A combined software and hardware data compression approach in PLATO

Loidolt, Dominik, Ottensamer, Roland, Luntzer, Armin, Kerschbaum, Franz, Ottacher, Harald, et al.

Dominik Loidolt, Roland Ottensamer, Armin Luntzer, Franz Kerschbaum, Harald Ottacher, Jorge Tonfat, Manfred Steller, Stefano Pezzuto, Mauro Focardi, Rosario Cosentino, "A combined software and hardware data compression approach in PLATO," Proc. SPIE 11443, Space Telescopes and Instrumentation 2020: Optical, Infrared, and Millimeter Wave, 114434S (13 December 2020); doi: 10.1117/12.2561137

Event: SPIE Astronomical Telescopes + Instrumentation, 2020, Online Only

A combined software and hardware data compression approach in PLATO

Dominik Loidolt[a], Roland Ottensamer[a], Armin Luntzer[a], Franz Kerschbaum[a], Harald Ottacher[b], Jorge Tonfat[b], Manfred Steller[b], Stefano Pezzuto[c], Mauro Focardi[d], and Rosario Cosentino[e]

[a]University of Vienna, Department of Astrophysics, Türkenschanzstr. 17, 1180 Vienna, Austria
[b]Austrian Academy of Sciences, Space Research Institute, Schmiedlstr. 6, 8042 Graz, Austria
[c]INAF-IAPS, Institute of Space Astrophysics and Planetology, Via del Fosso del Cavaliere 100, 00133 Roma, Italy
[d]INAF-OAA, Arcetri Astrophysical Observatory, Largo E. Fermi 5, 50125 Firenze, Italy
[e]INAF-FGG, Galileo Galilei Foundation, Rambla J. A. F. Pérez 7, 38712 Breña Baja, TF, Spain

ABSTRACT

PLATO is ESA's upcoming exoplanet-hunting mission. The spacecraft has 26 cameras equipped with a total of 104 individual CCDs, which together provide more than 2,000 megapixels, giving a combined optical sensitivity surface of $0.66\,\mathrm{m}^2$. This is more than twice as much detector area than on ESA's Gaia mission, the largest camera ever flown in space. To measure changes in stellar brightness, the CCDs are read out at a cadence of 25 seconds, resulting in a massive amount of data that has to be processed on-board.

In a pre-reduction step, hundreds of thousands of small windows of the target stars, called imagettes, are extracted from the detector arrays. This reduction process decreases the data volume down from several gigabytes to 25 MiB per acquisition period. Following this step, the remaining science data are sent to the instrument control unit (ICU), where they are processed and compressed in a lossless manner. While some science data products, such as measured backgrounds and fluxes, can be processed in software, the number of imagettes to be compressed (90 % of the total science data) exceeds the available CPU resources. To solve this critical problem, a specialised hardware data compressor logic was developed for an RTAX-2000 field-programmable gate array (FPGA).

The implemented compression method decorrelates the data temporally by a running average, which has an exponential tail. This pre-encoding step results in an almost geometric distribution of the residuals, a suitable input for the successive Golomb encoder. The set of parameters that control the encoder are semi-adaptive, i.e., they self-adjust to the data at certain intervals. While in principle being quite straightforward, the implementation turned out to be very challenging with the required handling of the data streams in real-time. With our approach we are able to meet the high requirements and managed to process the imagette data lossless with a speed of 2 MBps at a compression ratio up to 3.2.

This paper shows how the PLATO data compression concept works, which algorithms are involved in it, and discusses the specifics of the hardware and software implementation.

Keywords: data compression, lossless, FPGA, Golomb code, Rice coding, PLATO, adaptive compression, on-board, instrument control, SpaceWire network, SpaceWire core, RMAP

Further author information: (Send correspondence to D.L.)
D.L.: E-mail: dominik.loidolt@univie.ac.at, Telephone: +43 1 4277 51820
R.O.: E-mail: roland.ottensamer@univie.ac.at, Telephone: +43 1 4277 51883

Space Telescopes and Instrumentation 2020: Optical, Infrared, and Millimeter Wave, edited by Makenzie Lystrup, Marshall D. Perrin, Proc. of SPIE Vol. 11443, 114434S · © 2020 SPIE
CCC code: 0277-786X/20/$21 · doi: 10.1117/12.2561137

1. INTRODUCTION

The PLAnetary Transits and Oscillations of stars (PLATO) mission[1,2] is a transit survey to discover and study a large number of extrasolar planets around bright stars. Special attention is paid to detecting small earth-like planets in the habitable zone around sun-like stars. To achieve that, PLATO takes long (months to years) continuous high-precision photometric measurements of many bright stars ($V \leq 11$–13 mag) in the visible wavelength range. In combination with earth-based follow-up observation, it is possible to determine the primary parameters (radius, mass, age) of a huge amount of extrasolar planets in the following way:

- **Radius** from analysis of photometric transit light curves obtained by PLATO
- **Mass** from ground-based radial velocity (RV) follow-up spectroscopy
- **Age** from asteroseismic analysis of stars, based on PLATO's photometric light curves

PLATO is taking a multi-telescope approach to achieve the mission's objectives, which is novel for space telescopes. This means that the spacecraft has not one but 26 cameras, divided into 24 "normal" cameras and two "fast" cameras. The cameras are based on a fully dioptric design with six lenses and a pupil diameter of 120 mm. Using several telescopes has several advantages. The resulting large field of view (FoV) of 2232 deg^2 (almost 20 times the FoV of the Kepler instrument) allows the observation of many bright stars per pointing, which increases the probability of observing a planetary transit. The "normal" cameras are aligned in such a way that their fields of view partially overlap, resulting in areas covered by 24, 18, 12, and 6 cameras. This design gives the instrument a high dynamic photometric measuring range. The readout cadence of the "normal" cameras is 25 s, while the "fast" cameras have a higher cadence of 2.5 s. The two "fast" cameras observe stars with V 4–8 and also act as fine guidance sensors for the position control system. In contrast to the other cameras without filter, one "fast" camera is equipped with a red bandpass filter and the other with a blue bandpass filter.

At the end of the PLATO mission, a catalogue of thousands bulk characterised planets with accurate radii, masses, mean densities and ages, as well as about 1,000,000 stellar light curves will be available. This catalogue will enable us to get closer to the answers to the current questions of planetary research: How do planetary systems form and evolve? Is our solar system unique, or are there similar systems? Are there any potentially habitable planets? Furthermore, the huge, long-lasting legacy of PLATO will also form the basis for many other future extrasolar planetary missions such as ARIEL[3].

One challenge of the PLATO multi-telescope design is the resulting enormous detector area and the related large amount of data to be processed on-board. An important part of this on-board processing system is data compression to generate as much scientific data as possible with the limited uplink capacity. We have introduced a combined software and hardware data compression approach to handle the data volume.

2. THE DATA FLOW

The flow of the data starts at PLATO's 26 cameras, 24 of the 12 cm cameras are so-called "normal" cameras and read out with a cadence of 25 s, the remaining two "fast" cameras have a readout cadence of 2.5 s. Each camera is equipped with an array of four CCDs each with 4510×4510 pixels[*]. If we put all CCDs next to each other, we get a detector with 2.12 gigapixels and an area of $0.66\,\text{m}^2$. To put this into perspective, this roughly corresponds to the size of 10 pages of this paper. PLATO thus has a total detector area more than twice as large as ESA's Gaia mission and will replace Gaia as the largest camera ever flown in space when it is launched in 2026.

The four CCDs in each camera are operated by a front-end electronics (FEE). The FEE reads out and digitises the CCD data. The ancillary electrical units (AEU) are responsible for the synchronisation of the different cameras and subsystems as well as the power supply of the FEE.

The image data from two "normal" cameras are handled by one "normal" data processing unit (N-DPU), while the data from the two "fast" cameras are processed by two "fast" DPUs (F-DPU). The 12 N-DPUs are

[*]The CCDs of the "fast" cameras are operated in frame transfer mode. This mode allows faster readout times with the disadvantage that only half the area of each CCD can be used.

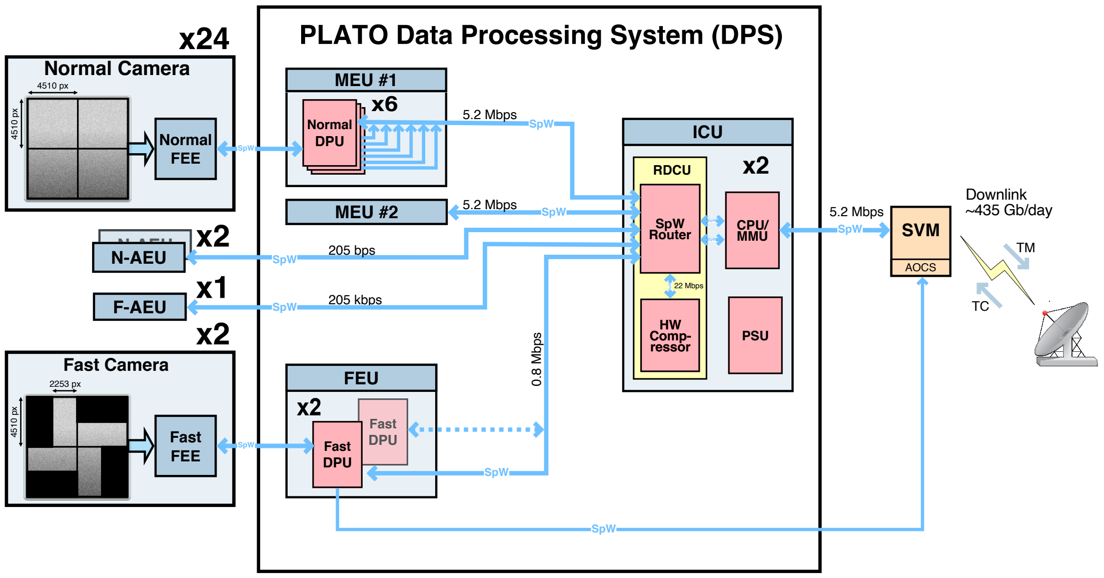

Figure 1. Visualisation of the PLATO data processing system (DPS). The CCDs are read out and digitised by the FEE. The DPUs cut out the small star imagettes and prepare the light curves and other data products. The data are sent to the ICU to compress them and to prepare them for the downlink. The SVM is the payload interface to the spacecraft, which sends the data to the ground. The AEUs synchronise the different units and cameras. Redundant links are not shown. The link speeds are based on Refs. 4.

divided into two groups of six units each, installed in two individual main electronics unit (MEU) boxes. The two F-DPUs are installed in the fast electronics unit (FEU) box. The uplink and on-board resources are not capable of handling all data generated by the cameras. To deal with the massive amounts of data from the detectors, the DPUs cut out a small window around the target stars with the help of the FEEs. This small image of the star is called an "imagette"[†] which typically is 6×6 pixels (9×9 pixels for the "fast" cameras). Furthermore, the DPUs are responsible for performing basic photometric tasks and generate a number of light curves, center of brightness, background values and other science data. All these data products, including the imagette data, are sent to the central instrument control unit (ICU).

The ICU[5,6] receives the science data over its SpaceWire (SpW) router located on the internal router and data compression unit (RDCU). The data handling is done on the processor and mass memory unit (CPU/MMU), which is the main data processing module of the ICU. The CPU/MMU board is responsible for the control of the payload and is the interface to the spacecraft for telecommand and telemetry. The used microprocessor of the CPU/MMU board is the UT700 SPARC V8/LEON 3FT processor from Cobham Gaisler, which works with a 133 MHz clock rate. To compress the imagette data, the CPU/MMU is supported by the hardware data compressor, which is located on the RDCU board and connected to the SpW router. The hardware data compressor helps the CPU/MMU board in compressing the massive amount of imagette data, which makes roughly 90 % of the total science data. The remaining science data products are compressed in software on the CPU/MMU board. The power supply unit (PSU) of the ICU supplies the other boards with the secondary supply voltages. To achieve cold redundancy, each of the three boards (CPU/MMU, RDCU, PSU) exists twice in the ICU.

After collecting and compressing the data, the ICU transmits them to the service module (SVM). PLATO will have an average downlink capacity in the K- and X-band of about 435 Gbit per day. In the last step, the SVM downlinks the data to earth, where the data flow finally ends.

[†]Imagette is the French word for a thumbnail image. The term was first introduced in this context by the French CoRoT mission.

3. THE COMPRESSION ALGORITHMS

The PLATO compression algorithms build on the experience and lessons learned from the on-board data compression for the Herschel[7] and CHEOPS space telescopes. The compression algorithm consists out six stages which are chained together.

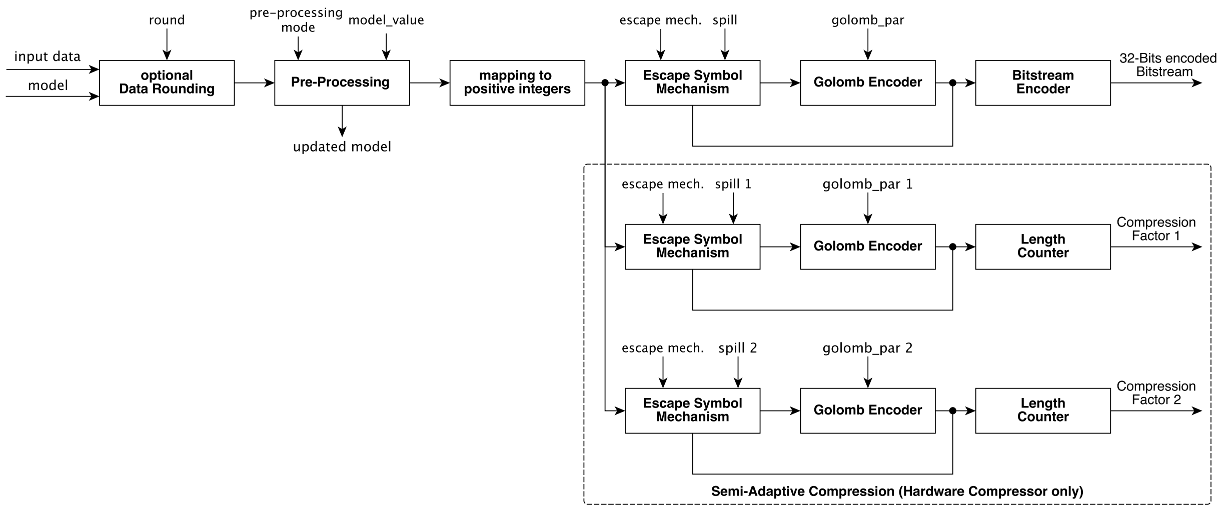

Figure 2. Visualisation of the compression chain. The first step is an optional lossy rounding step, followed by a pre-processing stage, which decorrelates the data. The output of the pre-processing stage follows a two-sided geometric distribution, which is then transformed into a one-sided distribution, by mapping the signed output to a positive value range. The escape symbol mechanism deals with outliers in our data set. The Golomb encoder gives small values short code words and large values long code words to compress the data. In the last step, the code words of different lengths are put together in the bitstream encoder. The semi-adaptive compression is a hardware compressor feature which allows to self-adjust the compression parameters to the data at certain intervals.

The compression chain, as shown in Figure 2, starts with an optional lossy step. At this stage, the least significant noise bits of the data can be rounded down if necessary. A simple right shift implements this. Currently, we do not intend to round the data in a lossy way. However, we have added them in order to remain flexible.

The next stage is the pre-processing stage, in which the data are decorrelated. Two different modes have been created for this: the 1D-differencing and the model mode. The 1D-differencing mode is used for the first data set of a time series. This mode forms a 1-dimensional array of the imagettes by arranging the lines of pixels side by side. After that, the difference between adjacent pixels is formed according to the following rule: $d_0 = a_0$, $d_i = a_i - a_{i-1}$, where d is the pixel difference, a the pixel value and i the array index.

For the other data sets of the time series, we use the model mode, in which the difference of the data and a local model of the data is formed ($\text{output}_i = \text{data}_i - \text{model}_i$). Afterwards, the model is updated to predict the next data set in a better way using an exponential weighted moving average:

$$\text{model}_{t+1} = \begin{cases} \text{data}_0 & t = 0 \\ \left\lfloor \dfrac{\alpha \cdot \text{model}_t + (16-\alpha) \cdot \text{data}_t}{16} \right\rfloor & t > 0 \end{cases} . \tag{1}$$

As an initial model, we use the first data set of the time series. The model weighting value α is an integer number in the interval [0,16][‡]. It determines how fast the model changes with a new data set. At the end of the

[‡]The base of the model update formula was chosen as 16 to avoid expensive floating-point operations on the one hand and on the other hand, it is much easier to divide by powers of two in the binary system than, e.g. by 10, especially in hardware.

pre-processing step, the data set follows a zero centred two-sided geometric distribution, as shown in Figure 3.

We now transform the signed output of the pre-processing stage into an unsigned value range. This is done by mapping all positive values to even numbers and all negative values to odd ones. This can be expressed mathematically by:

$$M(\epsilon) = \begin{cases} 2\epsilon & \epsilon \geq 0 \\ 2|\epsilon| - 1 & \epsilon < 0 \end{cases} . \tag{2}$$

With this operation, we transform the two-sided geometric distribution into a (one-sided) geometric distribution.

The escape symbol mechanism deals with outliers in the data record. These outliers can have many reasons. One of the most common are cosmic rays. This stage only has an effect if large values have to be encoded. We will skip the explanation of the details and come back to it later.

Now it is finally time to encode the data. When selecting the encoder, the high requirements regarding limited FPGA and memory resources, compression speed and last but not least, the compression factor had to be met. To fulfil the requirements, the Golomb code[8] was chosen to encode the data for several reasons. First, for geometrically distributed data, the Golomb code is equivalent to the Huffman code, which produces an optimal prefix code[9]. Second, the calculation of the codewords is straightforward because it depends only on two parameters. Namely, the value to be encoded itself and the Golomb parameter, which describes the geometric distribution of the data set. This is in contrast to the Huffman code, where the calculation depends on the probability of occurrence for each possible value. This way, we can use the advantage of the Huffman code, namely an optimal prefix code, and avoid the disadvantage of the costly calculation. This allowed us to ensure efficient and fast encoding with low FPGA/memory resource usage. Note that the Rice code is a subset of the Golomb code[§].

We take one step back to the escape symbol mechanism, which deals with outliers in our data set. If we encoded a very large value with a Golomb code, the result would be a long codeword which can be much longer than the unencoded symbol. To prevent this, an escape symbol is defined. The escape symbol is a value which signals that the next bits are unencoded data. If the input value of the escape symbol mechanism is greater than or equal to the defined spill threshold parameter, the escape symbol mechanism becomes active. In this case, the escape symbol is forwarded to the Golomb encoder (instead of the input value), and the input value of the stage is appended unencoded to the Golomb encoded escape symbol. Due to the different nature of the outliers in 1D-differencing and model pre-processing, we have developed two approaches for outlier encoding.

The first is the zero escape symbol mechanism, which always uses zero as the escape symbol. Due to the structure of the Golomb code, the value zero always has the shortest code word. To avoid the problem that zero can also be a legitimate value to encode, a one has to be added to each symbol when entering the zero escape symbol mechanism. If this precaution were not taken, it would lead to a situation in which it is impossible to distinguish whether the zero is only a normal input value or the escape symbol. Because of the short escape symbol, the zero escape symbol mechanism has advantages in distributions with a larger number of outliers. This is the case for the 1D-differencing mode.

The second method is called multi-escape symbol mechanism. If an outlier is detected (greater than or equal to the spill threshold parameter), the spill threshold parameter is subtracted from this value. Then it checks how many bits are needed to represent the difference, thus dividing the difference into groups. The first group needs a maximum of two bits to represent the difference, the second group needs a maximum of four bits, and so on in two-bit steps. Each group has its own escape symbol, starting with the first group escape symbol, which is the value of the spill threshold parameter. The second group uses spill+1 as escape symbol, the third spill+2 etc. After the Golomb encoded escape symbol, the difference is appended unencoded with the respective length of the group. The decoding of the multi-escape symbol mechanism works as follows: If a value is decoded which is greater than or equal to the spill threshold parameter, the group and thus the respective number of unencoded bits is determined ($\text{read\_bits} = (\text{decoded\_val} - \text{spill} + 1) \cdot 2$). Now the number of bits is read out, and the spill is added to get the original value of the outlier. We use this method to encode outliers from the model pre-processing.

[§]The special cases, when the Golomb parameter m is a power of two $m = 2^k$, correspond to the Rice codes.

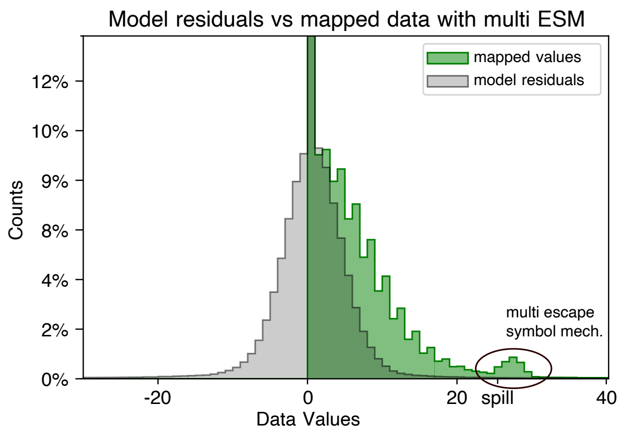

Figure 3. The grey distribution shows the data after the model pre-processing. The green distribution shows the data for the Golomb encoder, i.e. the grey distribution mapped into the positive value range and processed with the multi-escape symbol mechanism.

The last part of the compression chain is the bitstream encoder, which has the task of stringing together the generated code words of different lengths.

Semi-adaptive compression is an exclusive hardware compression feature and is explained in detail in Section 4.

4. HARDWARE DATA COMPRESSOR

To perform a data compression in hardware, the uncompressed data are first loaded via the remote memory access protocol (RMAP) into the SRAM on the RDCU board. RMAP is also used to set the registers required for compression, which describe where the data is located in the SRAM and which compression parameters (pre-processing mode, Golomb parameter, spillover threshold, etc.) should be used for the compression. By setting the start bit in the compressor control register, the compression is started. Then, the compressor works its way through all data until it has compressed the entire data set. To tell the CPU that the compression is complete, it sends an interrupt signal to the CPU/MMU board and also sets the compressor ready bit in the compressor status register. Then, the metadata, such as the length of the compressed data, can be read from the registers. Finally, the compressed data can be read from the SRAM to complete the compression.

4.1 Router and Data Compression Unit

As the name suggests, the router and data compression unit have two key components: the SpaceWire router and the RDCU FPGA logic, which contains the hardware compressor. An image of the RDCU board is shown in Figure 4.

The SpW router is responsible for the connection of the data processing system (DPS) with the ICU. It implements the central knot of the SpW network[10]. The used component is the dedicated SpW router chip GR718B from Cobham Gaisler. The component is configured to be able to operate with 100 Mbps link speed on all links. The router and routing map is configurable by the RMAP commands from the central processing unit of the ICU CPU/MMU board. The SpW router is capable of handling the data streams from the different sensors to the ICU and also to the RDCU logic. To add cold spare functionality on the SpW links, which is not natively embedded on the router chip, dedicated low-voltage differential signaling (LVDS) repeater circuits are added to each link.

The hardware data compressor is integrated into an FPGA. We use the reprogrammable Microsemi (formerly Actel) ProASIC3E A3PE3000 FPGA for the development, in the flight model we will replace the chip with a one-time programmable Microsemi RTAX2000S FPGA. For a smooth transition from one chip to the other, we use an Aldec ProASIC3E adapter board that is compatible with the footprint of the final RTAX device. An

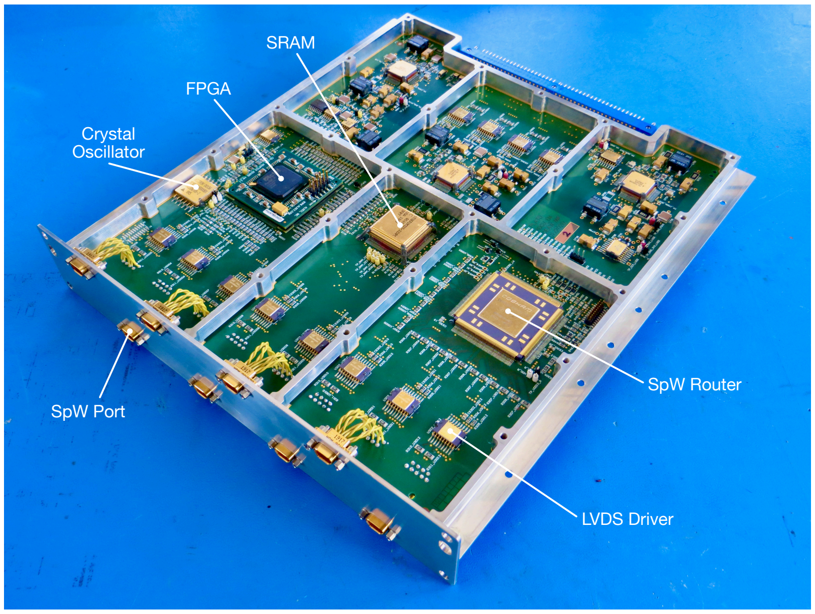

Figure 4. Picture of the router and data compression unit (RDCU) engineering model.

8 MB static RAM (SRAM) is available to store the data for the hardware compressor. The used SRAM is the Cobham UT8ER1M32.

4.2 Hardware Data Compressor Implementation

The FPGA logic of the RDCU board consists of several cores, as shown in Figure 5, which work together to perform the task of data compression. The cores are implemented in VHDL code. As the VHDL design method, the two-process method[11] is used, which has the advantage that an automatic correction of possible single event upsets of the hardware-implemented triple modular flip-flops is foreseen.

4.2.1 SpaceWire Core

The SpaceWire core is a SpW encoder-decoder implementation of the SpW protocol as defined in ECSS-E-ST-50-12C[12]. This core is based on the SpaceWire Light IP from OpenCores[13]. This IP core was adapted to fulfil the project requirements. The changes on the IP had the goal of achieving a link speed of 100 Mbps in both directions. The receiver side was modified to recover the clock from the data and strobe signals. Also, it was modified to avoid race conditions in the clock recovery circuit. The credit error reporting of the IP was corrected because it presented a different behaviour as it is defined in the SpaceWire standard. Finally, an empty packet handling feature was included to quietly discard empty packets.

4.2.2 RMAP Decoder Core

The RMAP decoder core is an “RMAP Target only” partial implementation of the RMAP protocol defined in ECSS-E-ST-50-52C[14]. It is able to receive RMAP commands and send reply packets. From the available RMAP commands, this core only implements a subset of them. This core can handle only read and write commands that request a reply and use the increment memory address option. For write commands, this unit can handle verify before writing and don’t verify before writing options. However, the verify data size is limited to four data bytes. It receives the packet header and data from the SpaceWire core. Then, the packet header is decoded, and the data is written to or read from the external SRAM memory or register through the WISHBONE bus. Finally, the reply is sent. Any errors during the packet processing are classified using the RMAP standard protocol error codes defined in the ECSS standard and own-defined error codes. When the unit receives an RMAP read command, the command will be decoded, and the read operation is started. If in addition a write command is received, the unit will execute the write command in parallel to the read reply sending. In this case, the core handles the sharing of the internal WISHBONE bus for the read and write operations.

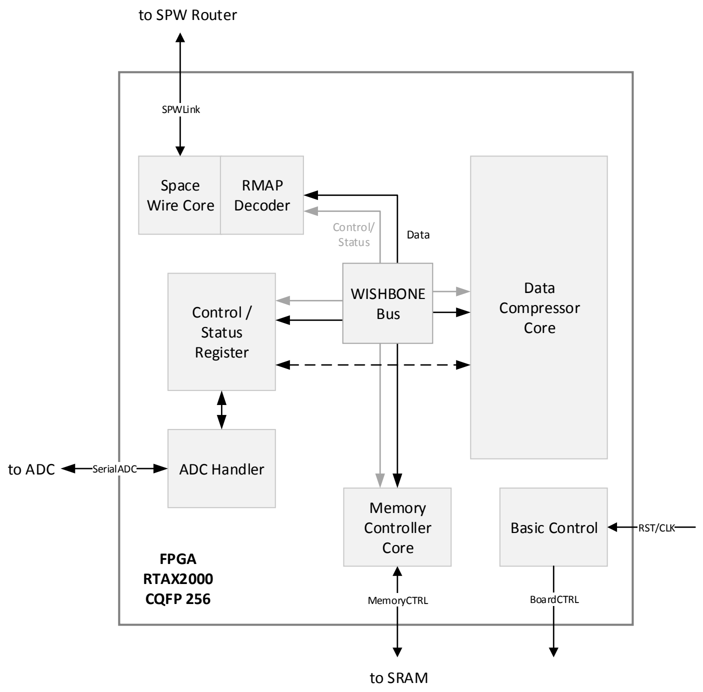

Figure 5. The internal core architecture of the FPGA design. Data are received and transmitted via the SpW/RMAP decoder core. The data compressor core implements the compression algorithms. The control/status register block realises the FPGA registers. Access to the SRAM and registers is possible over the internal WISHBONE bus. The ADC handler deals with the housekeeping measurements. The memory controller manages the external SRAM. Finally, the basic control core implements the global reset function and generates the needed clock frequencies.

4.2.3 Hardware Compressor Core

The data compression algorithms, as described in Section 3, are implemented in the hardware compressor core. The hardware compressor reads two 16-bit imagette pixels with one memory access. To compress these two values in parallel, the hardware compressor was implemented as a pipeline design. With this method, we compress the two values in 8 instead of 14 clock cycles. Furthermore, since memory accesses are expensive in terms of time, we have introduced an approach with two finite-state machines (FSM). One takes care of the communication with the memory, the other manages the compression of the data. This has the advantage that while the current data are compressed, the data for the next run can already be read in at the same time. These two concepts enable us to increase the throughput of the compressor logic in such a way that it is mainly limited by the memory access times.

It can be assumed that the structure of the collected raw data will change in the short term as well as in the long term due to various processes. For this reason, a technique is needed that adapts the compression parameters to these changes. A simple approach to implement such a method would be to compress the data with other compression parameters either in parallel or after the actual compression process. However, these ideas cannot be implemented due to the tight constraints of the compressor in terms of available time and SRAM memory. In order to still provide adaptive compression, a semi-adaptive method was developed. With this design, the compression factor for two additional compression parameter sets (consisting of a Golomb parameter and a spillover threshold value) can be calculated in addition to the actual compression process. This allows the user to try two more compression parameters in addition to the actual compression parameter and compare their compression results with each other. Under the assumption that the distribution of the next data differs only minimally from the current data, the best compression parameter set of the last process can now be selected for the next compression run. The semi-adaptive compression takes place in parallel to the actual compression, so this feature does not have a negative impact on the actual compression performance.

4.2.4 Other Cores

- **Basic Control** On the one hand, the basic control logic generates the main reset signal. On the other hand, it is responsible for generating the required clock frequencies and for distributing the clocks. There are several clock networks on the RDCU:
  - 100 MHz: for the SpaceWire router and the internal SpaceWire core
  - 25 MHz: for the FPGA logic (including the hardware compressor core) and the SpaceWire router
  - 2 MHz: for the internal FPGA logic used for timeouts and housekeeping measurements
  - 500 kHz: for the clock of the point of load converter to synchronize the power conditioning.
- **Bus Handler** The unit implements the internal data bus structure by using the WISHBONE specification[15]. It handles the requests from the masters, distributes the requests to the slaves, and handles the errors. Depending on the operation state the masters are differently connected with the available slaves.
- **Memory Controller** The memory controller manages the external SRAM access and generates the access signals for the external read and write operations. It also manages the access to the SRAM EDAC register and the memory scrubbing process.
- **ADC Handler** This core handles the serial communication with the analog-to-digital converter (ADC) chip and implements the housekeeping value register. It measures the housekeeping values periodically and stores the values in dedicated value registers.
- **Control/Status Register** The unit handles the access to the FPGA registers via the internal bus and implements the register functions.

5. SOFTWARE DATA COMPRESSOR

While the task of the hardware (HW) compressor is to compress the imagette data, which takes up most of the data budget, the software (SW) data compressor is tailored to compress the other PLATO data products, such as fluxes and center of brightness measurements. The compressor is implemented as an ANSI C library for the ICU application software and also uses the algorithms which are described in Section 3.

One difference to the HW compressor is that the SW compressor can additionally process 32-bit values, while the HW compressor is limited to compress only 16-bit words. Another difference to the hardware compressor is that the semi-adaptive compression function is not available because it would cost more calculation steps and thus slow down the SW compression process.

We designed the interface for the SW compressor in such a way that the same interface can be used for controlling the HW compression. This makes it easier to integrate compression into the data processing flow of the ICU application software.

The process of optimising the data products is not yet fully completed, but we are confident that we can meet the requirement for a lossless compression rate of more than two.

6. LESSONS LEARNED AND FUTURE PROSPECTS

Although our compression strategy is straightforward, it proves to be effective in performing the task of data compression. The concept can be used in both hardware and software for fast implementation of the compression without consuming large amounts of computational resources. Here the golden rule of data compression is applied: know your data. It turns out that even with simple approaches, good compression results can be achieved and rocket science is not always necessary.

It has proven to be extremely useful to first develop the compression algorithms in software and then implement them in the hardware logic. Since software development is more flexible and agile, the algorithms can be more easily adapted to given requirements, so that a fast proof of concept can be developed. Furthermore, the software implementation is very beneficial as a reference for testing and validating the HW compressor core.

One lesson we learn is that the data transfer times are a significant part of the total compression time in the proposed hardware compression concepts. Originally, transmission accounted for about 60 percent of the total compression time. To improve this, the RMAP core has been extended so that it is possible to read and write data in parallel to the compressor memory. This optimisation makes it possible to download the compressed data and the updated model at the same time as uploading the next to be compressed data and their model, which reduces the transmission time part to 50 %. Summarised: Don't forget the data transfer.

An additional way to speed up the HW compression process would be to use a bigger SRAM that could hold all models of the processing data. This would eliminate the need for a model transfer between the CPU/MMU and the RDCU board. However, the problem with this idea is that there is no space to place six additional memory chips on the board that are needed to hold all the models.

The next tasks on the to-do list are the design transfer from the reprogrammable PROASSIC FPGA to the once programmable RTAX2000X used in the flight model. Another task is the integration of the hardware and the software compression into the ICU application software, plus a variety of documentation and testing tasks. We also plan to publish the hardware compressor IP core as well as the software compression code under the GPLv2 open-source licence on our website (https://space.univie.ac.at).

ACKNOWLEDGMENTS

This work has been supported by the Austrian research promotion agency (FFG) Austrian space applications programme (ASAP) project 6328535.

REFERENCES

[1] Rauer, H. et al., "The PLATO 2.0 mission," *Experimental Astronomy* **38**, 249–330 (Nov. 2014).

[2] Rauer, H. and Heras, A. M., [*Space Missions for Exoplanet Science: PLATO*], 1309–1330, Springer International Publishing, Cham (2018).

[3] Tinetti, G. et al., "A chemical survey of exoplanets with ARIEL," *Experimental Astronomy* **46**, 135–209 (Nov. 2018).

[4] Rotundo, M., Leoni, A., Serafini, L., Del Vecchio Blanco, C., Davalle, D., Vangelista, D., Focardi, M., Cosentino, R., Pezzuto, S., Giusy, G., Biondi, D., and Fanucci, L., "Simulation and validation of a spacewire on-board data-handling network for the plato mission," in [*2018 IEEE Workshop on Complexity in Engineering (COMPENG)*], 1–5 (2018).

[5] Focardi, M., Pezzuto, S., Cosentino, R., Giusi, G., Giorgio, A. M. D., Biondi, D., Blanco, C. D. V., Serafini, L., Vangelista, D., Steller, M., Jeszenszky, H., Ottacher, H., Laky, G., Ottensamer, R., Kerschbaum, F., Guedel, M., Noce, V., Pace, E., Pancrazzi, M., Westerdorff, K., Peter, G., Ulmer, B., Berlin, R., Plasson, P., Pagano, I., Tommasi, E., and Natalucci, S., "The design of the instrument control unit and its role within the data processing system of the ESA PLATO Mission," in [*Space Telescopes and Instrumentation 2018: Optical, Infrared, and Millimeter Wave*], Lystrup, M., MacEwen, H. A., Fazio, G. G., Batalha, N., Siegler, N., and Tong, E. C., eds., **10698**, 1334 – 1345, International Society for Optics and Photonics, SPIE (2018).

[6] Focardi, M., Pezzuto, S., Cosentino, R., Giusi, G., Pancrazzi, M., Noce, V., Ottensamer, R., Steller, M., Giorgio, A. M. D., Pace, E., Plasson, P., Peter, G., and Pagano, I., "The instrument control unit of the ESA-PLATO 2.0 mission," in [*Space Telescopes and Instrumentation 2016: Optical, Infrared, and Millimeter Wave*], MacEwen, H. A., Fazio, G. G., Lystrup, M., Batalha, N., Siegler, N., and Tong, E. C., eds., **9904**, 962 – 976, International Society for Optics and Photonics, SPIE (2016).

[7] Ottensamer, R. and Kerschbaum, F., "HERSCHEL/PACS on-board reduction flight software," in [*Advanced Software and Control for Astronomy II*], Bridger, A. and Radziwill, N. M., eds., **7019**, 477 – 488, International Society for Optics and Photonics, SPIE (2008).

[8] Golomb, S., "Run-length encodings (corresp.)," *IEEE transactions on information theory* **12**(3), 399–401 (1966).

[9] Gallager, R. and van Voorhis, D., "Optimal source codes for geometrically distributed integer alphabets (corresp.)," *IEEE Transactions on Information Theory* **21**(2), 228–230 (1975).

[10] Focardi, M., Cosentino, R., Pezzuto, S., Biondi, D., Giusi, G., Serafini, L., Del Vecchio Blanco, C., Vangelista, D., Rotundo, M., Fanucci, L., Davalle, D., and Dps, P., "The plato payload and data processing system space wire network," in [*2018 IEEE Workshop on Complexity in Engineering (COMPENG)*], 1–4 (2018).

[11] Gaisler, J., "A structured vhdl design method," *Fault-tolerant microprocessors for space applications* , 41–50 (2011).

[12] ECSS, "Spacewire — links, nodes, routers and networks," Standard ECSS-E-ST-50-12C, European Cooperation for Space Standardization, Noordwijk, NL (2008).

[13] van Rantwijk, J., "SpaceWire Light IP core." OpenCores, 5 December 2018 https://opencores.org/projects/spacewire_light. (Accessed: 25 November 2020).

[14] ECSS, "Spacewire — remote memory access protocol," Standard ECSS-E-ST-50-52C, European Cooperation for Space Standardization, Noordwijk, NL (2010).

[15] Herveille, R., "WISHBONE, Revision B.4 Specification." OpenCores, 22 June 2010 https://opencores.org/howto/wishbone. (Accessed: 25 November 2020).