OPS-SAT Space Lab:
How the in-flight implementation of powerful on-board communication protocols completely changed the operations on OPS-SAT-1

David Evans, OPS-SAT Space Lab Manager, ESA

STINT workshop (IEEE WiSEE 2022)

13/10/2022

What is the OPS-SAT Space Lab?

A free ESA service offering access to a family of powerful, open, reconfigurable laboratories in orbit

Each lab has a unique theme (e.g. OPS-SAT VOLT's theme is optical and quantum communication) but they all share three common characteristics

1. They allow 3rd party experimenters to load their own software/firmware to the spacecraft and execute it either in real-time or offline. Now the nature of those experiments cannot be known in advance and they cannot be extensively ground tested before. Hence it shall be ensured by design that if any potential experiment fails the spacecraft can be recovered without permanent damage
2. They all include a powerful system on module (powerful CPU with a reconfigurable FPGA) on which the experiments will run. From here the experiments can reconfigure and control all the spacecraft subsystems and payloads
3. They have uplink speeds of at least 256 kbps. This is needed to allow the software and firmware experiments to be uploaded in reasonable timeframes and we expect this to be done on a daily basis

OPS-SAT-1 is in orbit, OPS-SAT VOLT and OPS-SAT-2 on the way

OPS-SAT-1

3U cubesat launched by ESA on 18th Dec 2019

First nanosatellite to be directly owned by ESA and controlled by ESA/ESOC

The result of 7 years of development supported by GSTP and ESOC internal investment

- OPS-SAT looks like an advanced ESA spacecraft to the ground. The uplink rate is four times higher than any ESA spacecraft; it employs never flown before communication protocols and implements new ESA patents
- At the centre of OPS-SAT is a high-performance control processor. This allows “normal” software (Linux, Java, Python…) to control the entire satellite: rotate, take pictures, classify them, compress them, send them to the ground, etc
- European industry and institutions can use the platform to rapidly test their software and firmware experiments in space at no cost and no bureaucracy.

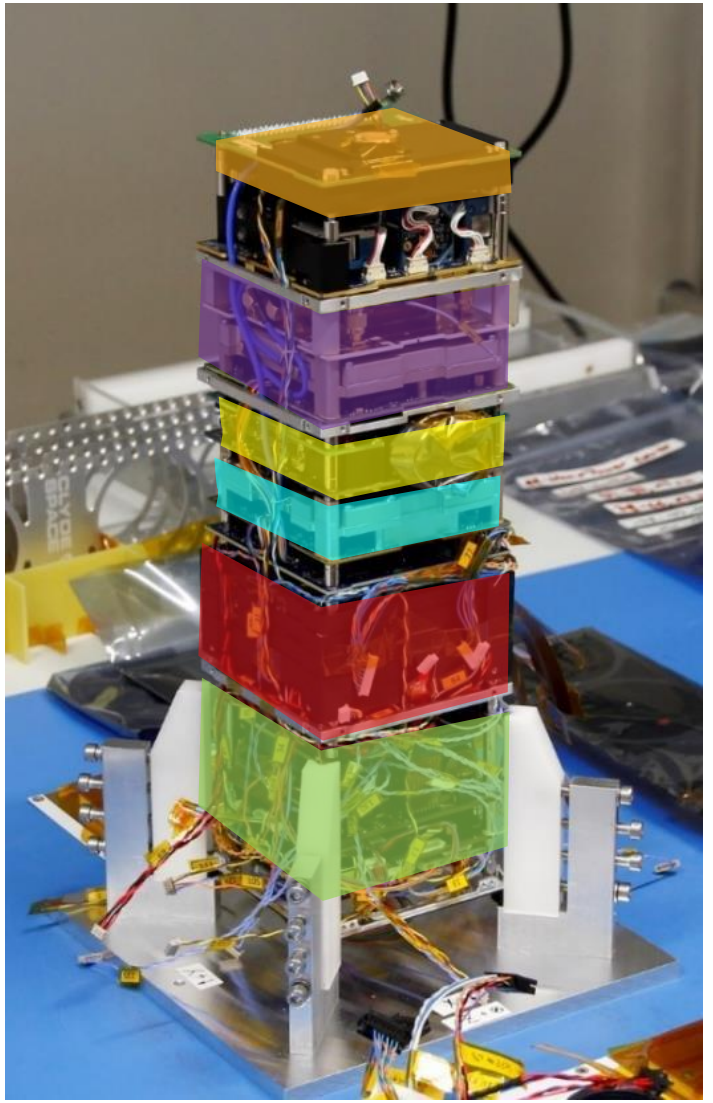
OPS-SAT-1 Space Segment

Satellite bus:

- Gomspace UHF AX100 radio + EPS/ACU
- Nanomind A3200 OBC (On-board computer, AVR32)
- S-band (2.2 GHz) TRX TMTC encoder/decoder (256kbps↑ 1Mbps↓)
- GNSS receiver

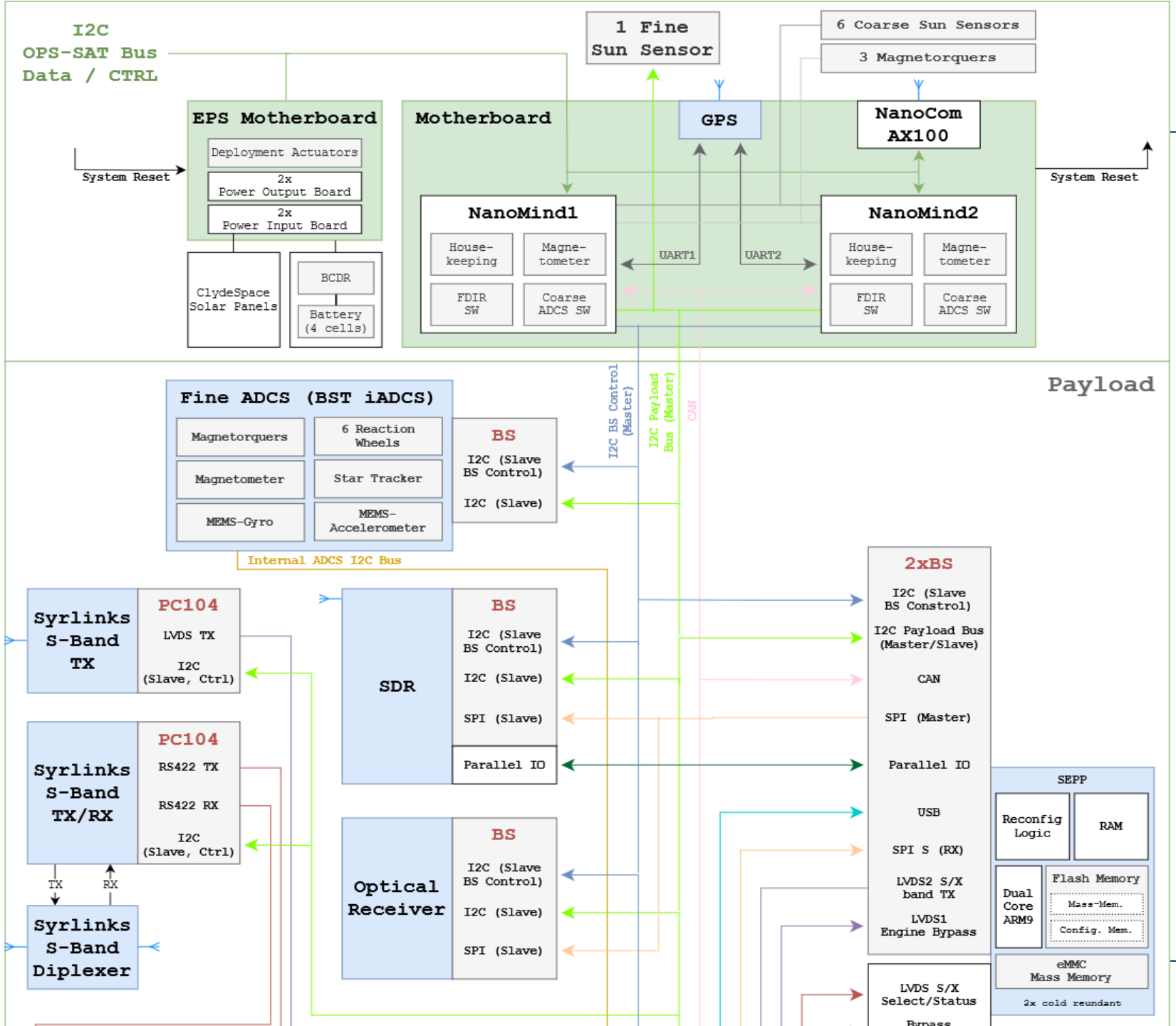
Satellite payloads available to experimenters:

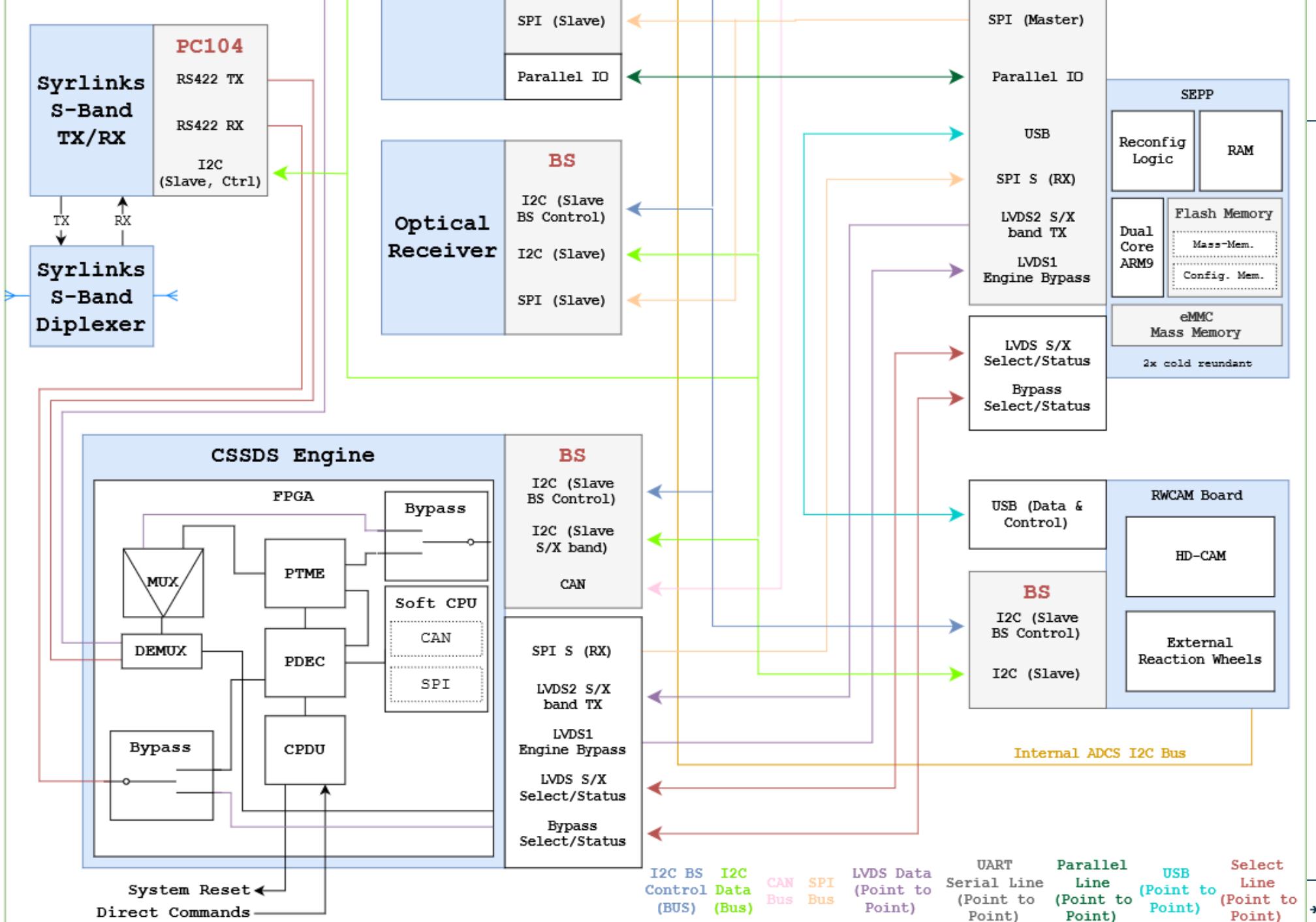
- Software Defined Radio (LMS6002D)
- HD-camera (Nadir-facing)
- Optical receiver (data uplink via laser)
- Advanced iADCS (Attitude Determination & Control Sys.)
- X-band transmitter (3-50MBit/s)
- 2x Cyclone V SoC (800MHz Dual Core ARM Cortex-A9 + FPGA fabric)



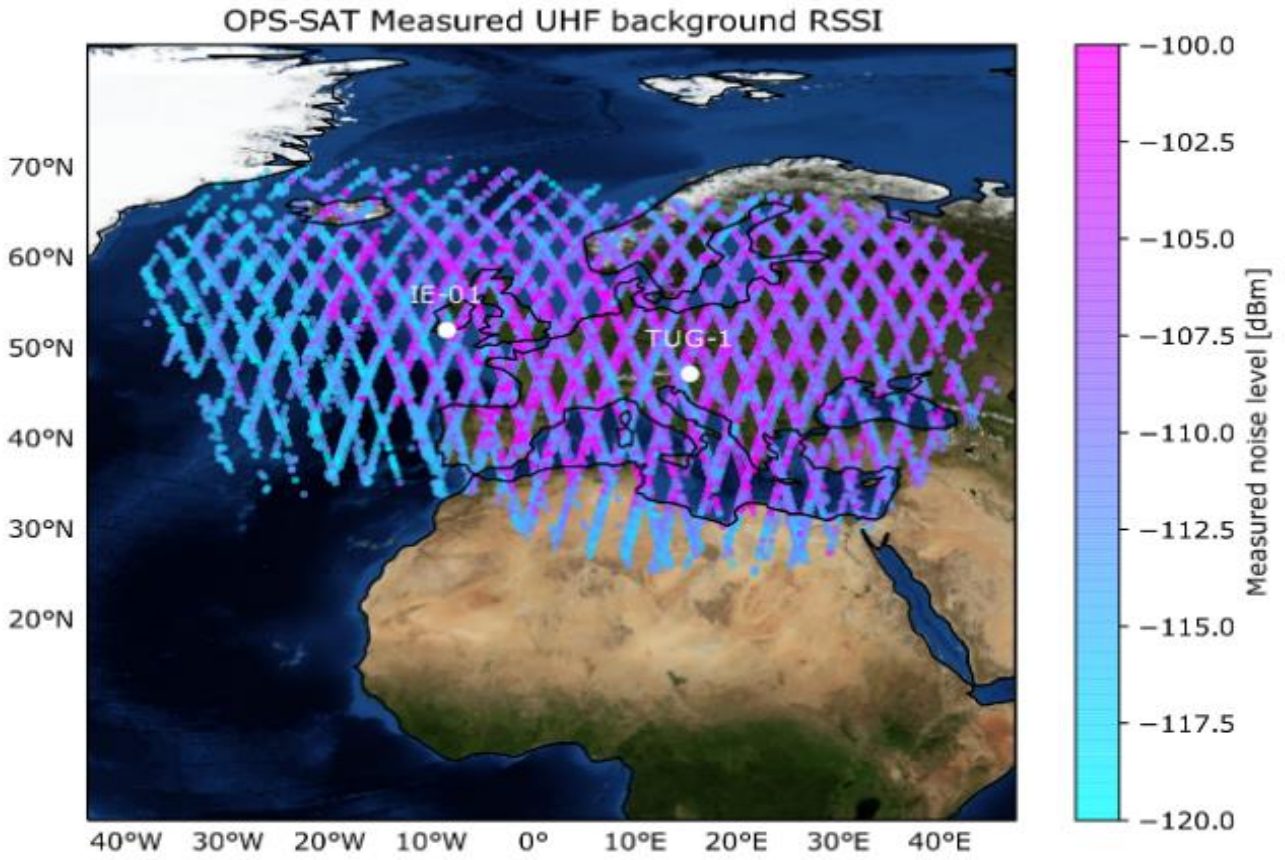
The heart of OPS-SAT-1

- SEPP - developed by TU Graz (mission prime)
- Running embedded Linux (built on Angstrom + Yocto 2.4.4 platform)
- Operated like a remote Linux machine (remote shell, package manager)
- Integrated Altera Cyclone V FPGA
- Software stack:
    - Java Runtime,
    - Python 3.5
    - Payload API user-space libraries
    - NanoSat MO Framework – high level application framework
    - TensorFlowLite – AI from Google….


I2C
OPS-SAT Bus
Data / CTRL
1 Fine Sun Sensor
6 Coarse Sun Sensors
3 Magnetorquers
EPS Motherboard
Deployment Actuators
2x Power Output Board
2x Power Input Board
System Reset
ClydeSpace Solar Panels
BCDR
Battery (4 cells)
Motherboard
GPS
NanoCom AX100
NanoMind1
House-keeping
Magne-tometer
FDIR SW
Coarse ADCS SW
UART1
UART2
NanoMind2
House-keeping
Magne-tometer
FDIR SW
Coarse ADCS SW
System Reset
Payload
Fine ADCS (BST iADCS)
Magnetorquers
6 Reaction Wheels
Magnetometer
Star Tracker
MEMS-Gyro
MEMS-Accelerometer
BS
I2C (Slave BS Control)
I2C (Slave)
I2C BS Control (Master)
I2C Payload Bus (Master)
CAN
Internal ADCS I2C Bus
Syrlinks S-Band TX
PC104
LVDS TX
I2C (Slave, Ctrl)
SDR
BS
I2C (Slave BS Control)
I2C (Slave)
SPI (Slave)
Parallel IO
Syrlinks S-Band TX/RX
PC104
RS422 TX
RS422 RX
I2C (Slave, Ctrl)
TX
RX
Syrlinks S-Band Diplexer
Optical Receiver
BS
I2C (Slave BS Control)
I2C (Slave)
SPI (Slave)
2xBS
I2C (Slave BS Constrol)
I2C Payload Bus (Master/Slave)
CAN
SPI (Master)
Parallel IO
USB
SPI S (RX)
LVDS2 S/X band TX
LVDS1 Engine Bypass
LVDS S/X Select/Status
Bypass
SEPP
Reconfig Logic
RAM
Dual Core ARM9
Flash Memory
Mass-Mem.
Config. Mem.
eMMC Mass Memory
2x cold reundant


Syrlinks S-Band TX/RX
PC104
RS422 TX
RS422 RX
I2C (Slave, Ctrl)
TX
RX
Syrlinks S-Band Diplexer
SPI (Slave)
Parallel IO
Optical Receiver
BS
I2C (Slave BS Control)
I2C (Slave)
SPI (Slave)
SPI (Master)
Parallel IO
USB
SPI S (RX)
LVDS2 S/X band TX
LVDS1 Engine Bypass
SEPP
Reconfig Logic
RAM
Dual Core ARM9
Flash Memory
Mass-Mem.
Config. Mem.
eMMC Mass Memory
2x cold reundant
LVDS S/X Select/Status
Bypass Select/Status
CSSDS Engine
FPGA
Bypass
MUX
PTME
Soft CPU
CAN
SPI
DEMUX
PDEC
CPDU
Bypass
BS
I2C (Slave BS Control)
I2C (Slave S/X band)
CAN
SPI S (RX)
LVDS2 S/X band TX
LVDS1 Engine Bypass
LVDS S/X Select/Status
Bypass Select/Status
USB (Data & Control)
RWCAM Board
HD-CAM
BS
I2C (Slave BS Control)
I2C (Slave)
External Reaction Wheels
Internal ADCS I2C Bus
System Reset
Direct Commands
I2C BS Control (BUS)
I2C Data (Bus)
CAN Bus
SPI Bus
LVDS Data (Point to Point)
UART Serial Line (Point to Point)
Parallel Line (Point to Point)
USB (Point to Point)
Select Line (Point to Point)

Communication Problems after launch



Bad UHF communication

- Timings
- Background noise

Dramatic increase in S band RX noise when TX is turned on

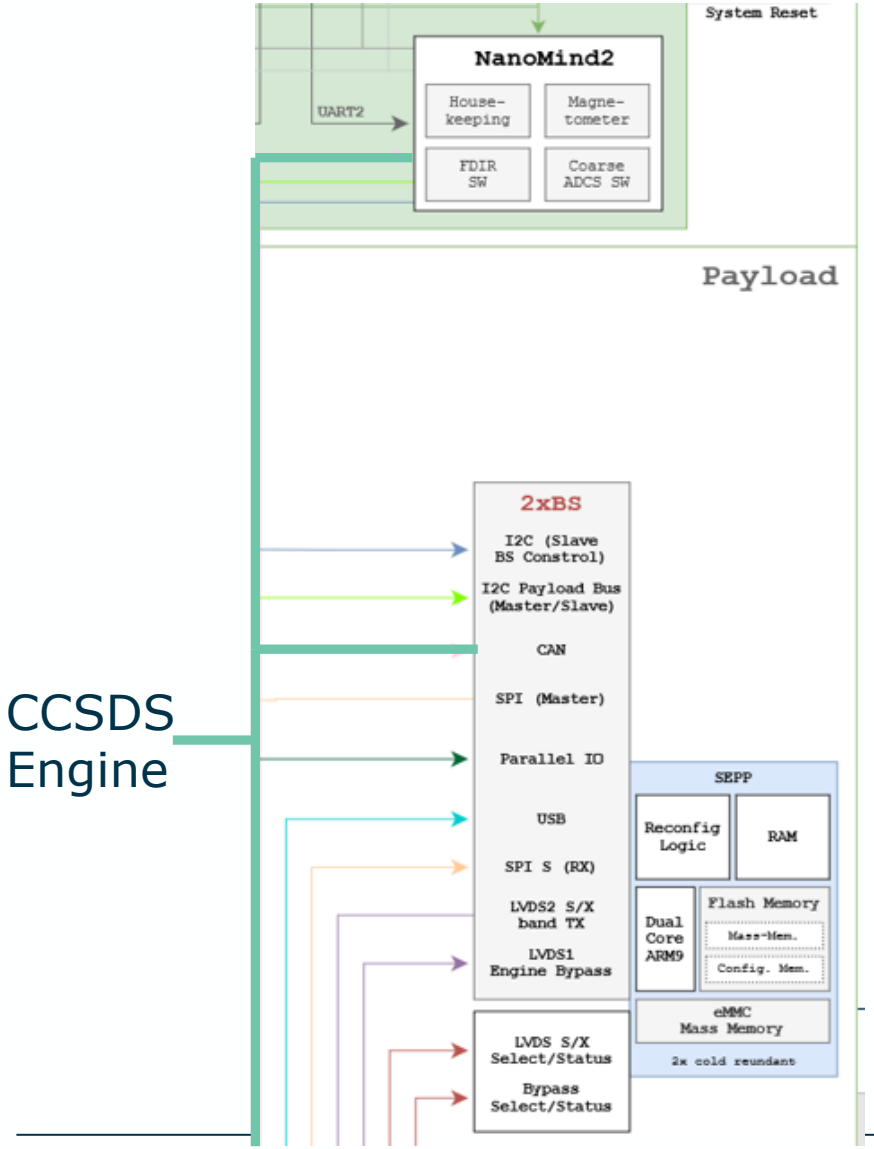
- 10 dB increase in RX noise level

ESOC-1 Ground Station High Power Amplifier failure

- 5-6 dB loss of uplink power

Result ➔ 2 mins of commanding a day and unpredictable when it would happen

Overcoming these initial challenges



How to update the OBSW under such conditions?

- When commanding is available load a compressed file to the SEPP that contains all the space packets needed for the OBSW load
- Spoof the Nanomind into thinking the SEPP is the ground and send them slowly out of ground coverage

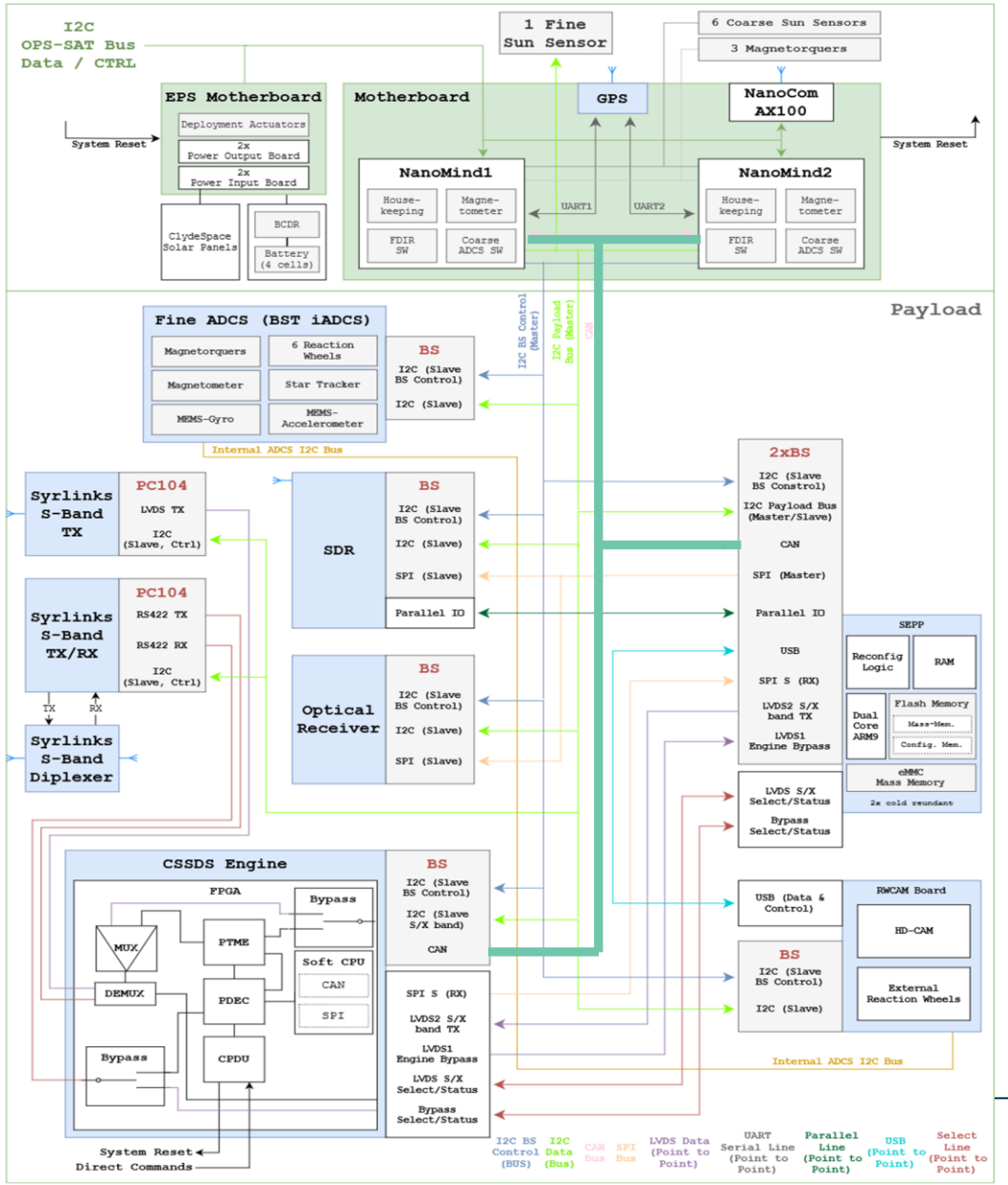
➔ An OBSW update that took one week is done between two passes

How to collect telemetry under such conditions?

- Send TM to the CCSDS engine even when the TX is off
- Load a SEPP application to sniff the CAN bus and collect the traffic
- Filter for the TM packets, collect in a file and compress it
- Download the compressed file when commanding available

➔ 10x increase in TM volume compared to ideal nominal case

CAN communication protocol choice



The CFP protocol already available on the Nanomond was implemented everywhere

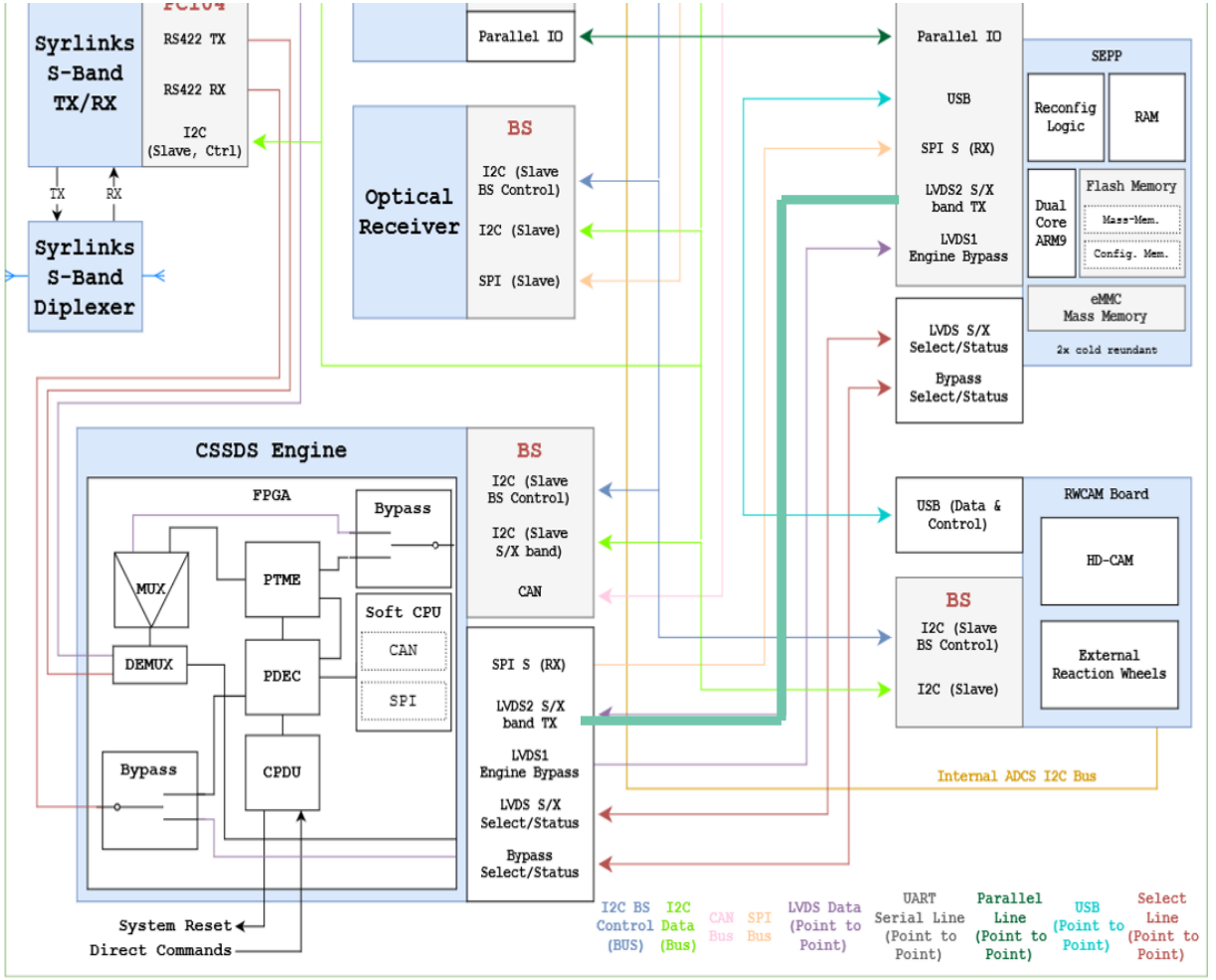
This was fine for the Nanomind connections as they were limited by the **application** speed

But for the SEPP-CCSDS connection it resulted in useful data rates of 150 kps on a link that could theoretically support 1 Mbps

Also implementing TCP/IP over CCSDS space packets on such a link would cause too many problems due to the small MTU sizes and overheads incurred

The back up plan

Before launch it was decided to use the LVDS line to implement a better communication protocol between the CCSDS engine and the SEEP – but there was no time to test it at system level!



Spacewire Lite implemented on the LVDS interface on the CCSDS engine side only. Then fused..

Loop back tested with the SEPP i.e. works up to the handshake level but not data

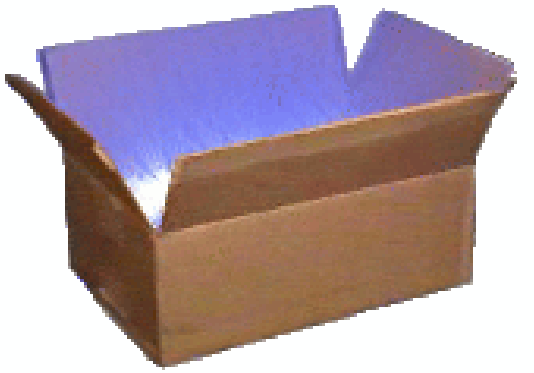
In orbit Spacewire Lite was loaded to the SEPP FPGA but it did not work initially. Analysis showed the CCSDS engine was not sending the last byte of the space packets!

Luckily this was part of a CRC 16 field that was not really needed and so the driver was altered to add the missing byte if the CRC was correct

It worked! We increased the effective data rate by six

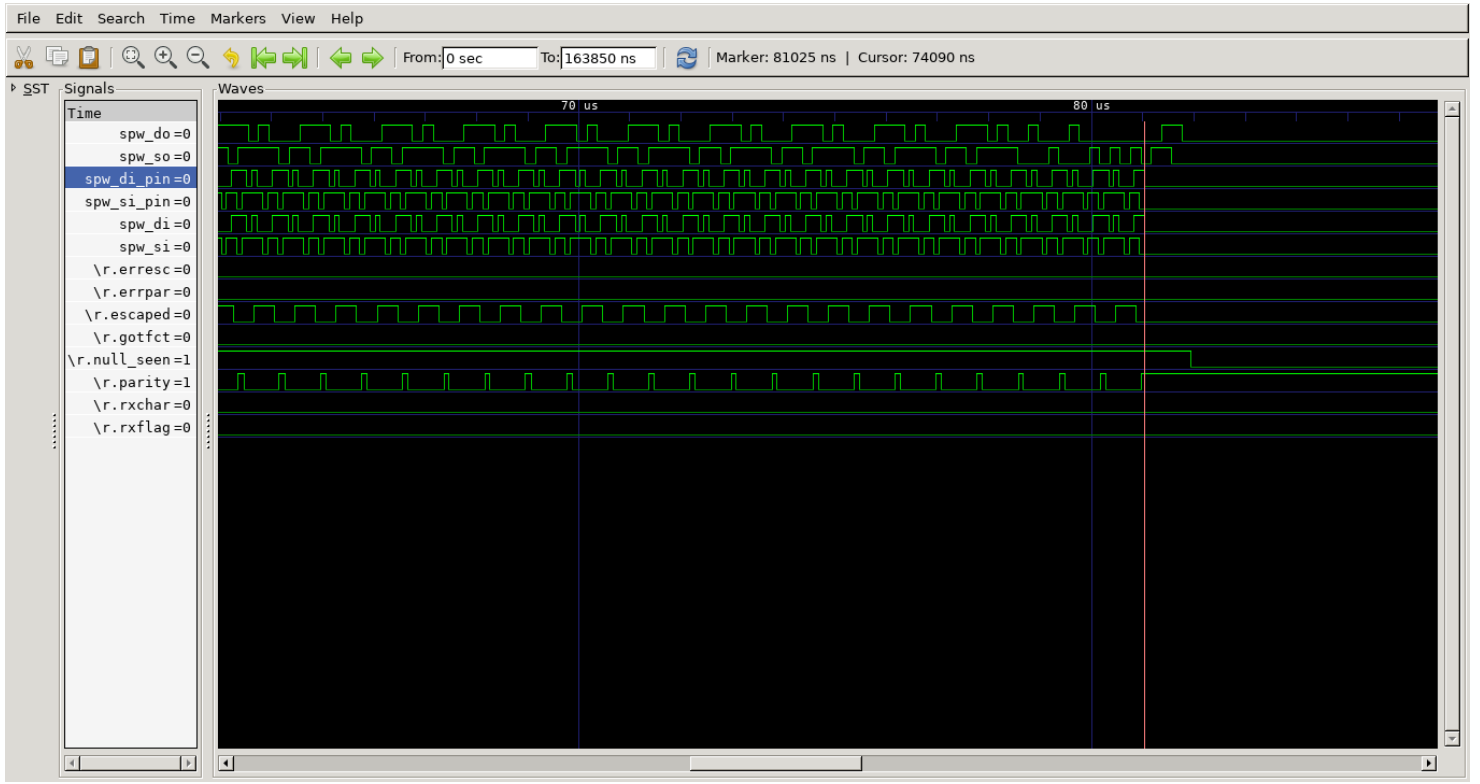
IP changes everything!

- With SpW the allowed MTUs were large enough to allow IP over CCSDS space packets to be implemented
- So a thin IP layer was added and the world changed!
- As the SEPP is running Linux and suddenly many of the native Linux services became available out of the box e.g. Rsync, SSH, remote kernel messages, demons, http and everything in Busybox!
- This allowed the mission control team to increase the productivity of the mission by an order of magnitude with very little effort. Functions that previously would have to written by us, tested and then loaded to the spacecraft became “one-liners”
- The experimenters also benefited from this “Cambrian explosion” - allowing new types of concepts to be easily tried out in space



We love Busybox

Using the FPGA to troubleshoot



Sometimes the SpW interface does not work and we have to fall back to CAN and no TCP/IP

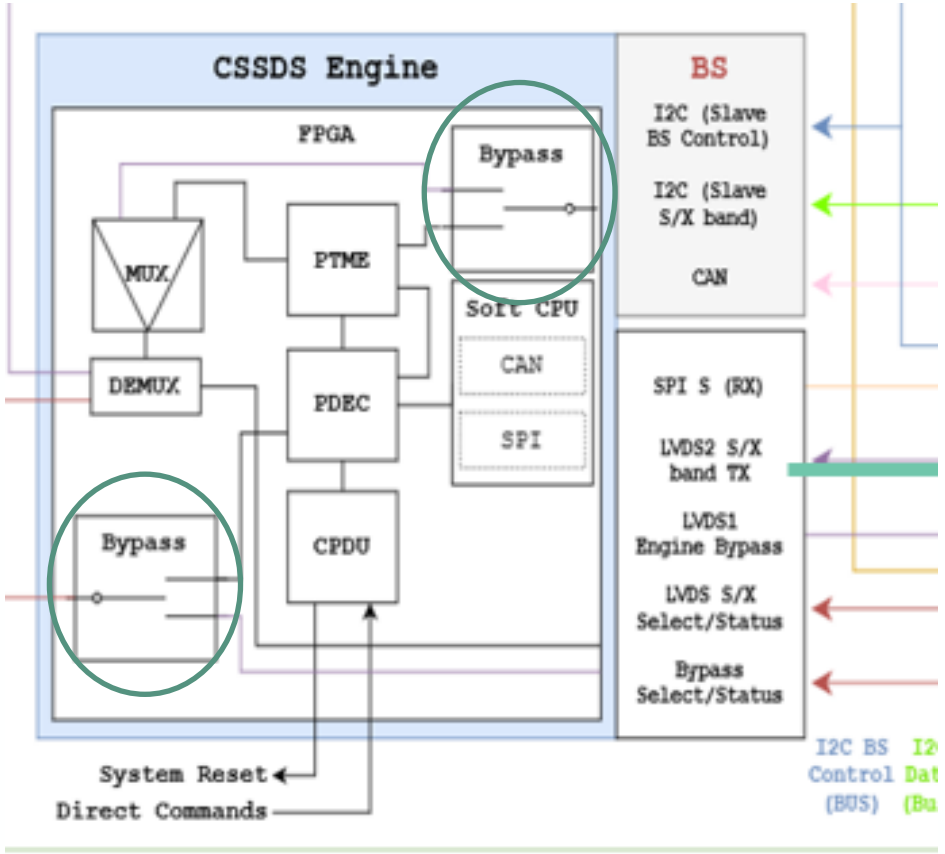
However the power of having a powerful Linux based system in combination with a reconfigurable FPGA has shown it's value in this area as well

We have been able to perform tests on the link in-orbit that are unthinkable on a traditional space mission

- Loading a logic analyser and recording the voltage levels on the SEPP pins
- Delaying the strobe and/or the data lines and sweeping +/- 300 nanoseconds to see what happens
- Inverting the lines
- Monitoring the statistics on the link up/down transitions, CRC errors, packets sent and received etc

Another solution?

TX/RX



TX/RX

Another solution is to exploit the bypass switches that were integrated into the CCSDS engine FPGA fabric

This allows us to turn the engine into a simple router with no data processing. Effectively we connect the SEPP FPGA to the transponders of the spacecraft

A project has already started to replicate the data processing elements on the CCSDS engine on the SEPP FPGA and then access the transponders directly using the bypass function

In theory we then have a completely reconfigurable system and should eliminate which ever error is still present

OPS-SAT-1 Space Lab Status

- 100+ companies from 20 countries registered 231 experiments
- JPL, JAXA, CNES, DLR, EU commission now on-board
- Many start-ups, research institutes and New Space
- ESA Academy, Fly your satellite, University courses (LUX, Zurich…)

GPS jamming experiments with Austrian Air Traffic Control, Army

1st ever Search and Rescue messages decoded in space for the first time

1st ever successful in-flight reprogramming of a Neural Network

D3TN ring road (interplanetary internet) successfully tested for 1st time

1st ever successful stock market trade in space with FlatexDEGIRO and Tradegate

On-board AI in daily use to classify camera pictures (SMART CAM)

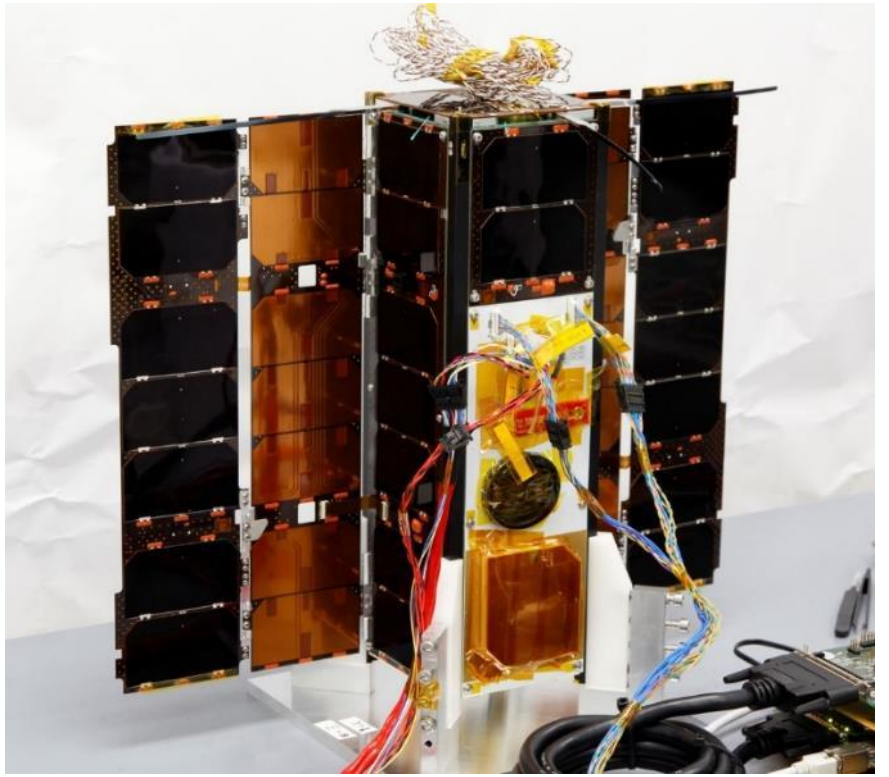
Direct commanding of satellite over the internet by experimenters now routine

TCP/IP direct connection to satellite, allowing standard IT tool use e.g. SSH, Rsync..

Space Wire successfully implemented in-orbit increasing data downloads by 10

1st ever in-flight control of a satellite using EGS-CC

1st ever European Hack-a-Sat competition announced for April 2022



Fastest submission to results time was 72 hours

Conclusions

- The OPS-SAT experience shows that on-board communications protocol choice can have a significant impact on mission productivity – far more than data rates need to be considered
- Reusing communication protocols that are already available is not always a good idea - the impacts of the choice can be masked by other constraints
- Having the ability to update the FPGA in orbit opened up a whole new world of operational possibilities, including changing the on-board communications protocol - and being able to trouble shoot it when it failed
- Launching with a partially tested communication protocol was risky. We have to acknowledge we had a lot of luck - we do not recommend it in an ideal world!
- On the other hand our experience shows that all of the difficult problems experienced centre on the CCSDS engine FPGA - which is fused. If it had been possible to update and troubleshoot it in orbit we would have rapidly solved these problems ☺ - well maybe…

Finally, many, many thanks to Maximilian Henkel of TU Graz who performed the majority of this work

Thank you!

More OPS-SAT-1 pictures on Flickr
https://www.flickr.com/photos/esa_events/albums/72157716491073681

David.Evans@esa.int