Miniaturized Radio Tranceiver for PocketQubes, Exceeding Performance of CubeSat Solutions

Broekhuizen, C.H.; Speretta, S.; Uludag, M.S.; van den Bos, M.F.; Haenen, J.; Menicucci, A.; Gill, E.K.A.

Publication date
2020

Document Version
Final published version

Published in
34th Annual Small Satellite Conference

Miniaturized Radio Tranceiver for PocketQubes, Exceeding Performance of CubeSat Solutions

Casper Hendrik Broekhuizen, Stefano Speretta, Mehmet Sevket Uludag, Michael van den Bos, Jasper Haenen, Alessandra Menicucci, Eberhard Gill
Delft University of Technology
Kluyverweg 1, 2629 HS Delft, The Netherlands; 0031 (0)15 27 81967
C.H.Broekhuizen@tudelft.nl

ABSTRACT

In this paper, a detailed design is presented of a communications module that is designed to fit tight PocketQube design budgets but still offer performance at least comparable to commercial off-the-shelf CubeSat solutions. The communications module features extremely efficient power usage, less than 2 Watt DC for 1 Watt RF output while fitting in an extremely small volume, (42 x 42 x 8mm, approximately a quarter of the volume of CubeSat solutions). Our system also features a new communication scheme based on Short Block LPDC codes that provides a very high code gain (approximately 6dB for hard-decision and 9dB for soft-decision) using a high code rate. A ground modem implementation based on GNURadio is also presented, taking advantage of a new implementation for low-latency asynchronous data transmission.

INTRODUCTION

PocketQube[1] are satellites even smaller than CubeSats, designed to push miniaturization even further as volume, mass, and available power have to shrink. This paper focuses on a communication system specifically targeted as PocketQubes but designed to compete in terms of performances with bigger CubeSat systems. The volume of one "unit" (called 1P, one PocketQube 'cube') is approximately 50 mm x 50 mm x 50 mm with a mass of less than 250g. The important challenges were the reduction of available design budgets, such as mass, volume, and power. Orbit average available power, for example, is approximately 400 mW for one single unit (in the case of Delfi-PQ this is approximately 1.25W considering 3 units, or 3P). However, thanks to the reduced size and mass of PocketQubes, they can be launched economically in large numbers: this opens up opportunities to build a distributed swarm of sensors for relatively low costs. The Delft University of Technology is in the process of developing its own PocketQube, Delfi-PQ, and expects to launch in late 2020. The same communications sub-system is also envisioned for a lunar rover mission, where high coding gain and power efficiency are extremely important. In this paper, we will present Delfi-PQ and provide an overview of the satellite. We will then focus on the communication module, presenting its hardware and software design. We will present a communication scheme based on the CCSDS Short Block LDPC codes that provides a very high coding gain while still guaranteeing a very simple implementation. We will present decoding performances of the selected protocol and the software design of our ground station modulator and demodulator. An automated test setup, that also allows for remote control of the board over the internet during the Covid19 crisis, will also be presented.

DELFI-PQ

Delfi-PQ (Figure 1) is the first PocketQube developed by the Delft University of Technology, with the aim to set a mechanical standard for this type of satellite and flight test the developed core bus[3]. The long-term goal of Delfi-PQ is developing a core platform with basic functionalities that will be developed in an iterative approach over time. Since this effort is being carried out by a university, it is desired that as many students as possible work on the satellite as part of their education, exactly as all previous missions had a clear objective for education, technology demonstration, and innovation. It is intended that once the first design is validated in flight, there will always be a satellite available ready for an eventual launch. The first mission objective is education and, in particular, giving students practical experience with a space project as part of their education. This is one of the reasons for the iterative design approach: by performing small design updates and going through the full testing and qualification cycle each time, we expect to give a realistic experience to students being involved in the different design phases. A second objective for Delfi-PQ is a technology demonstration, intended to provide a flight opportunity to different technologies developed by university researchers. The spacecraft has a size of 50x50x178 mm and internally is made by a single stack of boards stacked on top of each other. All boards rely on a connector providing electrical connectivity and on four rods providing mechanical support. The satellite is made by five main modules: the power module, the on-board computer, the antenna deployment module, and the communication module, which will be the emphasis of this paper.

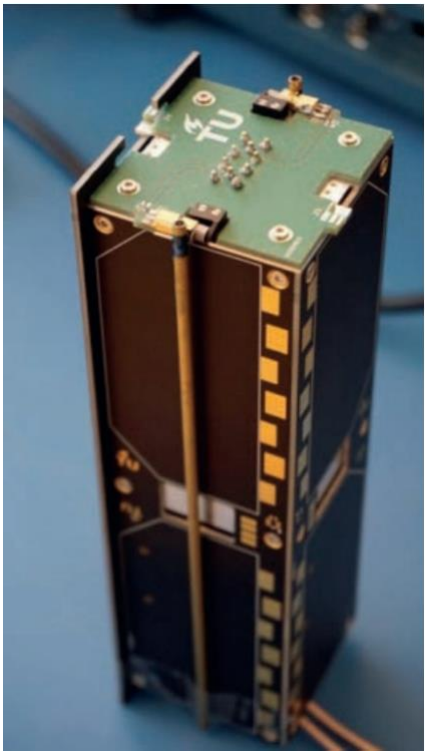

Figure 1: Delfi-PQ

COMMUNICATION MODULE

The communications module has been designed with modularity in mind, trying to make it feasible to accommodate different possible use cases with a single system. To achieve this, the system has been split into two separate boards: the mainboard, accommodating the microcontroller (MCU) managing the system, and two transceiver integrated circuits (ICs) implementing the transmitter and the receiver. Two separate chips allow for the system to be used for both half- and full-duplex communications. The daughterboard, plugs on top of the main one, is used to accommodate the power amplifier and the low-noise amplifier.

The overall form-factor has been designed to be compatible with the rest of the satellite [4] and to the PocketQube standard [2] to be able to rely on standard deployers for launch.

The use of a daughterboard also allows the implementation of different functionalities that were initially not considered when first designing this system. This is the case for one of the payloads to be flown on Delfi-PQ that will perform wide-band signal monitoring for very-low-frequencies (less than 30 MHz). For this application, we will re-use the receiver hardware and connect it to a mixer to accommodate a frequency band not natively supported by the hardware.

The main characteristics of this module are summarized in Table 1.

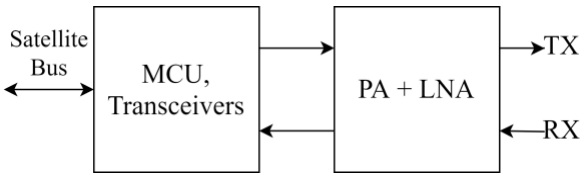

Figure 2: Communication Module Hardware Architecture

Table 1: Radio Service Replies

Feature	Value
TX Frequency Range	390 - 450 MHz
RX Frequency Range	140 - 170 MHz
Transmit Power	24 / 27 / 30 dBm
Receiver Sensitivity	-148 dBm
Modulation	GMSK
Data rate	9600 bps (up to 300 kbps)
Datalink	AX.25 or Advanced LDPC
Mass	30 g
Dimensions	42 mm x 42 mm x 8 mm
Power Consumption (RX)	60 mW
Power Consumption (TX)	0.5 / 0.9 / 1.8 W

HARDWARE DESIGN

All components used on the board are commercial-grade and have been selected based on the radiation tolerance performances found in literature. This alone does not guarantee full radiation hardness as lot-code screening has not been performed but it already provides a baseline for future improvements. Latch-up protection has been implemented to ensure eventual events do not become destructive: limiter resistors are present on all functional groups in the system to prevent overheating of the integrated circuits. A 3-level watchdog system has been implemented to protect the board and power-cycle all the systems in case one of the 3 separate protection triggers (see Figure 3). A load-switch controlling an external transistor is used to monitor the total power consumption and power-cycle in case over-current events are detected. An extra power-cycle can be triggered by the MCU, monitoring continuously the health of the different systems, and by an external watchdog monitoring the health of the MCU. Together with the limiting resistors, this system is expected to provide an effective way to address potential failures due to latch-up.

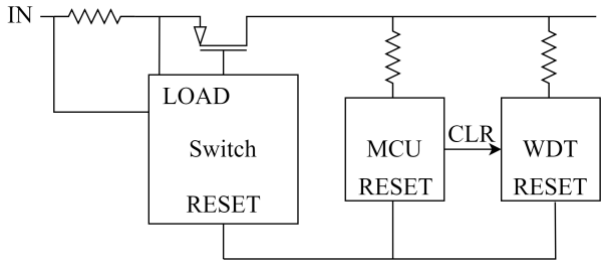

Figure 3: Watchdog Schematic

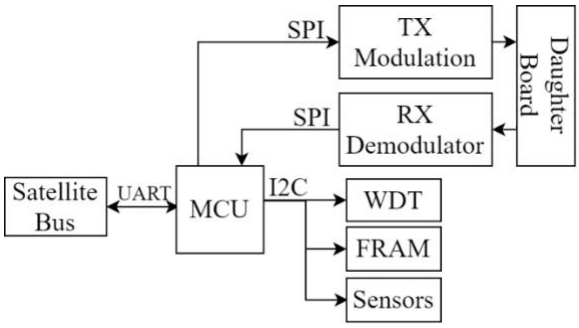

Figure 4: Mainboard Conceptual Schematic

Main board

The mainboard consists of a microcontroller (MCU), an MSP432, based on an ultra-low-power ARM Cortex-M4 core. This is a 32-bit MCU featuring 2 MB of internal Flash memory and 256 kB of SRAM. Besides this, the board is also equipped with a 512 kb FRAM chip to store configurable non-volatile parameters: this has been selected to take advantage of the extremely high number of read/write cycles, making it optimal for storing information that needs to be modified often. An overview of the board functionalities is depicted in Figure 4 and Figure 5 shows the mainboard.

The transmitter and receiver sections are based on an integrated transceiver IC from Semtech (SX1278): this device features an FSK/GFSK/GMSK radio engine and also a separate LoRa one. Despite the LoRa system is currently not in use, it provides an interesting option for future experimentation. This device is also capable of covering the full VHF and UHF frequency bands providing flexibility in choosing which specific portion to use (commercial or amateur, for example). The main rationale for employing two separate transmit and receive sections was to allow for full-duplex communication. This increases system complexity but also simplifies greatly operations.

The board features also a high-stability reference oscillator, used to feed both the transmitter and the receiver radio sections.

The board also includes the RS-485 transceiver used to communicate on the satellite bus and an extra communication line to connect directly to the satellite power system. This extra connection is used with a specific radio command (fire code) implemented to request a full-satellite power cycle in case of an emergency. This fire code has been designed as one of the emergency measures to force a satellite reset when the satellite bus, the power systems MCU, or the on-board computer would not be operating nominally.

Figure 5: Communication Module Mainboard

Daughterboard

The concept of a motherboard/daughterboard has been selected to leave design flexibility while not requiring the full system to be designed. For this reason, control and RF modulation/demodulation have been included in the mainboard. The current daughterboard design only features a low-noise amplifier, to provide extra selectivity to the radio, and a power amplifier.

Given the limited available power on Delfi-PQ, the power amplifier was designed to maximize the overall efficiency. Thanks to the modulation scheme selected (GMSK) providing a constant envelope, a fully saturated amplifier was selected. With a default 24 dBm (¼ W) RF power output, the system is capable of achieving a power-added-efficiency of 61%. By controlling the amplifier bias stage, selectable power output has also been implemented allowing to get an output power of 27 dBm and 30 dBm (½ W and 1W) while achieving an efficiency of 65%. This efficiency is considerably higher than comparable systems for small satellites and CubeSats: PocketQubes are much more constrained by the available resources and provide an extra challenge to improve the overall system performances while also reducing the volume and mass by a factor 4.

SOFTWARE DESIGN

The core software is using a simple (pseudo-) real-time task-based system with a round-robin scheduler following a static task list[5] specifically designed for Delfi-PQ. This minimal operating system implements most of the functionalities that are common to all subsystems in the satellite like the hardware low-level drivers, communication on the shared data bus, and the task manager. The module is controlled over the satellite bus and communications have been integrated into the operating system using so-called 'services'. They act as a way to multiplex different types of messages on the same bus, allowing them to create a standard set of functions (or services) that are present on all systems and some system-specific services. The following section will give an overview of the key tasks and services running on the board.

Satellite Bus communication

A dedicated task is running in the background to monitor the activity on the satellite bus and interact with the other modules present. The bus is based on RS-485 using multi-processor mode by means of a 9th bit. This is one of the standard solutions for such busses and it relies on an extra bit being set to a logic state high when the module address is sent: this alerts all modules and only the one with the matching address will respond. Our protocol is kept quite minimal and implements the data link layer, networking layer, and transport layer as described by the OSI model.

Figure 6 gives an idea of the data transfer units on the bus, showing also the different frame parts. The frame structure is very similar to the Packet Utilisation Service but providing only a subset of the functionality. The networking layer is used to address the different modules in the satellite and ensure the correctness of the frame collisions when two systems start to communicate at the same time. The transport layer (bottom part of Figure 6) provides separation to the different services (as in the PUS standard), and it allows to define a request and a reply.

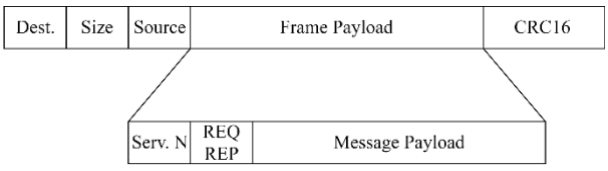

Figure 6: PQ9 Protocol consisting of a PQ9 Frame and PQ9 Message

OTA Reprogramming Capability

One of the features of the Delfi-PQ modules is Over-The-Air reprogramming capability. This capability allows the user to reprogram the module software in situ, either over the satellite bus or directly over the radio. This capability gives more flexibility and adaptability in terms of use, as the module can be completely reconfigured and its functionality can be changed, as long as the hardware allows. However, one of the important aspects of this functionality is failure tolerance. In order to allow an extra layer of safety to reprogram, multiple software versions can be stored on the internal FLASH memory, and the primary software version (Slot 0 in Figure 7) cannot be reconfigured, in order to ensure a 'safe' fallback.

The update will consist of one binary file with a maximum size of 512 kilobytes (one-quarter of the total memory size). Using one-quarter of the memory per each different slot is driven by the internal memory layout on the MSP432: memory is divided into two segments which can be individually write-protected. Because of this, it was decided to write-protect the first segment (to avoid unintentional reprogramming), leaving the second one open for reprogramming.

The binary file can be uploaded in chunks of 32 bytes and verified via a cyclic redundancy check. However, it is impossible to convert a chunk of 32 bytes into one unique CRC8 byte, so a collision might still happen. Therefore, at the end of the transmission, a check on the entire transmitted file is done to ensure its correctness using an MD5-hash. Once the full memory slot has been programmed and verified, the module default boot sector can be changed. In case of a boot failure, the system will fall back to the default program.

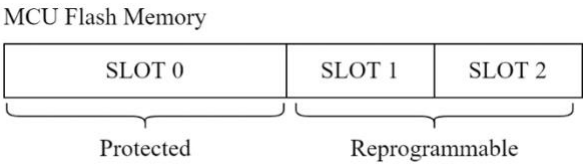

Figure 7: Flash memory of MSP432 divided into programmable slots

GROUND – SPACE PROTOCOL

The communication module implements two different Ground-Space protocols to cover two very specific use cases. Since many small satellites use amateur frequencies, supporting common protocols used by amateurs is often required and limits the effort in designing a custom ground segment: for this reason, AX.25[6] has been implemented to make use of the wide Amateur Radio community On the physical layer, this protocol is implemented using a scrambler

(G3RUH) and inverted non-return-to-zero encoding (NRZI)[7]. Thanks to the radio flexibility, multiple data rates are supported (besides 9600 bps that is commonly used with these settings).

To improve on the AX.25 required signal-to-noise ratio, an advanced packet-based communication protocol has been implemented based on Low Density Parity Checks (LDPC) to drastically improve performances. The protocol is based on a fixed block size to limit complexity: this could become a downside of this schema as often the fixed codeword size is often quite large. This forces the user to pad small data packets in order for them to be coded, which can become quite a sizable overhead when the communication only requires very short frames. Our goal became to define a scheme supporting small blocks (in the order of 32 - 64 bytes) and allowing to efficiently concatenate more of them without incurring into the penalty of transmitting synchronization sequences in between (that could become almost as long as the block size to guarantee synchronization with an E_b/N_0 approaching few decibels. Communication Link Transmission Unit (CLTU) as described by CCSDS[8] became a very attractive solution as it fulfills all our requirements and is documented in literature (see Figure 8 for more details).Short block length LDPC codes selected [9] have a block length of either 16 bytes, 32 bytes, or 64 bytes with a code-rate of ½, while still guaranteeing very high coding gain. Such a solution allows one to keep the required modulation and demodulation computational power to a minimum.

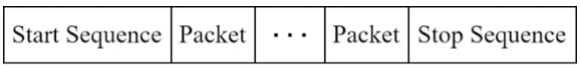

Figure 8: CLTU Overview

LDPC Decoding

Due to the hardware restrictions of the radio transceiver chips used, decoding in uplink is only possible using *hard-bits*, as a hard-decision is already made on the samples by the demodulator. Hard decoding algorithms for LDPC codes are bit flipping algorithms, such as the simple Gallager's Bitflip Algorithm[10]. This algorithm does error correction by flipping the bit that causes the most parity checks (syndromes) to fail:

1. Compute parity checks, if all checksums are successful, stop decoding.
2. Compute the number of failed parity checks per bit position.
3. Flip the bit(s) with the highest number of failed parity checks.
4. Repeat setup 1-3 until all checksums are satisfied or a maximum of iterations is reached (unsuccessful decoding).

On the downlink side, instead, such limitations do not apply as hardware complexity can be much higher allowing soft-bit decoding. Efficient decoding can be achieved using belief-propagation algorithms such as the Sum-Product-Algorithm (SPA) and its computationally optimized version, the Minimum-Sum-Algorithm (MSA)[11]. These algorithms estimate the original message using maximum a posteriori probability by passing information between the Variable nodes (The bits) and the Constraint nodes (the checksums):

1. Computer Log-Likelihood-Ratio of received bits (variable nodes).
2. Compute using the variable nodes the constraint nodes.
3. Using the calculated constraint nodes, re-calculate the variable nodes
4. Convert variable nodes back to bits (hard decision log-likelihood-ratios) and check if checksums are correct.
5. Repeat setup 2-5 until all checksums are satisfied or a maximum of iterations is reached (unsuccessful decoding).

Decoding simulations

Both the uplink and downlink decoding algorithms have been implemented and simulated using a virtual additive white gaussian noise channel. The simulation of the bit flipping algorithm used in the uplink communication shows a coding gain of approximately 6 dB when using a block-length of either 32 bytes (n=256) or 64 bytes (n=512) and a coding gain of approximately 4.4 dB when using a block-length of 16 bytes (n=512) for a bit-error-rate of 10^{-6} (see Figure 9 for further details).

Decoding has been simulated for the downlink which uses soft-bit decoding instead, allowing to greatly improve the code gain. Simulations show a coding gain of approximately 9 dB using a block-length of either 32 bytes (n=256) or 64 bytes (n=512) and a coding gain of approximately 8 dB using a block-length of 16 bytes (n=128) with the SPA algorithm (see Figure 10).

When using the complexity reduced decoding algorithm, MSA, decoding performances are slightly reduced (~0.5 dB). However, the algorithm has a much lower complexity and is numerically more stable[11](Figure 11).

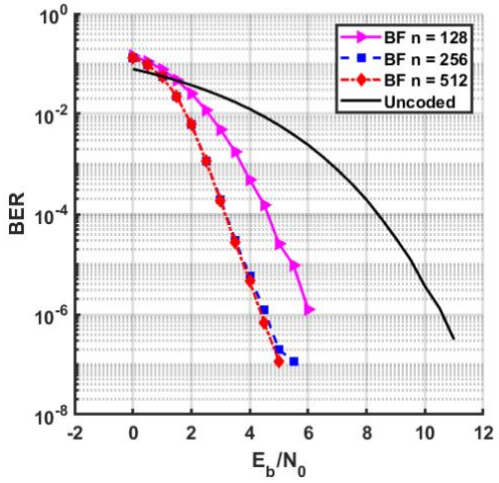

Figure 9: Bit flipping Algorithm: BER vs Eb/N0

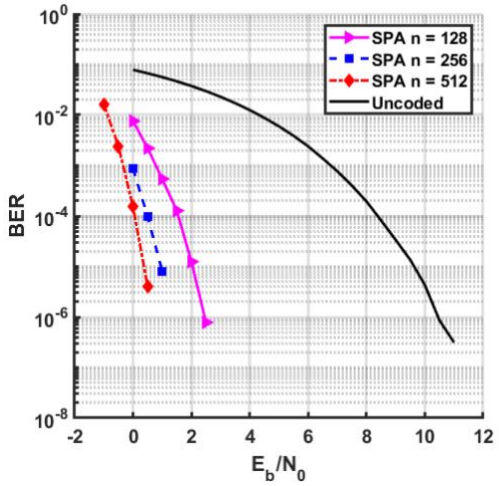

Figure 10: SPA Algorithm: BER vs Eb/N0

GROUND SEGMENT

Our ground segment is based on Software-Defined Radios (SDR), providing us with a lot of flexibility. Also, because of the two different communication schemes we are using (AX.25 and our custom protocol), we have the possibility of running two independent demodulators on the same machine. Our system is based on GNURadio and it is controlled by an external Python application. Our ground system is able to provide a persistent synchronization sequence (when no data is available, to help the satellite receiver to synchronize) and transmit messages asynchronously. One of the problems that appeared during the development phase is related to the way input streams are handled. GNURadio, due to the way its internal scheduler works, will constantly request our external application to produce more data (even if no data is required to be transmitted),

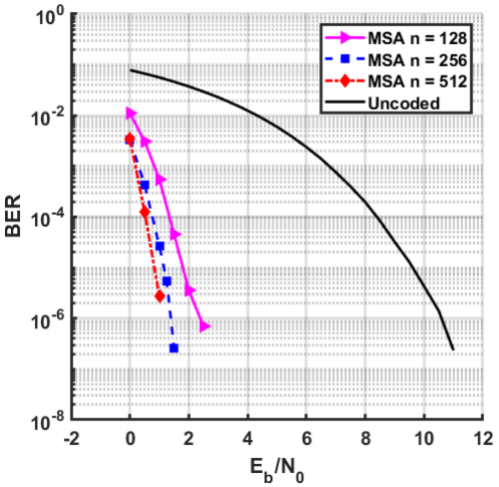

Figure 11: MSA Algorithm: BER vs Eb/N0

producing as many samples as possible to fill the internal buffer in order to avoid stream interrupts. This buffer can considerably grow (up to hundreds of Megabytes and, depending on the actual sampling rate, could require up to a few minutes to be completely flushed) and it causes considerable latency (from seconds to minutes) in case frames need to be sent.

There are different ways to reduce this latency, such as by providing the GNURadio new data at the signal's data rate. However, such a method will require a real-time application to supply samples. A more elegant approach is to create a feedback signal between a data consumer (the SDR) and the data producer (the incoming signal from the Python application), one can control how much data is being produced/consumed by only requesting new samples if the current samples are 'consumed' hence reducing the latency. This concept was already introduced in [12] but is not currently in use on deployed systems. This technique has been implemented by creating a real-time feedback line between the SDR output and the stream connection in GNURadio. Using this new system, the latency of the system has been measured to be 0.18 seconds over extended periods of time. Such a new development allows us to create a very representative implementation of a ground modem, also capable of more advanced features like ranging and tracking, while still taking advantage of the full GNURadio flexibility.

SYSTEM TESTING AND VERIFICATION

In order to properly test the functionality of the communications module (and other Delfi-PQ subsystems), an automated test setup has been created. This test setup directly connects to the satellite bus using a mounting board and allows the user to communicate directly with a computer. Additional

instruments (like a programmable power supply, multi-meters or an RF spectrum analyzer) can be connected to the computer to verify system performances in a fully automated way. A complete automatic test system has been created around the Python PyTest library, which allows running tests very similar with respect to software unit tests. This also includes the generation of a complete test report, that can be generated as soon as a new software version has been compiled and programmed into the hardware. This methodology, often referred to as Test Driven Development, allowed to quickly progress in the development of the board by verifying the complete functionality very often. This allowed us to identify bugs that were impacting other functionalities than the one that had been worked on.

The test equipment that was developed for Delfi-PQ (see Figure 12) heavily makes use of XTCE (XML Telemetric and Command Exchange[13]), a CCSDS standard defining data exchange structures based on an XML definition. This, together with dedicated libraries parsing the binary data to processed telemetry values, allowed an extremely quick development cycle. Telemetry and telecommands from/to the board have been defined using an XTCE file and also used by our application to generate a web-based user interface automatically.

Due to the CoViD19 crisis of 2020, remote testing and remote development also became a factor of the project. By creating a TCP interface on the Satellite Bus and the JTAG debug port on the MCU, one can remotely interact with the hardware, and by means of the Over-The-Air Programming functionality, the software can be reprogrammed by sending new binary files over the TCP Port directly onto the satellite bus. A dedicated setup was located in our cleanroom and connected to a computer that could be remotely accessed. Operators (working from home) could remotely perform all tests by using the developed web-based GUI(Figure 13).

Figure 12: Delfi-PQ Test Board

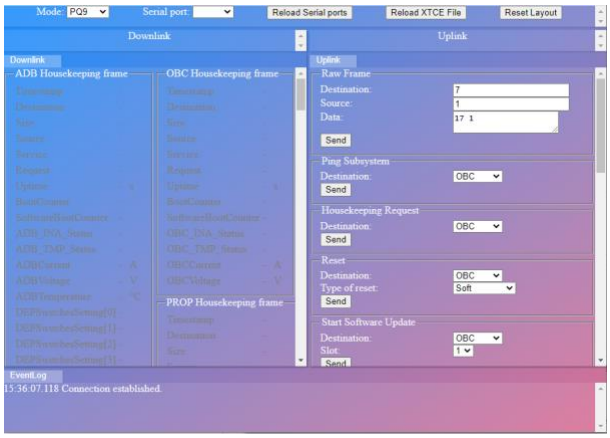

Figure 13: Delfi-PQ Web-based GUI

CONCLUSION

This paper presents the design for a miniaturized communication module for Pocketubes capable of competing with CubeSat solutions, having comparable RF performances, but with much higher efficiency. We developed a radio system capable of achieving 65% power-added-efficiency while producing a 1W RF output power. Moreover, besides legacy radio protocols, this system supports a new protocol based on CCSDS Short Block LDPC codes providing up to 9 dB and 6 dB of coding gain (with soft- and hard-decision demodulation).

REFERENCES

1. Speretta, S., Pérez Soriano, T., Bouwmeester, J., Carvajal Godínez, J., Menicucci, A., Watts, T., Sundaramoorthy, P., Guo, J. and E. Gill, Cubesats to PocketQubes: Opportunities and challenges, Proceedings of the 67th International Astronautical Congress, Guadalajara, Mexico, 2016
2. TU Delft, Alba Orbital, GAUSS Srl., "The PocketQube Standard", July 2018 https://dataverse.nl/api/access/datafile/11680
3. Radu, S., Uludag, M. S., Speretta, S., Bouwmeester, J., Gill, E., and N. C. Foteinakis, Delfi-PQ: The First Pocketqube of Delft University of Technology. Proceedings of the 69th International Astronautical Congress, Bremen, Germany, 2018
4. TU Delft DelfiSpace, PQ9-Template, Delft, Netherlands, May 2020: https://github.com/DelfiSpace/PQ9-Template
5. TU Delft DelfiSpace, DelfiPQCore: Operating System Overview, Delft, Netherlands, May 2020: https://github.com/DelfiSpace/DelfiPQcore/wiki/Operating-System-Overview

6. Tucson Amateur Packet Radio Corporation, AX.25 Link Access Protocol for Amateur Packet Radio, July 1998: http://www.ax25.net/

7. Miller, J., "9600 Baud Packet Radio Modem Design", April 1995: https://www.amsat.org/amsat/articles/g3ruh/109.html

8. Consultative Committee for Space Data Systems, CCSDS 231.0-B-3 TC synchronization and channel coding, Washington, DC, USA, September 2017

9. Consultative Committee for Space Data Systems, CCSDS 231.1-O-1 Short block length LDPC codes for TC synchronization and channel coding, Washington, DC, USA, April 2015

10. Kou, Y., Lin, S., and M.P. Fossorier, Low-density parity-check codes based on finite geometries: a rediscovery and new results, IEEE Transactions on Information theory, 47(7), 2711-2736, 2001

11. Ryan, W.E., and S. Lin, Channel Codes: Classical and Modern, Cambridge University Press, New York, NY, 2009

12. Ettus, M., Managing Latency in Continuous GNURadio Flowgraphs, GRCon19, 2019: https://www.gnuradio.org/grcon/grcon19/presentations/managing_latency/

13. Object Management Group, XML Telemetric & Command Exchange v1.2, January 2019: https://www.omg.org/xtce/index.htm