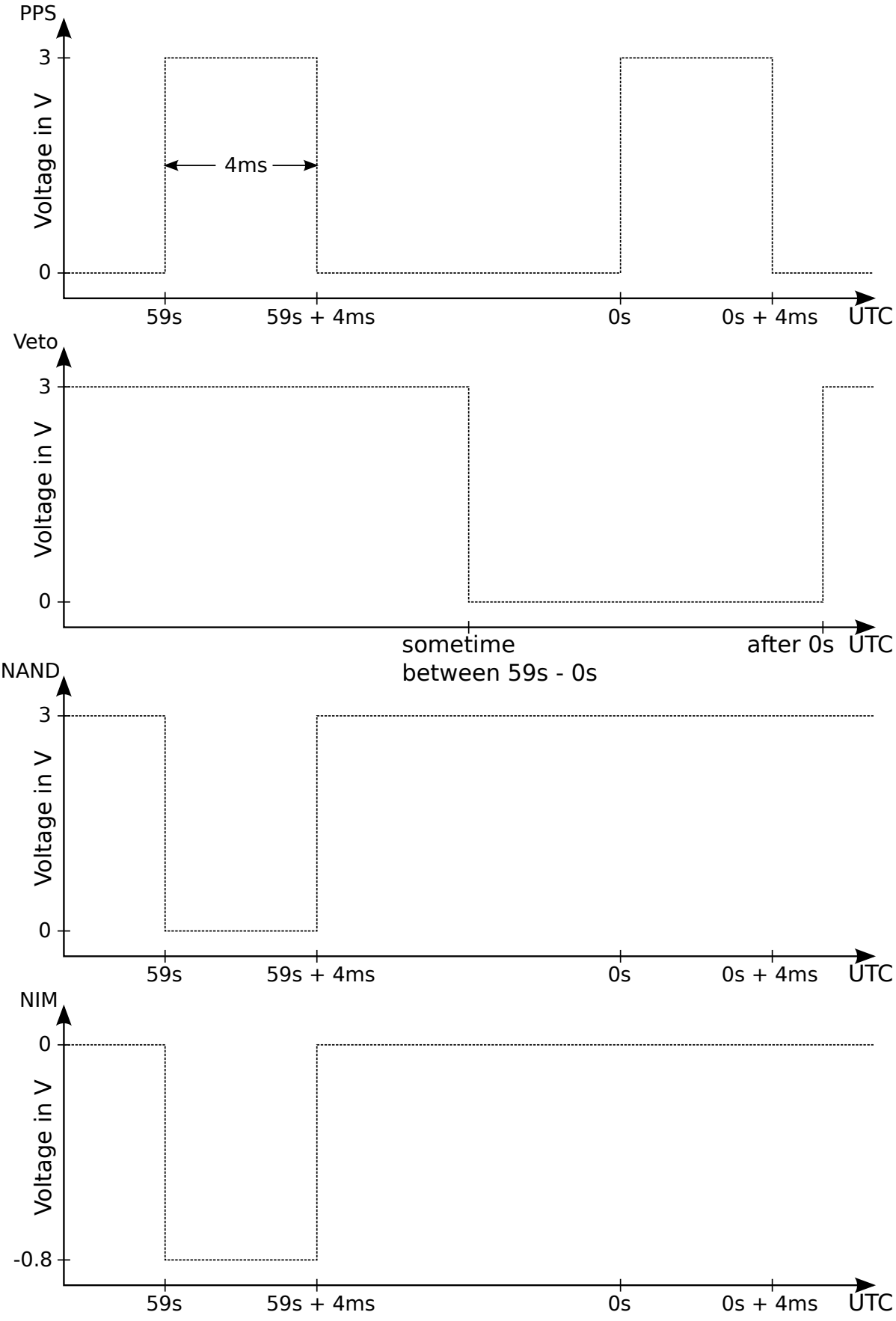


Figure 2.7: Sketch of the signals in the module

3 Firmware

3.1 Venus GPS

The Venus GPS chic was delivered with a firmware set to 9600 baudrate and to send all available NEMA messages once every second. This needed to change. In order to program the Venus GPS firmware, I downloaded the GPS Viewer / Configuration Software from the sparkfun website. The GPS module was connected to the computer via a standard FTDI cable, a prototyping breadboard, and some wires. The software runs under windows and the needed drivers were not preinstalled, but could be downloaded from the FTDI website (drivers -> VCP Drivers).

The firmware was changed to send only GPGGA as Nema message 8 times per second. In this way, the Arduino has 8 chances per second to enable the Veto. In order to send with this rate, the baudrate was increased to 115200.

3.2 Arduino

The full FACT GPS Arduino firmware vers. 1.0 is downloadable from the FACT La Palma repository:

```
svn co https://www.fact-project.org/svn/firmware/GpsArduino GpsArduino_firmware
```

3.2.1 Serial Communication, NEMA Parsing

The Arduino uses a loop() function and a setup() function in order to work. Additionally a state machine is implemented, that has three states: Veto off, Veto every full UTC minute and Veto on.

In the loop() first the serial communication is checked. if no byte was recieved via serial the arduino continues with ethernet. If a client is available, the arduino awaits the full ethernet message, responds accordingly and, depending on the message, changes its state.

It then continues with the next serial byte. If the full serial message was recieved, the message is parsed via sscanf:

```
//check if recieved message is gpgga compatible
  if ( byteGPS == '\n' ) {      // when '\n' read (i.e.
                                // end of message ),
                                // check if variables are
                                // at the correct pos. then continue
      if( sscanf( buf,"$GPGGA,%2d%2d%2d.%d,%d.%d,%c,%d.%d,
                  %c,%d,%d,%d.%d,%d.%d,M,%d.%d,M,%7s\r\n",
                  &dump,&dump,&seconds,&dump,&dump,&dump,&cdump,
```

```
                    &dump,&dump,&cdump,&dump,&dump,&dump,&dump,
                    &dump,&dump,&dump,&dump,&hexdump ) != 19 )
                {
    return 0;
    }

  blinkLed(0);          // message parsed, blink arduino led
  decideVeto();         // found the interesting part, do something
  nemaMsg=buf;          // save message to nema string
  // remove \r and \n from the string
  nemaMsg = nemaMsg.substring(0,counter-1);

  return 0;             // return 0 as we found the interesting part
}
```

The usual floating point designator %f does not work in Arduinos (because floating point stuff takes alot of computing power) and therefore a structure such as %2d.%d has to be used. In total 19 numbers are compared (all but the seconds are dumped) and only if all are sucessfully parsed, the Arduino decides to activate the Veto or not.

3.2.2 Ethernet

The arduino starts an ethernet server on port 23 (telnet port). The IP in La Palma is 10.0.100.112, but Watz has adopted our nameserver such that one can access it via "gps":

```
>telnet gps
```

In the following chapter the commands are explained in detail.

3.2.3 Special Remarks

Only after delivery, Dominik made me aware of a mode in which the loop() for the Arduino would "prefer" in a sense the recieving of the serial NEMA messages, and do so until the full message was recieved. In the current implementation, this is only true for the ethernet connection. That means one can saturate the Arduino with ethernet messages and prevent any interpretation of the serial message. In the real system the ethernet communication should therefore not be used more often than once per second and not during the critical 59s-0s UTC.

4 Ethernet Server Commandos

4.1 How to Connect

The arduino starts an ethernet server on port 23 (telnet port). Any communication can be realized with the help of the "socket" library in Python, the telnet program or any other way of calling the socket() system routine:

```
fact@gate:~> telnet gps
Trying 10.0.100.112...
Connected to gps.fact.local.
Escape character is '^]'.
get_nema
$GPGGA,133616.405,2845.7026,N,01753.4691,W,1,10,1.1,2197.3,M,34.8,M,,0000*7D
quit
Connection closed by foreign host.
fact@gate:~>
```

Listing 4.1: A small Python client

```
import socket
import sys
from time import sleep

#good old while true
while True:

        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # this is the arduino ip in the eth network
        server_adress = ('10.0.100.112',23)

        sock.connect(server_adress)
        # the command to get the answer from the arduino
        sock.send('get_nema\r\n')

        sleep(1000./1000.)
        # save the recieving message
        message = sock.recv(10000)

        sleep(1./1000.)
```

```
        print(message)
        sleep(5.)

        sock.close()
```

4.2 Commandos

There are seven commandos that can be issued by the client to the Arduino server. The first is *help*. This prints an explanation of all the other six commandos: get_nema, get_status, veto_off, veto_on, veto_60, quit:

```
fact@gate:~> telnet gps
Trying 10.0.100.112...
Connected to gps.fact.local.
Escape character is '^]'.
help
-- GPS Clock Arduino Help --
---- Autor: Max Knoetig ----
--- mknoetig@phys.ethz.ch --
get_nema  :  get the last complete NEMA message
get_status:  get the status, veto_off, veto_60 or veto_on
veto_off  :  switch off veto
veto_on   :  switch on veto
veto_60   :  veto every pps to the full GPS UTC minute
quit      :  close the connection
```

Please note that the commando *veto_on* activates the Veto and therefore *disables* the FACT GPS module. In return *veto_off* lets every PPS UTC trigger through.

The commandos veto_off, veto_on and veto_60 are answered by: veto now off, veto now on or veto 60 now on. The get_status commando is answered by either: veto_off, veto_on or veto_60. The get_status commando was introduced into the Arduino by Watz and me on the island and was not tested beforehand. It was, however, verified to work on the island.

4.3 Limitations

Remember: in the real system the ethernet communication should not be used more often than once per second and not during the critical 59s-0s UTC. The details are explained in the previous chapter about the firmware.

5 Lab Tests

5.1 Raw PPS from Venus GPS

The first test performed was to see what the PPS signal looks like.

Test set-up: Sparkfun Venus GPS, DC power supplied by voltage source. Antenna attached and hold outside of the window. The PPS signal from the Sparkfun module PPS pin was fed via a Lemo cable into a Tek digital scope with 50Ω termination. The serial communication was observed with a probe.

Test findings (Fig. 5.1): The PPS signal comes after the Venus chip has a lock on the GPS satellites. This can take from seconds up to minutes, depending on the view of the sky. The PPS signals do have a delay of 1s inbetween, as measured by the scope. They have a height of ∼-1.3V at 50Ω, good for our purposes (NIM ∼-0.8V at 50Ω usually). The serial connection was visible on the scope as well (Fig. 5.1)

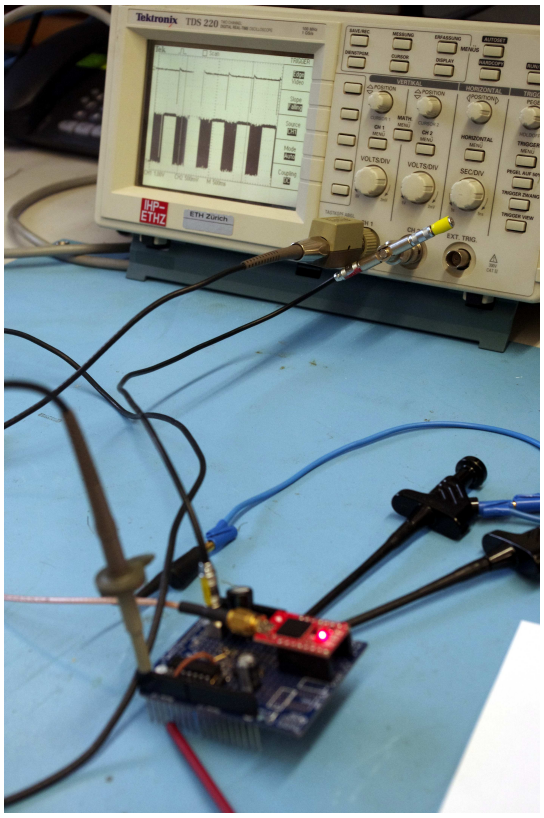


Figure 5.1: The test setup.

5.2 NAND @ DC-Voltage Source

Test set-up: the same setup as before, but in addition, the Veto pin was manually activated with a voltage source for one second.

Test findings (Fig. 5.2): NAND works just fine and the PPS is vetoed.

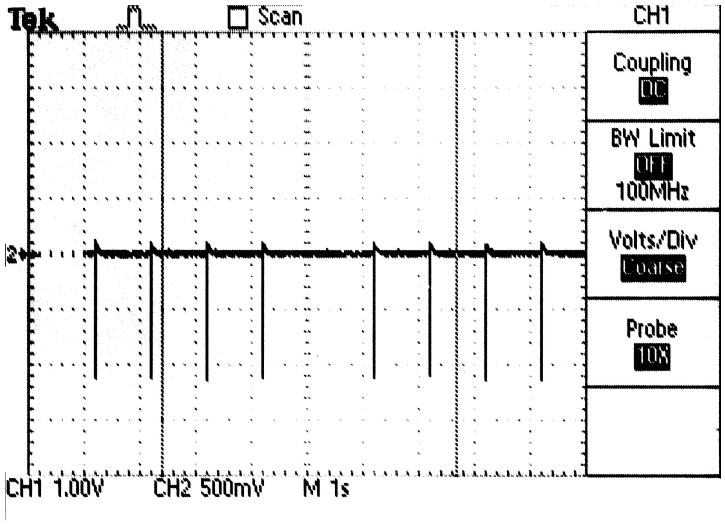


Figure 5.2: The vetoed PPS.

5.3 44m Lemo Cable Transmission

Test set-up: In order to test if the FACT GPS module can feed the FTM through 40m of coaxial cable, the signal was fed into a 22m long Lemo cable from the old L3 experiment, then fed into another 22m segment and then measured at the oscilloscope. in parallel the signal was directly fed into the scope via a T-Lemo element (without 16Ω impedance matching in all directions). The GPS module was tested when it was still unprotected by a plastic housing, but fully functional.

Test findings (Fig. 5.3): The initial "NIM" pulse is about -1.3V deep, 4ms long and has rising edges faster than 5ns (bandwidth limit by scope). After travelling through 44m of cable the pulse the pulse has lost less than 10% of voltage. The rising edge is now ∼50ns long. Both numbers are fine for us.

5.4 veto_60

Test set-up: Bare fully set-up FACT GPS module with antenna outside of the window and GPS lock, attached to a scope. The exact UTC second was measured by observing an accurate web clock. The state of the arduino was set by default to veto_60.

Test findings (Fig. 5.4): Every PPS triggered and every full UTC minute was vetoed by the arduino.

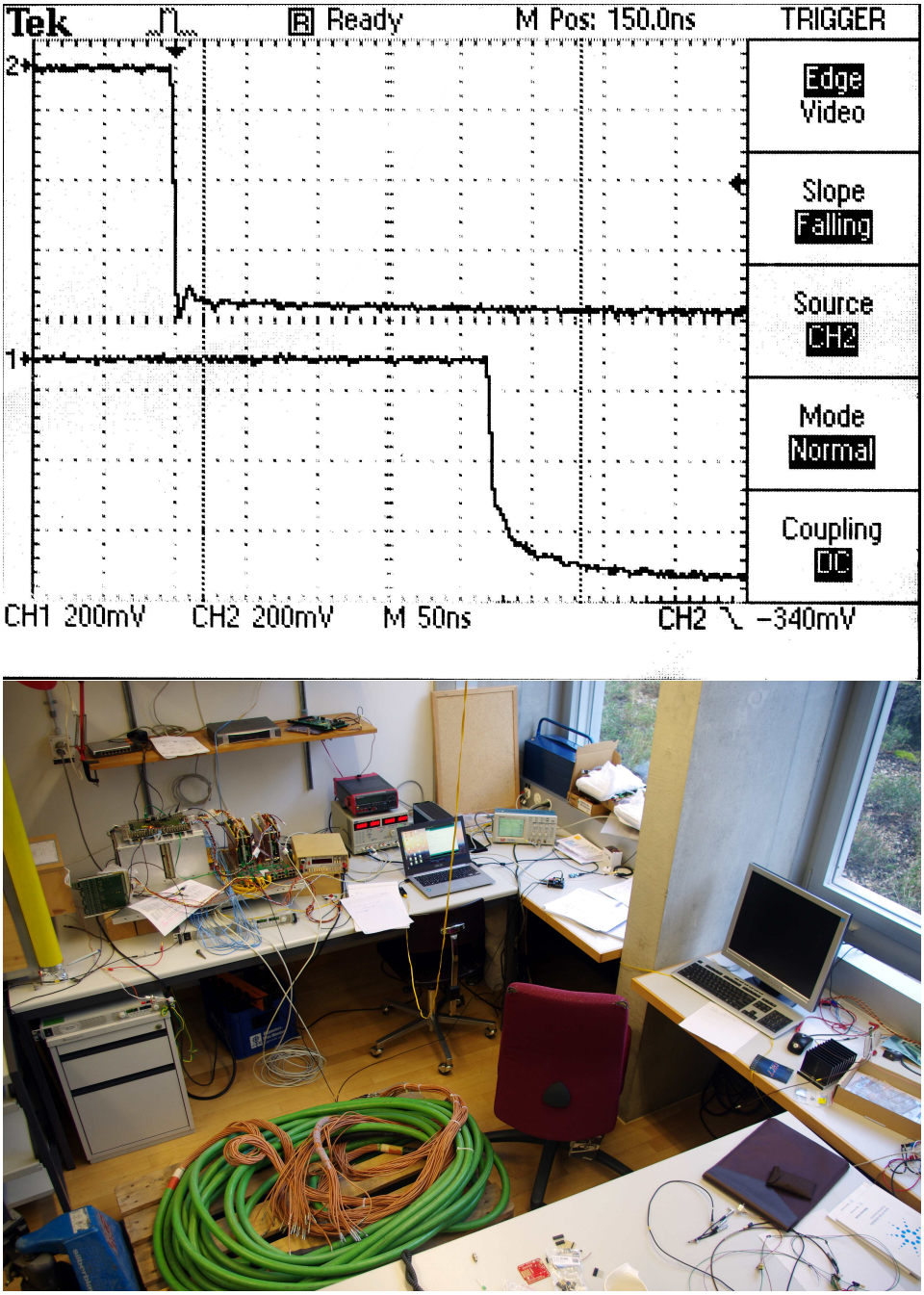


Figure 5.3: Top: The signals before and after the 44m cable. Bottom: Test set-up in the ETHZ basement next to a naked working sector of the FACT camera electronics

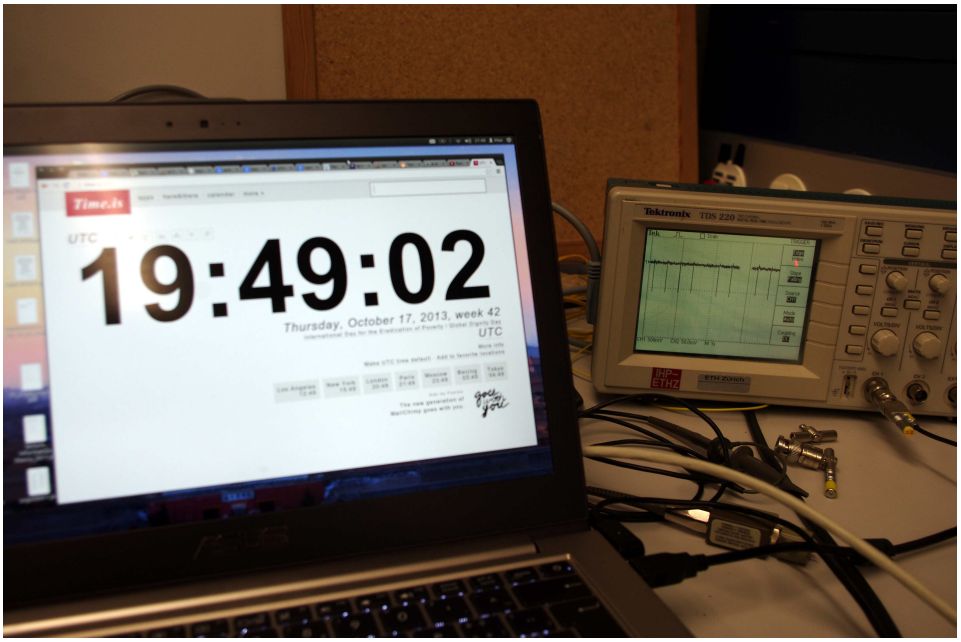


Figure 5.4: The "tooth gap". Please note the time on the laptop: 2 seconds after the full UTC minute. The scope shows 2 pulses to the tooth gap.

5.5 FTM Trigger In: Ext1 & Ext2

Test set-up: Full scale test of everything, including the FTM EXT1 & EXT2 inputs, the FACT GPS module, 44m of coax cable, and FACT++.

Test findings (Fig. 5.5): Everything worked as expected. FACT++ FTM tab showed a stable rate of 1Hz.

5.6 Ethernet Commandos

Test set-up: The assembled FACT GPS box was tested to work with the ethernet commands. At the same time, the LEDs were tested. A scope was used to verify the pulses were vetoed correctly.

Test findings: All commands worked fine. get_nema sends the last nema message, but the veto got stuck when trying to request it faster than ten times per second. The help command works, the quit command works. The veto_off and the veto_on commands work and switch on and off the Veto LED. The Arduino LED is blinking with a frequency of 4Hz, as expected. The GPS LED is ON in the beginning. After some time it blinks with 0.5Hz, as expected when the chip gets a lock on the satellites. The veto_60 command works as well, as verified by the same web clock like above.

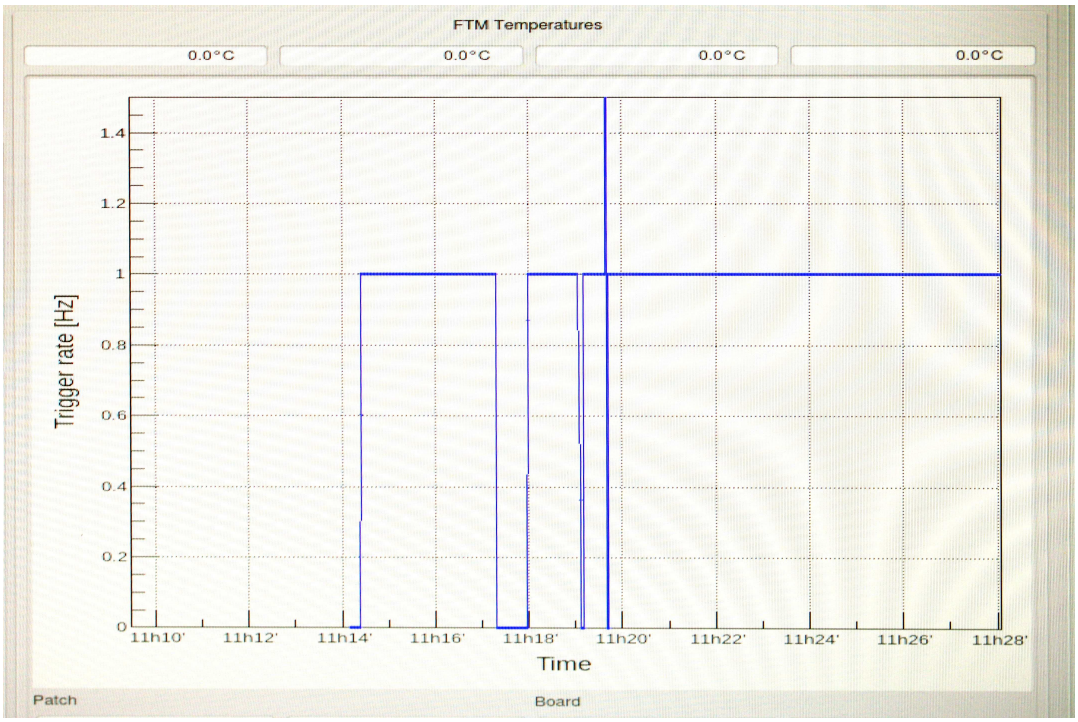


Figure 5.5: FACT++ FTM tab showing a stable rate of 1Hz. The large spikes to 0Hz are artificial because of switching the veto off and on. One spike seems to indicate 2Hz followed by 0Hz. This can be explained as two PPS measured in the same integration window.

6 Attachments

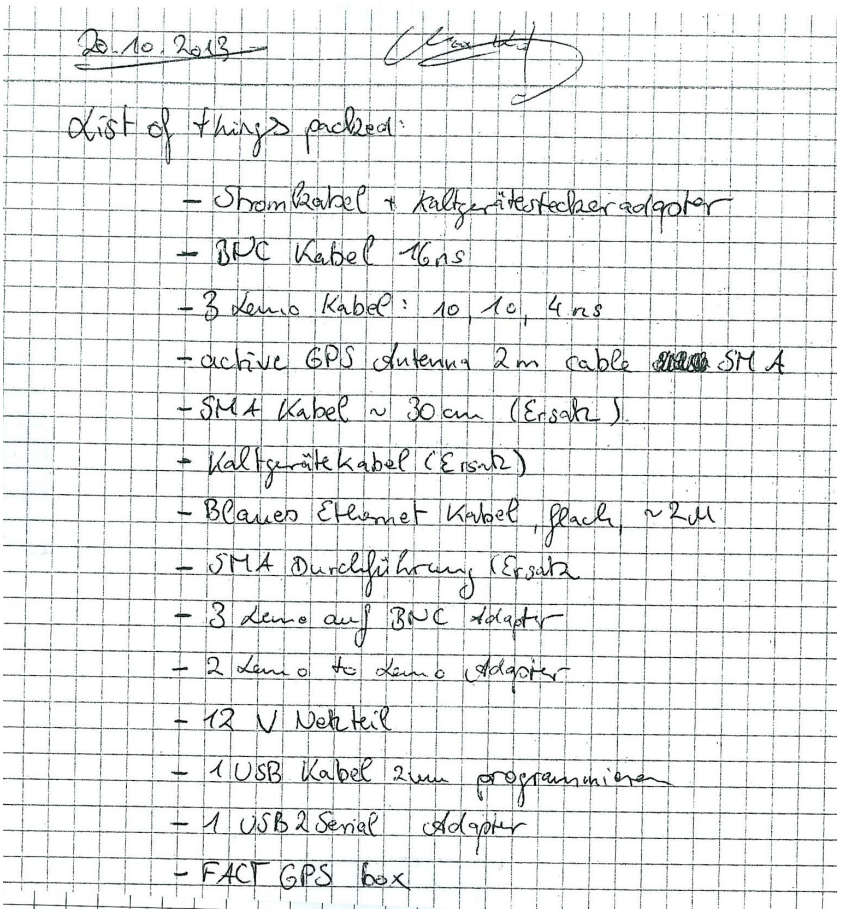
6.1 External Schematics

The external schematics are stored in the svn on La Palma, together with the firmware and this document:

```
svn co https://www.fact-project.org/svn/firmware/GpsArduino GpsArduino_firmware
```

6.2 Package List La Palma

This flew to La Palma:

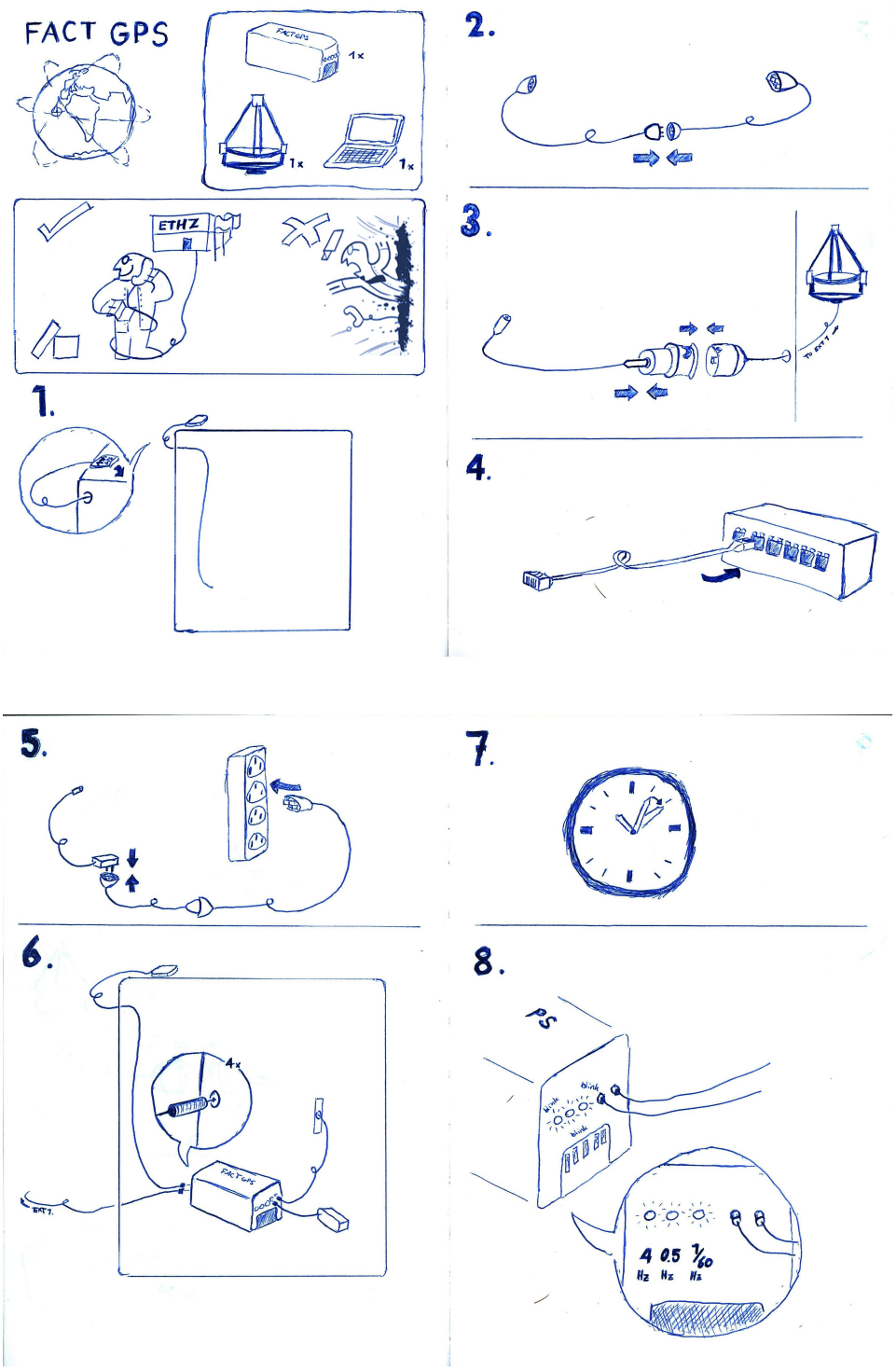


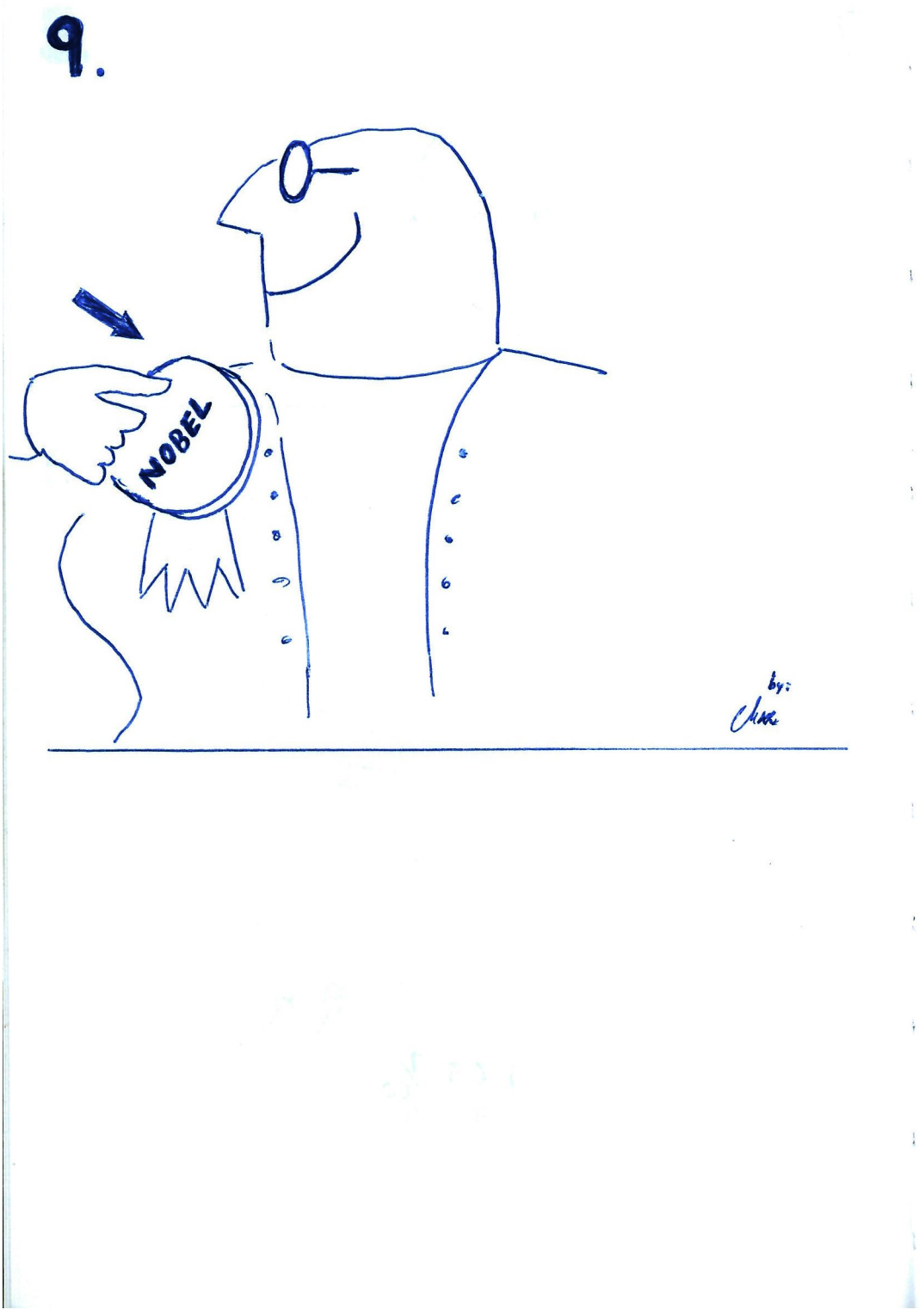
20.10.2013

List of things packed:

- Stromkabel + Kaltgerätestecker adapter
- BNC Kabel 16ns
- 3 Lemo Kabel: 10, 10, 4 ns
- active GPS Antenna 2m cable SMA
- SMA Kabel ~ 30cm (Ersatz)
- Kaltgerätekabel (Ersatz)
- Blaues Ethernet Kabel, flach, ~2m
- SMA Durchführung (Ersatz
- 3 Lemo auf BNC Adapter
- 2 Lemo to Lemo Adapter
- 12 V Netzteil
- 1 USB Kabel zum programmieren
- 1 USB 2 Serial Adapter
- FACT GPS box

6.3 Set-up Instructions for Watz




9.
NOBEL
by:

Bibliography

[1] Aleksic, J. *et al.* Phase-resolved energy spectra of the crab pulsar in the range of 50-400 gev measured with the magic telescopes. *Astronomy and Astrophysics* **540**, 69 (2012).

[2] Aliu, E. *et al.* Detection of pulsed gamma rays above 100 gev from the crab pulsar. *Science* **334**, 69–72 (2011).