# SmallSat Image Processing Using RISC-V


Cristian Jay Cobb
*Department of Electrical Engineering*
*University of Arizona*
Tucson, Arizona
cristiancobb@email.arizona.edu

Duy Do
*Department of Electrical Engineering*
*University of Arizona*
Tucson, Arizona
duydot@email.arizona.edu

Meghna Subramani
*Department of Electrical Engineering*
*University of Arizona*
Tucson, Arizona
msubramani@email.arizona.edu

Henry K Wong
*Department of Electrical Engineering*
*University of Arizona*
Tucson, Arizona
hkwong@email.arizona.edu



***Abstract*— This project explored the capabilities of the RISC-V instruction set architecture (ISA) in SmallSat image processing applications. The goal of the project was to run a single image processing application on the PULPino, RI5CY core and investigate, and introduce hardware optimizations for such applications. The image processing application chosen for this project, converts a grayscale image to a black and white image. Within the project timeframe, the simulation environment was setup, and the application was cross-compiled for RISC-V and run. Due to technical issues, with regards to the simulation environment, the research was suspended before any publishable results were generated. The paper describes the methodology used to conduct the research, details of the image processing applications, technical issues encountered with the simulations as well as future steps that could be taken to further this research project.**



***Keywords*— *RISC-V, PULPino, RI5CY core, SmallSats, image processing***


## I. Introduction

There are high costs associated with space missions making them low-risk tolerant. However, with the evolution of SmallSats and the use of cheap, commercial, off-the-shelf computing platforms more risk is being taken in space missions. RISC-V is an open source modularized instruction set architecture, which can be easily leveraged and expanded as necessary for generic applications as well as domain specific ones. It has the advantage of its openness, which contributes to shorter time to market and lower costs for reuse of hardware implementations. The openness of RISC-V allows for the adoption of terrestrial applications to space specific implementations with only the need of domain specific enhancements. Based on these benefits, this project investigates the performance and usability of RISC-V for SmallSat image processing applications, especially considering the wide use of such applications in both terrestrial and space domains.

Numerous researchers and companies are working on RISC-V and a number of processors based on this ISA have been created. The PULP platform builds on the RISC-V ISA, is reliable and provides a number of different designs with increasing complexity. The RISC-V ISA is divided into two parts: the base ISA and the extensions. The project focuses on the RISC-V, PULPino implementation, which has the RI5CY core. The base ISA used for the RI5CY core is RV32I, characterized by an instruction length equal to 32 bits. PULPino is characterized by:

- Two single port RAMs – one is for instructions and one is for data directly connected to the core.
- Central AXI interconnect
- Simple peripherals use the ABP protocol.
- AXI and ABP buses with 32 bits wide data channels.
- SPI slave that acts as a master on AXI. This provides access to the whole memory map of the SoC externally.
- Boot ROM – Bootloader loads program from SPI flash.

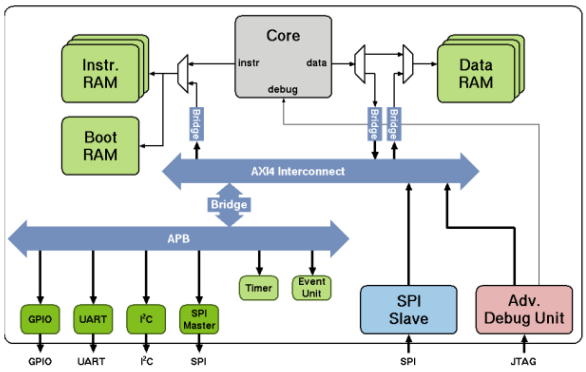


**Fig 1. PULPino block diagram [11].**

The project involves cross-compiling an image processing application for RISC-V and evaluating its performance on the PULPino implementation. Majority of the research conducted for PULPino focuses on IoT applications. Research in the domain of leveraging PULPino for space applications, let alone image processing applications is little or none.

## II. Related Work

The RISC-V implementation into the space specific domain has been slow but progressing nonetheless. The nature of open source software within the space industry has been hesitant given the requirements from the environment that the hardware must survive through. Some of the positive aspects of RISC-V in space is the openness and modularity features that it offers. As RISC-V gets more usage through different domains, the software ecosystem grows to benefit the space technologies as

well. The RISC-V ecosystem may not fulfill all the requirements for space like security and fault protection capabilities that are not provided by the software ecosystem [2]. Given an existing open source ISA would reduce cost, time, and complexity on systems and provide focusing resources toward improvements in system security and other critical subsystems to operate within information sensitive and volatile environments. Systems such as defense is one field of work that strongly needs highly secure designs that can handle any kind of data attacks. There are already studies that provide solutions for RISC-V operation in military applications that combat towards temporal and spatial memory attacks that comes in the form of extension of RISC-V [5].

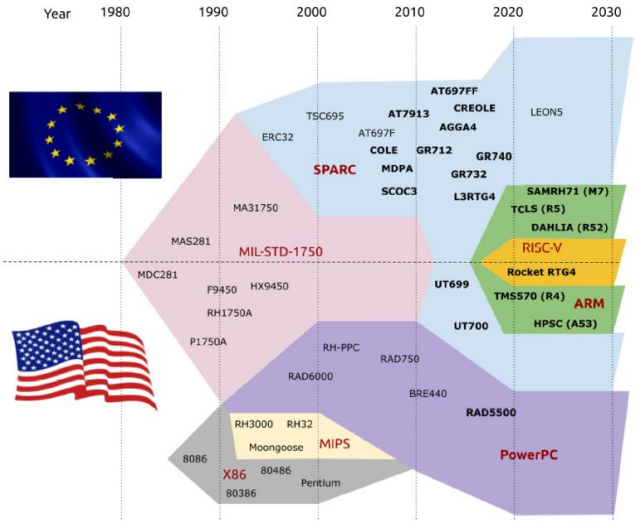


**Fig 2. Main CPU ISAs and their market share evolution in Europe and the United States (in bold SoCs). [1]**

Fault tolerance is another critical portion of the space domain that must be addressed within the system. It is extremely important that the system can handle issues within space since communication and hands on fixes are minimal and must be fixed quickly or it would cause catastrophic failure and loss of hundreds of millions of dollars of hardware and data. There are studies that utilized RISC-V, specifically Rocket, BOOM, and PULPino, for fault injection to test the performance [3]. Results from the study show that error can be smaller than 7%. Nonetheless, additional improvements and enhancements must be conducted to the IP to accommodate requirements such as Triple Modular Redundancy of the processor which typically handled by providing hardware redundancies which increased size, weight, power, and cost. To implement a fault tolerant IP Core from COTs products are too expensive and not a viable option. RISC-V IP can be modified for FT needs easily given the nature of open source IP. This path would provide systems the capability to fulfil the fault tolerance requirements and not designing the IP Cores from scratch [10].

Space domain also entails different classifications of a satellite that are designed to perform different types of missions based on requirements. This project, we chose to stick with the category of Smallsats and CubeSats. Microcontrollers are currently the best option for the SmallSat and CubeSat. Cost drives the need for Smallsats that could operate effectively and efficiently with minimal power, memory, processing resources, and zero-byte failures. Not only that, the Smallsats or CubeSats must respond to RF commands and communicate back to the ground, take high resolution images, process the images to see if the images are usable, and finally send the high resolution images back to the ground with no errors in a timely fashion for post-processing. Currently, some systems of the sort would take hours to send such images for analysis. Additional drivers for increased memory and processing requirements are the advent of new and improved technology in image sensors that are required to operate with greater memory, higher bandwidths, and greater power. This drives the need for low power and efficient ISAs that works well with image processing to determinate a good or bad image and filters them accordingly [4]. This trend cannot be sustained without methods to alleviate the need for such high data downlinking. Solutions are to improve on algorithm and machine learning to cope with the capabilities [9].

ISA that would provide minimal power usage and memory overhead is the solution. In the case of RISC-V, the three cores that will be discussed are Micro-riscy, Zero-riscy, and RI5CY in smallsat or cubesats platforms. Each core is optimized for specific applications. Micro-riscy is optimized for area with no hardware support for multiplication or division and implements the RVM RISC-V specification for the usage of 16 general-purpose registers. Main usage for Micro-riscy are for very small area and power usage for control-code tasks for always-on voltage domains, interacting with peripherals, off-loading tasks, and power management activities utilizing the RVC32E subset ISA. Zero-riscy is slighter larger in area and a significant jump in capability compared to the Micro-riscy. Optimization for Zero-riscy are for arithmetic-control mixed applications on complex systems that require peripheral control and accelerators.

Applications for Zero-riscy could be general-purpose applications and complex control systems that requires significant calculations but utilize minimal area and power. Finally, the RI5CY open source 32-bit RISC-V core implementing the RVC32IM ISA with enhanced performance, reduced code size, and increase energy efficiency. It is an in-order four pipeline stages configuration with a general-purpose extension and DSP-oriented extensions for great computational power. RI5CY core has been optimized to work in multi-core clusters with PULP to provide high performance in power-constrained embedded systems.

Studies comparing the three cores has proven that DPS instructions provide the most energy efficiency in data-intensive applications which places the RI5CY best for those capabilities. Zero-riscy for mixed arithmetic-control and Micro-riscy for pure control code given the minimal resource [6]. Studies were conducted to compare modern state-of-the-art ARM cortex equivalent series and compared them in performance, area, and area efficiency [6].

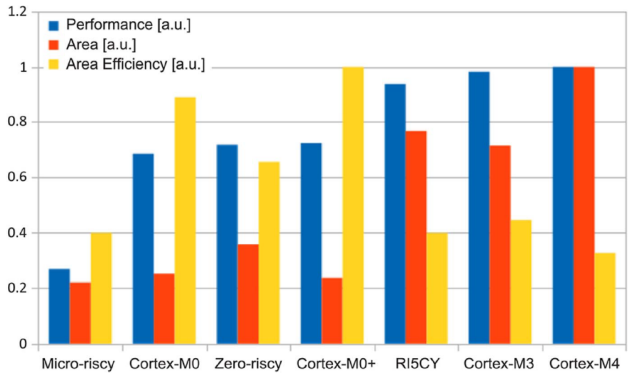


**Fig 3. Performance, area, and area efficiency for several implementations targeting microcontrollers. The data are derived from [6], except for the performance of the Cortex-M3 and Cortex-M4 [11].**

Additional studies associated with determining the number of pipelines were discussed and determined that for time-deterministic without speculations, the optimal setting of 2-3 stages are best for performance [7]. There is a homogeneous set of cores currently developed by Klessydra, designed for PULPino SoC and configured for 2-4 stages, 4 threads for fine-grain multithreading [8]. This configuration also utilizes the RI5CY with an F extension to assist with single-precision floating-point operations which based on studies show that is comparable to Cortex-M3 and Cortex-M4 in terms of performance.

Additional challenges that smallsat and image processing conflict with is the resource utilization. Image processing is a computation intensive capability that is growing in Smallsats. The resources needed for such computation to perform at a hard time constraint is the RI5CY IP cores RV32IMC(F) with short pipelines and no speculations to avoid penalties on time determinism [9]. Our effects for this project, focus more on payloads that utilize image processing algorithms that does on-board processing. The hope is to further understand the capabilities and benefits that the RI5CY has associated to the smallsat hardware specific performances. There are plenty of studies associated with terrestrial hardware implementation like Cortex-M3 and Cortex-M4 processors and ASIC SoC designs but not much in the realm of radiation hardened hardware, security and fault tolerant functionality that meet current requirements. Significant research is still needed and support from both the terrestrial and space industry is required to further expand the benefits of RISC-V.

## III. METHEDOLOGY

### A. *The Bitmap Image*

At the core of image processing is the act of processing or altering an existing image to achieve a desired look. Images are available in a number of formats such as JPEG, PNG, TIFF, BMP, etc. In this project we focus our discussion and experiments on the Bitmap file format (BMP). Compared to compressed file formats such as JPEG, BMP's are simple to understand and decode.

Bitmaps are defined by a regular mesh of cells called pixels, and are always oriented horizontally and vertically. A color value is contained within each pixel. Bitmaps are characterized by two fundamental parameters: number of pixels and color depth per pixel [12].

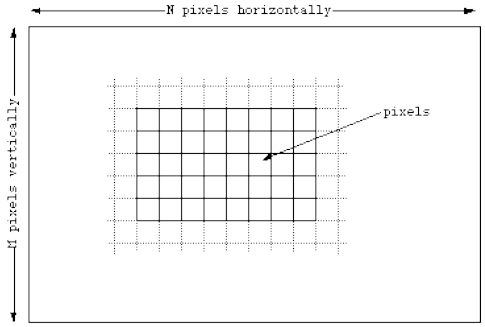


**Fig 4. Illustration of Bitmaps [12]**

Color depth is the information contained within the pixel. Color information standards for black and white images, and grey scale images are described below.

- *Black and White or Monochrome bitmap (1-bit)* – Each pixel can hold 1 bit. Pixels with a 0 are black, pixels with a 1 are white.
- *Greyscale ( 8-bit greys)* - Each pixel can hold 1 byte (8 bits), which results in 256 different states. Bits are mapped to a scale of greys (black to white). By convention, 0 is typically black and 255 is typically white. The numbers in between describe the gray levels; for example – the number 127 would be equal to a 50% grey level.


**Fig 5. Illustration of Greyscale (0 – black, 255 – white)**

Typically a BMP file consists of 3 – 4 parts: the header, the information section, color table, and image data. The color table is included if the image is tagged as colored, and helps identify the shade of the color based on the value. The image header is 54 bytes and the color table if present is 1024 bytes. The header contains information needed to process the image such as the image width and height, the offset to start of image data, number of bits per pixel, size of image data and number of colors in image [13].

### B. *Application - Greyscale to Black-White*

The image processing application used in the project, converts a grayscale file (lena512.bmp) to a black and white bmp file.

**Fig 6. Original grayscale image – lena512.bmp**

The application is written in C. To convert the original greyscale image to black and white, the first step is to open the file and strip out the image header and extract the width, height and number of bits per pixel (color depth). The second step is to read the color table if it exists and subsequently read the image data into a buffer. In order to convert the image into black and white, each pixel can hold either value 0 – black, or value 255 – white. All pixel values above the value 128 are converted to white and the remaining converted to black. Based on this, the new image data is written to the buffer and the new image is written to the output file.

**Fig 7. Processed black and white image- b_w.bmp**

*C. PULPino*

Following requirements need to satisfied in order to run the image processing application with PULPino:

- ModelSim – version 10.2c or higher
- CMAKE – version 2.8.0 or higher
- Python2 – version 2.6 or higher
- riscv-toolchain – The custom RISCV toolchain from ETH Zurich was cloned form git. With this toolchain all RI5CY ISA extensions can be leveraged.

First the toolchain was built via the command *make*. Once the PULPino repository was cloned, a new folder called *build* was created in */sw/* in order to simulate applications. Then the script *cmake_configure.riscv.gcc.sh* was copied from the *sw* directory into the *build* directory and run. The script initializes parameters necessary to simulate applications with ModelSim.

*cmake_configure.riscv.gcc.sh* selects the RIC5Y core and sets the environment to support the PULP-extensions and the RV32IM ISA.

ModelSim was used to run and analyze RTL simulations by first running the command *make vcompile*. Subsequently the simulation of the desired application was started using *make application_name.vsim* to execute the application via the GUI, or *make_application_name_vsimc* to execute the application via the console.

*D. Measuring Performance*

The RI5CY core supports a number of performance counters that can be leveraged to profile an application. Performance counters are able to count the total number of cycles, instructions, load hazards, load instructions, store instructions, jumps, etc. Access to these counters is available via the libraries provided with the PULPino installation (example – *bench.h*). The following four functions are required to setup the performance counters required to analyze the image processing application:

- *perf_reset(void)* – Performance counters are set to zero.
- *cpu_perf_conf_events(eventMask)* – Configure events that need to be counted in the program and begin count.
- *perf_stop(void)* – Stops all performance counters.
- *cup_perf_get(ID)* – Provides the value stored in the counter based on the specified counter ID.

An event can be configured in two ways: using a macro that maps to a specific event type or the function *SPR_PCER_EVENT_MASK*, which takes as an argument an integer called the *event ID*, that identifies the correct event type.

Following are events that can be counted:

- Clock Cycles – macro SPR_PCER_CYCLES or the *event ID 0.*
- Instructions – macro SPR_PCER_INSTR or the *event ID 1*.
- Load Hazard (stalls caused by load instructions) – macro SPR_PCER_LD_STALL or the *event ID 2*.
- Jump Stall (stalls caused by jump instructions) – macro PR_PCER_JMP_STALL or the *event ID 3*.
- Instruction miss – macro SPR_PCER_IMISS or the *event ID 4*.
- Load Instructions – macro SPR_PCER_LD or the *event ID 7*.
- Store Instructions – macro SPR_PCER_ST or the *event ID 8*.
- Jumps – macro SPR_ PCER_JUMP or the *event ID 9*.
- Branch – macro SPR_PCER_BRANCH or the *event ID 10*.

The events and macros were extracted from the PULPino repository */sw/apps/sequential_tests/convolution/convolution.h*

*E. Introducing & Testing Performance Improvements*

The primary goal of this project was to investigate and implement changes to the PULPino platform that could improve the performance of the chosen image processing application. Within the PULPino directory, System Verilog files are present in the */rtl/* subdirectory. These System Verilog files are organized by the various blocks present in PULPino (Fig. 1).

In order to change the configurations, System Verilog files need to be modified. It was intended that first changes would be introduced to the PULPino top-level System Verilog file, in order to modify the default memory map. The application would be re-run and the performance analyzed. Based on the initial

analysis, further changes would be introduced in order to gain a substantial amount of improvement.

## IV. Experimental Results

### A. Approach

To run the greyscale to black and white, application on PULPino, a new application folder was created in the applications directory (*sw/apps/*). The folder was named *black_white* and inside this folder the C program (*black_white.c*), the original image (*lena512.bmp*), and a *CMakeLists.txt* file were added. In the *CMakeLists.txt* file the line *add_application(black_white black_white.c)* was added. This allowed the application to execute in the modelSim GUI or console with the commands *make black_white.vsim* or *make black_white.vsimc* respectively.

In the directory *sw/apps* the CMakeLists.txt was modified to include the new application by adding the line *add_subdirectory(black_white)*.

### B. Issues Encountered

A number of issues were encountered with the setup of the PULPino environment. From start to finish, setting up the environment took 6 weeks. Issues included:

- Configuration changes required to PULPino files to support the 32-bit version of ModelSim
- Configuring the environment to ensure that ModelSim used the toolchain compiler instead of its native compiler.
- Errors generated while attempting to run the test application "Hello World", provided with PULPino. The issues were caused due to the fact that certain symbolic links need to be created.
- Debugging issues with the CMake configurations.
- PULPino is unable to run C++ applications. By default, for C++ applications, the toolchain uses the gcc compiler instead of the g++ compiler, which results in a number of standard library errors. This issues remains unresolved. This issue had a major impact on the team, as the original application chosen for the project was in C++. Additional time was spent trying to find an appropriate C image processing application and getting it to run.

Another impact on the results was the execution time of the RTL simulation. The greyscale to black and white application was running in ModelSim for multiple days. The simulation was forcefully terminated after a week, as it appeared that the simulation was stuck and not actually executing. This issue remains unresolved and further investigation is needed into the PULPino configuration files as well as ModelSim to resolve this issue.

PULPino also lacks sufficient documentation. Most of the documentation is available in the form of presentations. A considerable amount of time was invested in researching what performance counters are provided within the pulpino repository and how to leverage them. This information was extracted from code and examples provided within the PULPino directories.

### C. Results

Due to issues described above, the research was suspended before any publishable results were generated.

### D. Future Steps

During the investigation of performance counters, it was discovered that the RI5CY core provides a set of extensions for small vector manipulations typical of image processing applications [14]. These include:

- Dot product between vectors
- Mul/Add/Sub, round and normalization
- Shuffle operations for vectors
- Packed-SIMD ALU operations
- Bit manipulations

The vector extensions can be enabled with the following compiler options:

- no extensions – "*-mnohwloop -mnopostmod -mnomac*"
- with hardware loops – "*-mnopostmod -mnomac*"
- with post increment – "*-mnomac*"
- with register mac – "."

Within the PULPino repository there even exist common image processing algorithms that can be profiled using the previously described performance counters. An example of this is in the directory:

*/sw/apps/sequential_tests/convolution/conv_kernels.c*

*Conv_kernels.c* cover Convolutions which are import kernels in image processing. The examples provided in slides by ETH Zurich provide a glimpse into how matrix multiplications work within the RI5CY core and how they can be optimized using the vector dot product instruction [14]. The paper "*A near-threshold RISC-V core with DSP extensions for scalable IoT Endpoint Devices"*, is a valuable resource on how the RISC-V core has been extended to support hardware loops, post-incrementing addressing modes and SIMD instructions. It also utilizes the convolution code reference above [15].

Although this information was gathered much to late in the project timeline, the resources and paper reference above could

be leveraged for future research in this area. Since PULPino lacks sufficient documentation, it is advised to traverse the repository in order to find appropriate applications and examples.

## V. Conclusion & Future Work

The goal of the project was to run a single image processing application on the PULPino, RI5CY core and investigate, and introduce hardware optimizations for such applications. We have presented how to configure the PULPino environment and run RTL simulations. We have also researched an identified performance counters that can enable the profiling of an application run on the PULPino, RI5CY core. We also proposed a methodology for introducing and testing the performance improvements introduced by hardware enhancements. Due to technical issues and the project time constraint, the team was successful only with setting up the environment and starting the RTL simulation. The project concluded with no publishable results.

Future research in this area, could start with using the pre-existing image processing algorithm examples provided within the PULPino repository and then mimic the setup for newer applications. With the wide-spread usage of PULPino and its various cores, future work could also focus on providing documentation for utilizing the performance counters, running RTL simulations with ModelSim, and extending the RISC-V core.

## References


[1] Stefano Di Mascio, Alessandra Menicucci, and Eberhard Gill Delft University of Technology, 2629 HS Delft, The Netherlands and Gianluca Furano and Claudio Monteleone, European Space Agency, 2200 AG Noordwijk, The Netherlands "Leveraging the Openness and Modularity of RISC-V in Space ", JOURNAL OF AEROSPACE INFORMATION SYSTEMS, DOI: 10.2514/1. I010735

[2] Salmon, L. G., "A Perspective on the Role of Open-Source IP in Government Electronic Systems," 7th RISC-V Workshop Proceedings, Milpitas, CA, Nov. 2017.

[3] Cho, H., "Impact of Microarchitectural Differences of RISC-V Processor Cores on Soft Error Effects," IEEE Access, Vol. 6, 2018, pp. 41302–41313. doi:10.1109/Access.6287639

[4] Gillette, A.,Wilson, C., and George, A. D., "Efficient and Autonomous Processing and Classification of Images on Small Spacecraft," 2017 IEEE National Aerospace and Electronics Conference (NAECON), IEEE Publ., Piscataway, NJ, 2017, pp. 135–141.

[5] Menon, A., Murugan, S., Rebeiro, C., Gala, N., andVeezhinathan, K., "Shakti-T:A RISC-V Processor with Light Weight Security Extensions," Proceedings of the Hardware and Architectural Support for Security and Privacy, ACM, New York, 2017, pp. 2:1–2:8. doi:10.1145/3092627.3092629

[6] Schiavone, P. D., Conti, F., Rossi, D., Gautschi, M., Pullini, A., Flamand, E., and Benini, L., "Slow and SteadyWins the Race? A Comparison of Ultra-Low- Power RISC-V Cores for Internet-of-Things Applications," 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), IEEE Publ., Piscataway, NJ, 2017, pp. 1–8.doi:10.1109/PATMOS.2017.8106976

[7] Cheikh, A., Cerutti, G., Mastrandrea, A., Menichelli, F., and Olivieri, M., "The Microarchitecture of a Multi-Threaded RISC-V Compliant Processing Core Family for IoT End-Nodes," Apple Pies 2017: Applications in Electronics Pervading Industry, Environment and Society, Springer International Publ., Cham, 2019, pp. 89–97. doi:10.1007/978-3-319-93082-4_12

[8] Caprita, H.V., and Popa, M., "Design Methods of Multithreaded Architectures for Multicore Microcontrollers," 2011 6th IEEE International Symposium on Applied Computational Intelligence and Informatics (SACI), IEEE Publ., Piscataway, NJ, 2011, pp. 427–432. doi:10.1109/SACI.2011.5873041

[9] Doubleday, J., Chien, S., Norton, C., Wagstaff, K., Thompson, D. R., Bellardo, J., Francis, C., and Baumgarten, E., "Autonomy for Remote Sensing - Experiences from the IPEX CubeSat," 2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), IEEE Publ., Piscataway, NJ, 2015, pp. 5308–5311. doi:10.1109/IGARSS.2015.7327033

[10] Di Mascio, S., Menicucci, A., Furano, G., Monteleone, C., and Ottavi, M., "The Case for RISC-V in Space," Applications in Electronics Pervading Industry, Environment and Society, ApplePies, 2018, edited by S. Saponara, and A. De Gloria, Vol. 550, Lecture Notes in Electrical Engineering, Springer, Cham, 2019, pp. 319–325.

[11] https://github.com/pulp-platform/pulpino

[12] http://paulbourke.net/dataformats/bitmaps/

[13] http://paulbourke.net/dataformats/bmp/

[14] https://iis-people.ee.ethz.ch/~gmichi/asocd/exercises/ex_02.pdf

[15] Gautschi, Michael & Schiavone, Pasquale & Traber, Andreas & Loi, Igor & Pullini, Antonio & Rossi, Davide & Flamand, Eric & Gurkaynak, Frank & Benini, Luca. (2016). Near-Threshold RISC-V core with DSP extensions for scalable IoT endpoint devices. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. PP. 10.1109/TVLSI.2017.2654506.