A MODEL-BASED STUDY OF ON-BOARD DATA PROCESSING ARCHITECTURE FOR BALLOON-BORNE AURORA OBSERVATION

Chester N.V. Lim
Jet Propulsion Laboratory
College of Engineering
University of Hawai‘i at Mānoa
Honolulu, HI 96822

ABSTRACT

This paper discusses an application of ISAAC design methodology to a balloon-borne payload electronic system for aurora observation. The methodology is composed of two phases, high level design and low level implementation, the focus of this paper is on the high level design. This paper puts the system architecture in the context of a balloon based application but it can be generalized to any airborne/space-borne application. The system architecture includes a front-end detector, its corresponding data processing unit, and a controller. VisualSim has been used to perform modeling and simulations to explore the entire design space, finding optimal solutions that meet system requirements.

INTRODUCTION

The objective of the modeling methodology is to conduct trade space studies to evaluate the performance and effectiveness of system architectures. A good model has characteristics of a high level block diagram with functionality that incorporates low level requirements. The model discussed in this paper is for a ballooning mission as described by [1]. There are three high level blocks in the model, which includes the detector, processing unit (Ebox), and a controller as shown in Figure 1. For this paper the instrument being modeled will be one of JPL's developed near infrared (NIR) cameras. The Ebox includes an FPGA-based data processing platform (*iBoard*) developed by JPL and a storage element. Balloons from the National Scientific Ballooning Facility (NSBF) have their own separate computing element that acts as the master controller thus is being modeled as an external controller.

Figure 1: High Level System Design

This paper will go through the methodology of creating the model and some of the results that can be drawn from it. The structure of the paper is as follows, Section 2 will discuss an innovative technology that the ballooning mission incorporates which the model is a subset of; Section 3 will provide a background about one such ballooning mission that uses the NIR camera

to put things into context; Section 4 will formulate the problem, present assumptions and simplifications made by the model and discuss operational scenarios; Section 5 will give detail about the model; Section 6 will present results from the model and finally Section 7 will discuss about future extensions.

ISAAC DESIGN METHODOLOGY

ISAAC (Instrument Shared Artifact for Computing) is a JPL three year research and development task. Its main purpose is to provide designers a workflow that takes the users' high-level block diagrams and implement them into hardware. ISAAC technology targets a variety of instruments including radar, radiometer, and imager. It hopes to cut design time and cost by introducing a reusable FPGA data platform that has prebuilt cores which the designer can use.

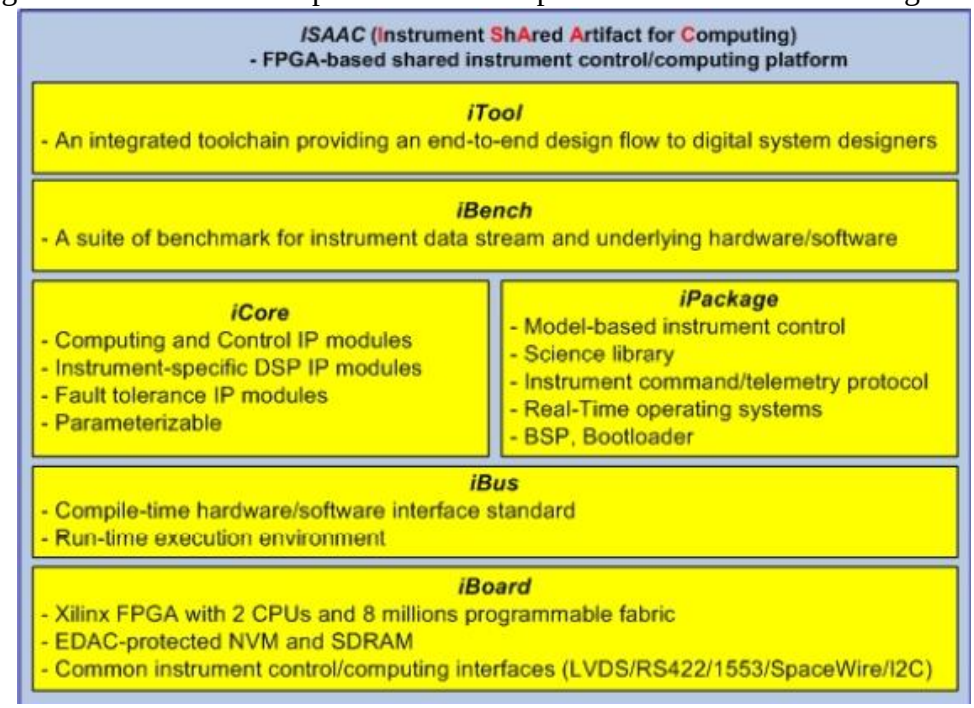

Figure 2: ISAAC concept diagram

Figure 2 shows the concept flow starting from the top flowing to the bottom. To begin, the user will use *iTool, which* is an integrated tool-chain that will aid in taking high-level designs to be implemented in hardware. *iBench* contains a suite of historical data streams from a variety of instruments. Data streams may be used through a common test bench architecture that tests and validates the ISAAC platform. *iCore* contains a set of standard IP cores, which the designer may select from. The designer does not need to spend time developing these cores since it's been provided. *iPackage* provides a front end interface to the user, which allows for instrument control and computing solution. *iBus* handles all the data flowing to and from ISAAC's FPGA platform to *iPackage. iBus* creates a standard protocol for the various instrument data transmissions. Finally, iBoard is the FPGA platform that features a high-performance Virtex 5 FPGA.

The ISAAC methodology is composed of two phases, high level modeling and simulation and low level implementation. This paper will focus on the high level models and simulations, exploring possible design spaces to determine optimal solutions for the system design.

NOTATIONAL MISSION OVERVIEW

Auroral observations are limited to places near the magnetic poles. In addition, it is desirable to be in a place that is not near any locations that may disrupt observations; this limits the locations to Antarctica. The NSBF restricts the window of opportunity of a balloon observation in Antarctica from December 1st to completion of the mission by February 1st. Based on the MAXIS balloon mission in 2000, the mission is expected to have a 15 day flight duration cruising at 35-50 km above ground.

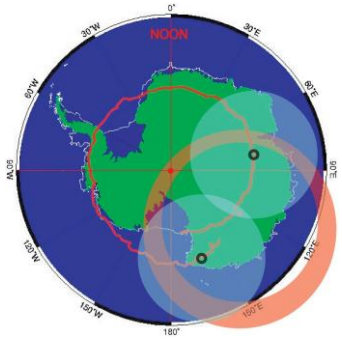

Figure 3: Balloon Flight Path [2]

The red squiggly line in Figure 3 shows the flight path taken by MAXIS in 2000 over Antarctica [6]. The red dot in the middle of the figure represents the geographical pole and the red dot in the lower right quadrant represents the magnetic pole. The red oval around the magnetic pole represents the auroral ring and the black circles are points at which the balloon is able to do observations as represented by the white circles. The balloon is expected to continuously change orientation while on its flight path. As a result, the cameras may possibly observe dayside aurora, night side aurora, aurora at lower or higher than the auroral ring.

The gondola on balloons provided by NSBF has separate control unit and power supply. The payload will reside in the Standard Instrumentation Package (SIP), which has been provided by the balloon. In the SIP, the payload will be able to communicate with the rest of the gondola. Because the processor unit and power supply from the gondola is being isolated from the SIP, the payload will need to supply them.

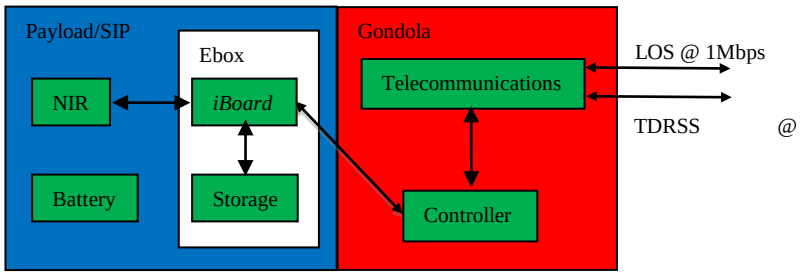

Figure 4: Top level diagram

Figure 4 shows the top level diagram of the system. The mission payload will reside in the science portion of the gondola. The science payload includes the NIR camera, the FPGA based reconfigureable on-board data processing platform *iBoard*, and stroage. Its mission is to capture data on daytime aurora so the scientists can analyse the results. The model will need to simulate the capturing of data by the NIR camera, processing and storaging functionality of the Ebox, and downlinking from the gondola.

ASSUMPTIONS/PROBLEM FORMULATION

4.1 Assumptions

The preliminary model presented in this paper assumes and simplifies the following:

1. A single NIR camera is used to capture data and left on for the whole duration of the mission.
2. One frame is captured per second
3. Image size is 625 KB
4. 6 hours of the mission is simulated
5. Aurora data fall into one of three categories
   - Not visible – frames with no aurora activity, Threshold = 1
   - Barely visible – frames with very faint aurora, Threshold = 2
   - Highly visible – frames with very obvious aurora, Threshold = 3
6. Aurora data categories are generated uniformly
7. BitMicro's E-Disk Altima Ultra320 SCSI Flash SSD specifications were used for the storage device
8. Batteries are not rechargeable
9. Data reduction is set to threshold detector, no data compression (raw images are stored)
10. Payload receives information about downlinking opportunities from the balloon's gondola
    - LOS <1% of the time, has data rate of 1 Mbps
    - TDRSS available 24/7, has data rate of 92 Kbps
11. Downlinking is always done when available
12. Data are downlinking in a FIFO basis
13. Ebox only handles processing of data and storage
14. Balloon gondola will send messages to the Ebox requesting data to be downlinked.
15. Model is only concerned with the payload side

As a reference, this model will use one camera that will continuously record data at one frame per second. The image size and downlink rates were provided by [2]. A threshold detector for aurora observations has been developed and shown to work; the model will use this method as a baseline algorithm (refer to [7] for details about method). A single simulated second maps to a single second in real time. There are 1296000 seconds in 15 days thus 1296000 frames are produced. This amount of frames is too large for the simulator to handle so 6 simulated hours or 21600 frames will be used (6 simulated hours is near the maximum data points the simulator can process).

4.2 Operational Scenario

Three major design spaces are being considered in this paper; they are storage capacity, power, and downlink. These three design spaces produce a large amount of operational scenarios. We will only consider a subset of these scenarios and consider all else a hybrid of them. The operational scenarios are as follows:

1. Store all data on-board during the entire mission without downlink it
2. All data are downlinked real time;
3. Data are process and stored first and only real-time data needs to be downlinked real time

The first scenario is the extreme case where all the data will be stored on-board. The second scenario is the opposite end of extremity where all the data is being downlinked. The third scenario is a hybrid of the first two and involves both storing and downlinking data in parallel; this will produce redundant data. The reasoning for considering these extreme cases is to gather information about them and know that any other scenarios will perform better than these.

The model will simulate the three operational scenario mentioned above based on the assumptions and simplifications mentioned earlier. The current model simulates the three scenarios and gets estimates on the storage capacity requirements as well as power usage.

4.3 Problem Formulation

The system consists of three blocks, instrument, Ebox, and external controller. The problem is how the system architecture should be designed such that all requirements will be satisfied. This section will describe three problems. Problem 1 describes a data reduction and compression architecture, Problem 2 describes the issues with power and operational states, and Problem 3 describes the issue with downlinking and the required storage capacity.

4.3.1 Data Reduction / Compression

The system consists of three blocks, instrument, Ebox, and external controller. The problem is how the system architecture should be designed such that all requirements will be satisfied. The first step in this approach is to determine the method of data processing. Before discussing this issue, let's define the difference between data reduction and data compression.

1. Data Reduction is the removal of information that **is not recoverable.**
2. Data Compression compacts the data into a smaller volume and **is recoverable to some extent** (there are compression algorithms that are *lossy* in which case, some bits are thrown away).

To put things into context with the aurora observation mission, video captured by the NIR camera does not always contain auroral activities. These "non-interesting" frames consume storage space and possibly processing power. In addition, the mission captures data on the order of terabytes (TB), which exceeds the maximum capacity of single storage devices.

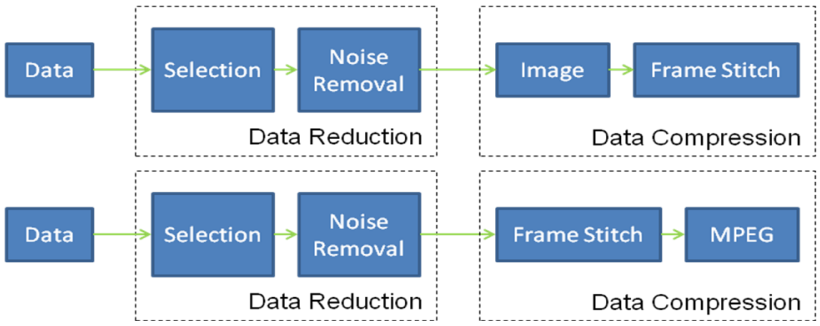

Figure 5: Data Flow

Figure 5 shows a possible data flow from processing raw data to a more manageable size. The first exploration that needs to be done is deciding whether to perform data reduction or data compression first. The order in this case matters because in the case where data reduction is performed first, fewer images will be compressed thus perhaps less processing time. However it may be the opposite case where perhaps compression should be performed first because the data reduction algorithm works better with fewer bits.

In addition to deciding whether data reduction or data compression should be done first, choosing the data reduction algorithm and data compression standard is also critical. There are several data reduction algorithm that may be considered like a threshold detector that throws away images that do not pass the threshold or an algorithm that learns the patterns an aurora and captures only those images. For data compression, there are quite a number of standards to choose from which needs to be determined.

4.3.2 Power and Operational States

Another problem to consider will be power management and operational modes. Different operational modes will draw different amounts of power. Choosing when to turn things on or off and what each operational mode will do is important.

Table 1: Power States

Operational States \ Devices	NIR Camera	E-Box
Initialization	Off	Active
Calibration	Active	Active
Standby	Standby	Standby
Active	Active	Active
Safe	Off	Off

Launch Stage
Functional Stage
Emergency
Power State

Initialization	Waking up iBoard and SSD to perform engineering telemetry
Calibration	Waking up camera and perform image/focus adjustments
Standby	Puts all devices on standby
Active	Cameras sampling the sky for aurora and data is processed and stored/downlinked
Safe	Survival mode turns everything off keeping storage intact

Table 1 shows all the possible power states the payload can go through during the mission. The possible states are: Initialization, Calibration, Standby, Active, and Safe.

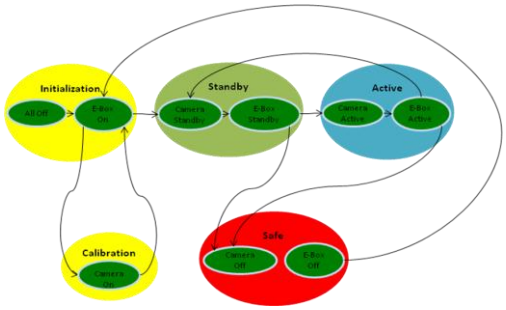

Figure 6: Power State Flow Diagram

Figure 6 shows a pictorial representation of the flow of states. The initialization state is called upon during the first few minutes of the mission. It wakes up the *iBoard* and storage device (Ebox) to perform engineering telemetry. In the initialization state, the sub-state, calibration is invoked which turn on the cameras to configure them properly. After initialization, the system will go into standby state in which both the cameras and Ebox are put into standby and is ready to capture data. The active state is the primary state of the system. In this state, the cameras and Ebox are capturing, processing and storing data. In the active state, the system has the option to go back into the standby state to conserve power. At any point after the initialization state, the system may determine that power levels are not enough to sustain operability and will go into the safe state. In the safe state, the payload is turned off, when power reach operational levels, the power states will reset and will be placed back into the initialization state.

4.3.3 *Downlink and Storage*

There are two choices of downlinking data for this mission. The line of sight (LOS) is the fastest but is only available for less than 1% of mission duration. The Tracking and Data Relay Satellite (TDRSS) is very slow but is always available. If the system does incorporate downlinking, when and how should it downlink? Downlinking will alleviate some of the load from the storage device but is not a significant amount compared to data volume captured. This model will focus on determining if downlinking will have an effect on the on-board storage device and determine the capacity needed.

VISUALSIM MODEL

VisualSim is a commercial modeling tool used to evaluate a system design [5]. The user is capable of designing complex systems and performing analysis and validation based on the concept specification. Its goal is to reduce the time in developing systems and aids the system engineer to extract low level implementation requirements from high level system architecture. Using VisualSim's hierarchy block enables specifying low level requirement without losing the big picture. VisualSim is the modeling tool that this project uses.

The approach for designing these models derive from the first two phases of the ISAAC methodology. Phase one starts with Figure 4. It shows the top level design that we would like to implement. Based on the assumptions and operational scenarios stated earlier, we can implement the design using VisualSim.

5.1 Top Level Balloon-based Model

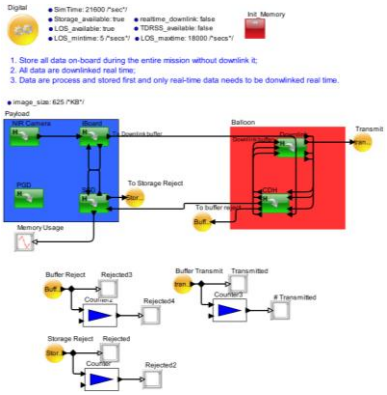

Figure 7: High Level Diagram

Figure 7 shows the VisualSim implementation of the top level system in Figure 4. The system can be broken down into two separate entities, the payload and gondola. Commands from the gondola are issued to the Ebox giving it information on when it is safe to turn on or off and requests data for downlinking when it becomes available. The Ebox will manage the payload's power and operational states and decides when to downlink. Components that make up the payload are the NIR camera, *iBoard*, storage device and the battery. The balloon gondola is assumed to have telecommunication equipment used for downlinking and a command and data handling (CDH) unit.

The payload's top level block diagram can be seen in the blue box. Data is generated by the NIR camera and processed in *iBoard*, the FPGA based data processing platform. From *iBoard*, data is sent to storage (SSD) or to the balloon's gondola for downlinking. The power generation and distribution (PGD) block holds the battery and monitors the usage.

The gondola consists of the downlinking unit and CDH. The downlinking unit detects when and which type of downlink is available and for how long. It is also responsible for receiving uplinked data and transmitting downlinked data. The CDH receives telemetry from its telecommunication unit and passes it on to the payload. It also notifies the payload when downlinking is available and if there were any uplinked messages.
Since simulating 15 days with one second resolution (1296000 points for one camera) is too much to handle with VisualSim, the total time spans 6 simulated hours (21600 points). Toggling the parameters Storage_available, realtime_downlink, LOS_available, TDRSS_available, LOS_mintime, and LOS_maxtime capture the three mission scenarios. Table 2 lists these parameters and provides a description of how it is being used.

Table 2: Operational Scenario Description

Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description
FALSE	FALSE	X	X	Does nothing
FALSE	TRUE	FALSE	FALSE	Does nothing
FALSE	TRUE	FALSE	TRUE	All data are real-time downlinked via. TDRSS
FALSE	TRUE	TRUE	FALSE	All data are real-time downlinked via. LOS
FALSE	TRUE	TRUE	TRUE	All data are downlinked real-time with both TDRSS & LOS downlink, but not at the same time
TRUE	FALSE	FALSE	FALSE	Data is stored
TRUE	FALSE	FALSE	TRUE	All data are stored first then downlinked via. TDRSS
TRUE	FALSE	TRUE	FALSE	All data are stored first then downlinked via. LOS
TRUE	FALSE	TRUE	TRUE	All data are stored first then downlinked via. TDRSS & LOS but not at the same time
TRUE	TRUE	FALSE	FALSE	All data are stored
TRUE	TRUE	FALSE	TRUE	Data are stored and downlinked via TDRSS in parallel
TRUE	TRUE	TRUE	FALSE	Data are stored and downlinked via LOS in parallel
TRUE	TRUE	TRUE	TRUE	Data are stored and downlinked via TDRSS & LOS, but not at the same time, in parallel

For notational purposes in the results section, the settings in blue will be Configuration 1, red will be Configuration 2, green will be Configuration 3, purple will be Configuration 4 and peach will be configuration 5. The configurations with both *Storage_available* and *realtime_downlink* set to true is not displayed in the results section because it yields the same results as Configuration 1 and 2 (*buffer_size)* and Configuration 3 (*storage_size*). The rest of the unassigned configurations have not been implemented into this preliminary system design.

5.2 Low Level System Models

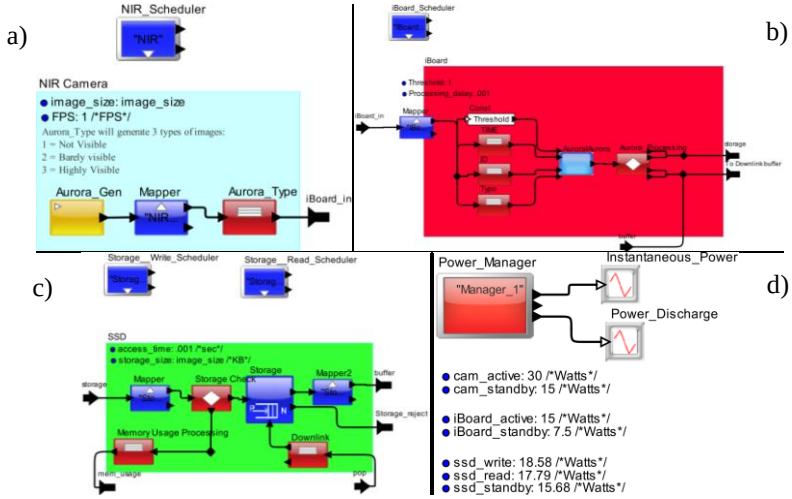

Figure 8: a) NIR camera b) *iBoard* c) SSD d) PGD

Figure 8 shows the low level detail implementation of the payload. Figure 8a shows the internal blocks of the NIR camera, 7b shows the internal block of *iBoard*, 7c shows the internal block of the storage and 7d shows the intern block for PGD. Figure 8 reside inside of VisualSim's hierarchy model denoted by the green blocks with an arrow and an 'H' as shown in Figure 7. This is a nice feature of VisualSim because the high level block diagram can be kept while still having functionality in the lower levels.

5.2.1 NIR Camera

Figure 8a consists of a traffic generator (yellow), a mapper (blue), and a processing block (red). Together it simulates the NIR camera. Data is being produced by the traffic generator every second based on the assumption that the camera takes images once every second. The traffic generator generates data in a data structure format, which contains information such as time and ID.

The data structure was extended to include information on the type of image. Data flows through the mapper, which is needed for simulations purposes. It allows different hierarchical blocks to run simultaneously instead of sequentially i.e. parallel processing. Mappers do not have any delay thus the data comes right out to the processing block. Here, the data is classified into one of three categories: not, barely, and highly visible. The processing block generates these three categories uniformly; this assumption was made to simplify the problem. It assigns 'not visible' to 1, 'barely visible' to 2, and 'highly visible' to 3 in **Type** of the data structure. Parameters of this hierarchical block are *image_size* and *FPS*. *image_size* is set to the provided information of 625 KB and *FPS* (frames per second) is set by assumption.

5.2.2 iBoard

Figure 8b models the *iBoard* FPGA based data processing platform. It consists of a mapper (blue), several processing blocks (red), and a CustomC block (light blue). After the data is categorized into one of the three types of frames, it is sent to the *iBoard* for processing. Once

again, data first flows through the mapper, which allows the data to be processed simultaneously as the camera block is producing data. Immediately after, the data is parsed into three points of interest, **TIME**, **ID**, and **Type**. **TIME** provides information on when the data generated and **ID** is the data's unique number. **Type** describes which category the data falls under (not visible, barely visible, or highly visible). The processing algorithm is a threshold detector. Three key information are passed along with the threshold value to the CustomC block. Inside the CustomC block is a C program that compares the threshold value to the Type. If the Type is larger than or equal to the threshold value, then the data is passed otherwise the data is thrown away. Data coming out of the CustomC block flows through the last processing block, which determines where to route the data. The possible options are to route directly to storage only, balloon gondola only, or both storage and gondola. Parameters in this hierarchy block consist of *Threshold* and *Processing_delay*. The *Threshold* value is a number between 1 and 3, which represents the three different types of images. *Processing_delay* is a parameter that describes how long in seconds the processor takes to perform an algorithm.

5.2.3 *Storage*

Figure 8c shows the model for the storage device. It is assumed that the mission will be using a solid-state device (SSD). This hierarchical block contains several mappers (blue), processing blocks (red), and a Smart Resource Block (blue with queuing diagram). Data from the *iBoard* goes through the mapper then through a memory usage counter that increment each time when data passes through. The data is then sent to the Smart Resource Block where it is being stored. The Smart Resource Block is configured as a very large FIFO (first-in, first-out) queue. Most of the time, the data will sit idle in this block, when *iBoard* determines it wants to downlink data, iBoard will send a message through 'pop' which takes the oldest data and sends it to the gondola. When that happens, the memory usage counter is decremented. The parameters in this block are *access_time* and *storage_size.* The time it takes to read and write data is characterized by the *access_time* parameter while the storage size is characterized by *storage_size*. The *storage_size* is a function of *image_size* because it is useful to know the amount of frames that can be stored.

5.2.4 *Power Manager*

Finally Figure 8d shows the model for the PGD unit. This unit consists of only the Power Manager (red). The Power Manager has a table of all the mappers being used and their power usage. Whenever a mapper is called on, the Power Manager will do a table look up for its corresponding power usage and adjust the current power level respectively. This mission assumes that the batteries cannot be recharged thus the Power Manager will only be decrementing. The parameters of this block are *cam_active, cam_standby, iBoard_active, iBoard_standby, ssd_write, ssd_read,* and *ssd_standby.* Their parameters are inputted into the Power Manger's table.

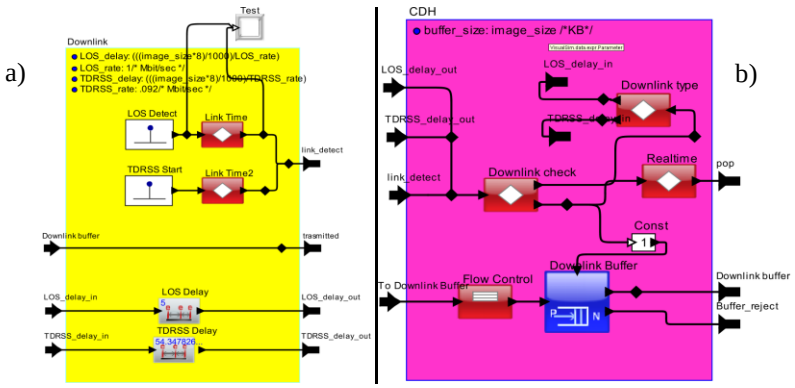

Figure 9: a) Downlink unit b) CDH unit

Similar to Figure 8, Figure 9 are hierarchy blocks that reside in Figure 7 under the balloon's gondola. Figure 9a shows the downlinking unit and Figure 9b shows the CDH unit. Both of these units are part of the gondola and modeled as an external controller to the payload.

5.2.5 *Downlink Unit*

Figure 9a shows the model for the balloon's gondola downlink unit. It consists of single event generator (white), processing blocks (red) and delay (gray). A flag is generated at a random time by the single event generator however for simplicity, both the single event generator will generate a flag at the start of the simulation. This flag will represent the telecommunications detecting available downlink. The flag goes through the processing blocks, which calculates the duration of the downlink and passes that information to the CDH unit. When *iBoard* decides to downlink, data will be pulled from the storage and passed to this module for downlinking. The parameters of this block are *LOS_delay*, *LOS_rate*, *TDRSS_delay*, and *TDRSS_rate*. *LOS_rate* is a given parameter and *LOS_delay* is calculated using *LOS_rate*. Similarly *TDRSS_rate* is given and *TDRSS_delay* is calculated using *TDRSS_rate*. These parameters model the time it takes to downlink a single frame.

5.2.6 *CDH Unit*

The CDH unit on the gondola is the main processing unit. It is responsible for keeping the balloon alive and functional. CDH will provide information about the balloon's location, power status, and downlink availabilities. The model assumes that the payload has no control over CDH. This model consists of several processing blocks (red) and a Smart Resource Block (blue with queuing diagram). A flag from the downlink unit will be sent to CDH notifying that a downlink option is available. CDH will relay this message to *iBoard* and waits for *iBoard's* decision. If downlink is taken, CDH will pull data from the storage and puts it into the Smart Resource Block buffer. Based on which high level parameters are toggled, the CDH block will determine which downlink will be used and routes the paths accordingly. CDH determines how much data can be downlinked by using the information generated by the downlink unit. It applies flow control and buffers as much data that can be transmitted in the window of opportunity. The only parameter in this model is the *buffer_size,* which is a function of *image_size* to determine how many frames are being sent.

RESULTS

The current model explores the storage size usage for each of the three scenarios. These results are for simulated 6 hour mission time during active state. The reason for this is because 6 simulated hours is how much the simulator can process. In addition, the active state shown in Figure 6 is the state which the system will be in most of the time so the simulator will model a typical operation. Both downlinks are available from the start of the simulation and are always taken. The image size is taken to be 625KB and a frame is captured every simulated second. The purpose of this experiment is to determine the required storage size under the configurations shown on Table 2 and with varying threshold.

In the first case where only downlink is available, the payload storage size is 'X' because it does not matter what the size is. The number of frames rejected plus one multiplied with the image size gives the buffer size on the gondola. These numbers are determined by setting *buffer_size* to equal *image_size* and viewing the number of rejected data. The number of rejected data will tell us how many frames need to be stored plus one. The same technique was used in determining the storage size. In the cases where *Storage_available* is true and *realtime_downlink* is false, *buffer_size* is set to *image_size* because the downlinked data are requested one at a time and so the gondola requires a buffer size of at least one frame. In cases where both *Storage_available* and *realtime_downlink* is true, the *storage_size* and *buffer_size* are exactly the same as when one is true and the other is false. This is because when *realtime_downlink* is true, data flows in parallel to both the storage and CDH and the payload storage and gondola's buffer are independent.

Table 3: Threshold = 1

Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description
FALSE	TRUE	FALSE	TRUE	All data are real-time downlinked via. TDRSS
Storage Size	**Buffer Size**			
X	**390**			
Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description
FALSE	TRUE	TRUE	FALSE	All data are real-time downlinked via. LOS
Storage Size	**Buffer Size**			
X	**2880**			
Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description
TRUE	FALSE	FALSE	FALSE	All data are stored
Storage Size	**Buffer Size**			
21599	**625 KB**			
Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description
TRUE	FALSE	FALSE	TRUE	All data are stored first then downlinked via. TDRSS
Storage Size	**Buffer Size**			
21202	**625 KB**			
Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description
TRUE	FALSE	TRUE	FALSE	All data are stored first then downlinked via. LOS
Storage Size	**Buffer Size**			
17999	**625 KB**			

Table 3 shows the case where none of the frames were rejected thus will be considered the baseline. Based on the first two results, we can see that LOS downlink will transfer more data than TDRSS. We can also conclude that if downlinking data in real-time were chosen, we would need a buffer size of at least 1.8 GB per 6 hours. The latter three results are when data is stored first and waits to be downlinked. In the worst case where no downlink is available, the storage size required to hold all the data is 13.5 GB per 6 hours. The best case among the latter three configurations is when data is being stored and LOS downlink is being used. This configuration requires a storage capacity of ~11.4 GB per 6 hours which means LOS downlink offloaded 2 GB of data per 6 hours.

Table 4: Threshold = 2

Storage Available	Real-time Downlink	LOS Available	TDRSS Available	AVG # of Rejected	SD
FALSE	TRUE	FALSE	TRUE		
			Storage Size	X	
			Buffer Size	385.5	0.55
Storage Available	Real-time Downlink	LOS Available	TDRSS Available		
FALSE	TRUE	TRUE	FALSE		
			Storage Size	X	
			Buffer Size	2512.6	9.69
Storage Available	Real-time Downlink	LOS Available	TDRSS Available		
TRUE	FALSE	FALSE	FALSE		
			Storage Size	14449.5	62.12
			Buffer Size	1	
Storage Available	Real-time Downlink	LOS Available	TDRSS Available		
TRUE	FALSE	FALSE	TRUE		
			Storage Size	13922	70.29
			Buffer Size	1	
Storage Available	Real-time Downlink	LOS Available	TDRSS Available		
TRUE	FALSE	TRUE	FALSE		
			Storage Size	10786.5	75.15
			Buffer Size	1	

Table 4 is the case where frames with no event are not stored. Each configuration was ran seven times and averaged; the standard deviation was also calculated. Similar to Table 3, when downlinking real-time data, LOS can downlink more data but requires a larger buffer size. Comparing this configuration with Table 3's, the buffer size is ~1.6 GB per 6 hours which is ~0.2 GB less. In terms of storage, the worst case requires ~9 GB per 6 hours, ~4 GB per 6 hours less than in Table 3. The best case where LOS downlink is being used to offload storage requires a storage capacity of ~6.7 GB per 6 hours, nearly half the capacity required in Table 3.

Table 5: Threshold = 3

Storage Available	Real-time Downlink	LOS Available	TDRSS Available	AVG # of Rejected	SD
FALSE	TRUE	FALSE	TRUE		
			Storage Size	X	
			Buffer Size	374.71	.76
Storage Available	Real-time Downlink	LOS Available	TDRSS Available		
FALSE	TRUE	TRUE	FALSE		
			Storage Size	X	
			Buffer Size	1499	7.07
Storage Available	Real-time Downlink	LOS Available	TDRSS Available		
TRUE	FALSE	FALSE	FALSE		
			Storage Size	7189.29	101.82
			Buffer Size	1	
Storage Available	Real-time Downlink	LOS Available	TDRSS Available		
TRUE	FALSE	FALSE	TRUE		
			Storage Size	6786	29.72
			Buffer Size	1	
Storage Available	Real-time Downlink	LOS Available	TDRSS Available		
TRUE	FALSE	TRUE	FALSE		
			Storage Size	4074.57	47.44
			Buffer Size	1	

Table 5 is the case where frames with highly visible aurora are stored and the rest are rejected. Similar to Table 4, each configuration was tested seven times and averaged with standard deviation calculated. When downlinking real-time data, LOS transferred the most, requiring ~0.9 GB per 6 hours of buffer capacity. This was approximately half the capacity required in Table 3. The worst case for storage required ~4.5 GB per 6 hours, approximate 1/3 of Table 3. The best case for storage requires ~2.5 GB per 6 hours which is approximately 1/4 of Table 3.

Table 6: Storage and Buffer Size Summary

Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description – Configuration 1
FALSE	TRUE	FALSE	TRUE	All data are real-time downlinked via. TDRSS
Image Type	**Storage Size (KB)**	**Buffer Size (KB)**		
1	0	243750		
2	0	241607.1429		
3	0	234821.4286		
Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description – Configuration 2
FALSE	TRUE	TRUE	FALSE	All data are real-time downlinked via. LOS
Image Type	**Storage Size (KB)**	**Buffer Size (KB)**		
1	0	1800000		
2	0	1572678.571		
3	0	937500		
Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description – Configuration 3
TRUE	FALSE	FALSE	FALSE	All data are stored
Image Type	**Storage Size (KB)**	**Buffer Size (KB)**		
1	13499375	625		
2	9022321.429	625		
3	4493928.571	625		
Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description – Configuration 4
TRUE	FALSE	FALSE	TRUE	All data are stored first then downlinked via. TDRSS
Image Type	**Storage Size (KB)**	**Buffer Size (KB)**		
1	13251250	625		
2	8738125	625		
3	4241875	625		
Storage Available	Real-time Downlink	LOS Available	TDRSS Available	Description – Configuration 5
TRUE	FALSE	TRUE	FALSE	All data are stored first then downlinked via. LOS
Image Type	**Storage Size (KB)**	**Buffer Size (KB)**		
1	11250000	625		
2	6750803.571	625		
3	2547232.143	625		

Table 6 compares the results with respect to each configuration. In Table 3, the two worst case scenarios Configurations 1 and 2 (strictly downlinking) and Configuration 3 (strictly storing), the required storage size was ~13.5 GB and buffer size was ~1.8 GB. Looking at Configurations 4 and 5 in Table 3 (data is stored first then downlinked), we see that there was a slight improvement in storage capacity but as the threshold becomes higher, the improvement

increases. Keeping in mind that this is for 6 hours, a 15 day mission will require ~810 GB of storage capacity and ~108 GB of downlink buffer. To make matters worse, this model only has one camera; the mission will require four cameras with larger frame size. We can see based on the results that downlinking data in real-time will require a relatively large buffer while downlinking from the storage unit can offload some of the storage requirements in the Ebox.

These results implement the threshold detector data reduction architecture with no compression. Seeing that the data is not being reduced sufficiently, Problem 1 will need to be investigated further. The assumption for made for the camera was to let it be continuously on. Modifying Problem 2 such that cameras can be turned off and on during the operational state may reduce some of the data volume. Finally Tables 3 through 6 directly relate to Problem 3. From these tables we were able to see the effects of downlinking and get a better understanding what the storage requirements are.

FUTURE WORK

This paper has explored the first part of the ISAAC methodology of high level design simulation. Through modeling and simulations, we can conclude that downlinking data from the storage device may prove useful in reducing the storage capacity required. Downlinking real-time data however does not improve the storage capacity and will require a large buffer size. After determining the final system design the next step is to continue with ISAAC methodology. The high level designs will be mapped into the low level FPGA implementation.

ACKNOWLEDGEMENTS

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, and was sponsored by the Hawaii Space Grant Consortium and the National Aeronautics and Space Administration.

REFERENCES

[1] Zhou, X. Y, D. Lummerzheim, G. R Gladstone, S. D. Gunapala, S. B. Bandara, J. Trhne, L. Herrell (2008), Magnetospheric application of high-altitude long-duration balloon technology: Daylight auroral observations, *Adv. Space Res., 42*, 1676-1682

[2] Zhou, X. Y, D. Lummerzheim, G. R Gladstone, S. D. Gunapala (2007), Feasibility of observing dayside aurora using NIR camera onboard high-altitude balloons, *Geophys. Res. Lett, 34*, L03105, doi:10.1029/2006GL028611

[3] Gorham, P *et al*., "Antarctic Impulsive Transient Antenna: A Long Duration Balloon High Energy Neutrino Observatory," University of Hawaii at Manoa., Honolulu, HI, 2003

[4] He, Y *et al.*, "ISAAC: Highly-Reusable, Highly-Capable, Integrated Instrument Control-and-Computing Platform," Jet Propulsion Laboratory., Pasadena, CA, 2008

[5] *VisualSim.* Mirabilis Design [Online]. Available at http://www.mirabilisdesign.com.

[6] ***M****eV* ***A****uroral* ***X****-ray* ***I****maging and* ***S****pectroscopy 1999/2000 Long Duration Balloon Flight Around the South Pole.* University of Washington., Seattle, WA, 2000

[7] Lim C., Detecting Daytime Aurora Using Digital Image Processing, *Class Project Report,* Oct. 2009.