

DAQ: Software Architecture for Data Acquisition in Sounding Rockets

Mohammad Ahmad, Thanh Tran, Heidi Nichols, Jessica N. Bowles-Martinez

Jet Propulsion Laboratory, California Institute of Technology

Pasadena, CA 91009

818-458-0709

mohammad.ahmad@jpl.nasa.gov

Abstract—A multithreaded software application was developed by Jet Propulsion Lab (JPL) to collect a set of correlated imagery, Inertial Measurement Unit (IMU) and GPS data for a Wallops Flight Facility (WFF) sounding rocket flight. The data set will be used to advance Terrain Relative Navigation (TRN) technology algorithms being researched at JPL. This paper describes the software architecture and the tests used to meet the timing and data rate requirements for the software used to collect the dataset. Also discussed are the challenges of using commercial off the shelf (COTS) flight hardware and open source software. This includes multiple Camera Link (C-link) based cameras, a Pentium-M based computer, and Linux Fedora 11 operating system. Additionally, the paper talks about the history of the software architecture's usage in other JPL projects and its applicability for future missions, such as cubesats, UAVs, and research planes/balloons. Also talked about will be the human aspect of project especially JPL's Phaeton program and the results of the launch.

TABLE OF CONTENTS

1. INTRODUCTION.....	1
2. SYSTEM DESIGN	2
3. FLIGHT HARDWARE.....	3
4. SOFTWARE ARCHITECTURE	5
5. PHAETON.....	8
6. FUTURE USES	8
7. CONCLUSIONS	9
ACKNOWLEDGEMENTS	9
REFERENCES	9
BIOGRAPHY	9

1. INTRODUCTION

One of the long term goals of NASA's planetary exploration program calls for robotic missions to extraterrestrial planetary bodies in the solar system such as the Moon, Mars, Europa, Titan, comets and asteroids. In particular there are some high value targets on these surfaces, such as craters, dry lakes, and meteorite impact sites that have greater scientific value than a generic location. The problem arises in that quite often these targets are not large enough to be navigated to by current state of the art spacecraft guidance and navigation technology; current technology has a 3-sigma landing error of 1 km from a planned target. [1], [2]

A proposed solution to this problem is to utilize TRN. TRN technology utilizes camera images of a planetary surface and inertial measurement unit (IMU) data of a spacecraft as inputs into a Kalman filter based algorithm which can determine the position of a moving spacecraft in real-time and with relatively high-accuracy. The image processing technique is called Map Matching, which utilizes a large preloaded map of the targeted landing area to compare against the images taken by the spacecraft cameras and create a vector of landmarks. The vector of landmarks is then fed as an input into the Kalman filter, along with a correlated set of inertial measurements. Another image processing technique that TRN is developing is called feature tracking. This technique uses the location of the landmarks from one image to the next to help determine the speed of the spacecraft as well as do hazard detection. The spacecraft then utilizes the position information to continually do course corrections during the Entry Descent and Landing (EDL) phase resulting in a landing accuracy of < 100 meters. [2]

However, currently TRN is at a Technology Readiness Level (TRL) of 4, i.e. "technology development phase". To advance to the next stage of "technology demonstration" a flight-like data set must be collected. A realistic and cost-effective method of doing so here on Earth is to use a sounding rocket with a trajectory similar to a spacecraft's planetary entry as an analogue.

To this end JPL has conducted sounding rocket flights in conjunction with WFF's NASA Sounding Rocket Operations Contract (NSROC) for collecting the necessary data. To date there have been two of these flights conducted. The first was 41.068/Seybold in 2006 and the second one, followed by 41.087/Heyne in 2010. [3]

TRN technology development requirements dictated sounding rocket system requirements such as minimum frame rate for images, minimum image size, minimum data rate for IMU (measurements), minimum GPS location accuracy, and maximum timestamping errors. In addition to these science requirements, any hardware utilized had to survive the extreme g forces, temperatures, and pressure changes experienced during a sounding rocket flight.

A systems architecture that leveraged the suite of legacy flight-proven hardware provided by NASROC and COTS devices designed to survive in similar extreme environments

¹ 978-1-4244-7351-9/11/\$26.00 ©2012 IEEE.

² IEEEAC paper #1201, Version 1, Updated December 1, 2011

was deemed to be the most cost effective method of meeting the mission objectives of gathering the necessary data.

The resulting architecture utilized a COTS computer that interfaces with NSROC provided hardware such as the GPS and IMU and with the COTS devices such as cameras and a Solid State Drive (SSD). The data is stored on board the SSD during flight and it's recovered from the rocket after it landed. The COTS computer utilizes a Linux OS running a C programming language based software application to accomplish this. An important reason for utilizing COTS Linux and C is the desire to reduce development time and to create a product that could be easily tailored to meet the needs of a similar mission operating with different instruments. [3]

This paper will introduce in later sections the details of the hardware and software as well as the how the software architecture was adapted from the needs of the first sounding rocket flight to the second flight. There will also be an explanation of the mission-operating scenario as well as a discussion of the testing effort utilized to verify that the software developed on the COTS devices was meeting the stringent TRN requirements. Some fault-tolerant techniques used to perform successfully in one-shot and relatively expensive sounding rocket flights will be discussed including aspects such as software quality assurance, JPL rules, and balancing time and money with the scale of the project.

This paper will also talk about the human aspect of the two sounding rocket flights and its impact on the software effort. Particularly highlighted will be relationship between the workforce on the first flight and the second flight; especially the role that JPL's Phaeton program played in leveraging the experience of the scientists and engineers from the first flight to support the scientists and engineers working on the second flight.

At the end of this paper will talk about the role this particular software architecture has played in other JPL projects and its possible uses of it on future projects. Under consideration are projects involving other sounding rockets, weather balloons, UAVs and autonomous vehicles. In the last section conclusions will be presented relating to the effectiveness of the DAQ software architecture and the usage of COTS software and hardware to satisfy soft real-time requirements.

2. SYSTEM DESIGN

TRN requirements required that the trajectory of the sounding rocket follow as closely as possible an EDL trajectory of a spacecraft attempting to land on a planetary body such as the Moon or Mars. To meet this goal the sounding rockets were designed for a 15-20 minute flight. The trajectory is such that the rocket reached an apogee of 120 km about 2 minutes into the flight and it stays in the exo-atmosphere (aka space) for about 2 minutes. The

atmospheric reentry phase begins at about 5 minutes after launch, with rocket at about 100,000 feet. Shortly following this the parachutes deploy at approximately 15,000 feet and the payload begins descending through the atmosphere until touchdown about 8-15 minutes later.

The design of the sounding rocket was broken up among the JPL team and the NSROC team. The NSROC team utilized their vast experience in sounding rocket development to build a number of active systems on board the rocket. These range from the fabrication of a custom aluminum shell, to building a power supply for the electronics on board the payload, to creating the software that controls the trajectory and maneuvers of the rocket, as well as supplying the Terrier Improved Orion rocket motor which actually launches the payload. Another important instrument provided by the NSROC team was the GPS and IMU system. [4] IMU measurements consisted of x, y, z accelerations, velocity, and delta thetas (turning angles). The GPS measurements were the x, y, z position, velocity, and GPS time. The GPS measurements served as a ground truth so that in post-processing the TRN algorithm's performance could be compared against a known truth. [2] All this data is combined into a large packet by the NSROC's microcontroller and transmitted out via RS-422.

The JPL team's contribution to the rocket payload was less broad, but just as important. Other than working with the NSROC team to define the rocket trajectory and TRN required activities, the JPL team was responsible for providing the onboard cameras and their associated peripheral devices such as lenses and lens heaters. Their additional contribution was the onboard computer, which interfaced with the cameras as well as the NSROC provided GPS and IMU units. The computer ran a software application developed by JPL to collect data from these sensors and timestamp it and store the timestamped data on computer's main drive, which was a Solid State Drive (SSD).

As mentioned earlier, there were two sounding rocket flights; however, each flight had a slightly different mission. When the first flight (41.068/Seybold) was launched its main purpose was to provide a proof of concept of the idea of using sounding rockets as way to advance TRN. [3] In keeping with this goal, the design was kept relatively straightforward. Cameras were only present in one section of the rocket known as the "descent" section. This section was aligned such that when the rocket was parachuting down through the atmosphere, the camera in the descent section was pointing towards the ground. The on-board computer utilized the Camera Link (C-Link) protocol to control the cameras and snap images at the desired frame rate dictated by TRN.

The set of IMU/GPS data was fed to JPL provided on-board computer through the computer's RS-232 port (with a RS-422 to RS-232 converter in between the NSROC microcontroller and the JPL computer's port). The data

from this port was timestamped by the same clock and in the same format used to timestamp the camera images. The result is a correlated set of images, IMU and GPS data.

The system design for 41.087/Heyne built upon the work and experience from the first flight and had new requirements so that TRN could be advanced further. The main goal was to develop a complementary data set to the “descent” phase by getting images from when the rocket is in the “exo-atmospheric” phase. This exo-atmospheric phase was representative of a spacecraft’s approach from orbit above a planet to just before it entered the atmosphere. In the case of the sounding rocket flight, this phase was approximately 2 minutes after the rocket approached the apogee. During this phase the camera was required to take images of the ground, in particular the location on the ground where the rocket was expected to touchdown. When the rocket reentered the atmosphere and the parachutes were deployed a second camera took images of the ground, in a manner similar to the 41.068/Seybold. The design chosen to accomplish this used two different cameras in two different sections of the rocket payload. The first camera was in the “exo-atmospheric” section, which was maneuvered so that the side containing the camera pointed down towards the ground. This descent section was similar to the descent section from the first flight, with a camera located on the portion of the payload opposite the parachute such that it always points downward. (see Figure 1).

C-Link cameras were used once again to interface cameras with the on-board flight computer; however, now there was

the addition of an extra camera. The IMU/GPS interface mechanism remained the same as being via the RS-232 port attached to the computer. The IMU/GPS dataset was logged for the duration of the flight similarly to 41.068/Seybold.

The key requirements that applied to the flight software from a TRN point were for minimum imaging frame rates of 6 Hz in the descent section and 3.5 Hz in the exo-atmospheric portion of the flight. An imaging blur requirement translated into a maximum exposure time of 1.1 ms for the cameras. The requirements for the IMU were to collect data at a rate of 100 Hz, and the GPS to collect at a rate of 20 Hz. The timestamping requirement was such that all data be timestamped within an accuracy of 10 ms. This requirement includes any timestamping jitter as well as any latency uncertainties: latencies by themselves are not bad as long as it’s known to enough precision.

In both flights a user remotely logs in to the flight computer via an Ethernet connection during the launch countdown and initiates the software application. After the rocket landing, a team would recover the payload shortly after the launch (approx 2 hours). Because of the possibility that the system could remain powered on during some of this time, it was also critical for the software not to overwrite any flight data, if/when it ran out of space on the SSD.

3. FLIGHT HARDWARE

The hardware instruments provided by NSROC included the JAVAD GPS and the LN200 IMU. Both of these

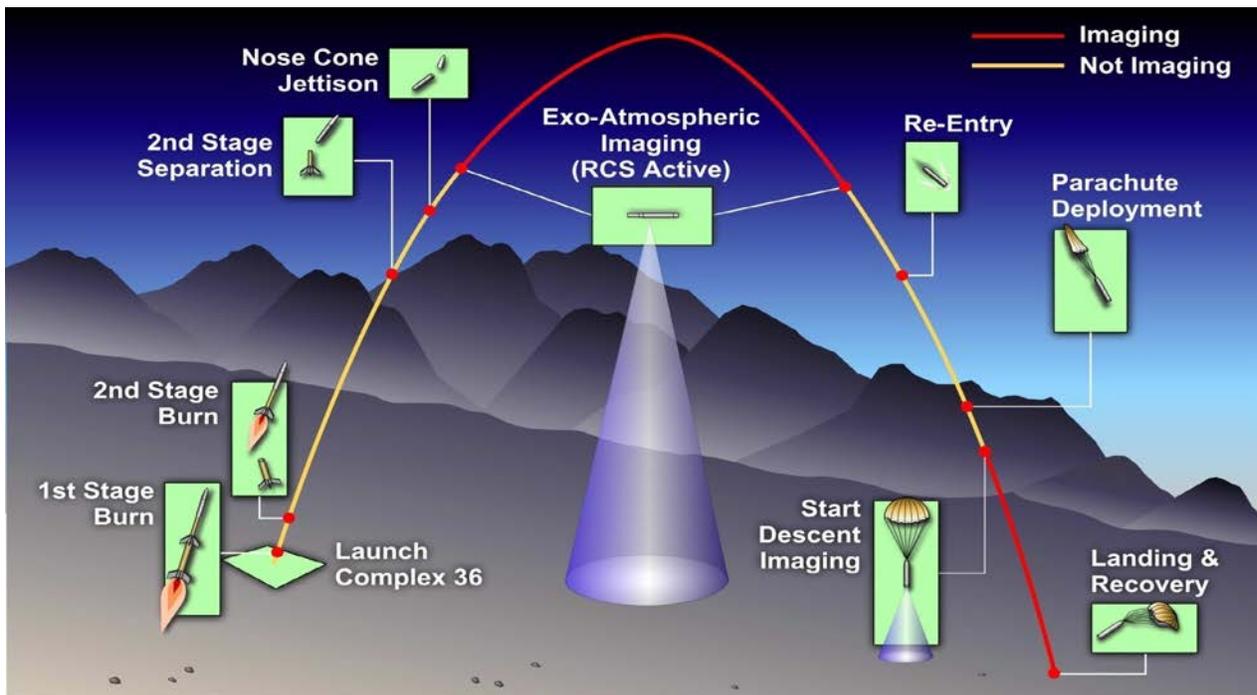


Figure 1 – flight and trajectory of the 41.087/Heyne sounding rocket flight, also shown are the major events such as the imaging portions of the flight. Image courtesy of the Phaeton project. [6]

instruments had been used extensively by NSROC and are customized for optimal performance on NSROC sounding rockets. The task of choosing the instruments was relatively simple as their versions of the IMU and GPS met the science requirements. NSROC's LN200 IMU's data rate is 100 Hz while their data rate of the GPS is 20 Hz. As mentioned previously IMU measurements consisted of (x, y, z) accelerations, velocities, and delta thetas (turning angles). The GPS measurements were (x, y, z) position, velocity, and GPS time. [4]

The GLN-MAC contains the embedded flight microcontroller, which interfaces with a LN200 IMU and the JAVAD GPS device to create a packet of data containing measurements from these two devices as well as an internal timestamp. [4] The GLN-MAC then formatted the data into a 179-byte packet to be transmitted at a rate of 50 Hz. Each packet contained two sets of IMU measurements (as the IMU's rate is 100 Hz). Only a single set of GPS measurements were in each packet; however, some measurements were repeated across packets, as the GPS rate is 20 Hz and the GLN-MAC simply grabs the latest set of GPS measurements to put them into the packet. This is important to keep all the packets a consistent size of 179 bytes. The GLN-MAC microcontroller also timestamps each measurement (IMU and GPS) based on its internal clock. The packet outputted at a baud rate of 115200, no parity, 8 data bits, and 1 stop bit. Each packet also had a 3-byte sync word at the beginning of the packet. The physical port on the GLN-MAC that output the data used the RS-422 protocol. The data line was passed through an external MAX233 chip that converts the signal into RS-232 format for the JPL flight computer. The GLN-MAC continuously transmits the data at the 50 Hz rate (in other words a new packet every 20 ms). To ensure that no spurious signals interfered with GLN-MAC operations, the receive pins on RS-422 port were removed.

The JPL team then had the responsibility for building the imaging subsystem. One of the key requirements for any camera to fly successfully on a sounding rocket is its ability to survive the high vibrations during the launch period as well extreme temperatures of up to 140 degrees Fahrenheit. In addition there is a need for a high degree of customization of camera settings such as exposure time, gain and reference voltage to allow for images that are optimized for the light conditions of the rocket's environment. An EEPROM in the camera saves the settings so that the camera could power on with user-desired settings. There are a number of commercial available cameras for this kind of need. JAI/Pulnix supplied the camera chosen by JPL for both flights. The camera used in the first flight was the TM1020-CL, which had an image size of 1008x1018. Each pixel was 8 bits, making a single image 1 Mb, the maximum frame rate was approximately 15 Hz. Due to improvements in camera technology between the time frame of 41.068/Seybold to 41.087/Heyne, a new model of the camera was chosen for 41.087/Heyne: the TM-2030CL with an image size of 1920x1080 (HD quality

images, with 8-bits/pixel each image is approx. 2 Mb, the maximum frame rate of the camera is 16 Hz). The same model camera would be used in both portions of the 41.087/Heyne flight, but with different field of view lenses. The cameras were grayscale and progressive scan (non-interlacing).

The cameras also utilized the C-Link protocol as means of transferring images and changing settings. The C-Link protocol is a high-speed imaging protocol developed by Automated Imaging Association (AIA) for machine vision application utilizing a special 28-pin connector and transfers data using LVDS for rates upto 3.6 Gbps.

The COTS flight computer had to satisfy similar environmental requirements as the cameras. The JPL team decided on a highly ruggedized and compact computer known as the Kontron CVX-server. The Kontron utilized a 1.7 GHz Pentium M processor, had 4 USB ports, 3 RS-232 ports, a PCI slot, and 1 Gb of DRAM. In addition the Kontron was disassembled and special potting material was applied to key joints and screws to further ruggedize it.

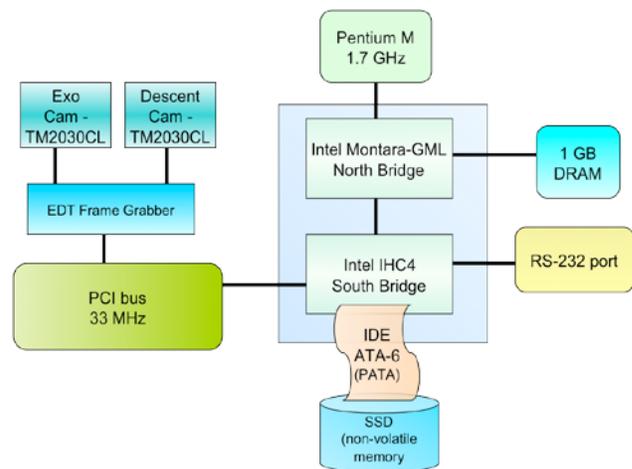


Figure 2 - detailed layout of the COTS Kontron usage in 41.087/Heyne

A C-Link frame grabber was installed on the PCI slot to enable interfacing with the cameras. The frame grabber was provided by the EDT company and had two separate channels (see Figure 2). Only channel 1 had to be used in 41.068/Seybold; channels 1 and 2 were used in 41.087/Heyne.

The Kontron also had a Parallel ATA-6 (PATA) interface for its hard drive. The original hard disk drive was replaced by a shock resistant solid-state drive (SSD), as the hard disk's mechanical components were not rated to survive the launch vibrations [3]. As SSD technology had progressed in the time between the first and second flights, the capacity of the SSD was increased from 4 Gb to 128 Gb. The SSD boots the Operating System (OS) as well as serving as the location where the flight data is stored by the software.

As described in the previous section, the imaging rate requirements for 41.087/Heyne were 3.5 Hz and 6 Hz for the descent and exo-atmospheric phase respectively. TRN technology development could possibly benefit from imaging rates greater than the required rates. A throughput analysis/testing of the Kontron, the frame grabber, and the SSD revealed that the biggest throughput bottleneck was the Kontron's PATA interface with the SSD, which limited the maximum write speed to 19-20 Mb/s. This analysis affected the design of the software for 41.087/Heyne. As the desired write speed for operating both TM2030CL cameras simultaneously was a minimum of 19 Mb/s, there was no margin. The design for 41.087/Heyne then was changed to capturing only images from exo-atmospheric camera during exo-atmospheric mode and capturing only descent camera images during only the descent portion of the flight. In this scenario, the imaging rate could be increased to 8 Hz, with each camera only operating during its required phase in the flight and software controlled switching between phases.

4. SOFTWARE ARCHITECTURE

The Kontron's processor and the memory use a 1.7 GHz Pentium M processor and 1 GB of RAM and it is capable of supporting a large COTS OS such as Windows XP or Linux. There was a need for high degree of OS customization to maximize performance and to support drivers for the various sensors. This drove the decision to use a Linux based OS such as Fedora. Fedora 6 was used in 41.068/Seybold and Fedora 11 in 41.087/Heyne (each the latest version available at the time of their usage). A real-time operating system was strongly considered, as it would've made the task of ensuring the timestamping accuracy much simpler; however, due to budget and especially schedule constraints, the project chose to go the route of a highly optimized non real-time OS in order to minimize development time.

Software Drivers for Sensors

The next step in software development was to identify and install any drivers for the hardware instruments. In the 41.068/Seybold, the two instruments interfacing with the Kontron were the TM1020-CL C-Link camera and the RS-232 output from the GLN-MAC. In 41.087/Heyne, the three instruments interfacing with the Kontron were two TM2030-CL C-Link cameras and the RS-232 output from GLN-MAC (see Figure 2).

As the EDT C-Link frame grabbers were the method of interfacing the Kontron with the C-Link cameras it was necessary to install drivers only for the frame grabber. Linux compatible drivers are available from EDT and install bug-free, at least on Fedora. The default install directory is the `"/opt/EDTpdv"` location in Linux. All necessary *include* files containing the camera drivers' Application Programming Interface (API) were by default installed to this location. Any program using the drivers simply

included the proper file from the install directory and called on the appropriate methods as defined in the API.

A challenging aspect of properly interfacing the cameras with the frame grabber was the need for a camera specific "config" file. This file had a special format and was used by the frame grabber to load camera characteristics such as image size, number of bits per pixel, and interlacing or non-interlacing mode among many other things. The EDT drivers had a large selection of config files for many different cameras; however, the cameras chosen by JPL (TM1020CL and TM2030CL) were not included. For 41.068/Seybold a significant effort had to be devoted to creating a proper config file. Much of the same task had to be repeated for 41.087/Heyne as the cameras had changed although the learning curve was much smaller due to previous experience.

The drivers necessary for using the COM ports on the Kontron came as part of the Fedora package. It is represented as the `"/dev/ttyS0"` device file and is very commonly used by a very large number of users in various applications [5]. There are also plenty of online resources dealing with proper usage of these drivers with sample source code.

DAQ Software Architecture

Software architecture is based on a multithreaded design; two threads are responsible for data from each sensor. The first thread acquires data and writes the acquired data to a ring buffer; while the second reads data from the ring buffer and writes this data to the SSD (see Figure 3). The threads are Linux posix threads and utilize the standard Linux "pthread" library. This library is available on all versions of Fedora.

This design is implemented in such a way as to create a standard approach to adding or removing new instruments. Two top-level global arrays hold instrument specific data and function pointers for all instruments.

The first array is of structure types defined as SENSOR. The SENSOR struct contains the following information:

- size: an integer representing the number of bytes per data point (example: 179 bytes per GLN-MAC packet or 1920*1080*8 bytes per TM2030CL image). Assuming that all packets of data from a specific sensor are always of the same size.
- rate: an integer specifying the rate of samples per second (example: 8 Hz imaging rate for 41.087 cameras)
- description: a string describing the sensor (example: "Pulnix TM2030CL")

The second array is of DAQ_TASK structure types and contains function pointers (C language data type

FUNCPTR) to functions of required tasks for all instruments. DAQ_TASK is defined as containing the following:

- *start()*: This function typically contains a loop which runs continuously and acquires data from the sensor. The acquired data is associated with the latest timestamp and then passed on to a ring buffer, through special functions defined later in this section. The sensor specific functionality may include such things as calling the driver’s API for a function such as “*snap_image()*”.
- *stop()*: A function to do a clean stop of the data acquisition by freeing some of the variables associated with the setup.
- *saveData()*: Function which takes as input a data point and its associated timestamp and writes these to disk in the proper format. For example the image data from the frame grabber drivers is in a raw image format, in this method for camera sensor a PGM format header is attached and the timestamp is put in the comments section of the header. In the case of the sounding rocket flights, the data was written out to a binary log file on the main drive of the OS. Since the OS was installed on the SSD, all data was written to it.

buffer per sensor is limited by the largest contiguous of space block available on the DRAM. For Fedora 11 running on the Kontron with 1 Gb DRAM, the available largest block was 100 Mb. The total allocation could also be limited through a special parameter input by the user. The C library function used to allocate the space is the generic “*malloc()*” function.

The ring buffer is a queue type design utilizing two pointers. A “*log_queue_insert()*” method, called from the while loop in *start()*, moves data from sensor’s memory space (DRAM) into the ring buffer memory space location pointed to by the insert pointer. The insert pointer gets incremented every time new data is inserted. An integer variable, *log_queue*, also gets incremented with every successful insert.

A second pointer is used by the “*log_save()*” method to move data out of the ring buffer and into the location specified by the “*saveData()*” method. This pointer is incremented with every successful pop from the queue. The pointers loop around the ring buffer until they reach the last location. The “*log_save()*” pointer only pops data if its’ value is less than the “*log_queue_insert()*” pointer, preventing data from being written out twice. The *log_queue* gets decremented with every successful pop.

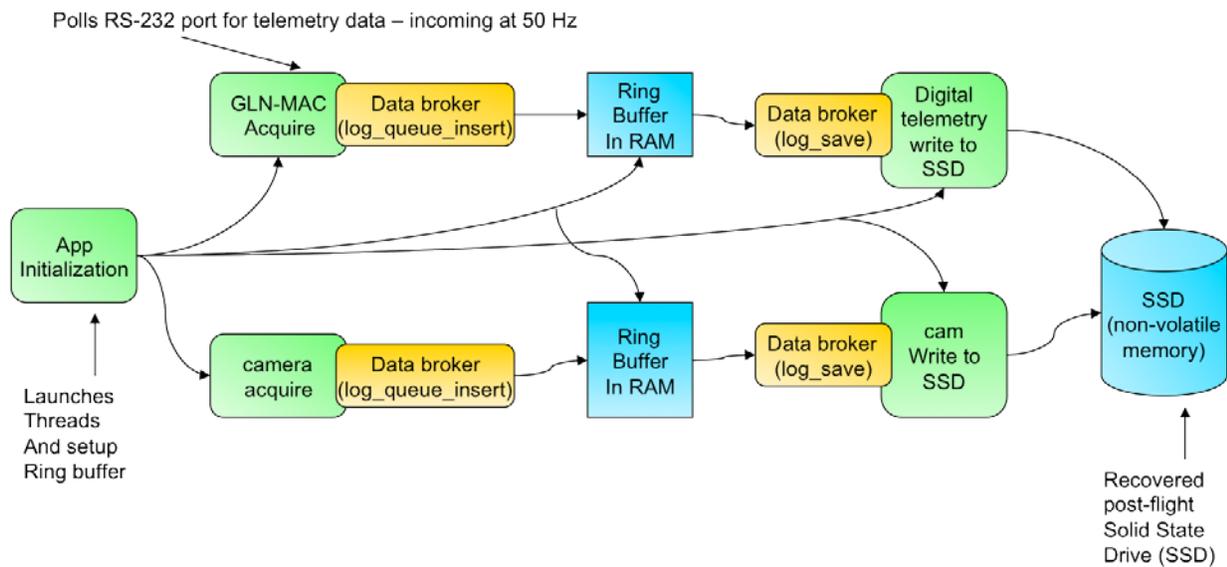


Figure 3 – flow diagram showing threads launched by “App Initialization”.

An *init()* function for each of the sensors is called by the *main()* function and initializes the DAQ_TASK and SENSOR structure type arrays. The next function called by *main()* is responsible for setting up the ring buffers which handle the data between the acquire threads and the write threads. Each sensors’ unique ring buffers are allocated in the DRAM and setup before the threads are spawned. The memory allocation in the ring

The *main()* then calls the “*acq_start()*” method. This function spawns an acquire thread for each of the sensors identified in the DAQ_TASK array. This is done through a call to the “*taskSpawn()*” method from the Linux pthread library. Each thread writes data to the ring buffer by calling the “*log_queue_insert()*” function.

“*Main()*” then launches the “*log_start()*” function. A mirror image of the “*acq_start()*” function, except now a thread is spawned to read the data written into the ring buffer and write it out to disk. This function uses the “*taskSpawn()*” to spawn a thread of the “*log_save()*” function, which contains a loop to continuously grab the data from the ring buffer and write it out using the “*saveData()*” method defined for each sensor type.

Mutexes are used to protect the data flow between the acquire threads and the save threads. The “*log_save()*” threads also use a thread mutex to block the thread if there is no data in the ring buffer to save. This is meant to save CPU cycles.

A race case condition exists if the “*log_save()*” thread can not write data and clear out the ring buffer as fast as the threads spawned from the “*acq_start()*” were writing it to disk. If the acquire threads encounter a full ring buffer, the data is dropped, until the sensor’s “*log_save()*” thread can read the data out and clear space in the ring buffer. This is detected if the *log_queue* variable is equivalent to the capacity of the ring buffer.

Soft real-time camera switching

As mentioned in Section 3, the 41.087/Heyne’s software on the Kontron was required to switch cameras from exo camera to descent camera at the proper time. This was implemented by utilizing the information in the incoming packets from the GLN-MAC. A special timer in 179-byte packet was such that its value only started incrementing when the rocket left the launch pad. This timer was also accurate enough to be useful in knowing whether the rocket is in the two-minute portion of the flight between exo-atmospheric imaging and descent imaging phase. The GLN-MAC acquire thread continually analyzes the timer to see when it reached the “desired value”. At the programmed time the GLN-MAC acquire thread incremented a specific global variable. This global variable was read by the camera acquire thread and at that time, the camera acquire thread switches to acquiring from the descent camera. (Note: The exo-atmospheric camera is on a specific channel of the frame grabber, while the descent camera is on the other channel, the software simply switches channels) Once the switching occurred, only a restart of the software could switch it back to exo-atmospheric mode. A fault tolerance mechanism was built in to prevent a spurious signal from causing a premature switch. Each frame’s timer analyzed by the GLN-MAC acquire thread would have to be at the desired value for at least seven consecutive frames and each timer value would have to be greater than the previous value. If any frame’s timer contained a value less than the pre-programmed “desired value” or the previous timer value, the count would start from zero once again.

Timestamping Analysis

The timing tests on the software/hardware revealed a total latency uncertainty of 2 milliseconds. The latency uncertainty for applying timestamp to an image was very low, approximately 50 microseconds. This was due to the fact that the frame grabber drivers used the PCI bus to write data directly into the DRAM. The frame grabber drivers utilized Direct Memory Access (DMA) by transferring directly from the camera’s CCD into the Kontron’s DRAM with no intermediate buffering in the frame grabber. The 50 microseconds latency number was provided by the frame grabber manufacturer as a maximum latency for DMA transfer over the PCI bus. As the frame grabber driver also timestamps the data after receiving

The latency for applying the timestamp to a RS-232 packet was higher at about 1.8 milliseconds. This was due to the lower priority the UART interrupt has in the Linux kernel’s default drivers. A test byte of data was sent from one COM port to another COM port on the Kontron, the byte was time tagged before it was transmitted and after it was received; the maximum delay observed in these tests was no more than 1.8 ms. The actual latency for just receiving serial data and timestamping it was almost certainly less than this value, as this added in the transmit latency on the Kontron’s COM part as well. Serial port latency is higher because the data is buffered by the UART chip’s internal register, rather than using DMA. Part of the 1.8 ms latency comes from the RS-232 chip on the Kontron, which is an UART16550A. It has an internal buffer of 16 bytes. At a baud rate of 115200 symbols/sec a maximum of .11 milliseconds worth of data can be stored on the chip before the processor accesses it and is able to timestamp it. As the system requirement was for an under 10 milliseconds latency uncertainty, no further testing was conducted to narrow down the RS-232 latency uncertainty and to be safe the whole RS-232 latency of 1.8 milliseconds was included as part of the latency uncertainty.

Linux Optimizations

The versions of Fedora used in 41.087/Heyne (Fedora 11) has default built-in (Graphical User Interface) GUI interface. This GUI consumed valuable CPU cycles, which directly affected the latencies of any timestamps applied by the DAQ software application and the priority of interrupts used by the sensor drivers. The GUI can be deactivated by editing the “*/etc/inittab*” file and changing the runlevel from 5 to 3. [6] This had no detrimental impact on the usage of the software application, as the user would log in to the Kontron through its Ethernet connection to use it.

Another Linux modification used was to launch the DAQ application as soon as the Kontron runs through its boot sequence, through editing the “*/etc/rc.local*” file. [6] This was implemented as a fault tolerant feature in case the Kontron temporarily lost power in mid-flight.

The data recovery portion of the mission involved removing the PATA SSD from the flight Kontron and installing it on an identical model spare Kontron. As the OS is installed on the SSD, the spare Kontron would boot with flight OS and data, which could then be copied over to an external hard drive using the USB port on the Kontron.

5. PHAETON

An important human aspect of the sounding rocket flights was the involvement of the Phaeton program at JPL. Phaeton is a program at JPL whose primary purpose is to train young engineers and scientists at JPL a hands-on experience to allow them to experience the full life cycle of a flight project. Typical flight projects at JPL can last up to a decade from project conception to actual launch. Some projects are also very large so that engineers only experience a very limited aspect of it. In 2008, a group of young JPL engineers conceived of the Phaeton program so that young engineers could get hands-on experience working on a small short life-cycle project. [7] Importantly Phaeton required that projects be useful to JPL in their scientific and technological objectives and accomplishments. Another important aspect of the Phaeton program is that all the team members on the Phaeton project must be young engineers (aka Early Career Hires (ECH): defined at JPL as having received a degree within the last 3 years). Once selected into the Phaeton program, the ECHs are assigned a mentor in the aspect of the project they were chosen to work on, and meet with them on a weekly basis and guide them in all aspects of their projects. [7] The mentors for the software development effort included Kenny Meyer, Calina Seybold, and Paolo Bellutta. The software architecture is primarily based on Paolo Bellutta's original source code from 41.067/Seybold.

A sounding rocket flight was considered a natural project to be part of the Phaeton project; they are typically small projects with budgets from \$1 million to \$5 million (compared to \$1+ billion for a flagship JPL mission), and a quick design turnaround time of 1-2 years. Another useful aspect of a sounding rocket flight as part of the Phaeton program was the recent flight of 41.068/Seybold to advance TRN. TRN technology development plans called for more sounding rocket flights to advance TRL levels; and much of the scientific justification for a new sounding rocket flight had already been done as well as the willingness of team members from 41.068/Seybold to serve as mentors of the next sounding rocket flight to advance TRN through another sounding rocket flight.

At the time of the formation of Phaeton a NASA proposal known as Hands-On Project Experience (HOPE) also started up. However, this was a announce of opportunity (AO) released by NASA for proposals of projects that use a sounding rocket flight via NASA WFF and NSROC to train young engineers. It was to be competed among the different NASA centers. Ultimately the JPL team won the

competitive proposal, which became known as 41.087/Heyne. [8]

The 41.087/Heyne sounding rocket was launched in December 2010, and was for the most part successful; there was a mechanical problem with the deployment of the exo atmospheric door, resulting in the collection of black images for the exo atmospheric portion of the flight. The problem was traced to unknown problem with the pyrotechnic screws that did not properly deploy. The JPL software performed exactly as planned, with the collection of complete correlated set of IMU/GPS data along with images from the exo-atmospheric and descent cameras; however, only images from the descent camera were of scientific value for the reason mentioned above. Although without a doubt an exo-atmospheric dataset would have been useful; a second descent dataset is still of scientific value as it provides an opportunity to verify the TRN algorithms with a second dataset and the data quality of the images and IMU measurements is far superior. Work on the data processing is ongoing and will be published at a later time by JPL.

Software Lifecycle

An important part of being a Phaeton project was the requirement to fulfill many of the JPL flight project practices followed by typically larger project. These included activities such as detailed requirements drawn up by the software cognizant-engineer, Mohammad Ahmad, with support from project system engineers (Benjamin Solish, Heidi Nichols), scientists (Martin Heyne, Shane Brennan, Nikolas Trawny) and project managers (Don Heyer). The design and the requirements also had to pass through several major peer reviews such as a Requirements Review, Preliminary Design Review, and a Comprehensive Design Review among other smaller internal reviews.

Other JPL internal reviews included those led by the Software Quality Assurance Engineer (SQAE), Ken Evensen, for verifying that the software met JPL institutional software development requirements.

The Integration and Test (I&T) Engineer (Thanh Tran and Jessica Bowles-Martinez) were responsible for verifying independently that the software meets all the software requirements and validating the functionality of the software meets the planned mission objectives. The I&T Engineers also had the key responsibility for certifying that the flight software interfacing properly with external interfaces such as Wallops Flight Facility software and hardware.

6. FUTURE USES

The software architecture and the implementation of the code make the DAQ software very amenable to being used on different projects. In fact many JPL research projects are technology development projects or science projects on earth based "vehicles" such as balloons, helicopters, sounding rockets, autonomous vehicles and UAVs. It is likely to be used in another Phaeton project involving

weather balloons. Currently the DAQ architecture is under consideration to be used on another Phaeton project involving a research balloon flight into the upper atmosphere, with the intent of collecting data from some instruments under development at JPL.

The software architecture has been used on various JPL technology development tasks to collect data on board autonomous vehicles (PredaTOR project and (Mars Science Lab) MSL technology development tasks for machine vision related activities). The functionality was similar, to collect data from various sensors (typically cameras) and store it on board for post-processing.

7. CONCLUSIONS

The utility of the DAQ architecture can be measured by its use over the last couple of years in many different projects at JPL, under different developers. The key reasons for the utilization is its use of generic Linux libraries to make it usable across many different machines as well as the well defined function structure which allows a user to easily customize it to any number of sensors. The main weakness identified with the DAQ architecture is that all data points from a single sensor must be of the same size. The workaround to this weakness is for the sensor to be initialized to a certain size and smaller sized data must be padded with garbage bits and bigger sized data must be broken up into smaller packets by the sensor's acquire thread.

The analysis of the timing test data from 41.087/Heyne also shows that COTS Linux's timestamping capabilities are capable of achieving millisecond level accuracy, essentially a soft-real time framework. Linux performance can also be improved with certain optimizations. The DAQ architecture can also be adapted to an actual real-time OS that uses posix threads; fortunately most real-time Linux OSes have posix libraries included.

ACKNOWLEDGEMENTS

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. The work was also carried out through the Advanced Computer Systems and Technologies Group at JPL in the Flight Electronics and Software Section, which provided key management support and technical reviews.

The NASA Sounding Rocket Office Contract (NSROC) operating out of Wallops Flight Facility (WFF) provided significant technical and scientific support. Funding was provided via JPL's Phaeton program and NASA's HOPE program. The authors would especially like to thank Dr. Johnny Kwok, Ross Jones, and Valerie Duval of JPL for their leadership and support of the Phaeton program. Also

we would like to thank the Flight Electronics and Software Section's management team for their support and leadership, especially Bill Whitaker, Naomi Palmer, Elihu McMahon and Michael Sierchio.

REFERENCES

- [1] NASA. (2006, Sep.) "Solar system exploration roadmap," [Online]. Available: http://solarsystem.nasa.gov/multimedia/download-detail.cfm?DL_ID=302
- [2] A. I. Mourikis, N. Trawny, S. I. Roumeliotis, A. E. Johnson, A. Ansar, and L. H. Matthies, "Vision-Aided Inertial Navigation for Spacecraft Entry, Descent, and Landing" IEEE TRANSACTIONS ON ROBOTICS, VOL. 25, NO. 2, APRIL 2009
- [3] C. Seybold, G. Chen, P. Bellutta, A. Johnson, L. Matthies, and S. Thurman. "Suborbital Flight Test of a Prototype Terrain-Relative Navigation System", AIAA Infotech@Aerospace 2007 Conference and Exhibit 7-10 May 2007, Rohnert Park, California
- [4] D. J. Krause "NASA's Sounding Rocket Program NSROC, Accomplishments and the Future". Proceedings of the 17th ESA Symposium on European Rocket and Balloon Programmes and Related Research, Sandefjord, Norway. 30 May – 2 June 2005 (ESA SP-590, August 2005) Available: <http://adsabs.harvard.edu/full/2005ESASP.590..305K>
- [5] "Setting up Permissions" [Online] Available: <http://gphoto.sourceforge.net/doc/manual/permissions-serial.html>
- [6] "Manpage of INITTAB" [Online] Available: <http://www.netadmintools.com/html/5inittab.man.html>
- [7] Phaeton TRaiNED project status website [Online] Available: <http://phaeton.jpl.nasa.gov/internal/ProjectStatus/TRGS/>
- [8] NASA (2009, Fall) "HOPE for the future" [Online]. Available: http://askmagazine.nasa.gov/issues/36/36s_hope_for_future.html

BIOGRAPHY



Mohammad Ahmad is a software / hardware engineer at JPL's Flight System Avionics section. He has worked on various projects at JPL such as Mars Science Laboratory, research on wireless avionics, and FPGA development for real-time image processing. Currently he is working on the Soil Moisture Active Passive (SMAP). He

graduated from Carnegie Mellon University in 2007 with a Masters in Electrical and Computer Engineering. His role on 41.087 was as the software cognizant engineer for the JPL's experiment software.



Thanh Tran is an integration and test engineer at JPL. He graduated from UCLA in 2006 with a Masters in Electrical Engineering. His main job at JPL for the past 3 years has been to conduct rigorous testing of the functionality of the on-board FPGA and electrical system of the MSL rover. His role on the 41.087 project

was as the lead I&T engineer for the JPL subsystem.



Jessica N. Bowles-Martinez is a electrical and computer engineer at JPL. She graduated from MIT and Johns Hopkins University. Her specialty is analysis of effect of radiation on FPGA's and computer memory in space environments. Her role on 41.087 was as environmental test engineer and

I&T engineer.



Heidi Nichols is an engineer at JPL. She graduated from Chico State University with a Masters in computer engineering. Her specialty at JPL is as a parts environmental reliability analysis. Her role on 41.087 involved aiding the I&T, systems engineering, and software engineering teams. Other project experiences include the Juno

mission to Jupiter and the SMAP satellite.

