LiveCheckHSI: a Hardware/Software Co-verification Tool for Hyperspectral Imaging Systems with Embedded System-on-Chip Instrument Avionics

Irene Wang[1], Didier Keymeulen[2], Danny Tran[4], Elliott Liggett[2], Matthew Klimesh[2], David Dolman[3], Daniel Nunes[2], Peter Sullivan[2], Michael Bernas[2], Michael Pham[5]
[1]California Institute of Technology, Pasadena, CA
[2]Jet Propulsion Laboratory, California Institute of Technology,4800 Oak Grove Dr., Pasadena, CA
[3]Alpha Data Inc., Denver CO
[4]University of California, Irvine, CA
[5]University of California, Los Angeles, CA
siwang@caltech.edu

***Abstract* — The emergent technology of system-on-chip (SoC) devices promises lighter, smaller, cheaper, and more capable and reliable space electronic systems that could help to unveil some of the most treasured secrets in our universe. This technology is an improvement over the technology that is currently used in space applications, which lags behind state-of-the-art commercial-off-the-shelf (COTS) equipment by several generations. SoC technology integrates all computational power required by next-generation space exploration science instruments onto a single chip. This paper describes hardware/software co-verification tools for the Xilinx Zynq-based control and data handling system that have been developed at the Jet Propulsion Laboratory (JPL) for visible-infrared imaging spectrometers. The system acquires and compresses images in real-time, in addition to programming the spectrometer (frame rate, exposure time), focus step motor, and heaters and reporting telemetry.**

TABLE OF CONTENTS

1. INTRODUCTION 1
2. SOC INSTRUMENT AVIONICS 2
3. ON-LINE DATA VERIFICATION 3
4. OFF-LINE DATA VERIFICATION 5
5. SUMMARY 9
ACKNOWLEDGEMENTS 9
REFERENCES 9
BIOGRAPHY 10

1. INTRODUCTION

Hyperspectral images are three-dimensional data sets, where two of the dimensions are spatial and the third is spectral (Figure 1). They can be regarded as a stack of individual images that represent the same spatial scene viewed in a distinct, narrow part of the electromagnetic spectrum. The Airborne Visible/Infrared Imaging Spectrometer-Next Generation (AVIRIS-NG [16]) instrument is an example of an instrument that can acquire such imagery. Hyperspectral imagery can also be acquired from spacecraft. Because one hyperspectral image can be comprised of hundreds of spectral bands, it can represent a large volume of data. Future spacecraft that acquire hyperspectral imagery [26] may have modest bandwidths available to transfer the data to the ground, and there is much interest in compressing the data efficiently [1][2][3].

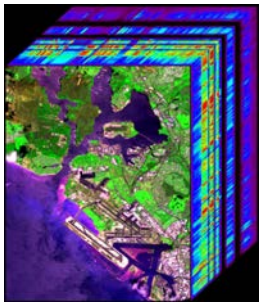

Figure 1. An example of a hyperspectral data cube: Pearl Harbor, Hawaii, taken by the AVIRIS instrument

The JPL-developed Fast Lossless hyperspectral compressor [23] is a low complexity compression technique. The compression effectiveness is derived from an adaptive filtering method. The technique is currently implemented on both software and hardware platforms for various space applications. In particular, the algorithm has been programmed on a Field Programmable Gate Array (FPGA), which is a hardware platform that can be used in space for data processing. The FPGA implementation is capable of compressing data in real-time as it is acquired.

Hybrid System-on-Chip (SoC) devices that embed one of the world's most energy efficient processor (ARM Cortex-A9 [8]) and the latest and most powerful FPGA architecture (Xilinx 7-series [9]) into a single chip (Xilinx Zynq [10]) promise new opportunities due to the performance, power consumption, weight and volume benefits they bring. Indeed, programmable logic and processors are usually combined in most spacecraft

978-1-5386-2014-4/18/$31.00 ©2018 IEEE

subsystems as separate components distributed along one or several circuit boards [11][12]. NASA and other space agencies are considering SoC technology for its high computation capabilities and power efficiency, hoping to pave the way for future space exploration missions that are becoming so performance-demanding that currently available space-grade technology (e.g., RAD750 [13]) cannot meet their needs[24]. Despite the fact that currently there are no space-qualified SoC parts, NASA is testing commercial Xilinx Zynq SoC devices in the International Space Station (ISS) as well as in precursor CubeSats operating in Low Earth Orbit (LEO), where the exposure to radiation is limited [19][20][21][28].

The complexity of hardware, software and HW/SW integration that arises from the convergence of so much functionality in such small hybrid SoC devices has driven both hardware and software innovation at almost break-neck speed, while the development methodology that brings hardware and software together lags behind. Sequential development, with software development waiting for available hardware, is still the prevailing norm. But sequential development often fails to deliver quality products within the short windows that rule the faster, cheaper and robust space missions today [7]. A more robust systems engineering approach is to have both HW/SW co-developed and genetically linked. To help in the parallel agile development of hardware and software for SoC technology [14], hardware/software co-verification tools have been developed for the hyperspectral imagers. These tools make sure that embedded system software works correctly with the hardware, and that the hardware has been properly designed to run the software successfully before deploying the image spectrometers in expensive space or airborne missions [5]. One of the co-verification tool is LiveView [4]. It was initially designed for real-time calibration of focal plane arrays. It is performing real-time analysis of images acquired through the Camera Link interface of the hyperspectral camera. This calibration analysis is used to identify sources of electronic noise and interference in the imaging spectrometer and to characterize the FPA [15]. In the co-verification process, it is detecting errors in the firmware or software implementations.

This paper presents two techniques of hardware/software co-verification for instrument avionics for hyperspectral spectral imagers: on-line and off-line. Section 2 presents the SoC for imagers. Section 3 presents the on-line verification tool. The tool controls the instrument to perform all the required functionalities and recognizes and handles unexpected behavior. It is used as ground support equipment (GSE) connected to the spectrometer and its avionics. Section 4 presents the off-line verification tool. It is parsing the data acquired by the spectrometer. Its goal is to guarantee that the raw data are consistent to be processed for calibration and analysis data processing.

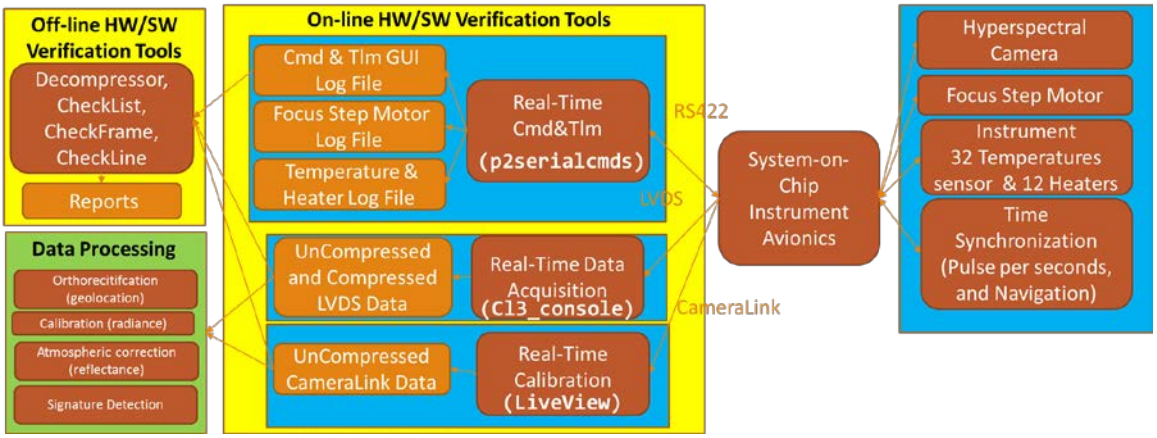

Figure 2. System-on-Chip instrument avionics architecture for hyperspectral imagers

2. SoC Instrument Avionics

The SoC instrument avionics performs data acquisition, cloud-screening and compression computing system for hyperspectral imagers (Figure 2). It is implemented on the Xilinx Zynq-based custom Alpha Data hardware assembly which fits into a 120mm by 190m by 40mm assembly and uses 9 watts at peak performance (Figure 3) [25]. The computing element is a Xilinx Zynq Z7045Q which

Figure 3. System-on-Chip Instrument Avionics for Hyperspectral Imaging Systems

includes a Kintex-7 FPGA and dual-core ARM Cortex-A9 Processors.

Hyperspectral images are acquired, screened for atmospheric clouds [27], and compressed (either losslessly or lossily) in real-time using JPL's "Fast Lossless Extended" (FLEX) compressor [22], implemented into the Programmable Logic (PL) of the Zynq SoC, and sent to on-line HW/SW verification tools either through Camera Link (LiveView[4]) or LVDS protocol (cl3_console). In addition the PL interfaces with Focus Step motor, 32 temperature sensors, 12 heaters and an IMU/GPS device providing inertial/position information and time synchronization with GPS or UTC time. The processing system (PS) of the Zynq SoC implements Command and Data Handling to program the Hyperspectral Camera (frame rate, exposure time), acquires telemetry (temperature, pulse-per-seconds counts, frames count), control heaters, focus motor and data flow inside the PL. The PS interfaces with the Real-Time Cmd&Tlm (p2serialcmds) performing on-line HW/SW verification.

The on-line HW/SW co-verification tool allows the parallel agile development of the SoC platform while the off-line tool provides a method to verify the integrity of the ancillary data included in each image.

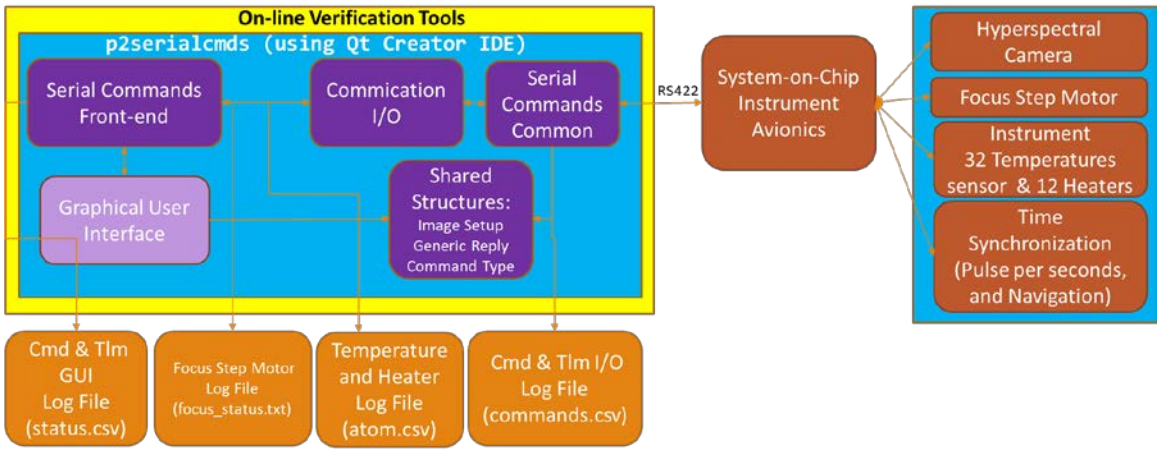

Figure 4. Software Architecture of the on-line Verification tools for SoC instrument avionics

3. On-line Data Verification

The on-line co-verification tool is implemented by 3 pieces of software running on the GSE Linux machine: `p2serialcmds`, `cl3_console` and `LiveView` (Figure 4). The `cl3_console` acquires real-time compressed and non-compressed frames which are verified by the off-line verification tool. The `LiveView` tool detects errors in the non-compressed image acquisition of the SoC such as flickering of pixels (Figure 5). The `p2serialcmds` tool programs the instrument and the SoC in different verification and operation modes by sending single commands sequentially to the SoC instrument avionics (Figure 6). For example, it can program the image to be a synthetic pattern generated by either SoC or the focal plane interface electronics (FPIE) or the focal plane array (FPA) of the hyperspectral camera. It can also program the cloud screening parameters, the compression parameters, line rate, the FPA, the heaters and the focus step motor and reports telemetry on their status. Finally it allows commanding the acquisition of compressed and non-compressed frames.

These on-line co-verification tools detect and identify faults in either the software, FPGA firmware, hardware interface or peripheral devices in real-time and locate the modules responsible for the errors.

Software Architecture

The graphical user interface (GUI) is connected to the instrument avionics via the serial command front-end. Each graphic component that makes up the GUI has a corresponding listener method that receives the events and responds to them in the serial commands front-end. The listener methods do so by building an image setup structure with the parameters based on input through the GUI. Possible request to the GUI consists of the following: acquiring the states of the various parameters of the instrument avionics, querying the instrument avionics registers, cloud screening results of images and thresholds parameters, cloudy pixels values, cloudy cross-track pixels, and timestamps for the aforementioned pixels. The various parameters of the instrument avionics include the number of least significant bit drop, the number Pulse per second (PPS), the PPS captured timestamp, and GPS time. Each spectrometer command is represented in the software as an enumerated type, which is a data type that consists of a set of named elements. The enumerated types are used to identify the custom command's values that are sent as a query to the spectrometer avionics.

Qt Creator operates on a framework where each object on the GUI is paired with a function in the code. This model is called "signal and slot", where a specified GUI object "signals" a particular function, called a "slot", to be triggered in response. User-input data introduced by the GUI is received and parsed via "signal and slot." A "signal"

can trigger one or more serial commands. The data is incorporated into a reply structure that the `serial commands common` and `communication I/O` can process.

The user-entered data from the GUI is passed to the `serial commands common` by building a generic reply structure with a command type, which is comprised of either an instrument command, program command or custom command. The command is then parsed in the

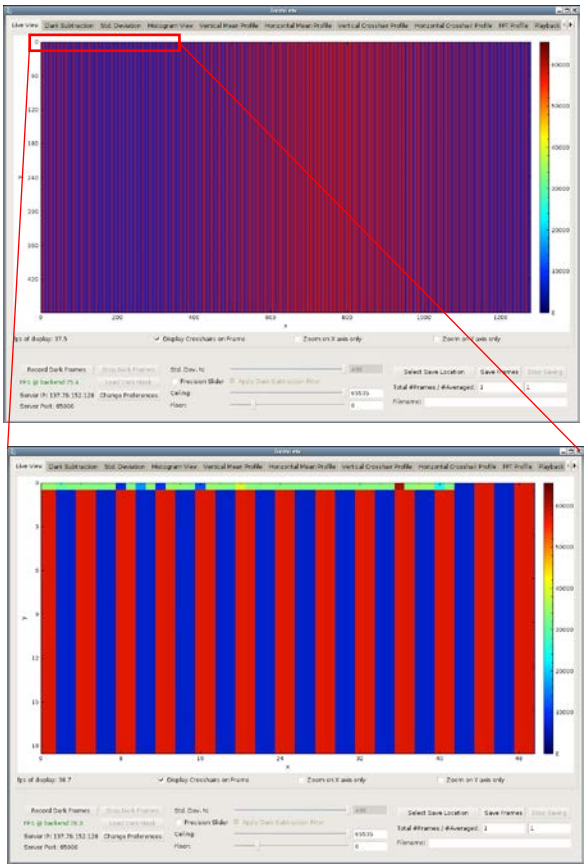

Figure 5: Real-time visualization of image (top) and ancillary data in first band (bottom) using the LiveView on-line Verification tool for SoC instrument avionics.

`serial commands common` after being passed to the `communication I/O` in the form of a signal. The commands are processed in the `serial commands common`, which then queues up the commands and sends them to the spectrometer avionics. The `serial commands common` receives replies from the SoC instrument avionics and passes said data to the communication port in the `communication I/O`. Any data received from the instrument avionics is parsed and copied by the `communication I/O` into another generic reply structure and thrown via "signal and slot" to the `serial commands front-end`.

Qt and Qt Creator

The p2serialcmds software was built using the cross-platform application framework, Qt5. The open-source graphical user interface library was selected to leverage Qt

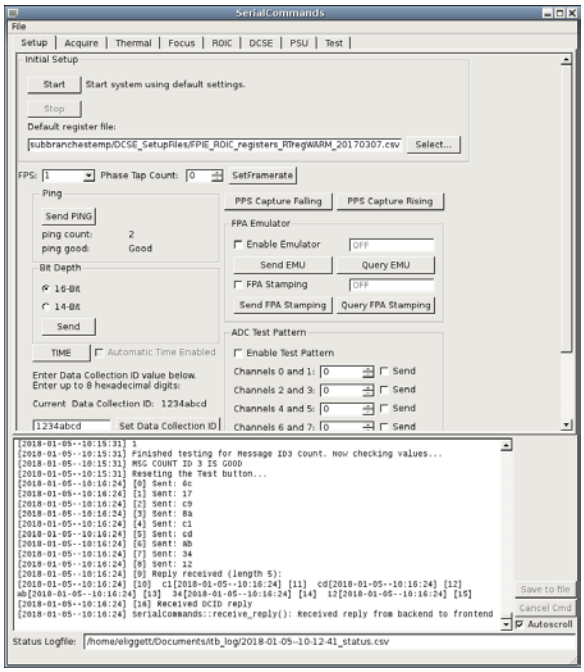

Figure 6: Graphical User Interface of the p2serialcmds on-line Verification tool for SoC instrument avionics

Creator, the C++ integrated development environment that is part of the software development kit for the Qt graphical user interface application development framework. The library enables immediate interface creation and connection of said graphics to the back-end. Qt Creator is integrated with version control systems used in the project like Git and includes a debugger plugin to connect the Qt Creator core and external debugger to debug the C++ code.

Results

A graphical window running on the GSE Linux computer displays results received from the spectrometer instrument avionics (Figure 6). The primary tab of the user interface enables the initial setup of the spectrometer to its default settings and include basic functions such as sending pings to check serial communication, fixing bit depth of the image samples to either 14-bit or 16-bit to reduce the sample noise, and setting PPS to be captured at its rising or falling edge depending on the GPS/IMU device. The FPA emulator section of the GUI allows the user to turn on or turn off an emulation of the focal plane array inside the instrument avionics. It is used upon power-on of the instrument avionics to confirm that the image data communication through low voltage differential signal (LVDS) or camera link is working properly.

Two log files are updated in real-time while the p2serialcmds software is being used. The user specifies the

status log file and command log file. For every signal sent by the GUI, status text is updated in real time on the GUI as well as written to both the status log file and the command log file. On exit from the software, both log files are closed and saved in a specified directory.

Several tabs focus on different sections of the spectrometer. The ADC tab deals with the analog-to-digital converter (ADC) registers and sends either test patterns or custom values to the 8 channels. Each channel needs to be reset before being programmed and includes error-checking for valid input from between 0 to 65535 (2^{16}). Current temperature data can be acquired in a separate tab and the heaters can be set to warm or cool the spectrometer. The set-up temperature of the heaters can then be sent to the spectrometer and the temperature of 32 sensors can be queried from the spectrometer. Another section is reserved for focal plane array (FPA) register values. FPA register files, which include register settings, are selected and sent to the spectrometer. FPA timing values like frequency index and integration duration can be retrieved and set with built in error-checking that inputs are within a valid range. For the FPA, the user can also power on or off the FPA on the power supply unit (PSU). Manipulation of spectrometer registers is implemented on a separate tab and allows for queries and displays for various parameters and cloud screening outputs.

A focus step motor log file is generated each time the focus step motor state is queried. The GUI shows the new log position based on the current absolute position of the step motor, the user chosen the number of half step and the direction to be executed by the step-motor. The GUI includes functionalities for changing log file extensions, rerouting default files and directories, and reorganizing the order of procedures. When a focus step motor query is made, the log file position is shifted by the accumulated half steps executed by the focus step motor in the chosen motor direction. During each query, a validation is made to ensure that the projected position is within the mechanical limits of the focus step motor, which is user chosen or a default of 4000 half steps. When a focus step motor query is received by the p2serialcmds software, the current focus step motor position is written to the focus log file along with the current phase of the step motor position. All parameters of the focus query are also displayed to the graphical user interface, including the next and current phase, accumulated half steps, acceleration, velocity, and current motor status and status time stamp. The corresponding focus log file is stored in a separate folder and is automatically saved on exit of the software. The log file is re-opened on reboot of the p2serialcmds software and the current absolute position is retrieved on valid reading of the log file.

On initial use of the p2serialcmds, it is important to verify that the actions of the software are being sent to and retrieved from the spectrometer correctly. Corresponding verification tests are included to test some basic operations. One such test verifies that "Synchronization Time" messages are being retrieved from the spectrometer correctly. To do this, a pre-determined number of "Synchronization Time" messages are sent to the instrument avionics. If the avionics is working as expected, it counts the number of "Synchronization Time" messages received. p2serialcmds is computes the difference between the initial count and final count. Each instance a time is queried, a "Synchronization Time Message" is sent, so the count is correspondingly incremented, which means that the initial and final count should differ by the pre-determined value. A command is sent to request various parameters, one of which includes the "Synchronization Time Message." When the final message count is received by the p2serialcmds, this difference is compared, and the verification test then displays if the messages and time queries are being transmitted successfully. If no response is received by the p2serialcmds within a pre-determined interval of time, p2Serialcmds is reporting the failure to command the instrument avionics.

Figure 7. Thumbnail of science portion of a flight line

4. OFF-LINE DATA VERIFICATION

The off-line verification is done following the acquisition of a flight line (Figure 7), while in flight or on the ground.

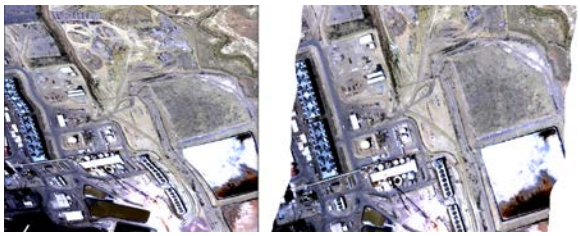

Figure 8. A flight-line after orthorectification post-processing with incorrect ancillary data on the left showing distorted roads and buildings compared to a correctly orthorectified image on the right

It can also be run on a multitude of flight lines. The verification accounts for multiple formats of data. For example, it was applied to data generated by 3 different airborne spectrometers flying on TwinOtter, King Air B200, ER2: AVIRIS-NG [16], HyTES [18], and PRISM

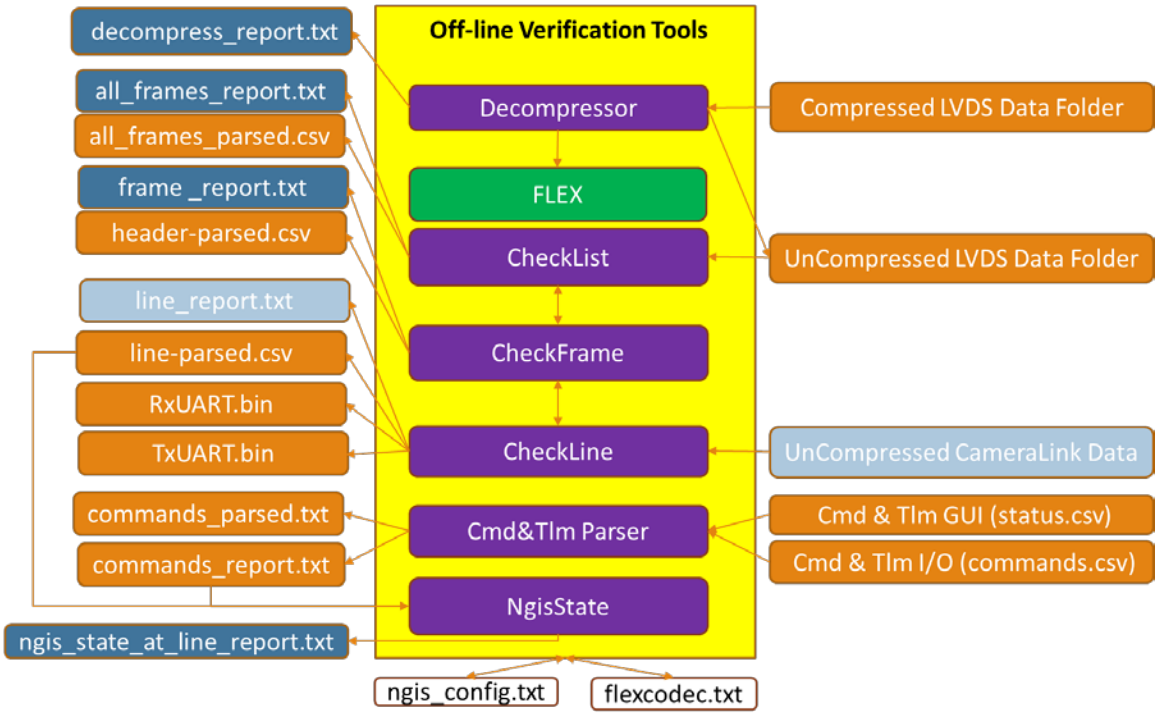

Figure 9. Software Architecture of off-line Verification Tools for SoC instrument avionics

[17]. It allows detection of ancillary data with incorrect timing prior applying intensive post-data processing such as orthorectification (Figure 8). It also allows reporting performance such as compression ratio and can detect multiple other issues with the hyperspectral camera and the SoC instrument avionics such as decompression errors, missing lines, missing frames, wrong frames rate as explained in the next sections.

Data Format

The acquisition of images is performed through multiple acquisition of frames. Each frame has a header part and a data part which includes 32 lines. Each line is an image with 640 cross-track pixels by 480 bands which is programmable. The 32 lines of each frame can be either compressed or non-compressed. Each frame has a fixed size non-compressed header. Each line has ancillary data in its first band (Figure 5). The frame and line headers contain information used by the verification tools and describing the state of the instrument during the acquisition of the frame and the line. For example, the header includes frame size, frame count, data frame identifier (start/acquire/stop flag), data collection identifier, the line count of the first line in the frame, the time stamp of the first line in the frame, the UTC time of the acquisition of the first line in the frame, the number of cloud pixels in the frame, temperature of the Focal Plane interface electronics, the position of the step motor during the acquisition of the frame. The line header contains the line count, the time stamp of the lines, the number of cloud pixels in the line, the line frequency and the clock programming parameters, the UTC time at the time of acquisition of the line.

Software Architecture

The frames recorded during multiple acquisition by SoC instrument avionics commanded by the Command and Telemetry Real-time control are saved in a single directory. If the data directory contains compressed raw frames, the Python decompression script needs to be used to decompress the frames before using other verification, validation and diagnostic scripts, which are comprised of three software: `CheckList`, `CheckFrame`, and `CheckLine` (Figure 9). The decompression script runs the Flex Codec decompressor software on all the compressed frame files inside a given directory. It first identifies all of the frames files and then parses out the data collection start/acquire/stop identifier value in the frame header of each image file. The data collection start/acquire/stop identifier is set to one of three values: 0, 1, or 2. An ID of 1 indicates that the current frame is the first of an acquisition, an ID of 2 indicates the frame is the final frame in an acquisition, and an ID of 0 indicates the current frame is neither the first nor final of an acquisition. If a frame has a data collection value of 2; the frame is simply copied and given the extension *.decomp*. Otherwise, the decompressor software is applied. At the end, the script generates a report containing a list of frames already decompressed. The `CheckList` software loops through all the frames sequentially inside a given directory under the assumption that all the frames have first been decompressed. At each iteration, `CheckList` performs the `CheckFrame` software on a single frame.

In the CheckFrame software, fields of both the frame header and the line headers of the raw image files are parsed. Each frame's fields are verified for correctness, and test results are written to both a text file and csv file. The data for each frame is split into acquisitions determined by using the data collection start/acquire/stop identifier values. After the loop on the frames acquired ends, the CheckList software generates the "all_frame_report" file containing checks at the all frames level.

The "all_frames_report" reports first the number of acquisitions found and the start and final frame of each acquisition. If a start or final frame is labeled "MISSING", the user will know to begin debugging here and can assume that the rest of the report is invalid. The remainder of the report contains checks for multiple acquisition of frames by the instrument avionics.

```
======================================================================
FRAME ngis_check_frame SOFTWARE
          Repository: https://github.jpl.nasa.gov/aviris/NGIS_Check_Line_Frame.git
          Commit used for processing:
                    Revision: e3a6425b54c015175c3504f0b79f0c635a1ba044
                    Date/Time: 2017-12-19 16:10:45
======================================================================
FRAME processing: image_116.640.15361.L16.xio
TIME processing: 2017-12-29 16:26:59
REPORT OUTPUT name: image_116.640.15361.L16.xio_20171229_framereport.txt
======================================================================
FRAME SUMMARY
======================================================================
Bytes Count:
          >> Bytes in Files: 19662080
          -> Check:  PASS
                    Expected: (640x480x32x2 + 640x2) = 19662080

Dynamic Range:
          Turned OFF

Test 1: Missing Lines in a Frame
          >> Number of Lines = 32
          -> Check: PASS
                    Expected: 32

Test 2: Line Frequency Check
          >> Difference between last and first line Time Stamp: 0.51292 sec
          -> Derived Data:
                    Estimated Lines Rate from table (same for all lines): 60.4562236288 lines/sec
                    Lines Rate using Time Stamp (1 / (Time Stamp / 32)): 62.387896748 lines/sec
                    Margin of Error: 5.0%
          -> Check: PASS
                    Actual Lines Rate ~= Estimated Lines Rate
          -> Note:
                    NGIS Project: L1-NGIS-11; 4.1.9; Sample rate; The sample rate shall be selectable over the range of 5 to
125 samples per second.

Test 5: Data Translation Error in Header
          >> Frame Header Checksum: 2720865089
          -> Check: PASS
                    Computed Checksum: 2720865089

Test 6: Cloud Screening Test
          >> Cloud Screening Cross-Track Pixels Count in Frame for lines 0-30: 0
          -> Check:
                    Computed the Cloud Screening for each of the 31 Lines (see report in LINE PARSING): All Line PASS
                    Computed sum of Clouds Screening Bytes for lines 1 to 31 = 0: PASS
          -> Derived Data:
                    Cloud Screening Frame Threshold: 640
                              FRAMES_NOT_CLOUDY
                    Cloud Screening Register Values
                              Time Stamp: N/A, values are taken from ngis_config.txt
                                        C1: 16
                                        C2: 32
                                        C3: 64
                                        C4: 128
                                        C5: 256
                                        T1: 0
                                        T2: 0
                                        T3: 0
                                        T4: 0
                                        T5: 0
                                        TB: 640
          -> Note:
                    NGIS  Project:  L1-NGIS-5;  4.1.5;  Cloud  detection;  The  instrument  shall  implement  a  cloud-detection
algorithm.
```

Figure 10: Frame Report generated by off-line Verification Tool

The CheckFrame software parses the header of a frame and calls the CheckLine software to parse the lines of the frame. Both software generate their own report text file along with a csv file with all of the parsed data. The CheckFrame software then combines the two text files into one report called "frame_report".

The CheckLine software can also be used independently to parse a non-compressed data file acquired through the LiveView [4]. It will generate a report text file

"line_report.txt" and a csv file "line_parsed.csv" in the same format as the ones for the LVDS data. Additionally, for all lines with ancillary data including a copy all navigation data sent by the GPS/IMU device through the Rx UART , the software will concatenate the Rx UART data over multiple lines and deliver the stored concatenation as a binary file "Rx_UART.bin" at the end. The same process is done for all the commands sent to the GPS/IMU devise through Tx UART data. These binary files can be parsed to extract location (longitude, latitude, altitude) and inertial information (velocity, acceleration, roll, pitch) of the instrument.

The CheckList, CheckFrame, and CheckLine scripts uses a configuration file ("ngis_config.txt"). This file allows customization of the checking (algorithm and error margin) for different instrument avionics hardware and mode of operation (with or without IMU/GPS device, in the air or in the ground, image size, cloud screening parameters). The Decompressor script utilizes its own configurable text file, "flexcodec.txt" that allows the user to pass commands option to the Flex decompressor software.

Results

Each script provides a suite of verification. Example of these checks are described below:

- *All Frame Level*: these checks are done over multiple acquisition of frames by the instrument avionics.
  - *Missing Frames in an Acquisition* – Checks for no missing frame by verifying that the frame count is incrementing by 1 for all the frames inside an acquisition. Also, reports if all the *Missing Lines in a Frame* checks at the *Frame Level* are passing for the acquisition.
  - *Pulse Per Second* – Checks for no missing detection of PPS by verifying that the PPS counts are incrementing by 1 at every second. In addition, checks that the PPS time stamps generated by the instrument avionics is incremented by 100,000 Clock ticks (Clock tick is 100kHz) at every second, with margin of error taken into account.
  - *Compression Ratio* – Reports the average compression ratio of all the frames in the data directory, and the average compression ratio for each acquisition.
  - *End of Acquisition* – Check if the data acquisition is terminated correctly by verifying that the frame header of the final frame of an acquisition is the same as that of the frame before it.
  - *Summary Frame level Checks*: Reports the results of the *Line Frequency* and *Cloud screening* checks performs at the frame level
- *Frame Level*: these checks are done over a single frame acquisition. Each frame contains 32 lines (See Figure 10).
  - *Missing Lines in a Frame* – Checks that all 32 lines are in the frame by verifying that the line count is incremented by one.
  - *Data Transmission Error in Frame Header* – Checks that the computed checksum of the data header is equal to the parsed frame header checksum ancillary data.
  - *Cloud Screening* – Checks that the computed sum of clouds screening ancillary data for lines 1 to 31 is equal to the parsed frame header clouds screening ancillary data. Also reports the result of the *Cloud Screening Test* at the *Line Level* (see below).
  - *Summary Line level Checks:* Reports the results of the *Line Frequency* checks performs at the line level
- *Line Level:* these checks are done over acquisition of multiple lines
  - *Ancillary Data* – Checks if the ancillary data is formatted correctly, quickly allowing the user to know if the rest of the report file is accurate.
  - *Summary Line level checks* – Reports if all the checks at line level are passing and if it fails, report the location of where fails are at. It provides to the user a quick diagnostic of the instrument avionics.
  - *Line Frequency* – Checks that the actual lines rate is equivalent to the estimated lines rate provided by the parsed line header line frequency ancillary data.
  - *Bytes Count* – Check that the data file size is equal to the expected file size based on the number of lines acquired as reported in the parsed line header line count ancillary data.
  - *PPS, Time Message and Line instrument avionics Timestamps:* The time stamps associated with the PPS, the line and the Time message are tested. The timestamp, generated by an instrument avionics 100kHz Clock, provides a synchronization mechanism for all events captured by the instrument avionics with an external time (such as GPS or UTC time) through the Pulse per Seconds signal.
  - *Cloud Screening* - Checks that the computed cloud screening matches the parsed clouds screening ancillary data for each of the 31 lines. The parameters used by the clouds screening algorithm are either extracted from the clouds screening parameters sent to the instrument avionics (reported in the "status.csv" file) or, if not provided, the default values extracted from the configuration file "ngis_config.txt" .
  - *Data Transmission Error in Line Ancillary Data*–Checks for Time, Rx UART, Tx UART messages with data checksum that the computed

checksum is equal to the parsed checksum ancillary data.

In addition to checking the images recorded, the verification tool also correlates the commands and telemetry issued by the Instrument Avionics with the images recorded. First, `ngis_cmdstlm_parser` parses the commands and telemetry file, which contains all the bytes sent and received to and from the Instrument Avionics. If the log file of the Command and Telemetry graphical user interface (GUI) is provided as input to the script, it will associate the user configuration filename with the commands inputs. The script output are "`commands_parsed.txt`" file with the parsed commands based on a dictionary and "`commands_report.txt`" file correlating the commands with GUI events.

Finally the `Ngis_State` script retrieves the state of the Instrument Avionics at a specified line number using the "`commands_report.txt`" is in conjunction with the "`line_report.txt`". The `Ngis_State` script sorts the commands in the "`commands_report.txt`" with time stamp. Same commands are grouped together. The script extracts the time stamp of the specified line number from the "`line_report.txt`". Finally, for each group of the same command, the script searches the command with the latest timestamp equal to or before the line's time stamp. If found, then the script reports the command at that time instance. If there are none, then for that group, the script reports that the logging of the commands started after the acquisition of this line.

5. SUMMARY

We have developed hardware/software co-verification tool for hyperspectral imagers SoC instrument avionics. The on-line HW/SW co-verification tool allows the parallel agile development of SoC platform while the off-line tool provides a verification method of the integrity of the ancillary data includes in each images prior intensive post-data processing such as orthorectification. These on-line co-verification tools detect and identify faults in either the Software, FPGA firmware, Hardware interface or peripheral devices in real-time and help the developer to locate the modules responsible for the errors. In the future, these tools will be port into the instrument avionics itself on the next generation Multi Processor System-on-Chip (MPSoC).

ACKNOWLEDGEMENTS

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The authors wish to thank the Caltech Summer Undergraduate Research Fellowship's (SURF) program and the JPL Year-Round Internship Program (JPLYIP) program for the funding necessary to carry out the implementation of the validation, verification, and diagnostic software.

REFERENCES

[1] D. Keymeulen, H. Luong, T. Pham, A. Kiely, M. Klimesh, M. Cheng, D. Dolman, C. Holyoake, K. Crocker, "Hardware implementation of Lossless and Lossy Compression of Space-based Multispectral and Hyperspectral Imagery," in *Proceedings of 2016 HyspIRI Science and Applications Workshop*, 18 - 20 October 2016, California Institute of Technology, Pasadena, CA.

[2] Consultative Committee for Space Data Systems (CCSDS), *Lossless Data Compression*, Recommendation for space data system standards vol. 121.0-B-1: CCSDS, 1997. (http://public.ccsds.org)

[3] C. Hartzell, L. Graham, T. Tao, H. Goldberg, J. Carpena-Nunez, D. Racek, C. Taylor, C. Norton, "Data System Design for a Hyperspectral Imaging Mission Concept," in *Proceedings of IEEE Aerospace Conference* 2009.

[4] N. Levy, J.P. Ryan, Elliott H. Liggett, "LiveView: A new utility for Real-Time Calibration of Focal Plane Arrays using Commodity Hardware," in *Proceedings of IEEE Aerospace Conference* 2016.

[5] "IDE Overview." *Qt Creator Manual*, The Qt Company, 2017, https://doc.qt.io/qtcreator/

[6] J. Andrews, *Co-verification of Hardware and Software for ARM SoC Design*, 2004, Elsevier.

[7] T. De Schutter, "The Power of Developing Hardware and Software in Parallel," In *Design & reuse, EE Times*, April 29, 2013.

[8] E. Blem, J. Menon, and K. Sankaralingam, *Detailed Analysis of Contemporary ARM and x86 Architectures*, Technical report, University of Wisconsin - Madison, 2013.

[9] N. Mehta, *Xilinx 7 Series FPGAs: The Logical Advantage*, Xilinx WP405, 2012.

[10] Xilinx Inc., *Zynq-7000 All Programmable SoC: Technical Reference Manual*, UG585, 2015.

[11] R. C. Wiens et al., "The ChemCam Instrument Suite on the Mars Science Laboratory (MSL) Rover: Body Unit and Combined System Tests," *Space Science Reviews*, pp. 167-227, Springer, 2012.

[12] D. Petrick, N. Gill, M. Hassouneh, R. Stone, L. Winternitz, L. Thomas, M. Davis, P. Sparacino and T. Flatley, "Adapting the SpaceCube v2.0 Data Processing System for Mission-Unique Application Requirements," in *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems* (AHS'15), 2015.

[13] BAE Systems Plc., *RAD750 Radiation-Hardened PowerPC Microprocessor*, 2008.

[14] Ken Beck et al. , “Manifesto for Agile Software Development” (http://agilemanifesto.org/)

[15] Peter Sullivan, Michael Bernas, Elliott Liggett, Michael Eastwood, Robert Green, “Characterization of the Teledyne CHROMA HgCdTe Detector for Imaging Spectrometers,” in *Proceedings of IEEE aerospace conference*, MT, USA, March 2017.

[16] Airborne Visible/Infrared Imaging Spectrometer-Next Generation (AVIRISng), http://avirisng.jpl.nasa.gov/

[17] Portable remote imaging spectrometer (PRISM), http://prism.jpl.nasa.gov

[18] Hyperspectral Thermal Emission Spectrometer (HyTES), https://airbornescience.jpl.nasa.gov/instruments/hytes

[19] D. Mandl, “Intelligent Payload Module Update,” in *Proc. of the HyspIRI Symposium*, 2015.

[20] M. Amrbar, F. Irom, S. M. Guertin and G. Allen, “Heavy Ion Single Event Effect Measurements of Xilinx Zynq-7000 FPGA,” in *Proc. of the Radiation Effects Data Workshop* (REDW'15), 2015.

[21] M. Wirthlin, “Neutron Radiation Test Results of the Linux Operating System Executing within the CHREC Space Processor (CSP),” in *Proc. of the Military and Aerospace Programmable Logic Device International Conference* (MAPLD'15), 2015.

[22] A. Kiely, M. Klimesh, N. Aranki, M. Burl, M. Cheng, S. Dolinar, D. Dolman, M. Gilbert, G. Flesch, D. Keymeulen, M. Le, J. Ligo, H. Luong, T. Pham, F. Sala, D. Thompson, W. Wu, H. Xie, and H. Zhou, “Multispectral & Hyperspectral Image Compression Development at the Jet Propulsion Laboratory,” presented at the 2016 Onboard Payload Data Compression Workshop (OBPDC16), 2016.

[23] Consultative Committee for Space Data Systems (CCSDS), *Lossless Multispectral & Hyperspectral Image Compression*, Recommendation for Space Data Systems Standards, CCSDS 123.0-B-1, May 2012.

[24] G. Flesh, D. Keymeulen, D. Dolman, C. Holyoake, D. McKee, “A System-on-Chip Platform for Earth and Planetary Laser Spectrometers”. In *Proceedings of IEEE Aerospace Conference, March 2017, MT, USA.*

[25] *ADM-XRC-7Z1 User Manual*, Revision V2.2, Alpha Data Parallel Systems, Denver, CO, 2014 (https://www.alpha-data.com/esp/)

[26] Committee on the Decadal Survey for Earth Science and Applications from Space, Space Studies Board, “Thriving on Our Changing Planet:A Decadal Strategy for Earth Observation from Space”, The National Academies Press, 2018.

[27] Brian D. Bue, David R. Thompson, Michael Eastwood, Robert O. Green, Bo-Cai Gao, Didier Keymeulen, Charles M. Sarture, Alan S. Mazer, and Huy H. Luong “Real time Atmospheric Correction of AVIRIS-NG Imagery”, Journal of IEEE Transaction on Geoscience and Remote Sensing, IEEE, December 2015.

[28] Fernanda Lima Kastensmidt,. Lucas Tambara, Eduardo Chielle, Jorge Tonfat and André Flores, “Using Programmable System on Chip for Aerospace Applications” in Single Event Effects (SEE) Symposium /Military and Aerospace Programmable Logic Devices (MAPLD) workshop, La Jolla, CA 2016 (https://www.seemapld.org/2016/)

BIOGRAPHY

Irene Wang *is a current senior at the California Institute of Technology and will graduate in 2018 with a B.S. in Computer Science. Beside her academic career, she was a fellow of the 2017 Caltech Summer Undergraduate Research Fellowship (SURF) and is a fellow of the JPL Year Around internship performing research in the JPL adaptive embedded systems Lab that leads to the development of high performant computing, intelligent, autonomous and adaptive instrument avionics technology for hyperspectral imagers based on System-on-Chip (SoC).*

Danny Tran *is currently a sophomore at the University of California, Irvine and will graduate in 2020 with a B.S. in Computer Science and Engineering. He began his fellowship at JPL under the JPL Year Around internship back in August 2017 in the adaptive embedded systems Lab under the guidance of Didier Keymeulen. During his time at JPL, he has developed software to process, verify and validate data produced by hyperspectral imagers instrument avionics based on System-on-Chip (SoC).*

Didier Keymeulen *received the BSEE, MSEE and Ph.D. in Electrical Engineering and Computer Science from the Free University of Brussels, Belgium in 1994. In 1996 he joined the computer science division of the Japanese National Electrotechnical Laboratory as senior researcher. Currently he is principal member of the technical staff of JPL in the Flight Electronics Section. At JPL, he is responsible for DoD and NASA applications on evolvable hardware for adaptive computing that leads to the development of intelligent, autonomous and adaptive instrument avionics technology. He participated also as test electronics lead, to Tunable Laser Spectrum instrument on Mars Science Laboratory. He served as the chair, co-chair, and program-chair of the NASA/ESA Conference on Adaptive Hardware.*

Elliott Liggett *is an Analog Circuits and DSP engineer specializing in Imaging Spectroscopy applications at the Jet Propulsion Laboratory. An active ham radio operator since he was a teenager, Elliott has an extensive background in hands-on analog design. Before receiving his bachelors in EE at the University of Arizona in 2012, he worked as an independent consultant in the music production industry, designing equipment and software. At the University of Arizona, he worked under Dr. Chris Walker in the Steward Observatory Radio Astronomy Lab on SuperCam, a 64 pixel superconductor THz receiver.*

Matthew Klimesh *received B.S.E., M.S.E., and Ph.D. degrees, all in electrical engineering from the University of Michigan in Ann Arbor, in 1989, 1990, and 1995, respectively. He spent one year as a research fellow (postdoc) at Michigan. Since 1996 he has been with the Information Processing Group at Caltech's Jet Propulsion Laboratory, working primarily on research and development of data compression algorithms for space applications. He is a primary developer of the Fast Lossless and FLEX algorithms for compression of hyperspectral imagery and co-developer of the ICER image compression algorithm and software that has been used on multiple space missions including the Mars Exploration Rovers.*

David Dolman *David Dolman MEng. MIET, graduated from Edinburgh University UK. He has spent 10 years working with Alpha Data Parallel System Ltd. in Scotland in the fields of board design, testing, software, drivers, firmware, and RTL/HDL based FPGA design. Here he has lead the development of a number of software and FPGA based application frameworks, supporting devices from Virtex 2 to the latest Xilinx FPGA devices include the Zynq-7000 SoC. Over the last 3 years he has consulted with JPL (working for Alpha Data Parallel System Ltd.) to advise on, and architect, FPGA designs and software APIs for a number of applications in the area of Hyperspectral image acquisition and data compression.*

Daniel Nunes *(Ph.D., Earth and Planetary Sciences, Washington University (2004); M.A, Earth and Planetary Sciences, Washington University (2000); B.S., Astronomy, University of Kansas (1997)) is a Research Scientist in the Science Division, Geophysics and Planetary Geosciences Group at JPL. In addition to his planetary geophysics research, he worked in the initial development and field deployment of the off-line verification software for airborne hyperspectral imagers. He is currently working as the Investigation Scientist for both the SHARAD and the RIMFAX ground-penetrating radar instruments, respectively on the Mars Reconnaissance Orbiter (MRO) and the MARS2020 missions.*

Peter Sullivan *is an electrical engineer specializing in mixed-signal design and infrared instrumentation at the NASA Jet Propulsion Laboratory. He has previously worked at the Johns Hopkins Applied Physics Laboratory and holds a B.S. from Cornell University and a S.M. from the Massachusetts Institute of Technology. He has characterized image sensors for applications ranging from Earth science to exoplanet detection.*

Michael Bernas *is an electrical engineer specializing in mixed-signal design for FPA applications related to imaging spectroscopy at the NASA Jet Propulsion Laboratory. He holds a B.S. from University of California, Santa Barbara and a M.S. from the University of Southern California. He has previously worked on a 5 megapixel IR camera at Lockheed Martin Santa Barbara Focal plane.*

Michael Pham *is currently a senior at the University of California, Los Angeles, and will graduate in 2018 with a B.S. in Electrical Engineering. He began his fellowship at JPL under the Caltech Summer Undergraduate Research Fellowship (SURF) program in June 2015 in the Adaptive Embedded Systems lab under the guidance of Daniel Nunes and Didier Keymeulen. During his time at JPL, he has developed MATLAB software to process, verify, and validate data produced by airborne hyperspectral imagers such as AVIRISng, PRISM, and HyTES based on server computers.*