Horizon Detection Hardware Accelerator for FPGA Based Space Computers

Benjamin J. Macht
Electrical and Computer Engineering
Montana State University
Bozeman, Montana, USA
benjanim.macht@student.montana.edu

Zachary Becker
Electrical and Computer Engineering
Montana State University
Bozeman, Montana, USA
bitbybiteco@gmail.com

Lucas Ritzdorf
Electrical and Computer Engineering
Montana State University
Bozeman, Montana, USA
lritzdorf@ieee.org

Mackenzie Smith
Electrical and Computer Engineering
Montana State University
Bozeman, Montana, USA
kenziegm@gmail.com

Hezekiah A Austin
Electrical and Computer Engineering
Montana State University
Bozeman, Montana, USA
hezekiah.austin@student.montana.edu

Brock J. LaMeres
Electrical and Computer Engineering
Montana State University
Bozeman, Montana, USA
lameres@montana.edu

Chris Major
Resilient Computing
Bozeman, Montana, USA
chris.major@resilient-computing.com

2025 Intermountain Engineering, Technology and Computing (IETC) | 979-8-3315-1282-8/25/$31.00 ©2025 IEEE | DOI: 10.1109/IETC64455.2025.11039393

***Abstract*— As humanity probes deeper into the solar system and becomes increasingly dependent on satellite technology for everyday life, the need for more powerful, resilient, and cost-effective space computing systems is increasing. This paper presents an image processing hardware accelerator design implemented on a Xilinx Artix-7 200T FPGA and integrated into a RISC-V soft processor system housed on the same FPGA. The system described performs horizon detection and computes orientation information from live, streaming camera data received from an on-board camera. It then displays the live data along with horizon and orientation correction overlays on an on-board liquid crystal display. Design solutions along with verification test setups and results are discussed. Testing results showed that the system was able to display 320 x 240-pixel live video at 30 frame per second with horizon and orientation overlay calculations being refreshed at 15 Hz using a processor running at 32 MHz. The design demonstrated the performance enhancing capabilities that custom hardware accelerator blocks can have on small satellite computers without requiring the use of additional resources.**

Keywords—FPGA, hardware accelerator, image processing

I. Introduction

Navigation and orientation are essential but highly complex tasks for landing spacecraft. When uncrewed spacecraft attempt to land, they must rely entirely on onboard computing systems to execute critical functions, such as horizon detection for orientation feedback. However, spaceborne computing faces significant challenges, primarily due to the disruptive effects of high-energy particles on the logic of integrated circuits. While there are existing technologies to mitigate these issues, they often come with drawbacks like added weight and high costs. Recently, the aerospace industry has begun exploring the use of commercial off-the-shelf (COTS) parts in space that run fault-mitigation procedures to defeat the effects of radiation. The use of COTS parts has a significant cost and performance benefit compared to the traditional "radiation hardened" processors.

This paper explores integrating an image processing hardware accelerator into a Field Programmable Gate Array (FPGA) based computer called *RadPC* developed by Montana State University (MSU) and Resilient Computing, LLC. RadPC implements redundant RISC-V processing cores [Fig. 1] configured in quad modular redundancy (QMR) with a voter to produce the system output. RadPC also implements redundant memory scrubbing and configuration memory soft error correction. These systems provide fault detection and recovery from radiation-induced Single Event Effect (SEE) [1], [2].

Previous iterations of the RadPC have undergone rigorous testing on high-altitude balloons (8x), sounding rockets (2x), the International Space Station (3x), small satellites (2x) [3], and, most recently, aboard the *Firefly Aerospace* Blue Ghost lunar lander launched from Kennedy Space Center on January 15, 2025 [4]. The project described in this paper showcases the flexibility of the FPGA-based soft processor platforms by utilizing RadPC's memory mapped hardware accelerator

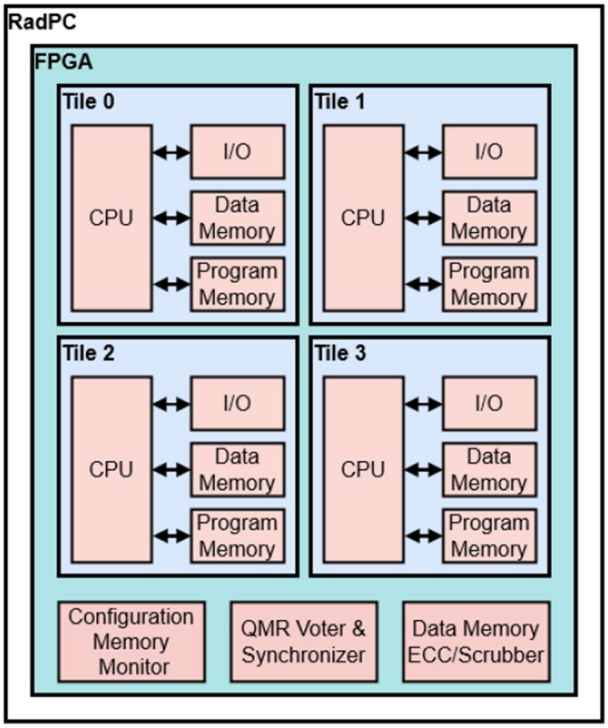


Fig. 1. RadPC computer architecture.

interface to integrate with a custom image processing / horizon detection system to provide real-time image processing capabilities at 30 frames per second that could be used by landing spacecraft as a radiation-tolerant orientation method on future missions.

II. Subsystem Design

A. System Overview

Fig. 2 depicts the general layout of the system described below. At the top level, there are four primary elements: an OV7670 complementary metal oxide semiconductor (CMOS) enabled, 640 x 480 pixel camera, a 240 x 320 pixel, LCD display that utilizes an ILI9341 single chip driver, a custom designed power system, and a RadPC Single Board Computer (SBC) that includes a Xilinx Artix-7 200T FPGA. The system's operational flow is as follows: live camera data is fed into the hardware accelerator where full pixel data is passed on to and written to the display. Additionally, specific pixel bytes are fed downstream to the image processing subsystem. After receiving a full frame, the accelerator peripheral asynchronously performs several steps before the next frame is received. These steps include identifying a horizon in the frame, deriving a mathematical parameterization for the identified horizon, rendering the parameterized horizon arc, and relaying this rendering to the display as an overlay element. The system then calculates the roll angle of the parameterized horizon, renders and displays a directional indicator representing the direction of roll required to make the parameterized horizon level, and provides numerical orientation data to the user through a memory mapped register interface.

Upon startup of the system, the RadPC's 32 MHz processor initializes the camera using a Serial Camera Control Bus interface that is similar to I2C communication protocol. During operation, the processor exerts control over the accelerator peripheral via a memory mapped interface adhering to the RadPC platform's coprocessor specifications. Notably, the processor does *not* directly handle the streaming of frame data from the camera or perform any calculations. Upon completion of the startup process, RadPC has no responsibility for the operation of the accelerator and is free to perform other tasks. However, at any point during operation RadPC has access to registers to alter parameters of the accelerator and can retrieve correction angle data calculated by the processor.

B. Data Management Unit

The Data Management Unit (DMU) is the component of the system responsible for managing the live stream of camera data, delivering the proper bytes of this data to image processing, and writing the correct data to the display. The DMU resides completely in the fabric of the FPGA and is split into two parts, the front-end and back-end. The front-end takes in the live, streaming Red Green Blue (RGB) camera data and passes the live data downstream to the image overlay component. In addition to passing through live data, the DMU front-end converts each pixel word of RGB 565 to a chroma value and passes the first byte of this converted pixel word to image processing for edge detection. The back end of the DMU is responsible for the horizon and correction image overlay process, the display initialization process, and the pixel write sequence required to write pixel data to the display.

The front-end of the DMU uses a standard Vivado dual port, dual clock ram IP block as a first in, first out (FIFO) memory unit that writes pixel byte data on the rising edge of the 24 MHz, camera generated pixel clock and reads data out on the rising edge of the 50 MHz system clock. Pixel bytes are read out of the FIFO one at a time, concatenated to form a complete pixel word, and then sent downstream to the RGB to luma conversion block and image overlay. After luma conversion, the first byte of each pixel word is sent downstream to image processing. Column counting is performed after a complete column of 480 bytes has been written and read from the FIFO and the read and write pointers are reset. Complete pixels are counted, and the resulting pixel address is sent downstream with each pixel. Pixels are counted until a complete frame of 76,800 pixels has been read before resetting the pixel and column counters. The DMU then relies on synchronization signals generated by the camera to restart the read and write process.

The image overlay component of the DMU [Fig. 3] is used to overlay the horizon and the correction indicator on top of the live camera data before an image is written to the display. The overlay component works by first receiving a pixel word

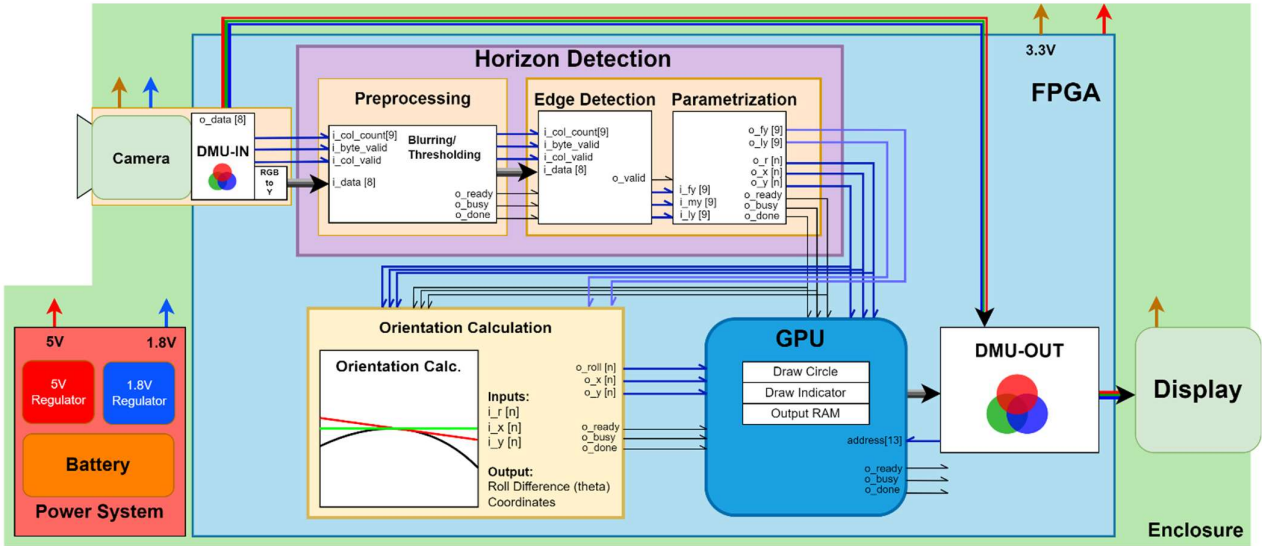


Fig. 2. Conceptual block diagram of the entire system.

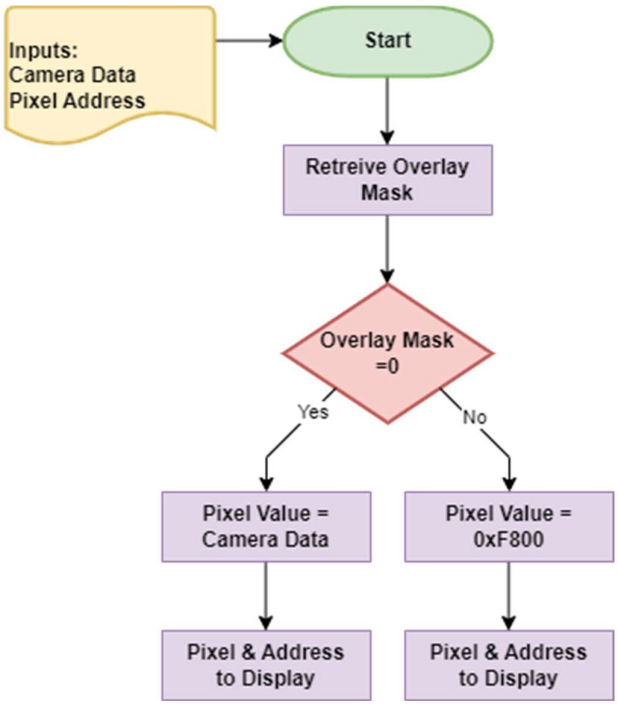


Fig. 3. Flowchart showing the functionality of the image overlay component of the DMU.

and address from the front-end of the DMU. The current pixel address is then used to retrieve an overlay bit mask located in a block RAM populated by the GPU. If the overlay bit is a zero, the image pixel is passed downstream to be written to the display. If the overlay bit is a one, the hexadecimal value for red (F800) is passed downstream and a red pixel is written to the display instead of the camera image data for that pixel.

The back end of the DMU component handles the initialization and pixel write process to the display. Upon reset, the back end of the DMU performs an initialization sequence that writes proper setup data to the ILI9341 control registers. After initialization, the back-end component enters a state machine that waits for a new pixel, writes that pixel to the current display address, and resets the display address to start writing a new frame after pixel address 76,7799 has been received.

C. *Image Processing*

The Horizon Detection (Image Processing) subsystem is the component of the system that is responsible for taking real-time pixel data from the DMU, determining the horizon line, and calculating a parameterized form of this line to provide inputs for the graphics processing unit (GPU). This sequence of operations is broken down in four different sub-components: preprocessing, column loading, edge detection, and parametrization.

The functionality of the preprocessing block is simple in concept as well as in construction. Data values enter the component in the form of a byte. This data byte is then compared to a threshold value, an adjustable parameter of the system that is set through the memory mapped register interface with RadPC. If the data byte is greater than this threshold value, the output of the component is set to be the maximum value of an unsigned byte (2^8-1 = 255). However, if the data byte is less than this threshold value, then the output of the component is set to be the minimum value of an unsigned byte (zero).

The column loader then loads three sets of two columns into indexed locations in D-flip-flops. For a two-dimensional first-difference algorithm to function, the edge-detection component needs to have data indexed in memory such that two different first-difference calculations can take place. To facilitate this indexing, a custom protocol was developed between the DMU and the Image processing subsystems that loaded the first byte of each gray scaled pixel into an array for each 240-pixel column.

Upon completion of a pair of 240-byte columns being loaded, the edge detection algorithm can occur. The results of those two calculations are then logically OR'd together to produce the edge-detection component's final result. This was then done for three sets of two 240-byte columns, essentially sampling the image three times across the x-axis for y-axis values, resulting in three x, y coordinates on the horizon line. To accomplish the first-difference algorithm as fast as possible, the use of a shift-register comprised of D-flip-flops was used. This allowed for the row-wise calculations to be made in one clock cycle and the column-wise calculations to me made in another clock cycle, allowing for extremely low latency through the edge detection component.

The result of the first difference algorithm is a singular 240-bit long column [Fig. 4], a binary mask, which has zeros where there is no edge and ones where an edge has been found. Since the x-axis values are constants within the system, and are consequently known, the remaining procedure is to determine what the index value (which represents the y-axis value) is the for the top-most high value that is in the mask. By doing this the system is looking for the highest edge, vertically speaking along the y-axis, of the image frame. Since we are using digital logic to accomplish this, we have implemented the use of a priority encoder, whose function is to produce an output address for the input with the highest priority. The resulting y-axis value when paired with the respective x-axis value produces one of the coordinate-pairs needed to parameterize the horizon line. The above procedure is then repeated two more times.

The three coordinate pairs are then fed into the parameterization subsystem. Each set of coordinates are then plugged into (1) producing three quadratic equations, each representing the circle of the horizon line. Subtracting the first from the second of these equations and the first from the third results in a system of two linear equations with two unknowns. Solving this system of equations results in the x, y coordinates

$$(x - x_c)^2 + (y - y_c)^2 - r^2 = 0 \qquad (1)$$

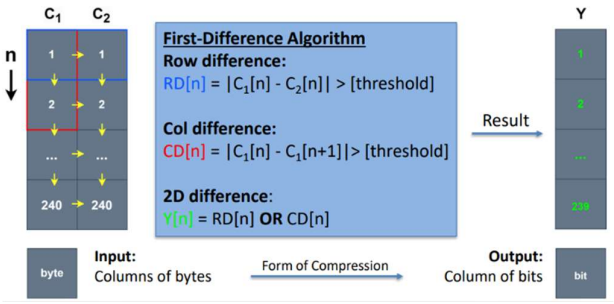


Fig. 4. Edge detection diagram.

of the center of the circle. The origin coordinates are then plugged into one of the previous quadratic equations to find the radius-squared value. A Vivado division IP block was used in the parameterization subsystem to perform the required division to derive the x, y coordinates. The origin coordinates and the radius-squared values are then passed to the GPU subsystem for circle rendering.

D. Graphics Processing Unit

The Graphics Processing Subsystem (GPU) is the component of the system that takes the circle center coordinates and radius squared value from Image Processing and renders the horizon arch, calculates the correction angle needed to be parallel with the horizon, and provides image overlay data. The GPU subsystem consists of three primary stages: circle rendering, indicator rendering, and output RAM.

The circle rendering stage consumes three circle parameters, center x- and y-coordinates and a radius-squared value from the upstream horizon detection component. Using these parameters, it identifies which pixels of the display are to be "drawn" as part of the circle's border. This is done if the pixel under consideration, whose coordinates are known, is part of the border stroke if it has a radius-squared value which is either greater than the ideal circle's radius squared, minus a predefined margin, and less than the ideal circle's radius squared, plus the same predefined margin.

These conditions, taken together, create a ring shape. For intuitive purposes, it may be useful to consider the case of a Venn diagram, in which one region completely encompasses the other. Then, the area between the regions' boundaries (i.e. which includes the larger region but excludes the smaller one) is a ring shape. This is exactly the sort of logic which the GPU utilizes to identify which pixels are to be part of the circle's border stroke.

Following this, dataflow continues to the orientation indicator rendering/overlay stage. Here, the GPU consumes inputs which indicate whether the detected horizon is inclined clockwise or counterclockwise using a predefined lookup table that compares the y values of the horizon endpoints and converts them to a horizon angle. This angle is then used to determine if the roll needs to be corrected clockwise or counterclockwise and written to a memory mapped register where the user can access the value in real time. Additionally, the proper indicator is selected, based on the correction direction, from a set of predefined glyphs that are stored as simple binary image masks in memory. The appropriate mask is then overlaid onto the circle-border mask produced by the previous stage [Fig. 5]. This design implements the overlays as a Boolean XOR. Because of this, if a region of the image is specified as filled by both masks, the XOR operation produces an un-filled output for that region.

After the indicator glyph and rendered circle are overlaid, the resulting completed frame is ready to be stored. This is handled by the third and final stage of the GPU: the output RAM. This is a standard block RAM component in the FPGA fabric, with a few pieces of surrounding infrastructure. Most importantly, it converts the counted pixel-based addresses to RAM-style word indices. For instance, since the display is 240 by 320 pixels, the pixel index 300 corresponds to the 60th pixel of the second row, which is the address format required in order to retrieve the relevant pixel from block RAM. Following this conversion, the GPU returns a single bit, whose value represents the state of the rendered mask at the pixel requested by the downstream DMU.

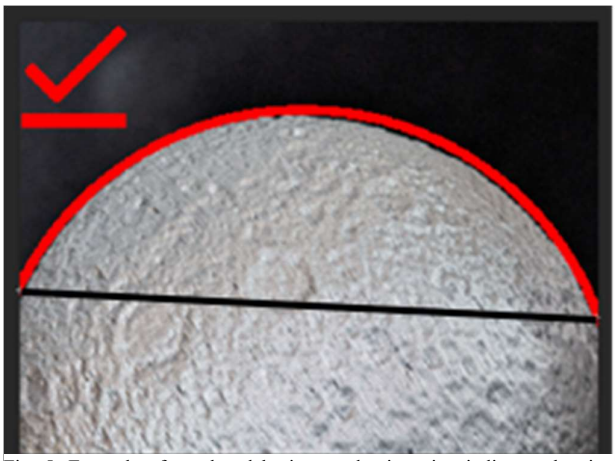


Fig. 5. Example of overlayed horizon and orientation indicator showing horizon endpoints used for correction angle calculation.

II. Verification

Upon completion of subsystem fabrication, proper functionality of each subsystem was verified before integration into the final system. Verification of the DMU subsystem included measuring the framerate of camera data being written to the display, checking for image distortion on the display [Fig. 6], and proper pixel data was being sent downstream to image processing [Fig. 7]. External and internal logic analyzers were used to verify timing and data transfer, and measurements were taken to ensure no image distortion was occurring. A framerate of 30 frames per second, proper data and timing, and no image distortion were verified with these tests.

Verification of the Image Processing subsystem included verifying the edge detection and circle parameterization outputs were correct. To test the edge detection component an Arduino Uno was used to emulate a frame of data being loaded into the image processing subsystem [Fig. 8]. This same frame of data was first run through a MATLAB proof-of-concept script to generate three y values and their associated x values. Then the values generated by the horizon detection component were compared to these values and found to be within one pixel of the proof of concept values. This error was attributed to the rounding difference between the double precision software calculations and the fixed-point calculations in hardware.

To test the proper functionality of the parameterization component a similar test set up was used. Three known points were run through a MATLAB proof-of-concept algorithm and

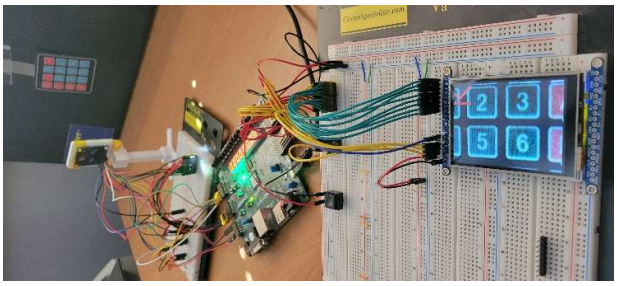


Fig. 6. Test bed used to test for image distortion.

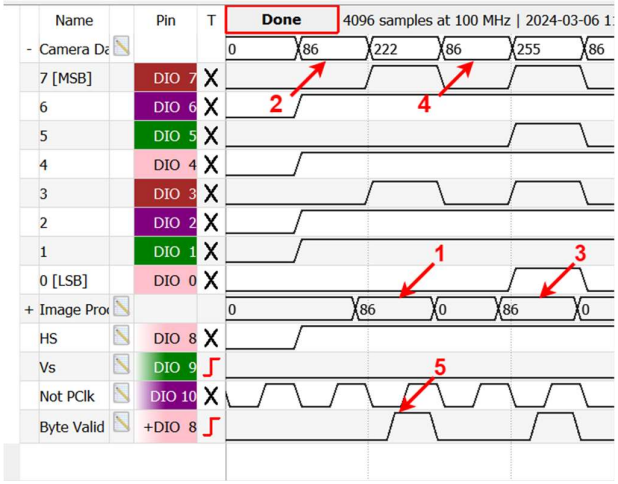


Fig. 7. External logic analyzer waveforms used to verify the proper functionality of the DMU. Every other camera pixel byte (2 & 4) can be seen being sent downstream to image processing (1 & 3) along with the byte valid signal (5) being set in the middle of valid image processing data.

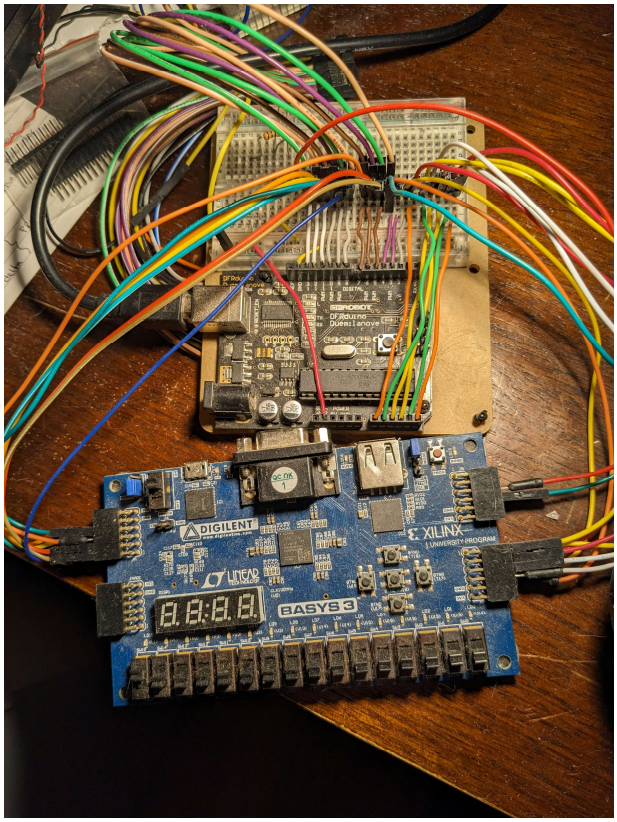


Fig. 8. Test bed used to verify the proper functionality of the image processing subsystem.

the MATLAB center coordinates and radius squared value was compared to the output of the parameterization component processing the same three points. Again, the hardware generated values were found to be within one pixel of the expected values and the error was attributed to rounding of the fixed-point hardware calculations.

Verification of the GPU subsystem included testing the accuracy of the rendered horizon line and correction angle calculations along with verifying proper indexing of the overlay bitmask in the output RAM. A test bed that included the actual logic circuitry on the FPGA was fed circle center coordinates and radius squared values using a Virtual IO module. Values from a known image were used to generate a horizon overlay and calculated correction angle and compared to measurements taken from the original image. The rendered horizon was found to match the test image and the correction angle was within 0.2 degrees of the measured angle, well within the required five-degree of allowable error.

After verifying proper functionality of the individual subsystems, all components were integrated into the final system [Fig. 9]. Live image framerate was once again measured and found to be 30 frames per second. Additionally, the latency of the entire data path through the image processing and GPU subsystems was measured using an internal logic analyzer to monitor start and finish signals. The refresh rate of overlay and correction angle data was measured at 15 Hz, or once every other frame.

FPGA resource utilization was examined at this point [Fig. 10]. It was found that the most heavily used resources, look-up tables (LUT) and digital signal processing (DSP) blocks, were only at 41% and 26% respectively. These usage numbers mean that there are plenty of resources available to refine the system described above or to add additional hardware accelerator blocks to the FPGA.

III. Conclusions

A properly functioning image processing hardware accelerator was designed and built. This system demonstrated the versatility and performance capabilities of the RadPC SBC platform using custom hardware blocks and RadPC's custom memory mapped peripheral interface.

A. Future Work

While the system described in this paper meets basic image processing requirements of a simple arc, there are improvements that could be made to improve the overall effectiveness of the system. These ideas include the addition of a blurring preprocessing component, the detection and rendering of more complex horizon shapes, and adapting some components of this system so they can be swapped out using dynamic partial reconfiguration.

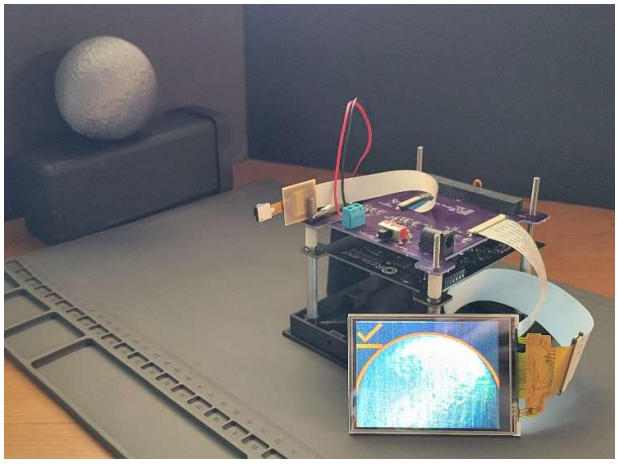
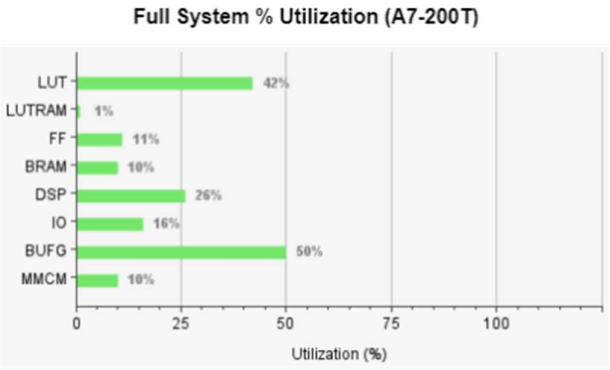


Fig.9. Completed system.



Full System Resource Utilization (A7-200T)

Resource	Utilization	Available	Utilization %
LUT	55849	134600	41.49
LUTRAM	6	46200	0.01
FF	28377	269200	10.54
BRAM	36.50	365	10.00
DSP	193	740	26.08
IO	47	285	16.49
MMCM	1	10	10.00

Fig.10. Nexys A7 200T FPGA resource utilization

A preprocessing blurring component was originally planned for this project, but due to time limitations, this component was never completed. Finishing the blurring component will provide better horizon detection and produce a more stable overlay of the horizon on the displayed image.

The final system delivered by our design team currently uses three points on the horizon to parameterize a circle for overlay on the displayed image. While this approach is effective in detecting roughly circular horizons, it does not accurately detect non-circular objects. By adjusting this approach to include multiple points on a horizon and connecting these points with lines, more detailed, complex edges could be accurately detected and overlayed on to the displayed image.

By using partial reconfiguration, FPGA, computing, and power resources can be conserved, improving the capability and flexibility of small satellites without requiring the size or power requirements to increase. The stand-alone hardware design and register interface of this project make it a good candidate for future partial reconfiguration research.

IV. Acknowledgements

This research was supported by NASA under award number 80NSSC23CA147 and by Resilient Computing, LLC under subcontract number 4W9082. Any opinions contained herein are those of the authors and do not necessarily reflect those of NASA or Resilient Computing, LLC.

References

[1] C. M. Major, A. Bachman, C. Barney, S. Tamke, and B. J. LaMeres, "Radpc: A novel single-event upset mitigation strategy for field programmable gate array-based space computing," Journal of Aerospace Information Systems, vol. 18, no. 5, pp. 280-288, 2021.

[2] J. Williams, C. Barney, Z. Becker, J. Davis, C. Major, B. J. LaMeres, and B. Whitaker, "Radpc@scale: A novel approach to radpc single event upset mitigation strategy," 2022 IEEE Aerospace Conference, 2022.

[3] "RESILIENT COMPUTING." Accessed: Feb. 08, 2025. [Online]. Available: https://resilient-computing.com/wp-content/uploads/2024/08/RadPC-SBC-001_Product_Overview.pdf

[4] "Blue Ghost Mission 1," *Firefly Aerospace*. https://fireflyspace.com/missions/blue-ghost-mission-1/