Designing PACAS Pilot

Bachelor's Thesis

Triyan Bhardwaj

tbhardwaj@ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Peter Belcák
Prof. Dr. Roger Wattenhofer

July 29, 2022

Acknowledgements

I am grateful to Prof. Dr. Roger Wattenhofer and my supervisor Peter Belcák for offering such an enticing bachelor's thesis to work on and for providing me their time, support, and resources to see it through to completion. I would also like to thank the Texas Instruments E2E community for contributing key insights during development.

Abstract

This work implements PACAS Pilot – a prototype embedded system which will serve as a test bed and development platform for distributed algorithms designed to prevent collisions between light-sport aircraft (LSA). The final device consists of a main board with a CPU, a GNSS receiver, a low power, long range radio transceiver, and a Bluetooth Low Energy (BLE) transceiver. The main system is assisted by a remote board, which is to be positioned externally on target aircraft, as it relays back barometric altitude and airspeed data via a BLE connection.

Evaluation of the system showed very good 3D position accuracy (within 5 m in the worst case) thanks to the GNSS module. Similarly, the differential pressure sensor performed as expected, providing usable indicated airspeed values while in motion. Furthermore, testing the transceiver revealed that it is capable of transmitting large payloads up to approximately 750 m at 5.5 kbit/s.

Contents

Acknowledgements **i**

Abstract **ii**

1 Introduction **1**

1.1 Motivation 1

1.2 Related Work 1

2 System Design **2**

2.1 Design Objectives 2

2.2 Overview 2

2.3 Choice of Hardware 3

3 System Implementation **5**

3.1 PACAS Remote 5

3.1.1 Limitations of microcontrollers 5

3.1.2 Pressure sensor for altitude 5

3.1.3 Differential pressure sensor for airspeed 7

3.1.4 Bluetooth, communication, and configuration interface . . 8

3.1.5 Application design 9

3.2 PACAS Main 11

3.2.1 The plug-in module 11

3.2.2 Sensor data retrieval 11

3.2.3 GNSS module 11

3.2.4 Long range transceiver 12

4 System Evaluation **14**

4.1 Testing the Sensors and GNSS Module 14

4.2 Testing the Transceiver's Range 18

4.3 Analysing Power Consumption 19
4.4 Measuring Execution Time 21

5 Conclusion **23**
5.1 Summary 23
5.2 Future Work 23

Bibliography **24**

A PACAS Pilot Documentation **A-1**
A.1 PACAS Pilot Sub-devices A-1
A.1.1 PACAS Main A-1
A.1.2 PACAS Remote A-1
A.2 PACAS Pilot Bluetooth Protocol A-3
A.2.1 Packet structure A-3
A.2.2 Implemented packets A-3
A.2.3 Central state machine A-4
A.2.4 Peripheral state machine A-4
A.3 PACAS Main C/C++ Library A-5
A.3.1 Modules A-5
A.3.2 Classes and data structures A-5
A.3.3 Functions A-6

B Schematics and PCB Layout **B-1**

Chapter 1

Introduction

1.1 Motivation

Airborne collision avoidance systems (ACAS) are safety mechanisms designed to reduce the risk of midair collisions between aircraft. They continuously monitor the airspace in the immediate vicinity of the aircraft and issue audiovisual warnings or even resolution advisories (recommended manoeuvres for averting imminent collisions) to pilots, should a threat arise.

The collision avoidance problem has been solved by the likes of TCAS I and II, for large aircraft that are equipped with transponders or radar, whose trajectories are quite predictable, and where size, cost, and power consumption of the system prove to be milder constraints. There exists no universal solution for light-sport aircraft (LSA) such as gliders, small single engine aircraft, or non-fixed-wing aircraft. A wide-spread technology in use today, FLARM, is deemed suitable only for human-controlled, less agile, fixed-wing aircraft, thereby excluding paragliders, hang-gliders, hot air balloons, and drones, to name a few. These diverse aircraft tend to cluster densely in uncontrolled airspace around landmarks or areas of high lift. The absence of a system tailored towards the aforementioned vehicles presents itself as an opportunity to develop a practical and reliable embedded device, which competes with FLARM for the purpose of collision detection and avoidance.

1.2 Related Work

This thesis is part of a larger project – Practical ACAS (PACAS), which aims to implement a solution geared towards LSAs. While this work focuses on developing an early version of the actual system, a preceding project designed algorithms to perform trajectory prediction and collision detection [1]. The proposed algorithms consider the effects of thermals, which is a big advantage over FLARM as the predictions become more accurate.

Chapter 2
System Design

This chapter presents a high level overview of PACAS Pilot's design.

2.1 Design Objectives

The following features were deemed necessary in order to facilitate collision detection and avoidance:

- an independent power supply for portability,
- a high precision pressure-temperature sensor,
- a pitot tube based airspeed sensor,
- an interface for wireless communication with the above sensors,
- an on-board serial interface for configuration,
- a low power GNSS module,
- a long range, low power transceiver, and
- a central processing unit managing the above and capable of running collision prediction algorithms

2.2 Overview

The PACAS Pilot device is split into two subsystems – PACAS Main and PACAS Remote. The former manages the GNSS module, transceiver, and Bluetooth connection to PACAS Remote, a separate board (with both sensors) to be mounted externally on aircraft. Figure 2.1 shows a block diagram of the device containing the aforementioned modules and how the microcontrollers (MCUs) communicate with them.

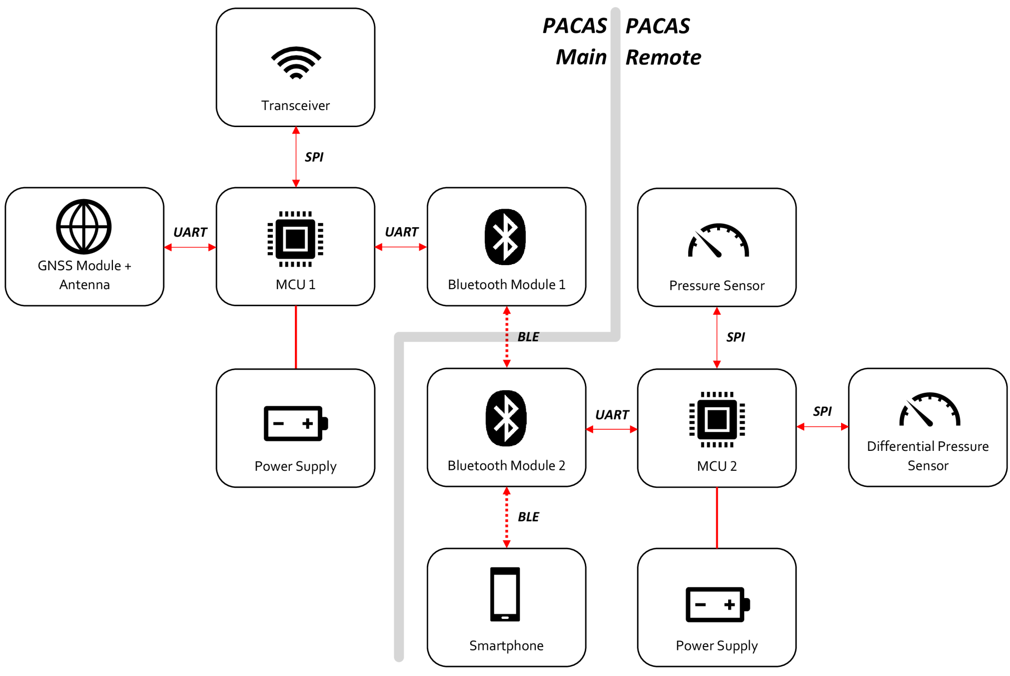


Figure 2.1: PACAS Pilot block diagram as per Section 2.2. The grey line separates PACAS Main blocks from those of PACAS Remote. Red, solid, double headed arrows indicate a wired serial communication bus, whereas dashed ones indicate a wireless connection. Red, solid lines represent a wired connection with no communication.

2.3 Choice of Hardware

The chosen parts had to be easily available, reasonably low cost, and designed with low power consumption in mind. Texas Instruments' MSP series of microcontrollers are well known for their versatility and ultra-low power consumption. Therefore, they were chosen to be the backbone of the entire system. The criteria for the sensors were maximum digital resolution, suitable pressure ranges, and a common communication interface. The remaining parts were picked based on their flexibility and low power consumption. Table 2.1 lists the concrete parts for the blocks in Figure 2.1.

Block	Part name
MCU 1	Texas Instruments MSP-EXP432P401R
MCU 2	Texas Instruments MSP430G2553
Pressure sensor	Measurement Specialties MS5607-02BA
Differential pressure sensor	Honeywell ABP2MRRN005NDSA3XX
Bluetooth modules	DSD Tech HM-19
GNSS module	M5Stack GPS with u-blox NEO-M8N
Transceiver	Semtech SX1276MB1MAS

Table 2.1: Part listing for PACAS Pilot corresponding to the blocks in Figure 2.1

Chapter 3

System Implementation

This chapter provides detailed insights into both sub-devices of PACAS Pilot including reasoning behind some of the design decisions.

3.1 PACAS Remote

3.1.1 Limitations of microcontrollers

Before describing the integration of the sensors, it is necessary to consider how calculations are carried out by the device. The on-board 16-bit microcontroller is a Texas Instruments MSP430G2553, which features neither a floating point unit (FPU) nor a hardware multiplier. Thus, all affected operations must be emulated in software, resulting in slower execution and larger code size. To tackle this issue, Texas Instruments provides IQmathLib, an optimised 32-bit fixed point arithmetic library whose functions run considerably more efficiently than equivalent ones from the standard `math.h` C library [2].

A brief summary of the number system used is as follows (further details can be found in the user guide [3]). To approximate real numbers, A 32 bit binary integer is split into N bits representing the fractional part with $M := 32 - N$ bits remaining for the integer part. This format has advantages over IEEE-754 floating point such as much higher, uniform precision over its range of values. There are some caveats, however. Special care must be taken to select the correct N and convert between different N-formats during a calculation such that all intermediate results are represented correctly. As an example, multiplications and divisions necessitate conversion between formats due to the varying scales of the numbers.

3.1.2 Pressure sensor for altitude

The chosen sensor (a Measurement Specialties MS5607-02BA) provides raw digital temperature and pressure values. These are combined to calculate a tem-

perature compensated pressure reading in Pascals. The 1976 US Standard Atmosphere defines a model for how atmospheric pressure P varies with altitude h above a reference altitude h_b, which is given by the barometric formula (Equation 3.1) [4].

$$P = P_b \left[\frac{T_b + (h - h_b) L_b}{T_b} \right]^{-\frac{g_0 M}{R^* L_b}} \tag{3.1}$$

Where:

$P_b \equiv$ reference pressure (Pa)
$T_b \equiv$ reference temperature (K)
$L_b \equiv$ temperature lapse rate (K/m)
$h \equiv$ altitude at which pressure is calculated (m)
$h_b \equiv$ altitude of reference above sea level (m)
$R^* \equiv$ gas constant: 8.314 32 J/(mol·K)
$g_0 \equiv$ gravitational acceleration: 9.806 65 m/s^2
$M \equiv$ molar mass of Earth's air: 0.028 964 4 kg/mol

In this model, P_b, T_b, L_b, and h_b are multi-valued constants which correspond to 7 different layers of the lower atmosphere. The subscript b ranges from 0 to 6. Only the values for $b = 0$ are relevant to this application since the aircraft involved are not expected to fly higher than 11 km. The values for the first 3 layers up to 32 km are listed in Table 3.1.

b	P_b (Pa)	T_b (K)	L_b (K/m)	h_b (m)
0	101 325.00	288.15	−0.006 5	0
1	22 632.10	216.65	0.000 0	11 000
2	5 474.89	216.65	0.001 0	20 000

Table 3.1: Constants for the barometric formula – up to 32 km, by layer of atmosphere

With $b = 0$, Equation 3.1 can be solved for the altitude h to yield

$$h = h_0 + \frac{T_0}{L_0} \left[\left(\frac{P}{P_0} \right)^{-\frac{R^* L_0}{g_0 M}} - 1 \right] \tag{3.2}$$

$$= \frac{T_0}{L_0} \left[e^{-\frac{R^* L_0}{g_0 M} \ln\left(\frac{P}{P_0}\right)} - 1 \right], \tag{3.3}$$

where the power has been rewritten in exp-log form to suit the available functions in IQmathLib. The resulting equation is the one implemented on the device and is

quite computationally intensive, requiring an estimated 13 951 cycles (1.16 ms at 12 MHz) to complete. A second order Taylor series approximation (Equation 3.4) is also available to reduce CPU usage to 3 695 cycles (0.31 ms at 12 MHz) at the cost of accuracy at higher altitudes. Figure 3.1 depicts the error of the approximation.

$$h = a_1 (P - P_0) + a_2 (P - P_0)^2 + \mathcal{O}\left((P - P_0)^3\right) \tag{3.4}$$

Where:

$$\begin{aligned} a_1 &= \frac{Ar}{P_0} \\ a_2 &= \frac{Ar(r-1)}{2P_0^2} \\ A &= \frac{T_0}{L_0} \\ r &= -\frac{R^* L_0}{g_0 M} \end{aligned}$$

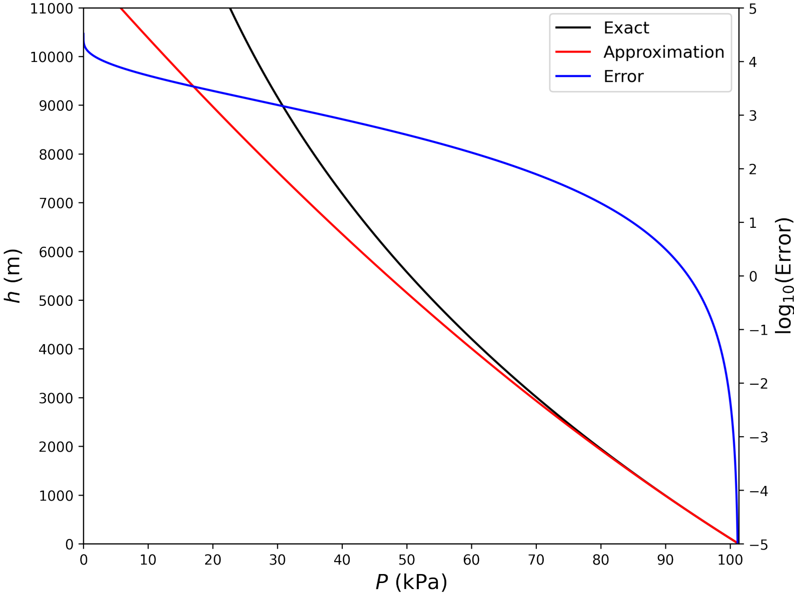


Figure 3.1: Comparison of barometric formula and Taylor series approximation. The series truncation error is a third degree polynomial in $(P - P_0)$. The true altitudes at which the absolute error becomes 0.1 m, 1 m, 10 m and 100 m are 309.6 m, 671.9 m, 1 471.2 m, and 3 286.9 m respectively.

3.1.3 Differential pressure sensor for airspeed

The Honeywell ABP2MRRN005NDSA3XX differential pressure sensor has two ports (to be attached to a pitot tube) – one to measure static pressure and the

other to measure dynamic pressure. Its measurement range is ±5 inH_20 (±1 245.4 Pa), ideally allowing for an airspeed of up to 45 m/s. The sensor outputs a raw digital pressure difference which, when converted to Pascals and combined with Equation 3.5 – a consequence of Bernoulli's incompressible flow equation, can be used to derive an indicated airspeed (IAS) value v_{ind}.

$$v_{ind} = \sqrt{\frac{2Q}{\rho_{air}}} \tag{3.5}$$

Where:

Q ≡ differential pressure (Pa)
ρ_{air} ≡ density of air at STP: 1.225 kg/m^3

The IQmathLib implementation of this formula requires around 3 925 cycles (0.33 ms at 12 MHz), which is acceptable. Before using it, the sensor should be calibrated to remove bias. This is achieved by taking the mean of multiple readings (e.g. 256) and subtracting this from all subsequent readings. For the best results, calibrations should be performed in a windless environment with the static and dynamic ports connected (i.e. exposed to the same pressure) while stationary.

3.1.4 Bluetooth, communication, and configuration interface

Both PACAS Main and PACAS Remote feature the DSD Tech HM-19 BLE module, which essentially takes a UART connection and makes it wireless. Its primary function is to relay sensor data to PACAS Main but the hardware also doubles as a configuration interface for PACAS Pilot. The remote board's module is configured in peripheral (slave) mode so that it advertises itself to a central (master) device, enabling PACAS Main or a smartphone to connect to it. PACAS Main devices are not capable of being connected to from central devices.

A request-reply based communication protocol has been designed on top of the Bluetooth link to facilitate data transfer. One transaction consists of a command/request sent by a central device followed by a response from the peripheral device (optionally returning data and the status of the transaction). The command syntax and state machines are shown in Appendix A. Currently, four commands are implemented, namely a power-up clear (PUC) request to restart the device, a sensor data request, a flash write request, and a flash read request. The sensor data command may be extended to include more sensors if needed. Expanding the command set to incorporate new functionality is also possible.

When a definite collision detection algorithm is implemented on the PACAS Pilot device, it will likely make available configurable parameters to the user. If these parameters need to be persisted in and retrieved from non-volatile memory,

the flash read and write commands may be used. The MSP430G2553 has 256 bytes of "information memory", split into four equal segments. One segment is reserved by the microcontroller for its calibration coefficients, leaving three (192 bytes) for application level data. It should be noted that only one segment may be written to per request and that a request always erases the entire segment in question before writing to it (due to the nature of flash memory used).

Currently, an Android smartphone connection is supported through Serial Bluetooth Terminal [5], which provides a console interface to send and receive data from Bluetooth devices. A screenshot of the app in use with PACAS Remote is shown in Figure 3.2. The final product should have an accompanying lightweight, more user-friendly smartphone app if the configuration feature is used.

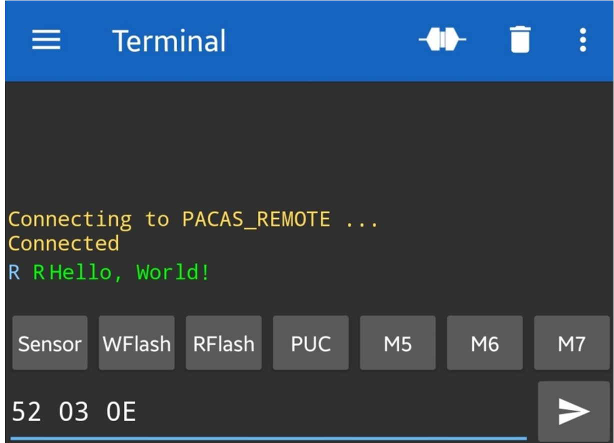


Figure 3.2: Sending a read flash command to PACAS Remote with the Serial Bluetooth Terminal app. The command `52 03 0E` seen in the screenshot requests the first 15 bytes stored in segment 0 of the microcontroller's flash. In this case, the string `Hello, World!<CR><LF>` was stored at the memory range in question.

3.1.5 Application design

Figure 3.3 and Table 3.2 summarise the design of the application running on the microcontroller.

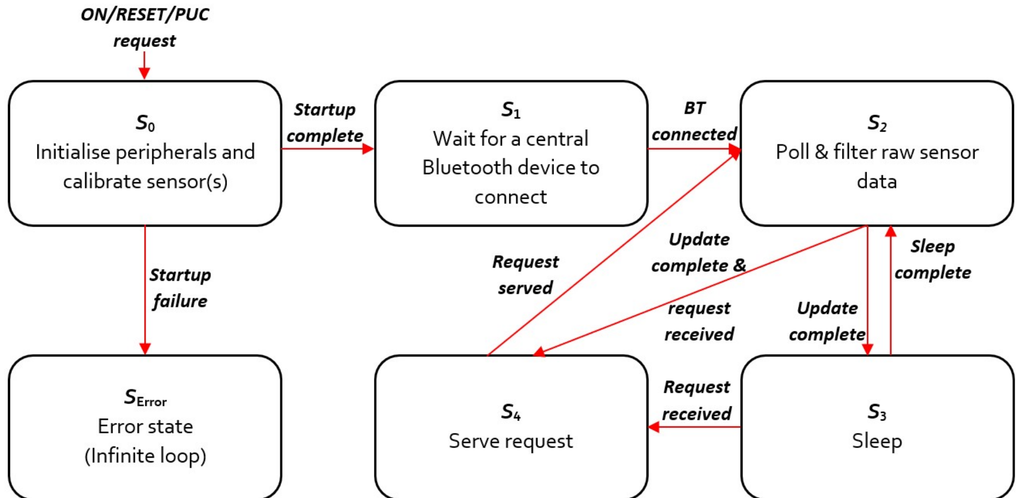


Figure 3.3: State diagram of the remote board. The red arrows represent state transitions, whose labels are events or conditions that must be fulfilled for the transition to occur.

State	Description
S_0	Upon powering up, receiving a power-up clear (PUC) request via Bluetooth, or pressing the reset button, the device begins setting up GPIO pins and serial communications. Then it initialises the sensors and calibrates the differential pressure sensor.
S_1	The device actively waits for a central Bluetooth device such as PACAS Main or a smartphone to connect.
S_2	The microcontroller polls the sensors for raw data which are then low-passed to reduce noise.
S_3	The microcontroller goes into low power mode 0 (LPM0), a power saving state in which the CPU clock is stopped.
S_4	The device serves a request received via Bluetooth. Details regarding the implemented requests and communication protocol can be found in Appendix A.2.
S_{Error}	This state is entered if an error is encountered during sensor initialisation. Possible causes are ROM corruption, SPI glitches, or lack of power. The only way to exit this state is by resetting the device.

Table 3.2: Application state details as per Figure 3.3

3.2 PACAS Main

3.2.1 The plug-in module

PACAS Main in its current state is a plug-in module (also referred to as a BoosterPack in the Texas Instruments ecosystem) for the MSP-EXP432P401R LaunchPad. It interfaces to a DSD Tech HM-19 BLE module, a u-blox NEO-M8N GNSS module, and a Semtech SX1276MB1MAS transceiver. As with PACAS Remote, this device may be powered by batteries for portability. The necessary steps are laid out in Appendix A.1.

The collision detection algorithm is to be implemented on the ARM Cortex-M4F based MSP432 in either C or C++. It is recommended to program and debug the microcontroller using the Micro-USB port and the Code Composer Studio IDE.

3.2.2 Sensor data retrieval

As described in Section 3.1.4, sensor data may be requested from PACAS Remote via the Bluetooth module. The `pacas_main.h` library provides a function to issue the request and set up a transfer of the data into a suitable data structure. The function only transmits the request. Interrupt service routines handle incoming data and execute a callback function when all bytes have been received. This enables other tasks to be executed while PACAS Remote handles the request.

3.2.3 GNSS module

The on-board concurrent GNSS module can simultaneously process signals from up to 3 of 4 satellite constellations (GPS, GLONASS, Galileo, and BeiDou). It supports the NMEA 0183 and the proprietary UBX protocols. These protocols implement "sentences" (packets) with various quantities of interest (e.g. position, velocity, and time). Although NMEA sentences are more universal, their disadvantage lies in the use of ASCII for all data in sentences. This results in poor information density (especially for numeric data) and higher processing overhead for the host system. The UBX protocol on the other hand encodes all its sentences in binary format and maintains compatibility with little-endian systems like the MSP432. Apart from this, data in the payload are aligned to memory meaning a complex parser is unnecessary, enabling values to be read directly into C structs (unlike NMEA). Having built a case for UBX, the chosen sentence is the 100 byte long UBX-NAV-PVT which provides a position, ground speed, heading, and UTC time solution among other, less vital pieces of information. The full sentence documentation can be found in the protocol specification [6].

The GNSS module's navigation solution update rate is set to 10 Hz and runs

in sync with UTC. Polling the module for data is not an option since this may take 100 ms plus processing time in the worst case if the request is made just after an iteration finishes. A solution to this problem is the MSP432's DMA, which can transfer data from various input channels to memory without CPU intervention. The GNSS module communicates with the microcontroller via UART, outputting a PVT message as soon as one is ready. The corresponding DMA transfer has been configured in the so-called "ping-pong" mode. This mode transfers data to two separate buffers in an alternating fashion, switching after each iteration. The user may then request a solution from the most recently updated buffer with a corresponding call to a `pacas_main.h` library function. Since the buffers are updated continuously, using two ensures that data part way through an update is not visible to the user.

3.2.4 Long range transceiver

PACAS Main supports Semtech's SX1276MB1MAS shield which features an SX1276 LoRa modem, matching circuitry, and SMA connectors for antennae. PACAS Main operates the shield in the licence-free EU868 (863-870 MHz) band. LoRa is a long range, low power radio technology typically integrated in IoT devices. A modulation technique similar to chirp spread spectrum (CSS) is used. Each symbol is encoded by a chirp – a sinusoidal signal whose frequency linearly sweeps a range defined by the centre frequency and bandwidth used. The achievable range and data rate are inversely correlated and depend on the selected bandwidth (BW) and spreading factor (SF). A summary of expected performance at different settings is given in Table 3.3.

The radio defines the physical layer, leaving higher level protocols up to users to implement. A widespread communication protocol is LoRaWAN, which most use cases employ. However, it requires the network to be arranged in a star topology with dedicated (ground based) gateway devices in the centre to act as message forwarders. In the context of PACAS Pilot, this is too restrictive and inefficient, as the application demands direct broadcasts to all aircraft in the vicinity. Therefore, the required unaddressed mesh network has been implemented by porting a subset of the RadioHead library, a project started by M. McCauley [7], to the MSP432.

When designing communication systems with a shared physical medium, appropriate access control mechanisms must be in place for unhindered transmissions. Slotted ALOHA and TDMA (time-division multiple access) are protocols that were considered. However, these call for clock synchronisation across nodes on an already bandwidth-limited network. Nevertheless, thanks to the LoRa modem's Channel Activity Detection (CAD) feature, sensing the preamble chirps and in many cases also the payload chirps on a specific channel in an energy efficient manner is possible [8]. As a result, certain features of CSMA/CA (carrier

sense multiple access with collision avoidance) are present on the device. When a frame transmission request is made, the device executes a distributed interframe space (DIFS) slot in which a fixed number (N_{DIFS}) of CADs are performed to check the channel occupancy state. If all CADs report an idle channel, another window is opened where a random number N_{BO} is generated followed by further CADs being performed, with each idle report decrementing N_{BO}. This back-off stage ensures a lower probability of frame collisions should multiple transmitters have started their DIFS slots at the same time. If a CAD reports a busy channel during a DIFS slot or the back-off stage, the process is reset to the DIFS stage (without regenerating N_{BO}). A timeout is in place to abort the transmission if too many unsuccessful DIFS slots occur. Once N_{BO} reaches 0, the frame may be transmitted.

Spreading Factor	**Bit Rate (bits/s)**	**Range (km)**
SF10	980	8
SF9	1 760	6
SF8	3 125	4
SF7	5 470	2

Table 3.3: Expected transceiver performance with a 125 kHz bandwidth setting, according to Semtech [9]. The trade-off between bit rate and range is clearly visible, as increasing the range by 2 km (roughly) halves the bit rate. The majority of LSAs fly slow enough for a range of 0.5 to 2 km to be adequate.

Chapter 4

System Evaluation

This chapter presents the experiments devised to test various components of PACAS Pilot along with their results.

4.1 Testing the Sensors and GNSS Module

Experiment setup

The primary targets for PACAS Pilot are gliders and similar LSAs. A typical glider such as the Schleicher ASK 21 has a nominal stall speed (minimum airspeed) of 35 knots (18.0 m/s) and a smooth-air never exceed speed of 150 knots (77.2 m/s). Since access to real gliders was not convenient, the performance of PACAS Remote and GNSS accuracy were tested on the road. The A51 highway in the vicinity of Zurich Airport was elected as the venue, allowing for speeds of 100 km/h (27.8 m/s). Weather conditions were good at the time of the experiment (few scattered clouds and a very occasional light breeze). During the test, care was taken to drive on the rightmost lane, keep distance from other vehicles to avoid effects from their slipstream, and maintain a constant speed of 100 km/h while travelling north on the highway.

A pitot tube was constructed for the differential pressure sensor by attaching a pen barrel to one port and to the other a small plastic tube closed at one end with a circular opening on its side. Appropriate tubing was used between the ports of the sensor and attachments to ensure a seal. The PACAS remote device was enclosed in a box and then mounted firmly on the roof of a car. PACAS main and the GNSS antenna were placed on the passenger side dashboard in good view of the sky (i.e. the very right side of the car). Data was updated approximately every 100 to 200 ms and no Taylor series approximation was used for altitude.

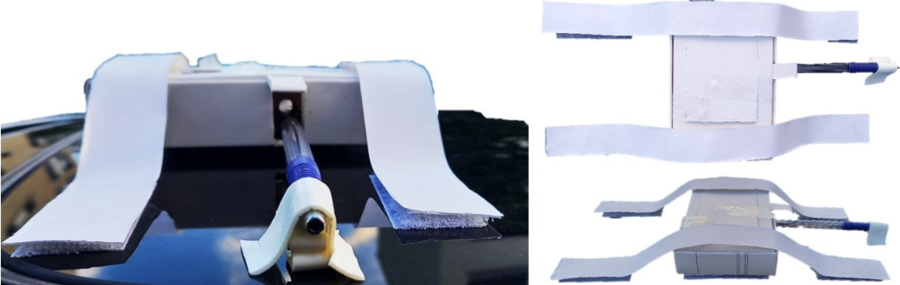


Figure 4.1: PACAS Remote enclosed and mounted to the roof of a car

Experiment results

GNSS: Figure 4.2 shows the route covered during the test according to the GNSS data. The GNSS module appears to give accurate results, especially when moving at higher speeds. Figure 4.3 is an example of a stretch with relatively high deviation from the expected path.

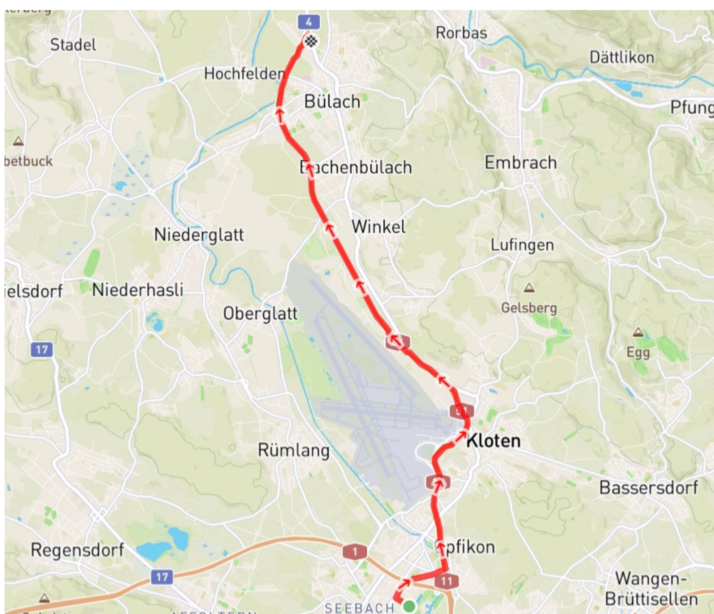


Figure 4.2: A plot of the coordinates recorded by the GNSS module during the highway test

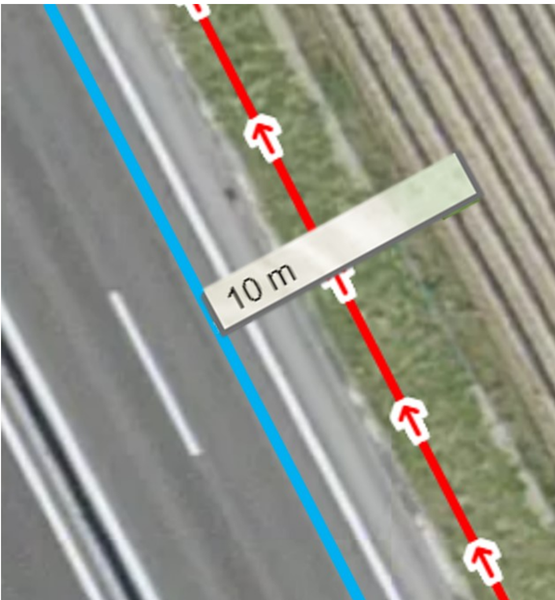


Figure 4.3: Deviation from the approximate expected path (light blue) is within 5 m. It is worth noting that satellite image accuracy is a limiting factor in this comparison.

Airspeed: Although a low pass RC filter (in software) was used to reduce noise for sensor data, the collected airspeed data is still quite noisy. To remedy this, a slower filter or a second stage may be implemented. During the time frame denoted T in Figure 4.4, the car was travelling at 100 km/h (with the help of cruise control). The mean airspeed during this period was 25.78 m/s ($\sigma = 1.02$ m/s), whereas the mean ground speed according to the GNSS module was 28.23 m/s ($\sigma = 0.11$ m/s). This discrepancy can be attributed to many things, such as imperfect alignment of the pitot tube into the airstream or the fact that the calculated value is indicated airspeed (IAS) and not true airspeed (TAS) which takes actual air density into account. Another source of error may be the wind. However, this is advantageous since a local estimate of wind speed and direction can be derived by factoring in ground speed and the vehicle's heading of motion (provided by GNSS). This estimate can prove useful in the collision detection algorithm were it to consider thermals and their evolution.

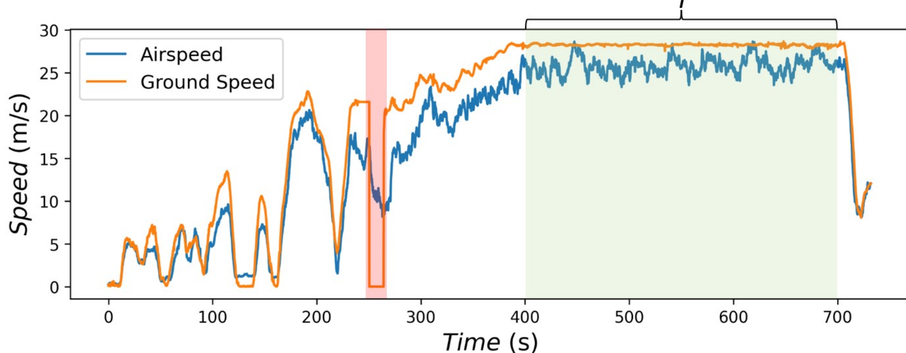


Figure 4.4: Measured airspeed and ground speed. The red interval marks a loss of GNSS signal due to a tunnel, invalidating the ground speed readings.

Altitude: Three sources of altitude data were considered – the barometer's converted reading, the GNSS module's solution, and a reference elevation profile generated offline by GPX Studio [10] based on the latitude and longitude data. It should be noted that the elevation profile is not ground truth since it is derived from location data with some, albeit small, error. Additionally, results from the GNSS depend on the positions of the satellites in the sky (satellites close to the horizon generally do not provide good altitude accuracy). In Figure 4.5, it can be seen that the GNSS readings exhibit a smooth profile and are in agreement with the reference. On the other hand, the barometer vaguely resembles the reference profile and shows more erratic behaviour (especially in the high speed regime) with sudden dips in altitude which may be explained by turbulent air flowing over the sensor. This problem is easily solved by constructing a better container for PACAS Remote. The offset of approximately 40 to 80 m from the reference is acceptable since all devices will use the same scale (the absolute altitude is of lesser importance).

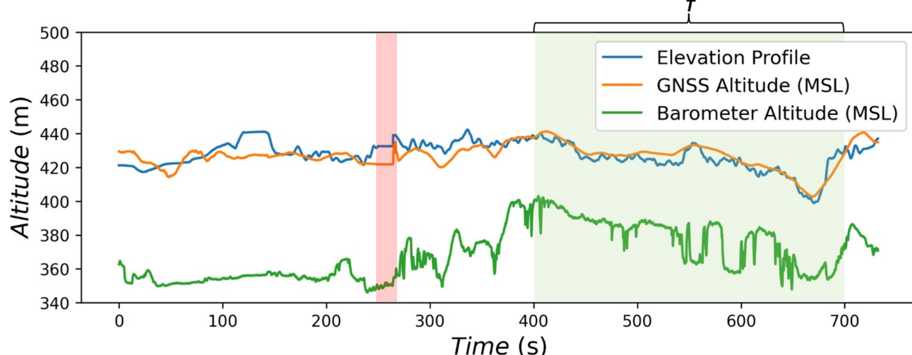


Figure 4.5: Altitude measured during the trip from various sources. The red interval marks a loss of GNSS signal due to a tunnel, invalidating the GNSS altitude readings and elevation profile.

4.2 Testing the Transceiver's Range

Experiment setup

Having an adequate range is critical to the application's success. To determine PACAS Main's range, a test with two transceivers was carried out in a park situated next to a suburban area. The channel settings were: 868.1 MHz centre frequency, 125 kHz bandwidth, spreading factor 7, and coding rate 4/5. These were chosen for their high bit rate and sufficient range as shown in Table 3.3. Positioned in line of sight, the transceivers were made to exchange 80 byte long payloads at various separations and the received signal strength indicator (RSSI) was recorded for one device. RSSI (measured in dBm) is an indication of the power of signals received from other transmitters, referenced to 1 mW. Since the intensity of radio waves follows an inverse square law, a 6 dBm decrease in RSSI (a reduction in power by a factor of 4) is expected whenever the distance doubles.

Experiment results

The measured RSSI values at various distances are plotted in Figure 4.6. The transceiver exceeds the minimum desired range of 500 m but does not achieve the 2 km mark predicted on paper. The transmissions already occur at a power setting of 14 dBm, which is the limit on the 868.1 MHz frequency (regulated by ETSI [11]). The transceiver may transmit at 15 dBm in the 869.40 MHz to 869.65 MHz band, which has a limit of 27 dBm. Other ways to increase range include using a lower bandwidth or higher spreading factor, both of which incur a cost in transmission speed.

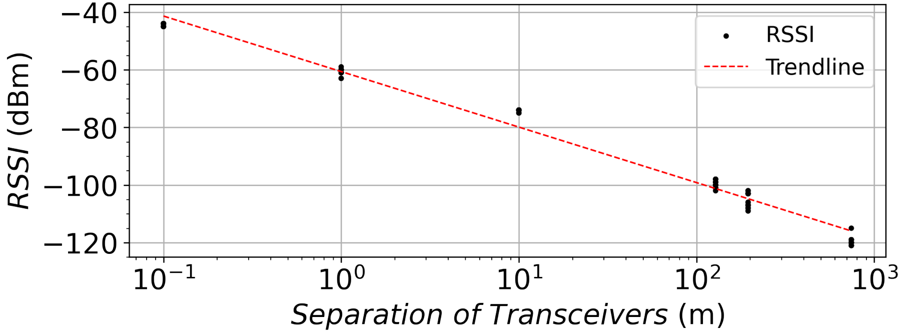


Figure 4.6: RSSI measured at separations of 0.1 m, 1 m, 10 m, 128 m, 194 m, and 743 m. There are 25 measurements for each distance. Messages were exchanged reliably up to a distance of 743 m. Beyond this distance, the RSSI drops below the transceiver's sensitivity on the chosen channel, meaning frames are not always demodulated successfully. The trendline shows a -19.3 dBm/decade gradient, which is in line with the expected -20 dBm/decade.

4.3 Analysing Power Consumption

Experiment setup

As with any battery operated embedded system, minimal power consumption is a desirable feature. EnergyTrace is a program in Code Composer Studio provided by Texas Instruments which allowed the power drawn by PACAS Pilot's subdevices to be measured, provided the current remained below a 75 mA threshold. The necessary measurement hardware is present on the MSP432 LaunchPad.

PACAS Main's performance depends heavily on the collision detection algorithm and frequency of transceiver usage. Therefore, only power consumption for individual operation modes could be determined. For PACAS Remote, measurements were made for the main loop of the application with sensor commands being served approximately every 100 to 200 ms.

Experiment results

PACAS Main: Table 4.1 profiles the device's power consumption in various modes. To clarify, the nominal state for PACAS Main is characterised by the following:

- The microcontroller is in LPM0
- PACAS Main is connected to PACAS Remote via Bluetooth
- The GNSS module is reporting a 3D fix
- The transceiver is not used (idle mode)
- Periodic interrupts (e.g. DMA and timer) are being served

An estimate for power consumption can be made assuming the final application's main loop takes one second and that this execution time is split up in the following way: 500 ms in active mode for updating trajectory and collision predictions, 200 ms for transmission, 300 ms for reception. This split results in a mean power of 102.5 mW. Two AA batteries in series with a capacity of 2500 mAh (27 kJ) could therefore power the device continuously for 73.2 hours.

PACAS Remote: Figure 4.7 shows the power consumed by the PACAS Remote application at two different main clock frequencies (12 MHz and 1 MHz). The average power at 12 MHz and 1 MHz were 13.48 mW and 10.41 mW respectively, meaning reducing the clock frequency 12-fold reduced the power by 23%. The reason for this small decrease is the Bluetooth module, which runs independently at 7.89 mW on average when connected.

Mode	**Mean Power (mW)**
Nominal	16.0
Nominal + MCU in active mode	21.8
Nominal + Update sensor data & PVT at 5 Hz	17.5
Nominal + Transceiver in RX mode	49.6
Nominal + Transceiver in TX mode	383.7

Table 4.1: PACAS Main power consumption in various modes (12 MHz clock). Note that the final measurement was performed using an oscilloscope since it exceeds EnergyTrace's current limit.

PACAS Remote's energy source is a 3 V CR2032 cell. Typically, these have a capacity of 220 mAh (2376 J). This implies that the device is capable of operating continuously for 49.0 hours at 12 MHz and 63.4 hours at 1 MHz before needing a battery replacement.

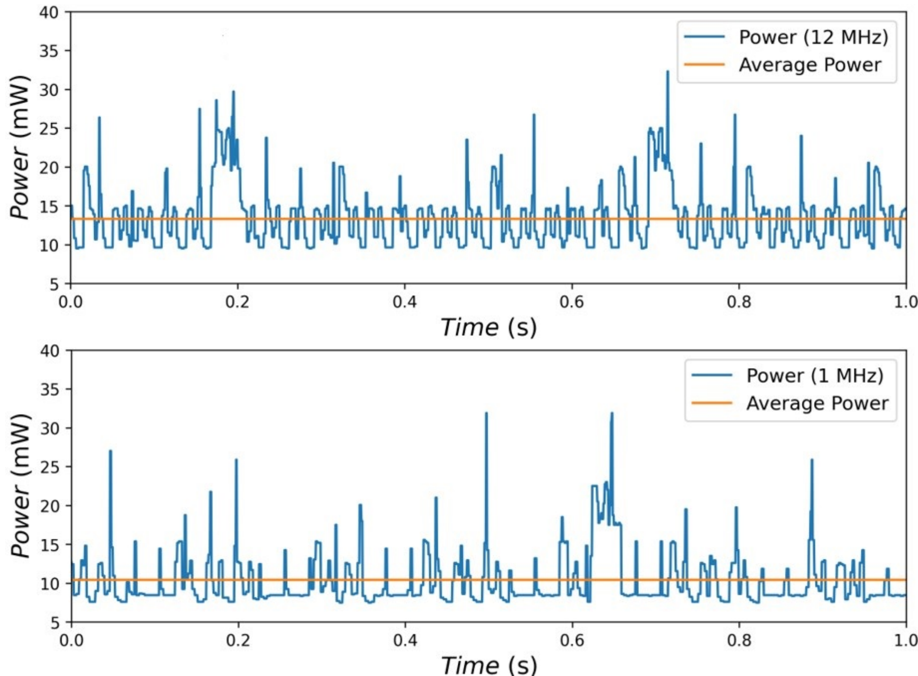


Figure 4.7: PACAS Remote main loop power consumption at 12 MHz (top) and 1 MHz (bottom)

4.4 Measuring Execution Time

Experiment setup

To determine the execution times of various tasks, a software timer with a resolution of 1 ms was used. Therefore all quoted execution times have an uncertainty of 1 ms. The measured tasks were: polling PACAS Remote's sensor data, updating PVT values, and sending messages through the transceiver with one concurrent transmitter in the same room. To estimate the Bluetooth transmission latency, a version of the sensor task was measured with a wired UART connection between PACAS Main and PACAS Remote. Both microcontrollers used a 12 MHz main clock for this experiment.

Experiment results

Sensor Task: This task has a high variance in execution time since the application on PACAS Remote may be polling the sensors when the request is made, which has a higher priority than serving them. Approximately 25 ms are required to update and filter raw sensor values. Converting raw values to pressures first and then deriving altitude and airspeed requires an additional 5 ms. This explains the range of values seen in the right histogram in Figure 4.8. On average, the response time without Bluetooth is 11.5 ms. Considering the version with Bluetooth, the average latency increases to upwards of 5 times that amount. In less than 1% of cases, the execution time of one request goes beyond 100 ms, cutting off at about 125 ms.

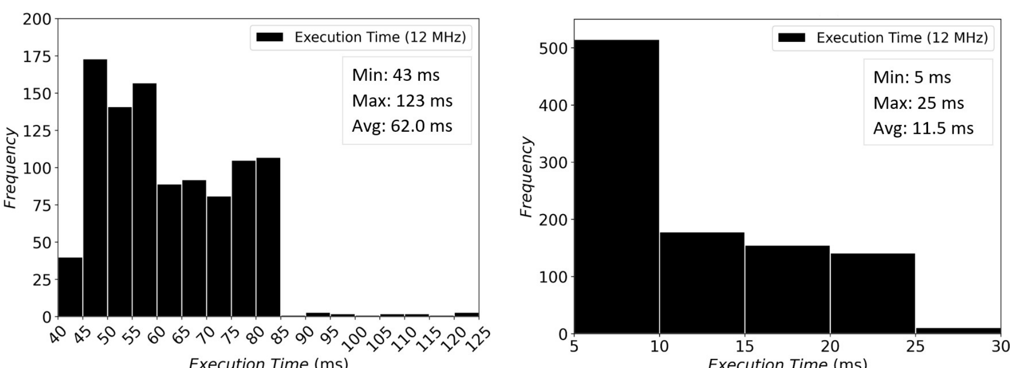


Figure 4.8: Sensor task execution time with Bluetooth (left) and without Bluetooth for comparison (right). There are 1000 requests in each case. With Bluetooth, 99% of requests are served within 98 ms, with a median of 59 ms. On the other hand, the 99th percentile for the UART version is 25 ms, with a median of 9 ms.

PVT Update Task: Updating the PVT struct is simply a matter of calculating a checksum over the received buffer and then copying it to the data structure. These operations are consistent and efficient, resulting in an execution time of less than 1 ms.

Transceiver Send Task: Packet transmissions are subject to the stochastic CSMA/CA procedure described in Section 3.2.4. Execution times vary based on channel settings, channel occupancy, as well as payload length. Test runs with two different payload sizes (80 bytes and 4 bytes) and channel settings (spreading factors 7 and 8) are shown in Figure 4.9.

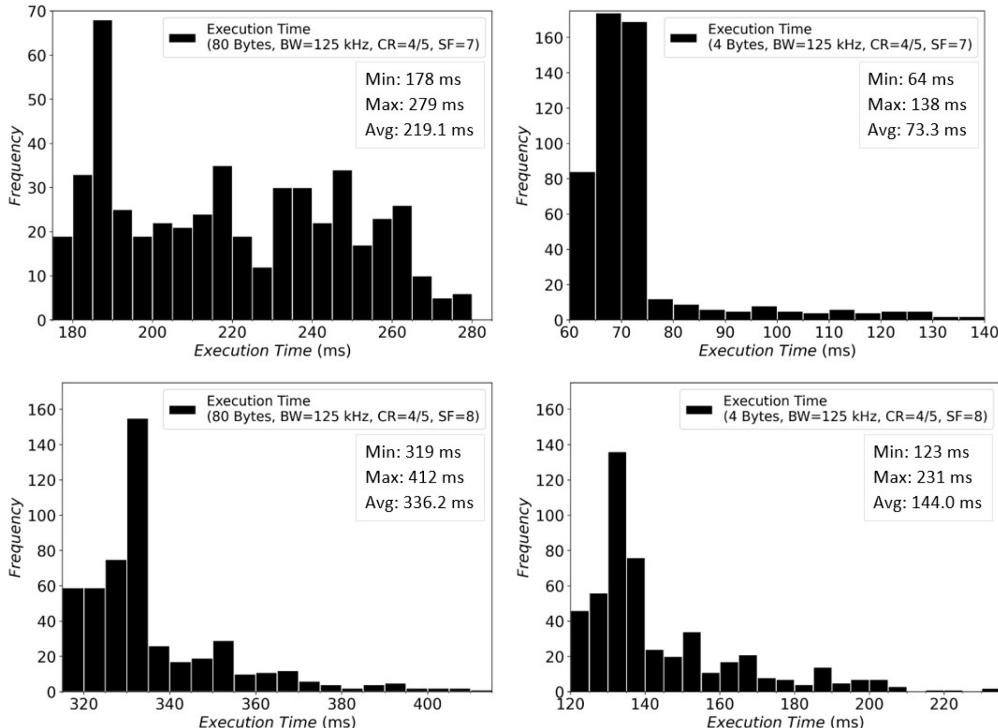


Figure 4.9: Transceiver send task execution time (including transmission time). The left and right columns correspond to 80 byte and 4 byte payloads respectively, whereas the top and bottom rows correspond to spreading factors 7 and 8 respectively. There are 500 transmissions in each case. The 99$^{\text{th}}$ percentiles for each case in clockwise order starting from the top left are: 275 ms, 129 ms, 208 ms and 400 ms

Chapter 5

Conclusion

This chapter summarises the thesis and provides an outlook.

5.1 Summary

The design objectives laid out in Section 2.1 are met satisfactorily and the way has been paved to implement and test concrete algorithms. Splitting PACAS Pilot into two devices, although necessary, increased system complexity and development time. Thanks to the flexibility and ease of use of chosen sensors and GNSS module, integrating these subsystems did not present itself as a hurdle. The 3D positional and velocity data provided by the GNSS and the airspeed measured by the differential pressure sensor all showed good accuracy during tests. PACAS Remote's barometer, however, despite its high resolution, behaved slightly erratically when faced with high speed. Special care must be taken when mounting PACAS Remote on aircraft to ensure its pressure sensor is exposed to static air only. PACAS Pilot's Bluetooth modules enabled seamless connectivity between its sub-devices and allowed for the implementation of a configuration interface, which may be employed by the application to change its behaviour according to user preferences. Finally, the LoRa based transceiver proved capable of transmitting data up to a distance of about 750 m at approximately 5.5 kbit/s.

5.2 Future Work

There is potential for the device to be improved and expanded upon. A low power display should be incorporated to inform pilots whether any collisions are likely, accompanied by aural warnings if necessary. Apart from this, an IMU and sensor fusion algorithm could be added to increase the 3D position accuracy and contribute to redundancy in case the GNSS signal is poor. Furthermore, a dedicated study could be conducted to determine how to optimise throughput and enable more reliable communication in the LoRa mesh network before attempting to scale the system up.

Bibliography

[1] P. Oberlin, “Collision Detection Algorithm for a Practical Airborne Collision Avoidance System,” Mar. 2022. [Online]. Available: https://pub.tik.ee.ethz.ch/students/2021-HS/BA-2021-30.pdf

[2] “MSP-IQMATHLIB Fixed Point Math Library for MSP,” Texas Instruments Incorporated, accessed: 21.04.2022. [Online]. Available: https://www.ti.com/tool/MSP-IQMATHLIB?DCMP=ep-mcu-msp-iqmath-en&HQS=ep-mcu-msp-iqmath-pr-sw2-en

[3] “MSP430 IQmathLib Users Guide version 01.10.00.05,” Texas Instruments Incorporated, Jan. 2015. [Online]. Available: https://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/IQmathLib/01_10_00_05/exports/MSP430-IQmathLib-UsersGuide.pdf

[4] “U.S. Standard Atmosphere 1976,” National Oceanic and Atmospheric Administration, National Aeronautics and Space Administration, and United States Air Force, Oct. 1976. [Online]. Available: https://www.ngdc.noaa.gov/stp/space-weather/online-publications/miscellaneous/us-standard-atmosphere-1976/us-standard-atmosphere_st76-1562_noaa.pdf

[5] K. Morich, “Serial Bluetooth Terminal,” May 2022, Vers. 1.38, Google Play Store. [Online]. Available: https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal&hl=en&gl=US

[6] “u-blox 8 / u-blox M8 Receiver description Including protocol specification,” u-blox, Nov. 2021. [Online]. Available: https://content.u-blox.com/sites/default/files/products/documents/u-blox8-M8_ReceiverDescrProtSpec_UBX-13003221.pdf

[7] M. McCauley, “RadioHead Packet Radio library for embedded microprocessors,” accessed: 10.05.2022. [Online]. Available: https://www.airspayce.com/mikem/arduino/RadioHead/index.html

[8] A. Gamage, J. C. Liando, C. Gu, R. Tan, and M. Li, “LMAC: Efficient Carrier-Sense Multiple Access for LoRa,” in *26th Annual International Conference on Mobile Computing and Networking (MobiCom ’20)*, Sep. 2020. [Online]. Available: https://wands.sg/publications/full_list/papers/MobiCom_20_1.pdf

[9] "LoRa and LoRaWAN: A Technical Overview," Semtech Corporation, accessed: 10.07.2022. [Online]. Available: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/

[10] "GPX Studio," accessed: 09.07.2022. [Online]. Available: https://gpx.studio/

[11] "ETSI EN 300 220-2 V3.1.1," European Telecommunications Standards Institute, Feb. 2017. [Online]. Available: https://www.etsi.org/deliver/etsi_en/300200_300299/30022002/03.01.01_60/en_30022002v030101p.pdf

Appendix A

PACAS Pilot Documentation

Finer details of the implementation are given in this appendix. The reader is encouraged to familiarise themselves with the material in Chapter 2 and Chapter 3 first.

A.1 PACAS Pilot Sub-devices

A.1.1 PACAS Main

Assembly and power

The PACAS Main board plugs into the MSP-EX432P401R LaunchPad's 40 pin BooterPack header through J1-J4. To achieve this, the jumper on the LaunchPad labelled "Blue P2.2" must be removed to make way for J14 to connect. PACAS Main reroutes the connection to the blue LED to P4.1. The Bluetooth module plugs into J9, GNSS into J10-J13, and transceiver into J5-J8. The assembled device is depicted in Figure A.1.

The device may be powered through the LaunchPad's Micro-USB port (e.g. by a computer or power bank) or by external batteries. For the second option, it is necessary to remove all jumpers from the LaunchPad's J101 isolation block save GND, RXD, TXD, and RST. Only then may a battery pack be connected to either VEXT or BATT and GND on J15 of PACAS Main. The corresponding position on switch SW1 must be selected to turn the 3.3 V and 5 V power regulators on.

A.1.2 PACAS Remote

Assembly, power, and programming interface

The only additional components to be added are a Bluetooth module which connects to J2 and a pitot tube which connects to the differential pressure sensor's

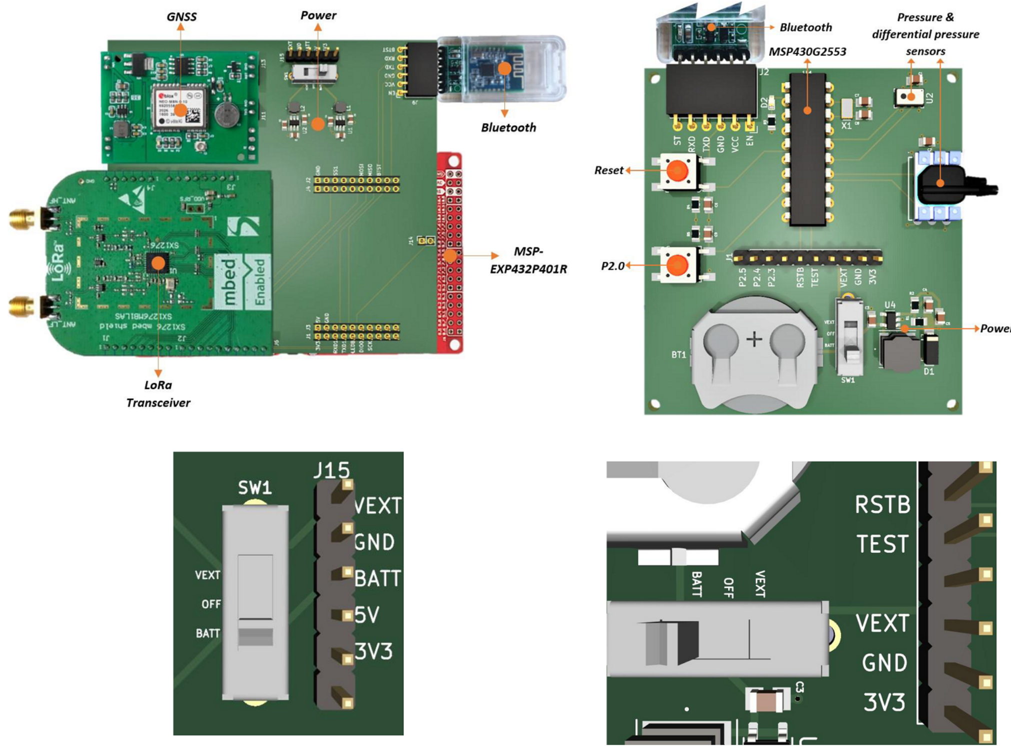


Figure A.1: Top Left: PACAS Main, Top Right: PACAS Remote, Bottom Left: PACAS Main power selection, Bottom Right: PACAS Remote power selection and programming interface

ports (the designation of static/dynamic port is up to the user).

PACAS Remote's microcontroller, Bluetooth module, and sensors require a stable 3.3 V power supply. To this end, the board has a CR2032 coin cell holder and a boost converter designed for the necessary voltage. Setting the slide switch to the BATT position will use the on-board battery. External batteries (of up to 3V) may be used instead by switching to the VEXT position and connecting a battery to the VEXT and GND pins. Finally, a 3.3 V supply sourced from an external regulator (e.g. from a microcontroller evaluation board) can be used by selecting OFF on the slide switch and connecting the 3V3 and GND pins to the source. As the name suggests, the OFF position also switches off the board (given that the 3V3 pin is left unconnected).

The microcontroller may be reprogrammed and debugged by connecting the 3V3, GND, RSTB, and TEST pins to their counterparts on an eZ-FET debugger which come with MSP430 LaunchPads (such as the MSP-EXP430FR5994 employed in this project). Other methods have not been tested.

A.2 PACAS Pilot Bluetooth Protocol

A.2.1 Packet structure

The protocol is a request-reply style one where the central device (master) issues commands to the peripheral device (slave). A packet consists of a 2 byte header containing the packet type and total byte count including the header. This is followed by a payload of up to 256 bytes. The first payload byte is usually interpreted as settings for the transaction in a request or status in a reply. Timeouts are in place to protect from incomplete or lost packets.

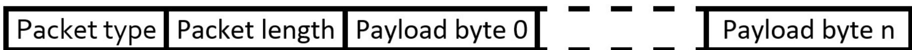


Figure A.2: Packet structure

A.2.2 Implemented packets

Figure A.3 lists the available packets. Library functions are available to send and act on these packets. For the sensor data request packet, the settings byte may be either 0 or `BT_SENSOR_SETTINGS_BARO_USE_TAYLOR` to choose whether a Taylor series approximation is used for altitude. In the case of read and write flash packets, the settings byte is the logical or of `BT_FLASH_SEG_M` and `BT_FLASH_XFER_NB` where `M` is 0, 1, or 2 and `N` ranges from 01 to 64. This sets the flash segment and number of bytes for the transaction. Generally, a status byte 0 indicates an error-free transaction.

Packet Type	Request Packet				Reply Packet			
Sensor data	53	03	Settings		53	0B	Status	8 bytes
Write flash	57	n + 3	Settings	n bytes	57	03	Status	
Read flash	52	03	Settings		52	n + 3	Status	n bytes
Power-up clear	50	02			No reply			

Figure A.3: Available packets (literals are in hexadecimal format)

A.2.3 Central state machine

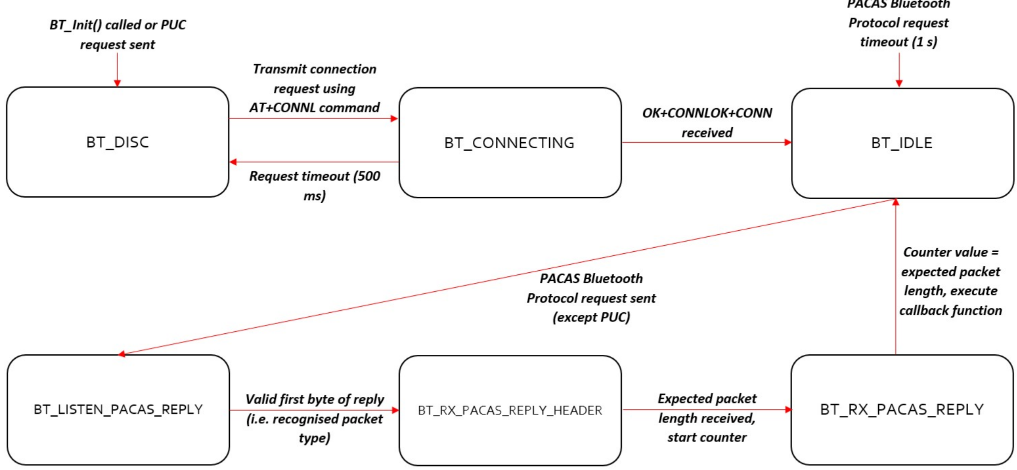


Figure A.4: PACAS Main Bluetooth state machine

A.2.4 Peripheral state machine

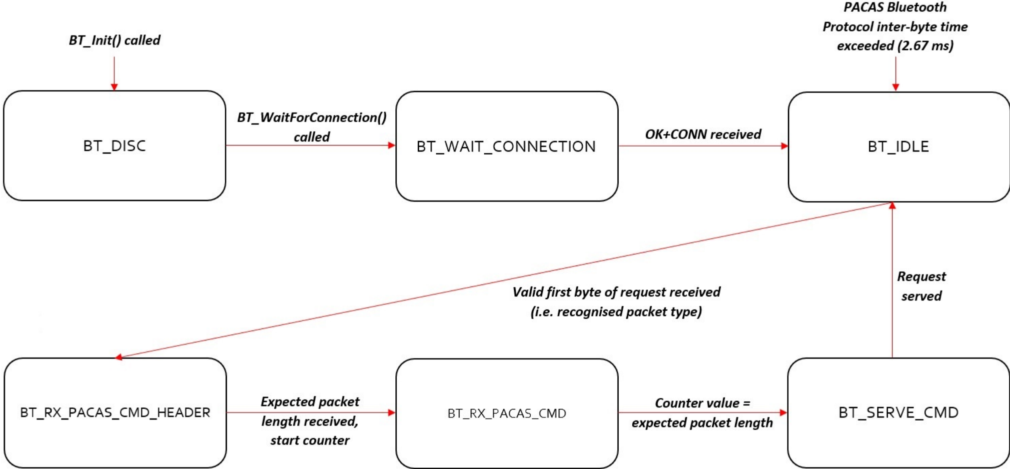


Figure A.5: PACAS Remote Bluetooth state machine

A.3 PACAS Main C/C++ Library

A.3.1 Modules

Module name	Files	Remarks
utils	utils.(h/c)	Functionality such as delay, random number generator and millisecond counter
serial	serial.(h/c)	Provides a printf style function
bluetooth	btll.(h/c), btmain.(h/c), bt_protocoldefs.h	Implements the Bluetooth communication protocol described in Appendix A.2. Microcontroller specific operations are handled by btll, with btmain providing higher level functionality.
gnss	gnss_dma.(h/c), gnssll_dma.(h/c)	Implements interactions with the GNSS module as described in Section 3.2.3. Microcontroller specific operations are handled by gnssll_dma, with gnss_dma providing higher level functionality.
radiohead	RH_RF95.(h/cpp), low level drivers	A port of the RH_RF95 module of the RadioHead project
pacas_main	pacas_main.(h/cpp)	A module that ties the above into one header file

Table A.1: Modules in the `pacas_main.h` library

The only dependency of `pacas_main.h` is on Texas Instruments' DriverLib for the MSP432 which must be included in the project directory.

A.3.2 Classes and data structures

Module *utils*

None

Module *serial*

None

Module *bluetooth*

None

Module *gnss*

`struct GNSS_PVT_t`: A 92 byte large data structure to store a UBX-NAV-PVT message [6].

Module *radiohead*

`class RH_RF95`: Driver to send and receive unaddressed, unreliable datagrams via a LoRa capable radio transceiver (e.g. SX1276) [7]. An instance of the radio is declared in pacas_main.cpp.

Module *pacas_main*

`struct PACAS_SENSOR_DATA_t`: Stores altitude and airspeed data.

A.3.3 Functions

This section documents most of the available functions, grouped by module.

Module *utils*

```
// Initialise utils module
void Utils_Init();

// Delay for the given number of ms (goes into LPM0, stopping CPU clock)
void delayMS(uint32_t ms);

// Generate a random integer from the set {from, ..., to}
// Uses a seed generated by the ADC and an LCG to generate the numbers
uint32_t random(uint32_t from, uint32_t to);

// Return the number of milliseconds since Utils_Init() was called
uint32_t millis();
```

Module *serial*

```
1  // Initialise EUSCIA0 for UART mode at the baud rate
2  // PRINT_UART_BAUDRATE (default: 115200)
3  void uart_println_Init();
4
5  // Print a formatted string to a host computer
6  // This function may be used exactly like printf()
7  // However, support for floats may need to be configured in the IDE settings
8  void uart_println(const char* str, ...);
```

Module *bluetooth*

```
1  // Initialise state machine and EUSCIA2 for UART mode at the baud rate
2  // BT_UART_BAUDRATE (default: 115200)
3  void BT_Init();
4
5  // Transmit a string of length n
6  void BT_TransmitStr(char* istr, uint8_t n);
7
8  // Request sensor data. The default callback function will transfer results
9  // to dst which may be an array or of type PACAS_SENSOR_DATA_t (or anything
10 // which uses contiguous blocks of memory)
11 // The value of settings may be 0 or BT_SENSOR_SETTINGS_BARO_USE_TAYLOR
12 // which controls whether a Taylor series approximation is used to calculate
13 // the altitude
14 void BT_SendSensorCommand(void* dst, uint8_t settings);
15
16 // Write to PACAS Remote's flash. The default callback function is empty
17 // src is a pointer to the beginning of the data source
18 // settings must be BT_FLASH_SEG_M | BT_FLASH_XFER_NB
19 // Where M is 0, 1, or 2 and N ranges from 01 to 64.
20 // This selects the destination flash segment and number of bytes to be
21 // transferred.
22 // Example to write a byte to flash segment 0:
23 // BT_SendWriteFlashCommand(src, BT_FLASH_SEG_0 | BT_FLASH_XFER_01B);
24 void BT_SendWriteFlashCommand(uint8_t* src, uint8_t settings);
25
26 // Read from PACAS Remote's flash. The default callback function will transfer
27 // results to dst
28 // dst is a pointer to the beginning of the destination
29 // settings must be BT_FLASH_SEG_M | BT_FLASH_XFER_NB
30 // Where M is 0, 1, or 2 and N ranges from 01 to 64.
31 // This selects the source flash segment and number of bytes to be transferred.
32 // Example to read the first 15 bytes from flash segment 3:
33 // BT_SendWriteFlashCommand(dst, BT_FLASH_SEG_3 | BT_FLASH_XFER_15B);
34 void BT_SendReadFlashCommand(uint8_t* dst, uint8_t settings);
```

```
35 // Restart PACAS Remote. This terminates the Bluetooth connection as a side
36 // effect. There is no reply for this command
37 void BT_SendPowerUpClearCommand();
38
39 // Returns a non-zero value if an error was encountered during the previous
40 // command such as a timeout or flash error
41 uint8_t BT_GetLastTransactionStatus();
42
43 // These callback functions are executed after a response is received.
44 // They may be implemented/expanded upon elsewhere in user code.
45 // These functions are called from within an ISR so best practice is to keep
46 // them short.
47 inline void BT_SensorDataCallback();
48 inline void BT_WriteFlashCallback();
49 inline void BT_ReadFlashCallback();
```

Module *gnss*

```
1 // Initialise DMA, EUSCIA1 for UART mode at the baud rate
2 // GNSS_UART_BAUDRATE (default: 115200)
3 // Also enables the PVT message on the GNSS module
4 // A delay of 5 s should be used after calling this function to enable the
5 // GNSS module to start up
6 void GNSS_Init();
7
8 // Disable DMA and PVT message
9 void GNSS_Deinit();
10
11 // Write the most recently received PVT message into pvt
12 // Returns a non-zero value if data is corrupted
13 uint8_t GNSS_UpdatePVT(GNSS_PVT_t* pvt);
14
15 // Return a UNIX timestamp in ns from the given PVT struct
16 uint64_t GNSS_TimestampNS(GNSS_PVT_t* pvt);
```

Module *radiohead*

The documentation for the ported RH_RF95 class and low level drivers can be found on the RadioHead website [7]. Some changes were made to the member functions of the RH_RF95 class which are described below.

```
1 // New default constructor for MSP432 with channel settings:
2 // 868.1 MHz, BW 125 KHz, SF 7, CR 4/5, TX power +14 dBm
3 RH_RF95::RH_RF95();
```

```
4  // Executes the CSMA/CA routine before sending a packet
5  bool RH_RF95::send(const uint8_t* data, uint8_t len);
```

Module *pacas_main*

```
1  // Initialise all other modules and connect to PACAS Remote
2  // Use verbose = 0 to suppress progress messages
3  // Returns a non-zero value if a module failed to initialise
4  uint8_t PACAS_SystemInit(uint8_t verbose);
```

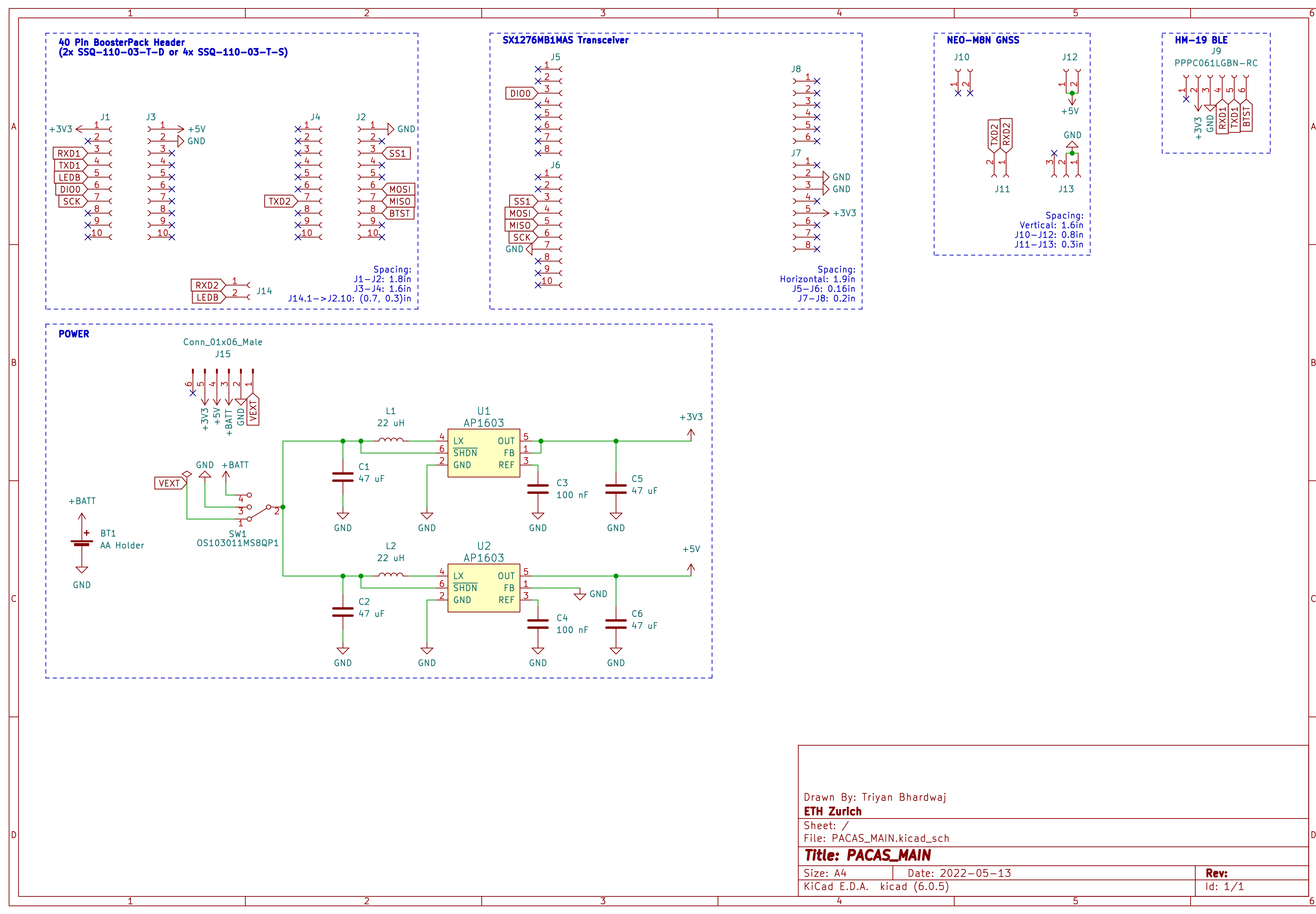
Minimal working example

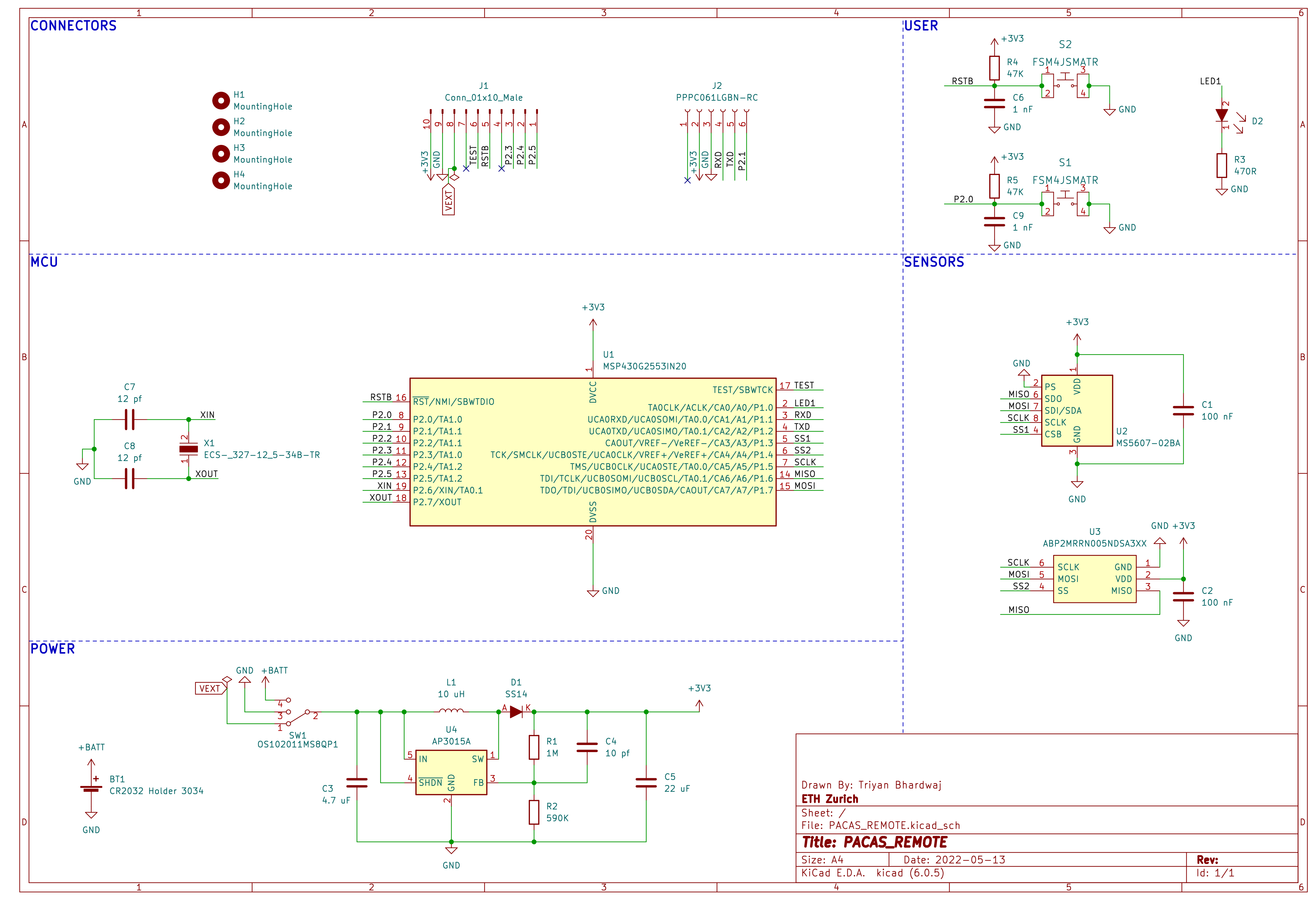
```
1  #include <pacas_main.h>
2  ...
3  PACAS_SENSOR_DATA_t sensor_data;
4  GNSS_PVT_t pvt;
5  char radio_buf[19] = {0};
6  ...
7  void main(void)
8  {
9      WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // Stop watchdog timer
10     if(PACAS_SystemInit(1))
11         while(1); // Some module failed to initialise
12     ...
13     while(1)
14     {
15         BT_SendSensorCommand(&sensor_data, 0); // Request sensor data
16         // Do something asynchronously (60-150 ms) here while computation occurs
17         // When you are ready to use sensor data, use the next 2 statements
18         while(BT_STATE != BT_IDLE);
19         if(BT_GetLastTransactionStatus())
20             uart_println("Error updating sensor data.");
21
22         if(GNSS_UpdatePVT(&pvt)) // Now update PVT data
23             uart_println("Error updating PVT solution.");
24
25         snprintf(radio_buf, 19, "%llu", GNSS_TimestampNS(&pvt));
26         sx1276.send(radio_buf, 19); // Transmit UNIX time via transceiver
27         ...
28     }
29 }
```

Appendix B

Schematics and PCB Layout

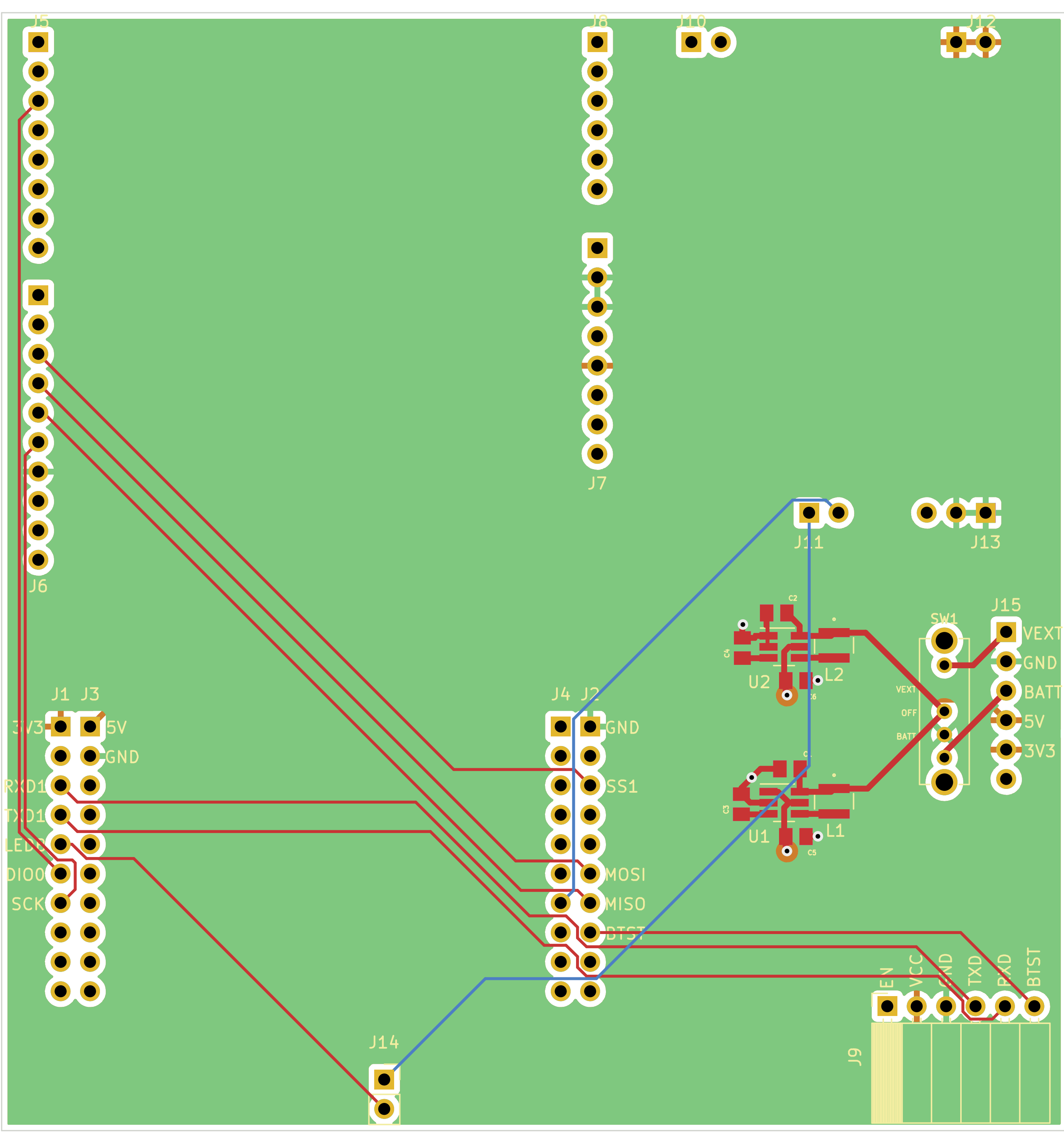
The schematics and PCB layouts of both boards are provided for reference.


40 Pin BoosterPack Header
(2x SSQ-110-03-T-D or 4x SSQ-110-03-T-S)
J1
J2
J3
J4
J14
+3V3
+5V
GND
RXD1
TXD1
LEDB
DIO0
SCK
TXD2
SS1
MOSI
MISO
BTST
RXD2
Spacing:
J1-J2: 1.8in
J3-J4: 1.6in
J14.1->J2.10: (0.7, 0.3)in
SX1276MB1MAS Transceiver
J5
J6
J7
J8
Spacing:
Horizontal: 1.9in
J5-J6: 0.16in
J7-J8: 0.2in
NEO-M8N GNSS
J10
J11
J12
J13
Spacing:
Vertical: 1.6in
J10-J12: 0.8in
J11-J13: 0.3in
HM-19 BLE
J9
PPPC061LGBN-RC
POWER
Conn_01x06_Male
J15
VEXT
+BATT
BT1
AA Holder
SW1
OS103011MS8QP1
L1
22 uH
L2
22 uH
U1
AP1603
U2
AP1603
LX
SHDN
GND
OUT
FB
REF
C1
47 uF
C2
47 uF
C3
100 nF
C4
100 nF
C5
47 uF
C6
47 uF
Drawn By: Triyan Bhardwaj
ETH Zurich
Sheet: /
File: PACAS_MAIN.kicad_sch
Title: PACAS_MAIN
Size: A4
Date: 2022-05-13
KiCad E.D.A. kicad (6.0.5)
Rev:
Id: 1/1



CONNECTORS
H1 MountingHole
H2 MountingHole
H3 MountingHole
H4 MountingHole
J1 Conn_01x10_Male
+3V3 GND VEXT TEST RSTB P2.3 P2.4 P2.5
J2 PPPC061LGBN-RC
+3V3 GND RXD TXD P2.1
USER
RSTB R4 47K +3V3 C6 1 nF GND
S2 FSM4JSMATR GND
P2.0 R5 47K +3V3 C9 1 nF GND
S1 FSM4JSMATR GND
LED1 D2 R3 470R GND
MCU
C7 12 pf
C8 12 pf
X1 ECS-_327-12_5-34B-TR
XIN XOUT GND
U1 MSP430G2553IN20
+3V3 DVCC DVSS GND
RSTB 16 RST/NMI/SBWTDIO
P2.0 8 P2.0/TA1.0
P2.1 9 P2.1/TA1.1
P2.2 10 P2.2/TA1.1
P2.3 11 P2.3/TA1.0
P2.4 12 P2.4/TA1.2
P2.5 13 P2.5/TA1.2
XIN 19 P2.6/XIN/TA0.1
XOUT 18 P2.7/XOUT
TEST/SBWTCK 17 TEST
TA0CLK/ACLK/CA0/A0/P1.0 2 LED1
UCA0RXD/UCA0SOMI/TA0.0/CA1/A1/P1.1 3 RXD
UCA0TXD/UCA0SIMO/TA0.1/CA2/A2/P1.2 4 TXD
CAOUT/VREF-/VeREF-/CA3/A3/P1.3 5 SS1
TCK/SMCLK/UCB0STE/UCA0CLK/VREF+/VeREF+/CA4/A4/P1.4 6 SS2
TMS/UCB0CLK/UCA0STE/TA0.0/CA5/A5/P1.5 7 SCLK
TDI/TCLK/UCB0SOMI/UCB0SCL/TA0.1/CA6/A6/P1.6 14 MISO
TDO/TDI/UCB0SIMO/UCB0SDA/CAOUT/CA7/A7/P1.7 15 MOSI
SENSORS
U2 MS5607-02BA
+3V3 GND PS SDO SDI/SDA SCLK CSB VDD GND
MISO MOSI SCLK SS1
C1 100 nF
U3 ABP2MRRN005NDSA3XX
GND +3V3
SCLK MOSI SS2 MISO
SCLK MOSI SS GND VDD MISO
C2 100 nF GND
POWER
VEXT GND +BATT
SW1 OS102011MS8QP1
L1 10 uH
D1 SS14
+3V3
U4 AP3015A
IN SW SHDN GND FB
R1 1M
C4 10 pf
R2 590K
C3 4.7 uF
C5 22 uF
GND
+BATT BT1 CR2032 Holder 3034 GND
Drawn By: Triyan Bhardwaj
ETH Zurich
Sheet: /
File: PACAS_REMOTE.kicad_sch
Title: PACAS_REMOTE
Size: A4
Date: 2022-05-13
KiCad E.D.A. kicad (6.0.5)
Rev:
Id: 1/1

PACAS_MAIN PCB Layout



PACAS_REMOTE PCB Layout