A flexible and heterogeneous framework for scientific image data processing on-board the Solar Orbiter PHI instrument

Lange, Tobias, Fiethe, Björn, Guan, Yejun, Michalik, Harald, Albert, Kinga, et al.

Tobias Lange, Björn Fiethe, Yejun Guan, Harald Michalik, Kinga Albert, Johann Hirzberger, David Orozco Suárez, Manuel Rodríguez-Valido, "A flexible and heterogeneous framework for scientific image data processing on-board the Solar Orbiter PHI instrument," Proc. SPIE 11155, Image and Signal Processing for Remote Sensing XXV, 1115506 (7 October 2019); doi: 10.1117/12.2532102

Event: SPIE Remote Sensing, 2019, Strasbourg, France

A flexible and heterogeneous framework for scientific image data processing on-board the Solar Orbiter PHI instrument

Tobias Lange[a], Björn Fiethe[a], Yejun Guan[a], Harald Michalik[a], Kinga Albert[b], Johann Hirzberger[b], David Orozco Suárez[c], and Manuel Rodríguez-Valido[d]

[a]IDA, Technische Univ. Braunschweig, Germany
[b]Max-Planck-Institut für Sonnensystemforschung, Germany
[c]Instituto de Astrofísica de Andalucía - CSIC, Spain
[d]Univ. de La Laguna, Spain

ABSTRACT

Present scientific space instruments generate a high amount of raw data while deep-space missions only have a very limited telemetry rate. Because the computation of the scientific relevant parameters is usually accompanied with the reduction of the data, the processing is desired to be carried out already on-board.
To accomplish this, the following paper presents a flexible image processing framework which makes use of a heterogeneous data processing module consisting of a space-grade General Purpose Processor (GPP) as well as two dynamically reconfigurable Field-Programmable Gate Arrays used for hardware acceleration. The flexibility and capabilities of the presented framework are proven by means of three exemplary processing tasks of the *Polarimetric and Helioseismic Imager* (*PHI*) on-board *Solar Orbiter*.

Keywords: Solar Orbiter, on-board processing, FPGA, Kuhn-Lin-Loranz, Hough Transform

1. INTRODUCTION

Among the scientific instrumentation of space missions, especially imaging instruments produce a very high amount of data. This amount is reduced by the calculation of the scientific parameters of interest, which typically is performed on ground. Since the telemetry rate of deep-space missions is very restricted, the required processing has to be carried out already on-board. As, on the other hand, the availability of qualified processing components (such as General Purpose Processors, GPPs) for such missions is very limited, space-grade Field Programmable Gate Arrays (FPGAs) are widely used for dedicated processing tasks. For complex instruments such as the *Polarimetric and Helioseismic Imager* (*PHI*) on *Solar Orbiter* [1], on-board processing is desired to be very flexible. *PHI* is a camera-based instrument which will acquire high-resolution and full disk measurements of the solar photosphere used to provide maps of the continuum intensity, the magnetic field vector and the line-of-sight velocity within the photosphere. This is done by scanning six different wavelength positions within the FeI-6173 Å line with four different polarization states, each. Final magnetic field vectors are determined by the inversion of the Radiative Transfer Equation (RTE). Because the instrument is equipped with an active pixel sensor of 2048×2048 pixels, a set of 24 acquired images would lead to an overall amount of approx. 265 MiB. Due to the very limited telemetry rate, the processing of the acquired data already has to be carried out on-board. Therefore, the Data Processing Module (DPM) for *PHI* utilizes a GPP in combination with two reconfigurable FPGAs to speed up data acquisition and processing. While data acquisition, image stabilization and the inversion of the RTE [2] are done by dedicated FPGA designs, the image pre-processing demands a high grade of flexibility. To retain the performance gained by the use of FPGAs, a heterogeneous on-board processing framework has been implemented which overcomes the bottleneck between FPGA and GPP.

Further author information:
Tobias Lange: E-mail: tlange@ida.ing.tu-bs.de

Image and Signal Processing for Remote Sensing XXV, edited by Lorenzo Bruzzone, Proc. of SPIE Vol. 11155, 1115506 · © 2019 SPIE · CCC code: 0277-786X/19/$21 · doi: 10.1117/12.2532102

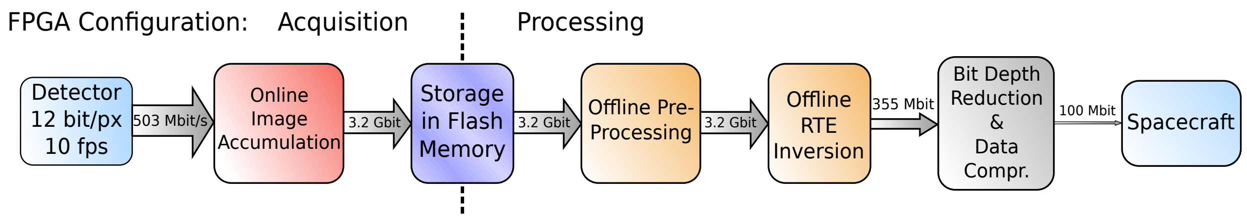


Figure 1. Data processing of the *PHI* instrument

1.1 Data Processing Module for Solar Orbiter PHI

The Data Processing Module (DPM) for the *PHI* instrument which is the underlying hardware platform for the presented framework is based on a combination of a General Purpose Processor and two reconfigurable FPGAs based on SRAM technology (RFPGAs) [3–5]. The architecture of the module is depicted in figure 2 and consists of a system controller, a system supervisor, the two reconfigurable FPGAs and a solid state mass memory for intermediate data storage.
A radiation hard *GR712RC* processor ASIC based on the fault tolerant *LEON3* architecture was chosen to implement the system controller of the DPM. It controls the instrument and its processing and interfaces the DPM to the spacecraft. It is connected to its own 256 MiB of working memory as well as non-volatile memory for storage of software and FPGA configuration bitstreams. The processor system is running at a clock frequency of 50 MHz.
Two reconfigurable *Xilinx Virtex-4* FPGAs are used for dedicated processing tasks in a time-shared manner. For buffering of image data, one of the reconfigurable FPGAs is connected to 1 GiB of dynamic memory.
The processor system and the two reconfigurable FPGAs are connected by radiation-hard and one-time programmable *RTAX2000* FPGA which acts as a system supervisor and configuration controller. It includes two dedicated controllers responsible for safe configuration and permanent scrubbing of the two RFPGAs via JTAG. It also connects the main components of the DPM via a dedicated *SoCWire* communication network. Furthermore, it includes a controller for a mezzanine 512 GiB solid-state NAND-Flash memory board for intermediate storage of image data [6].

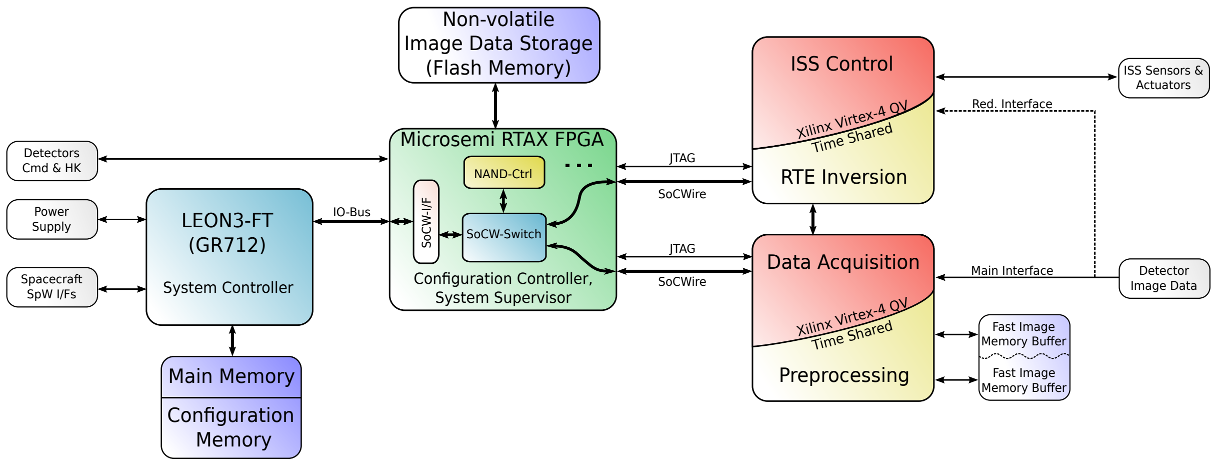


Figure 2. Architecture of the *PHI* Data Processing Module (DPM)

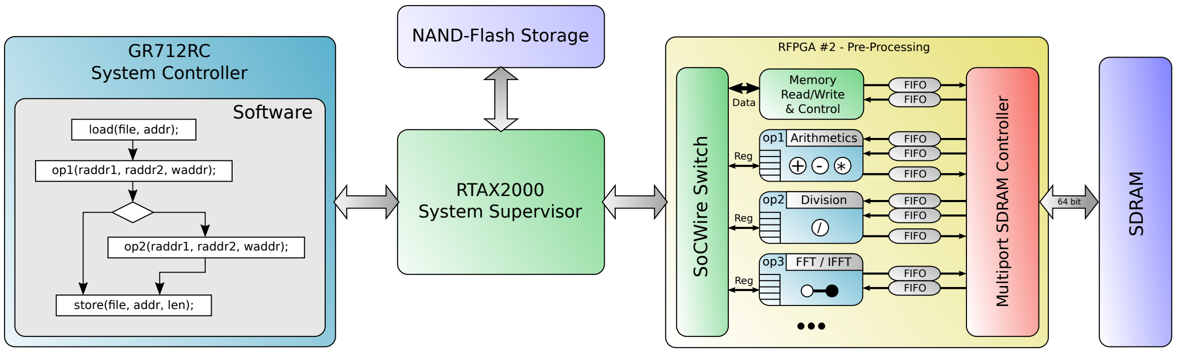
2. FLEXIBLE, HETEROGENEOUS IMAGE PROCESSING FRAMEWORK

In order to enable the flexible processing of image data which meets the performance requirements on the presented hardware, a framework has been developed which allows the instrument software to control the data flow inside the processing FPGA. Therefore, the framework consists of a set of specialized processing modules. These modules include e.g. simple addition, subtraction and multiplication as well as a Fast Fourier Transform (FFT) and median filtering. Each of these modules is connected to the multi-port SDRAM controller via simple FIFO interfaces, allowing to access the 1 GiB of SDRAM buffer memory. A dedicated *SoCWire* communication protocol is used for control, status and data transmission purpose between the various components of the DPM and the processing framework. The software layer enables the automatic detection and identification of processing modules after reconfiguration of the FPGA. Each module is parametrized and executed by a set of registers. These registers are controlled by the software layer which allows a seamless integration into the flow of the on-board software. Reading and writing of data to and from the other components of the DPM is done by a dedicated module which is included in each processing configuration.

In contrast to the acquisition of data, intermediate storage of data and the elliptical orbit of the spacecraft allow off-line processing of the data. Thus, the FPGA configurations for processing are protected to a lesser extend against single-event upsets within the space environment. Instead, faults are detected within the framework and processing steps will be repeated if necessary. Therefore, the framework includes a software-based self-test which makes use of small test vectors to detect malfunctioning processing modules. The tests run automatically after the execution of each module which implies a constant run-time overhead.

To meet the performance requirements, the buffer memory is using a bus-width of 64 bit (96 bit gross, including error correction). Because not all modules fit into a single FPGA configuration, the framework makes intense use of dynamic reconfiguration of the SRAM-based processing FPGA at run-time. An excerpt of processing modules arranged across three FPGA configuration bitstreams is shown in table 1. The principle of the processing framework is depicted in figure 3. The pixels of an image are typically represented by 32 bit fixed-point values. In the frequency domain, 64 bit complex floating-point format is used (32 bit, each for real and imaginary part) to avoid scaling issues. The set of hardware modules as well as the external SDRAM memory is also running at a clock frequency of 50 MHz.

In order to allow automated testing of the implemented modules as well as rapid prototyping and evaluation of more complex processing routines, the framework provides a set of libraries which enables to use the functions on the DPM from within *Python* running on a host computer.



```
load(file, addr);
op1(raddr1, raddr2, waddr);
op2(raddr1, raddr2, waddr);
store(file, addr, len);
```

Figure 3. Architecture of the processing framework

Table 1. Arrangement of needed processing modules within FPGA configuration bitstreams

Configuration 1	
`op_addsub`	Pixel-wise / scalar addition and subtraction
`op_mult`	Pixel-wise / scalar multiplication
`op_div`	Pixel-wise / scalar division
`op_thold`	Thresholding less/equal/greater than with optional replace
`op_shift`	Horizontal / vertical shift and roll of images
`op_multisum`	Minimum / maximum determination, sum, mean value and least squares fitting
`op_logic`	Scalar logic functions (and, or, xor, not)
...	
Configuration 2	
`op_fp_cext`	conversion between fixed-point real ↔ floating-point complex
`op_fp_fft`	Floating-point FFT/IFFT
`op_fp_mult`	Pixel-wise / scalar (complex) floating-point multiplication
...	
Configuration 3	
`op_log`	Calculation of logarithm
...	

3. PROOF OF CONCEPT

Besides regular pre-processing of the acquired science data, the framework is also used for in-flight instrument calibration [7]. Therefore, the framework shall be proven in terms of run-time and flexibility by means of the image pre-processing as well as the determination of the flat field. While the pre-processing requires an execution time of around 15 minutes per data set, the run-time of the calibration tasks is considered to be less critical. Since the final sequence of the science data pre-processing depends on the commissioning phase of the instrument, the basic processing pipeline is used as proof of concept.

For the determination of the flat field, no uniform light flux in front of the detector will be available during flight. Therefore, the flat field calibration for the full-disk telescope has to be done using the algorithm according to Kuhn, Lin and Loranz (KLL) [8]. The algorithm calculates the individual gain for each detector pixel using a set of displaced images of the solar disk. Before this is carried out, the input images have to be transformed to a logarithmic scale. The computation iterates through all possible combinations of the displaced images. For the *PHI* instrument, a set of nine images is used, resulting in overall 36 combinations. The detector has a size of 2048×2048 pixels, leading to an overall image size of 16 MiB for real valued images and 32 MiB for complex images. Therefore, the calculation of the flat field using the KLL algorithm requires a sufficient amount of processing power. In order to apply the KLL algorithm to the input images, the center position of the solar disk within each image has to be determined in advance. This is done by using a circular Hough transform. Instead of extending the presented processing framework by dedicated hardware modules, the two tasks can be broken down into the subset of already implemented basic functions as listed in table 1.

3.1 SCIENCE DATA PROCESSING

The data flow of the exemplary image pre-processing as required by the *PHI* instrument is depicted in figure 4. In the first step (1), the acquired set of images has to be corrected with the dark field of the optics and the flat field of the sensor. Therefore, intermediately stored acquisition data as well as dark and flat fields have to be loaded from the NAND-Flash memory. The next step corrects the influence of the telescope's Point-Spread Function (PSF) by a deconvolution with a Wiener Filter which can be carried out in the frequency domain (2). This also includes the necessary transfer of the prepared filter kernel from the NAND-Flash into the SDRAM buffer memory. In each of the six different wavelengths around the FeI-6173 Å line, the instrument acquires four intensities I_0, I_1, I_2 and I_3 at different polarization states. In order to allow further computation, these intensities have to be converted (3) into the four Stokes parameters I, Q, U and V. Equation 1 shows that the demodulation matrix $D(x, y)$ is dependent on the image position. Therefore, the computation requires D in form of a set of 16 individual images with an overall size of 256 MiB. Before hand-over to the second FPGA for the inversion of the RTE, the data has to be reordered and converted to floating-point format in accordance of the operation mode of the RTE inversion (4).

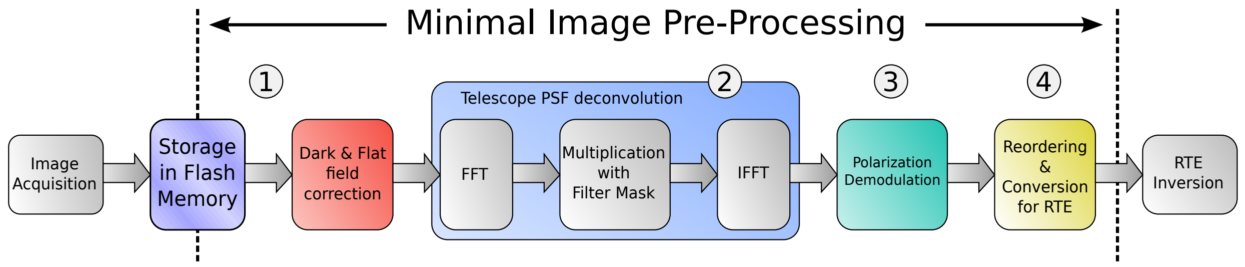

Minimal Image Pre-Processing
1
2
3
4
Image Acquisition
Storage in Flash Memory
Dark & Flat field correction
Telescope PSF deconvolution
FFT
Multiplication with Filter Mask
IFFT
Polarization Demodulation
Reordering & Conversion for RTE
RTE Inversion

Figure 4. Exemplary regular data processing

$$
\begin{aligned}
I_S(x,y,\lambda) &= \begin{pmatrix} I(x,y,\lambda) \\ Q(x,y,\lambda) \\ U(x,y,\lambda) \\ V(x,y,\lambda) \end{pmatrix} = D(x,y) I_{raw}(x,y,\lambda) \\
&= \begin{pmatrix} D_{0,0}(x,y) & D_{0,1}(x,y) & D_{0,2}(x,y) & D_{0,3}(x,y) \\ D_{1,0}(x,y) & D_{1,1}(x,y) & D_{1,2}(x,y) & D_{1,3}(x,y) \\ D_{2,0}(x,y) & D_{2,1}(x,y) & D_{2,2}(x,y) & D_{2,3}(x,y) \\ D_{3,0}(x,y) & D_{3,1}(x,y) & D_{3,2}(x,y) & D_{3,3}(x,y) \end{pmatrix} \cdot \begin{pmatrix} I_0(x,y,\lambda) \\ I_1(x,y,\lambda) \\ I_2(x,y,\lambda) \\ I_3(x,y,\lambda) \end{pmatrix}
\end{aligned}
\tag{1}
$$

3.2 HOUGH TRANSFORM

The Hough transform is a well known technique in image processing used for the extraction of features [9, 10]. In case of the *PHI* instrument, a circular Hough transform is needed to find the center of the solar disk within nine different images. After edge detection and the generation of a binary image, the circular Hough transform is usually carried out by adding up a circle of given radius at the position of each valid pixel in the binarized image into a separate accumulation buffer. Finally, the circular center can be determined by finding the maximum value inside the accumulation buffer.

Because the transform is similar to a convolution, the accumulation might also be carried out in the frequency domain under certain conditions [11, 12]. The principle of the circular Hough transform, only consisting of the functions listed in table 1, is shown in figure 5. It includes the actual Hough transform in the frequency domain as well as the edge detection and binarization. The circular filter kernel has to be precomputed and can be stored within the DPM's NAND-Flash storage.

The calculation shown in figure 5 uses the nine displaced images as needed for the KLL. To determine the correct center position, circular kernels for five different radii are used. Because the transform into the frequency domain of the kernel is done at run-time, multiple binary filter kernels with different radii can be stored into one single image. The currently needed radius can be picked by the use of the logic function module.

After successful inverse Fourier Transform, the center can be determined by finding the overall maximum position of the five output images for each of the nine input images. With the arrangement of configurations from table 1, the Hough transform only needs three reconfigurations of the processing FPGA.

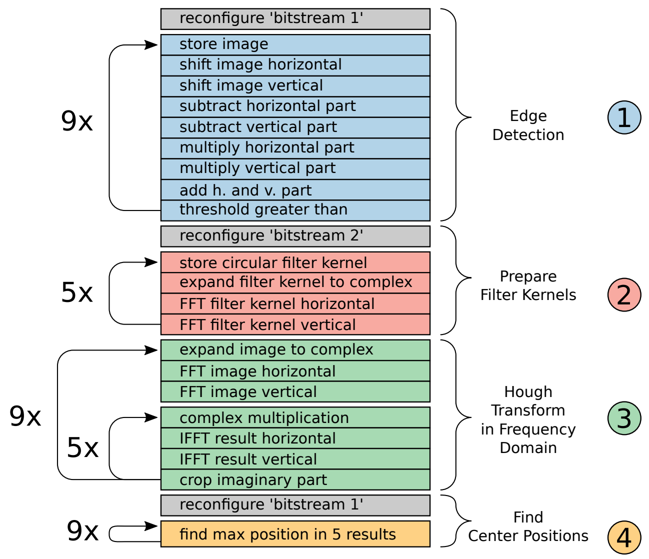


Figure 5. Hough transform

3.3 KUHN-LIN-LORANZ ITERATION

The actual iteration of the KLL works on the logarithmic representation D_i of each corresponding input image d_i. It can be described as a series by the following equation:

$$G^{r+1}(x) = K(x) + \frac{1}{n(x)} \sum_{i<j} \left[G^r(x - a_i + a_j) + G^r(x - a_j + a_i)\right] \tag{2}$$

This includes constant K, which is defined by:

$$K(x) = \frac{1}{n(x)} \sum_{i<j} \left[[D_i(x) - D_j(x - a_i + a_j)] + [D_j(x) - D_i(x - a_j + a_i)]\right] \tag{3}$$

The displacement is expressed by the vectors a_i and a_j, respectively. The algorithm sums up over all possible combinations of the input images ($i < j$). Pixels beneath a certain threshold will not be included in the calculation. Thus, the result has to be scaled by $\frac{1}{n(x)}$ which is used to accumulate valid pixel positions. The iteration starts with $G^0(x) = 0$. For the simulated data of the *PHI* instrument, we assume that the algorithm converges after five iterations.
Similar to the Hough transform, the KLL can be split up into basic function calls and processing modules which are given in table 1. The KLL can be divided into five major tasks:
The preparation of the input data (1) loads the nine displaced images from the NAND-Flash memory and converts them into a logarithmic scale. Furthermore, this steps generates binary masks which are necessary for the accumulation of $n(x)$. This is followed by the calculation of constant $K(x)$ which also is the initial value $G^1(x)$ of the iteration (2). The involved images have to be shifted to a common fiducial position. The step has to be done for all 36 valid combinations of the nine input images. The calculation of $G(x)$ is similar to the calculation of $K(x)$ but does not include the subtraction (3). This not only has to be done over the 36 valid combinations of the nine displaced images but also for the five necessary iterations. Therefore, this step is highly computation intense. The calculated $G(x)$ is divided by $\frac{1}{n(x)}$ and corrected in terms of scaling after each of the five iterative steps. Also invalid (NaN) results have to be replaced. This step has to be done for all the five iterations. Finally, the resulting flat-field has to be scaled back by exponentiation. Because this only affects the final result, the exponentiation is not calculated within a dedicated module but instead carried out in software within the system controller.

4. EVALUATION AND ANALYSIS

As mentioned in section 2, the framework includes a set of *Python*-based libraries to enable automated testing of modules and rapid prototyping of complex processing pipelines from within an external computer. Furthermore, this test software can be used for detailed evaluation of implemented algorithms. The detailed run-time of the three presented processing pipelines is analyzed in table 2, 3 and 4.

As can be seen, the computation for all three given examples is done in reasonable time. While the less critical flat field calibration (Hough transform and KLL) takes around 11 minutes, the run-time constrained minimal pre-processing is executed within less than 4 minutes.

Since the reconfigurable FPGAs need a full reconfiguration in order to update the processing modules, the results show that the overall time needed for FPGA reconfiguration is negligible related to the overall run-time of the three pipeline examples. The individual module run-time strongly depends on the utilization of the memory interface. Thus, functions which involve more pixel-wise operands are slower in execution. Another significant factor is the out-of-order access across row boundaries of the SDRAM memory. This can be observed in particular on the vertical Fast Fourier Transform.

All the shown run-time measurements include the optional self-test after the execution of each function. Measurements have shown a constant overhead of around 0.02 s. For functions with a slow execution time the overhead is measured to be around 7 % while the overhead for very fast functions can be up to 25 %.

Furthermore, the analysis shows that the FFT/IFFT, multiplication as well as addition/subtraction each have a significant share on the processing. Also the loading of acquired data sets and parameters is not to be neglected.

As a backup solution, a software version of the processing modules and the Hough transform is available to run on the *GR712* system controller. Although there is room for software improvements, the run-time of the minimal pre-processing is carefully expected to take several hours. The software implementation of the Hough transform was measured to take around 940 s for a single image. Thus, the complete set of nine images is estimated to take more than two hours whereas the computation within the hardware accelerated framework takes between two and three minutes.

5. SUMMARY AND CONCLUSION

In summary, the paper presented a flexible and heterogeneous framework for scientific on-board data processing. It makes efficient use of the dynamic in-flight reconfigurability of the two used space-grade *Xilinx Virtex-4* FPGAs which only results in a negligible overhead to the overall execution time. The flexibility of the framework has been proven by means of three different processing tasks needed for regular processing and calibration on the *Solar Orbiter PHI* instrument. It has been shown that even complex processing tasks which at a first glance demand for dedicated hardware acceleration and implementation, can be broken down into a subset of basic hardware modules. The control flow is seamlessly integrated into the on-board software running on the *GR712* system controller. Therefore, the processing can be easily adapted to mission specific changes without necessary update of the FPGA bitstreams.

Furthermore, the framework is not only restricted to be used on the DPM of the *PHI* instrument. The used *SoCWire* communication structure and the multi-port memory controller can be easily changed or adapted to support different FPGA platforms and memory technologies. Another possible application of the framework is the proposed *PMI* instrument on-board *ESA*'s Lagrange mission which would monitor space-weather conditions. The instrument requires processing similar to *PHI* instrument but instead of intermediate storage, computation has to be done in real-time.

Table 2. Run-time analysis of the implemented Hough transform

(1)	**Edge Detection**	**29.7 s**
1×	Reconfiguration	2.2 s
9×	Load Image	13.7 s
9×	Shift	3.5 s
36×	Addition/Subtraction	6.0 s
18×	Multiplication	3.0 s
9×	Threshold	1.3 s
(2)	**Preparation of Filter Kernels**	**10.5 s**
1×	Reconfiguration	2.2 s
1×	Loading Filter Kernels	1.5 s
5×	Expand to Floating-Point Complex	1.0 s
5×	FFT Horizontal	1.6 s
5×	FFT Vertical	4.2 s
(3)	**Hough Transform**	**87.9 s**
9×	Expand to Floating-Point Complex	1.8 s
9×	FFT Horizontal	3.0 s
9×	FFT Vertical	7.6 s
45×	Complex Multiplication	14.0 s
45×	IFFT Horizontal	37.9 s
45×	IFFT Vertical	14.8 s
45×	Conversion to Fixed-Point Real	8.8 s
(4)	**Detection of Center Positions**	**5.7 s**
1×	Reconfiguration	2.2 s
45×	Finding Maximum Position	3.5 s
Overall Run-Time		**133.8 s**

Table 3. Run-time analysis of an exemplary pre-processing

(1)	**Dark and Flat-Field Correction**	**90.6 s**
1×	Reconfiguration	2.2 s
49×	Load Image	74.3 s
24×	Subtraction	4.0 s
24×	Division	4.1 s
49×	Scalar Multiplication	6.0 s
(2)	**Deconvolution of the Telescope PSF**	**76.5 s**
1×	Reconfiguration	2.2 s
1×	Loading Filter Kernel	1.5 s
24×	Expand to Floating-Point Complex	4.9 s
24×	FFT Horizontal	7.7 s
24×	FFT Vertical	20.2 s
24×	Complex Multiplication	7.4 s
24×	IFFT Horizontal	7.7 s
24×	IFFT Vertical	20.2 s
24×	Conversion to Fixed-Point Real	4.7 s
(3)	**Polarization Demodulation**	**54.7 s**
1×	Reconfiguration	2.2 s
16×	Load demodulation Matrix	24.3 s
96×	Multiplication	16.1 s
72×	Addition	12.1 s
(4)	**Preparation for RTE Inversion**	**7.2 s**
1×	Reconfiguration	2.2 s
24×	RTE Reordering and Conversion	5.0 s
Overall Run-Time		**229.0 s**

Table 4. Run-time analysis of the Kuhn-Lin-Loranz iteration

(1)	**Preparation of Input**	**35.7 s**
3×	Reconfiguration	6.5 s
9×	Load Image	13.7 s
9×	Mask	1.3 s
9×	Logarithm	12.9 s
9×	Threshold and Replace	1.3 s
(2)	**Calculation of $K(x)$**	**90.5 s**
144×	Shift	27.8 s
144×	Multiplication	24.2 s
37×	Scalar Multiplication	4.6 s
180×	Addition/Subtraction	30.2 s
12×	Threshold and Replace	1.7 s
12×	Divide	2.0 s
(3)	**Calculation of $G(x)$**	**320.5 s**
720×	Shift	139.0 s
360×	Addition/Subtraction	60.5 s
720×	Multiplication	121.0 s
(4)	**Correction of $G(x)$**	**9.4 s**
10×	Addition/Subtraction	1.7 s
5×	Multiplication	0.8 s
5×	Scalar Multiplication	0.6 s
5×	Division	0.9 s
5×	Square Root	0.6 s
15×	Mean Value	1.2 s
25×	Threshold and Replace	3.6 s
(5)	**Exponentiation** (Software)	**55.0 s**
1×	Data Transfer	5.0 s
1×	Exponentiation	50.0 s
Overall Run-Time		**511.1 s**

REFERENCES

[1] Solanki, S. K., del Toro Iniesta, and et al, "The Polarimetric and Helioseismic Imager on Solar Orbiter," *arXiv e-prints* , arXiv:1903.11061 (Mar 2019).

[2] Carrascosa, J. C., del Moral, B. A., Mas, J. R., Balaguer, M., Jiménez, A. L., and del Toro Iniesta, J., "The rte inversion on fpga aboard the solar orbiter phi instrument," in [*Software and Cyberinfrastructure for Astronomy IV*], **9913**, 991342, International Society for Optics and Photonics (2016).

[3] Lange, T., Bubenhagen, F., Fiethe, B., Michel, H., and Michalik, H., "Fpga-based dynamically reconfigurable processing module for the solar orbiter phi instrument," in [*SEE / MAPLD 2011 Conference*], (April 2013).

[4] Fiethe, B., Bubenhagen, F., Lange, T., Michalik, H., Michel, H., Woch, J., and Hirzberger, J., "Adaptive hardware by dynamic reconfiguration for the solar orbiter phi instrument," in [*Adaptive Hardware and Systems (AHS), 2012 NASA/ESA Conference on*], 31–37 (jun 2012).

[5] Lange, T., Fiethe, B., Michel, H., Michalik, H., Albert, K., and Hirzberger, J., "On-board processing using reconfigurable hardware on the solar orbiter phi instrument," in [*2017 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*], 186–191 (July 2017).

[6] Lange, T., Michel, H., Fiethe, B., Walter, D., and Michalik, H., "Fault-tolerant nand-flash memory module for next-generation scientific instruments," *Proc.SPIE* **9646**, 9646–9646–7 (2015).

[7] Albert, K., Hirzberger, J., Busse, D., Lange, T., Kolleck, M., Fiethe, B., Suárez, D. O., Woch, J., Schou, J., Rodriguez, J. B., Gandorfer, A., Guan, Y., Carrascosa, J. P. C., Expósito, D. H., del Toro Iniesta, J. C., Solanki, S. K., and Michalik, H., "Autonomous on-board data processing and instrument calibration software for the so/phi," *Proc.SPIE* **10707**, 10707–10707–9 (2018).

[8] Kuhn, J. R., Lin, H., and Loranz, D., "Gain calibrating nonuniform image-array data using only the image data," *Publications of the Astronomical Society of the Pacific* **103**(668), 1097 (1991).

[9] Hough, P. V., "Machine analysis of bubble chamber pictures," in [*Conf. Proc.*], **590914**, 554–558 (1959).

[10] Hough, P. V., "Method and means for recognizing complex patterns," (Dec. 18 1962). US Patent 3,069,654.

[11] Hollitt, C., "Reduction of computational complexity of hough transforms using a convolution approach," in [*Image and Vision Computing New Zealand, 2009. IVCNZ'09. 24th International Conference*], 373–378, IEEE (2009).

[12] Hollitt, C., "A convolution approach to the circle hough transform for arbitrary radius," *Machine Vision and Applications* **24**, 683–694 (May 2013).