EECS Capstone

Aerospace Recorder for Graphical History

With Nvidia Jetson AGX Xavier

Anthony Kung, Caden Friesen, and Henry Chen

Revision 2: 28th January 2022

1. Overview

1.1. Executive Summary

The primary objective of our project is to create a graphical history (video) recorder for large commercial and military drones. The goal for our project is to be able to save the data from a GigE Vision camera and a USB3 thermal camera using the Nvidia Jetson AGX Xavier platform to allow other people to review it later. This is to facilitate an aerial reconnaissance mission where a large drone will be able to record aerial graphical data by taking multiple images every second. Our system needs to record, store, organize, and self monitor data collection to ensure whole hours of images can be taken without any errors. Our system is robust to power failures and has variables saved in ways that allow easy continuation in data from where the system left off when it lost power.

The system consists of multiple hardware components provided by Collins Aerospace: the Single Board Computer Nvidia Jetson AGX Xavier, the GigE Vision camera Imperx C3210, the USB Type C camera Flir Boson320, the power supply module V24C12T150BL with an evaluation board, a PoE switch TP-Link TL-SG1005P, and finally a GigE Vision camera from Flir which is regulated under ITAR regulations. One of the main goals of the project was to integrate all of these pieces into one system. Our power supply module powers the Jetson as well as the PoE switch, the PoE switch is powering the GigE Vision camera and the Jetson is powering the USB-C camera. A user interface is attached to the Jetson for basic operations, including recording time delays, a manual recording mode, a recording length setting, and saved preferred settings. Our system includes a Primary Executable, or APE in short, that will manage the entire software operations and runs upon system startup. The cameras take videos by capturing each video frame as an individual image, this process is done at high speed which allows for the conversion to video after. These images of video frames will be processed by the Storage Block to be organized into an SSD storage medium onboard the Jetson.

1.2. Team Contacts, Communication Protocols, and Standards

Team contact information

Table 1.1: **Team Roaster**

Team Member	Email	Block
Henry Chen	chenjunh@oregonstate.edu	1. Jetson Power Supply 2. Jetson GPIO

Team Member	Email	Block
Anthony Kung	kungc@oregonstate.edu	1. Primary Executable 2. GigE Camera
Caden Friesen	frieseca@oregonstate.edu	1. USB Camera 2. Jetson Storage

Table 1.2: **Team Role**

Role	Role description	Role Owner
Chief Finance Officer	Taking care of the budget, make sure it is not overspending. Ordering stuff for the team.	Caden Friesen
Project Partner Communicator	In charge of communication between the team and the project partner.	Anthony Kung
Team Meeting Coordinator	Making a schedule for the team meeting. Writing down notes during the meeting and making sure everyone is able to attend the meeting.	Caden Friesen
Project Progress Video Owner	In charge of editing the video for the weekly report. And make sure the video met the requirements.	Anthony Kung

Communication preferences of our project partner

- We will communicate via email. All of our email addresses will be included in each email so that all members can communicate and see all communication.
- We will use a google slides presentation to document progress, questions, and challenges that our project partner can check in on at any time.
- Calls with our project partner can take place on Zoom and will only be scheduled as needed and preferably early in the morning.

- Our communication just within our group will be done mostly through discord.

Table 1.3: **Team Communication Protocols and Standards**

Topic	Protocol	Standard
Time Management and Challenges	If a team member is running behind or is stuck on a task before a deadline, then they should contact the group to receive help on the task.	Work will be done by deadlines unless reassessed by the whole group.
Quality of Work	To avoid a team member's work not being of good quality, regular checkups of their blocks and sharing of work accomplished should be maintained.	Work will be completed to a professional standard that is expected by the project partner.
Division of Work	As new assignments are posted the team will meet to determine how to split up work equally among all members.	Team members will contribute an equal amount of work to the project to the best of their ability.
Decision-Making	Team decisions should be agreed on by the majority of the team. Team members should contribute and make informed decisions. Team members will make sure they understand what is being proposed and ask questions before agreeing to or voting on decisions.	Team members would participate in team meetings and contribute to the decision meeting to the best of their knowledge and the information provided. Team members will make sure they understand what is being proposed
Conflict Resolution	In the event of a conflict, all team members should participate to resolve the conflict by providing helpful input. Should the conflict be unable to be resolved by the team, the team should contact an external mediator such as an instructor to	Team members would communicate in a professional way to express concerns. Conflicts will not be personal within the group and be resolved methodically.

	attempt to resolve the conflict.	
Communication	Before new documents or project groups are created it will be discussed what is available to each team member to make sure all will have access.	Communication will be carried out in a way accessible to all team members.

1.3. Gap Analysis

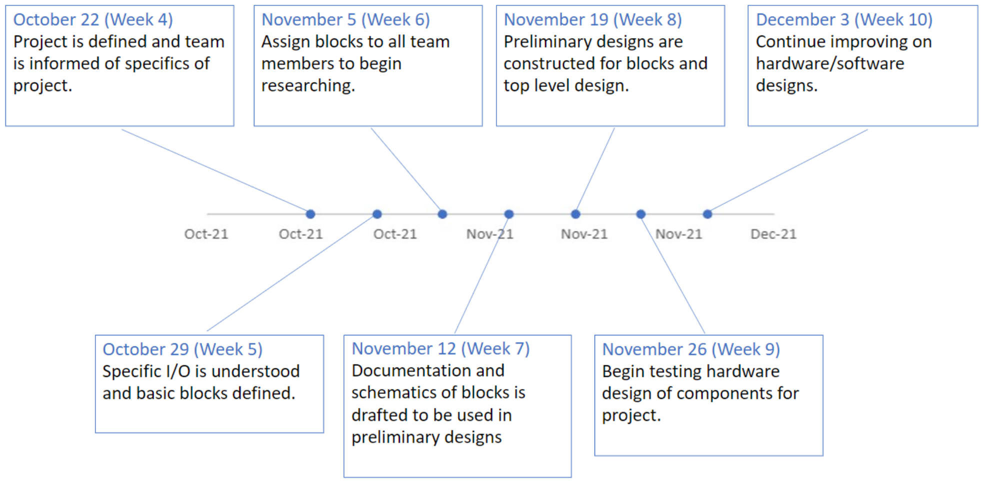
This project exists to fill a need for a graphical data recorder for large drones using GigE Vision and USB3 Vision industrial cameras. The project partner requires a system that is capable of recording graphical data using the Jetson Single Board Computer and the 3 specified cameras. Our project will fill this gap by integrating multiple system SDK into a single program which allows the Jetson to manage all 3 cameras on the drone. The recorded graphical images will be stored physically on an SSD for later viewing. Unfortunately, due to the sensitive nature of our project, we do not have any specifics of the exact purpose of the project.

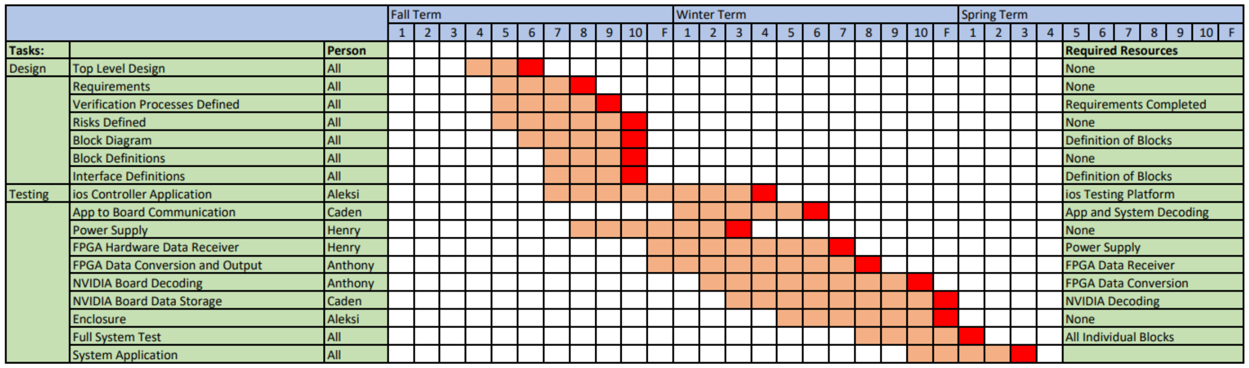
Since our system has to be self-sustainable and easily installable on the drone while supporting 3 different cameras. We will need to create a modular design in both hardware and software to allow this. We also need to make sure that the system can boot on power up and run the necessary programs including APE without user interaction. And our programs should be developed with the consideration of a power failure to ensure our Jetson does not get impaired during abrupt disconnections and allow auto recovery upon power restoration.

From what we have learned from the project partner, this project intends to take aerial footage in remote regions with a large jet-size drone. Our project will provide valuable data for Collins and provide an understanding of how to interface multiple camera protocols on the Jetson platform for their future implementation and product development. Collins Aerospace specializes in creating camera systems for commercial and military aircraft, the result of this project will be very helpful to the future development of Collins' products.

1.4. Timeline/Proposed Timeline

The timeline of the project extends throughout three terms each being 10 weeks long beginning with problem definition, transitioning into the preliminary design and system testing, and ending with the final presentation and documentation. Below is a proposed timeline for the fall term along with a secondary timeline of the full 30-week project. Editing this document now after the first two terms this timeline serves as a very good example of how things don't always go according to an original plan.


October 22 (Week 4)
Project is defined and team is informed of specifics of project.
November 5 (Week 6)
Assign blocks to all team members to begin researching.
November 19 (Week 8)
Preliminary designs are constructed for blocks and top level design.
December 3 (Week 10)
Continue improving on hardware/software designs.
Oct-21
Oct-21
Oct-21
Nov-21
Nov-21
Nov-21
Dec-21
October 29 (Week 5)
Specific I/O is understood and basic blocks defined.
November 12 (Week 7)
Documentation and schematics of blocks is drafted to be used in preliminary designs
November 26 (Week 9)
Begin testing hardware design of components for project.



			Fall Term											Winter Term											Spring Term										
			1	2	3	4	5	6	7	8	9	10	F	1	2	3	4	5	6	7	8	9	10	F	1	2	3	4	5	6	7	8	9	10	F
Tasks:		**Person**																											**Required Resources**						
Design	Top Level Design	All				🟧	🟧	🟥																					None						
	Requirements	All					🟧	🟧	🟧	🟥																			None						
	Verification Processes Defined	All					🟧	🟧	🟧	🟧	🟥																		Requirements Completed						
	Risks Defined	All					🟧	🟧	🟧	🟧	🟧	🟥																	None						
	Block Diagram	All						🟧	🟧	🟧	🟧	🟥																	Definition of Blocks						
	Block Definitions	All							🟧	🟧	🟧	🟥																	None						
	Interface Definitions	All							🟧	🟧	🟧	🟥																	Definition of Blocks						
Testing	ios Controller Application	Aleksi							🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟥												ios Testing Platform						
	App to Board Communication	Caden												🟧	🟧	🟧	🟧	🟧	🟥										App and System Decoding						
	Power Supply	Henry								🟧	🟧	🟧	🟧	🟧	🟧	🟥													None						
	FPGA Hardware Data Receiver	Henry											🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟥									Power Supply						
	FPGA Data Conversion and Output	Anthony											🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟥								FPGA Data Receiver						
	NVIDIA Board Decoding	Anthony													🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟥						FPGA Data Conversion						
	NVIDIA Board Data Storage	Caden														🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟥					NVIDIA Decoding						
	Enclosure	Aleksi																🟧	🟧	🟧	🟧	🟧	🟧	🟥					None						
	Full System Test	All																			🟧	🟧	🟧	🟧	🟥				All Individual Blocks						
	System Application	All																					🟧	🟧	🟧	🟧	🟥								

1.5. References and File Link

1.5.1. References

[1] "GigE Vision," Wikipedia, 03-Sep-2021. [Online]. Available: https://en.wikipedia.org/wiki/GigE_Vision. [Accessed: 12-Mar-2022].

[2] "USB3 vision," Wikipedia, 01-Sep-2021. [Online]. Available: https://en.wikipedia.org/wiki/USB3_Vision. [Accessed: 12-Mar-2022].

1.5.2. File Links

N/A

1.6. Revision Table

3/12/2022	Anthony Kung - Changed Executive Summary - Changed Gap Analysis - Added references
3/12/2022	Henry: edit the gap analysis
3/12/2022	Caden Friesen: Edited timeline piece. Reformatted section
3/6/2022	Anthony Kung - Changed Executive Summary - Updated Team Communication Standards & Protocol - Changed Gap Analysis - Removed old terms to avoid references to slavery in support of GitHub initiative
12/3/2021	Aleksi Hieta: Added revised timeline
12/3/2021	Henry: edit section 1.1 and 1.2, added team role.
11/19/2021	Henry: Added section 1 and change it from the feedback
11/12/2021	Caden Friesen: Added a team standard and edited 1.3. Added more to 1.1
11/12/2021	Aleksi Hieta: Made edits to 1.1, 1.3, and added broad timeline
10/29/2021	Anthony Kung - Incorporated instructor feedback - Incorporated project partner feedback

10/22/2021	Everyone: Edited sections 1.1 and 1.2
10/22/2021	Aleksi Hieta: Added Proposed Timeline
10/21/21	Henry: Added Executive Summary and Team Communication Protocols and Standards
10/21/21	Caden Friesen: Added gap analysis, references, and revisions table sections

2. Requirements Impacts and Risks

2.1. Requirements

1. Project Partner Requirement: Ability to record and store camera stream for Boson320

Engineering Requirement: The Jetson will capture video frames from the Boson320 camera and the video frames will be stored as image files at a 10FPS minimum in 320x256 pixel images.

Verified By:

1. Connect the Boson320 via USB C port on the Jetson along with the Imperx C3210 via the PoE switch.
2. Verify APE runs on startup
3. Set the recording time to 1 minute using the user interface.
4. View data recorded on the Solid State Drive and verify there are at least 600 photos.
5. Check that saved images are 320x256 pixel images.

2. Project Partner Requirement: Ability to acquire image from Imperx GigE Vision Camera

Engineering Requirement: The Jetson will capture image data from the Imperx camera and store it as an Imperx RAW image file with the original resolution of 3216 x 2208 pixels at up to 16 images per second.

Verified By:

1. Connect the camera to PoE switch

2. Connect Jetson to PoE switch
3. Power up Jetson board
4. Verify APE runs on startup
5. Verify images have been taken to /data/Imperx directory.

3. Project Partner Requirement: The device will store preferred settings to allow easy resetting.

Engineering Requirement: The Jetson will store the previously set recording time and preset delay time in a file. These should be able to be set with the user interface.

Verified By:

1. Powering on the Jetson
2. Configure the recording time and preset delay time to 60 seconds
3. Verify the configuration has been written to the file by checking the .txt files for the previous record and previously.
4. Power down the Jetson
5. Powering up the Jetson again
6. Verify the configuration is correctly restored when pressing the set to the previous button on the user interface.

4. Project Partner Requirement: The device will function electrically with what power is available on an aircraft without the need for extra pieces.

Engineering Requirement: The power supply block will take an input voltage of 28V with a 5V margin to produce a steady 12V supply for the Jetson.

Verified By:

1. Connect the DC voltage source to the input terminals of the Power Supply Unit.
2. Connect the PSU output wire to the NVIDIA Jetson.
3. Verify that Jetson runs the program and starts up with no noticeable abnormalities. This means being able to run the program and record the files.
4. Monitor the operating input voltage to see if the device is able to handle the variety of input voltage. The expected input from the aircraft is 28V DC. Vary the input voltage from 26V to 30V.

5. Project Partner Requirement: The system will have a GUI that provides functionality for setting up the device for future recording sessions.

Engineering Requirement: The system will allow 9 out of 10 users to perform the following functions: Manually starting/stopping recording, setting a preset delay for recording, setting a preset recording length, and resetting to previous settings.

Verified By:

There will be four buttons: A menu button for switching between options, a select button for performing actions, a plus button for adding, and a minus button for subtracting. This step-by-step guide will not explain how to perform each action by each button press but will only use the buttons above.

1. Connect the Boson320 to the NVIDIA Jetson with a USB C port.
2. Press the select button on the start/stop recording option. The status indicator on the display should change to stop. Verify on the computer monitor that the recording has stopped by no more files appearing on the solid-state drive (recording defaults to start on boot).
3. Add 1 minute to the preset delay option make sure to press select on these
4. Add 2 minutes to the recording length option, then verify that the option can be moved back down to 1 minute.
5. Press the select button on the start/stop recording option. The status indicator on the display should change to waiting, and the display should begin counting down the preset delay.
6. Once the preset delay reaches 0 the status should change to recording and this should be verified by viewing files being added to the solid-state drive. At this point, the preset recording length should begin counting down.
7. Verify that when the recording length reaches 0 the status changes to stopped.
8. Select the reset to the previous option. Verify that the one-minute recording length and preset delay reappear on the display.

This process also needs to verify that users can use it. A small manual for the functions of buttons will be made and given to participants along with the device.

1. Attach the Boson320 and Imperx camera to the NVIDIA Jetson with the USB C port and PoE switch.
2. Set the delay time to 5 minutes and the record time to 0 minutes.
3. Run the Primary Executable allowing the cameras to record.

4. Hand the participant a list of small tasks they will pass if they are able to accomplish all of these tasks without help.
    a. Stop the recording
    b. Change the delay to 1 minute.
    c. Change the recording time to 2 minutes.
    d. Resume the recording.
    e. At any point after this return the system to previous settings.
5. This will be verified if 9/10 people can accomplish these tasks.

6. Project Partner Requirement: Video is stored on an SSD in an organized fashion.

Engineering Requirement: The system will output image files to an SSD that will be organized by camera number.

Verified By:

1. Attach the Boson320 and Imperx camera to the NVIDIA Jetson with the USB C port and PoE switch.
2. Run the Primary Executable allowing the cameras to record.
3. Stop the recording after 10 or more seconds
4. Check that files are stored in the folder on the solid state-drive named "CameraX" where X is the number assigned in the primary executable included in the title (this number should be verified during a verification test).
5. Check that both cameras are stored properly in different folders.
6. Check that none of the files are still stored on the local memory.
7. Run this test one more time with this variation:
    a. Run the test with the folder for the camera deleted and verify this folder is created on running.

7. Project Partner Requirement: Images saved will have a way to easily check what images were taken at the same time.

Engineering Requirement: The system will output images with the timestamps on the file name.

Verified By:

1. Attach the Boson320 and Imperx camera to the NVIDIA Jetson with the USB C port and PoE switch.
2. Run the Primary Executable to record the cameras.
3. Stop the recording after one minute.
4. Check for the files stored in the Camera1 folder on the solid-state drive.
5. Verify that the files have a range of timestamps one minute long across all their names.

8. Project Partner Requirement: The system will be able to start recording when it gains power.

Engineering Requirement: The system will operate without user intervention when power is applied. If a preset delay or recording length was chosen this feature will continue from where it left off 10 out of 10 times.

Verified By:

1. Attach the Boson320 to the NVIDIA Jetson with the USB C port.
2. Run the primary executable to record the camera.
3. Disconnect power from the NVIDIA Jetson.
4. Reconnect power to the NVIDIA Jetson.
5. Check that recording has begun again by viewing if new files are appearing in the Camera1 folder.
6. Stop recording and create a preset delay of 1 minute, and a recording length of 1 minute.
7. Restart the system with this preset delay applied.
8. View on the display that the delay counts down from 60 to 30 seconds.
9. Repeat steps 3 and 4.
10. View on the display that the delay continues counting down from 30 seconds before it begins recording. This can be verified like step 5.
11. View on the display that the preset recording length counts down from 60 to 30 seconds.
12. Repeat steps 3 and 4.
13. Verify that once the system resumes recording it records for only 30 more seconds.

2.2. Design Impact Statement

2.2.1. Introduction

Our project is to create a camera system for a large commercial and/or military drone using industrial vision cameras. This project is created in collaboration with Collins Aerospace and will be using the hardware provided by them. This includes the Nvidia Jetson AGX Xavier platform as the controller of the system, GigE Vision cameras, and USB Vision cameras. Due to the nature of the project, we do not know its exact purpose of this project. To the best of our knowledge, this project will be used as an aerial reconnaissance system to survey the area in a remote location. This Design Impact Assessment is to evaluate the public impact caused by the development and deployment of the tool. By creating an impact assessment, project stakeholders can be aware of any potential impact this project may bring upon completion and the project design engineer will be able to look at the potential impact and include steps to mitigate it if necessary. This protects the project and the public from any undesirable outcome that can be preventable and foreseen.

The primary function of this project is to rapidly collect images on a drone using the provided hardware and store it locally to be accessed later. The project will also be modular for any expansion the end-user desires to expand the data being collected, for example, the ability to collect data with additional cameras.

2.2.2. Public Health, Safety, and Welfare Impacts

Since the primary purpose of this project are to collect graphical data or images from our industrial camera system, the impact of this project on public health, safety, and welfare is limited. The product will be a monitoring system that does not impede aircraft operations. However, since the device will be connected to an aircraft onboard power system, failure of the Power Control Unit (PCU) or the power supply unit (PSU) of the project could result in the loss of power to the aircraft. This will result in the loss of flight and the loss of aircraft, and thus, the loss of human life.

An example of flight system power failure resulting in the loss of 229 human lives would be Swissair Flight 111 [1]. The cause of this accident was believed to be a failure of the newly installed in-flight entertainment system as stated "Investigators identified evidence of arcing in the wiring of the in-flight entertainment network, but this did not trip the circuit breakers, which were not designed to trip on arcing" [2]. This arcing caused by a short circuit resulted in a fire mid-flight that destroyed the entire airplane rapidly before the pilots were able to land. To further elaborate on this point, installing a new piece of equipment on an aircraft may compromise the security of the aircraft. If wrongly installed, the new equipment could disrupt the power delivery system of the aircraft and cause power failures just like this.

The device itself can also pose a hazard such as exploding and damaging the aircraft. The FAA has a long list of aviation accidents caused by smoke, fire, extreme heat, or explosion involving lithium-ion batteries [3]. An example would be a recent laptop explosion onboard a

United Airlines flight in February 2020 [4]. Fortunately, no life was lost due to the accident, however, it did cause the aircraft to make an emergency landing. The Nvidia Jetson AGX Xavier can get very hot at times if there isn't a consideration of temperature control, the hot heat sink could come in contact with flammable material and cause harm.

If built inappropriately, our project may become a hazardous material, a catastrophe waiting to happen. To mitigate these potential hazards and risks, our project has to follow strict standards set by industries and government agencies. This includes adapting the IEEE 1156.1 standard [5] on microcomputers which has a special section for avionics, and the MIL-STD-810 military standard [6] which includes test methods for various scenarios of potential risks. We would also need to protect our system from any damages which could then cause harm to the aircraft, the Ingress Protection code will provide the necessary requirements to protect our device from water damage as well as dust that could spark a fire. The IEC 62262 standard [7] would ensure our device is rugged and durable enough to withstand the worst-case scenario of the aircraft such as turbulence or drastic maneuvers. At this point, there are no other recommendations that could effectively mitigate this risk, this is the same method used in the industries.

2.2.3. Cultural and Social Impacts

In this project, the Nvidia Jetson contains computer chips, but many chip companies treat people unfairly and give workers bad working conditions. Some workers do not even have enough protection to perform their jobs safely. Many companies provide some bad working conditions and environments, those companies also often pay less money to those workers. The environment can get lots of workers sick and not pay for their medical bills. The electronic workers in India show us how huge of an impact this is to them. Sheela is one of the thousands of invisible workers who toil night and day in India's electronics sweatshops. These sweatshops do not pay nearly enough for what conditions the workers are put through and how much money the companies make. “The industry's total production figure of US$21 billion (in both hardware and software) for the 2002– 2003 fiscal year did not mean much to her or to the many other thousands of workers, who find it difficult to make ends meet.”[8]. This shows they are paid way less than other people in the world. They work hard, they mostly work more than 8 hours a day. And then no one takes care of their health.

What we can do for our project is responsibly buy cameras and other stuff from companies who treat their workers well, so that other companies will start to treat the workers better because they want to sell their products. We can also help spread the word about this injustice outside our project and try to make a difference.

2.2.4. Environmental Impacts

For the environmental impacts, our project is helping the company to understand how to use Nvidia to collect the data for planes and drones. This will involve lots of aircraft. Aircraft today are powered by liquid aviation fuel, made mostly from fossil fuel sources. Fossil fuel has been a big problem for the world since the 1900s. Extra carbon dioxide in the atmosphere increases the greenhouse effect. More thermal energy is trapped by the atmosphere, causing the planet to become warmer than it would be naturally. This increase in the Earth's temperature is called global warming. Our planes are still using fuel now, there are no electric planes yet. “Fossil fuel consumption-based economic development brings the tricky issue that the over-emitted CO2 emissions pose a threat to people’s health and lives (Gong et al. 2017). Nine billion tons of energy-related CO2 emissions, accounting for 60% of the global output, were produced by China in 2016; however, in 1990, China’s CO2 emissions only accounted for 5% of the global output (Gu et al. 2019). According to the China Energy Outlook: World Energy Outlook 2017 (IEA 2018), China was once again the largest contributor to global CO2 emissions.”[9] We can see the C02 emissions from burning fuel are very huge, this makes a lot of impact on the environment. C02 emissions cause global warming and it has a bad effect on the environment of the earth and humans.

For this problem, the key point to solve is to reduce emissions. All we need to do is meet the project requirements. Once our project is working, this will help Collins more efficiently test their cameras, and this will speed up the testing and reduce the number of flights they take. This will reduce lots of emissions from aircrafts in the process.

2.2.5. Economic Factors

Economically this product should help Collins Aerospace provide better equipment in the future. There is a lot to be said for planes and how important they are in our world’s infrastructure. Every year 6.8 trillion dollars of cargo is shipped by air. That is 35% of internationally traded goods by value [10]. A device that allows Collins to better check and test their camera systems could be a step in the right direction towards some major future developments. As a company like Collins develops a better camera system it is possible that the world could leap closer to self-flying planes. Self-flying planes would be very major additions to the world’s shipping power. This year Dynamic Aviation released a plane along these lines that are being tested for package delivery. It is estimated that over the life of one self-flying plane a shipping company could save 6 million dollars in costs [11]. This would help bring down shipping costs for everyone, without the need for trained pilots in every vehicle. Many more planes would be allowed to ship products at one time. This would cause a negative side effect as well though as it would remove some jobs from the market so the cost and benefit would need to be weighed against each other. It is unlikely that self-flying planes would eliminate all need for pilots so it is likely it would be a general benefit. This is just one example of the type of developments that could be made using our project. Our project really opens the door to speed up the pace of Collins’ developments which could have a great impact on the economy.

2.2.6. Conclusion

In conclusion, although our project is relatively passive and has no physical output or interaction, and also has no commercial interests and public end-user, there is still some level of risk involved. Some of these risks can be potentially deadly or damaging to the aircraft.

Proper precaution in designing and implementing this system is required to ensure that it does not impact aircraft operations nor does it cause any harm. These problems often are mitigatable, take the avionic hazards, for example, there are standards to follow which would help in mitigation. However, some such as greenhouse gas emissions from power consumption are difficult to mitigate as the power would be coming outside of our system and out of our control.

Based on the impacts predicted in this assessment, the following mitigation methods are recommended to be included in the design and development of this project:

- Follow the regulatory standards for ensuring avionics are safe and do not cause harm to the aircraft itself. The standards include IEEE 1156.1 [5], MIL-STD-810 [6], IP, IEC 62262 [7], and if possible, obtaining an FAA certification for airborne devices [12].
- Ensure the security of data transfer by encrypting transmission with TLS certificates and being in compliance with PCI DSS v3.2 [13] security standards.
- Employ renewable energy where possible and select carbon-neutral cloud service providers.
- Take necessary ESD protection measures such as wearing ESD protection wristband [14], using ESD protection pads, and including ESD protection circuit in the design [15].
- Ensure the device is rugged and protected from external environmental factors such as vibrations and turbulence.

2.3. Risks

Risk ID	Risk Description	Risk Category	Risk Probability	Risk Impact	Performance Indicator	Responsible Party	Action Plan
R1	Falling behind schedule on tasks	Timeline	Medium	High	Deadlines met	The whole team	Discuss timetables regularly and reduce delays.
R2	Our product must be able to hook up to a drone of a project partner specified power level.	Technical	Low if mitigated	High	In-person tests are smooth	Henry	Discuss with Carlo exactly what this will be tested on to avoid miscommunication.
R3	Our product must be verifiable without a drone powering it.	Timeline	Medium	Medium	Our device can attach to a camera system on the ground.	Anthony	Has an imitation power device to retain verification methods.
R4	Different assigned portions of the project are not	Technical	Low	Medium	Individual block verifications are smooth and on time	Caden	Check in with teammates' progress to reduce errors.

	operating as intended.						
R5	Vendor Delays	Timeline	Medium	Medium	Hardware is not available to test with	Aleksi	Order parts early to avoid shortages or delays.
R6	Our product does not combine well within the team members	Technical	Medium	High	Each block is connecting according to specified interfaces	Whole team	Discuss and meet with the team to reduce conflicts, communicate regularly.
R7	Power supply does not supply the right current or voltage to the system	Technical	Medium	High	Power supply supplies the right amount of voltage and current	Henry	Testing it extensively to avoid the wrong voltage or current happening in a full system test or full system run.
R8	Materials are too expensive	Cost	Medium	High	Price change or getting sponsor	Whole team	Reduce costs with alternative components that are lower in cost.

2.4. References and File Link

2.4.1. References

[1] “Swissair Flight 111,” Wikipedia, 28-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/Swissair_Flight_111. [Accessed: 28-Oct-2021].

[2] “Swissair Flight 111,” Wikipedia, 28-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/Swissair_Flight_111#Probable_cause. [Accessed: 28-Oct-2021].

[3] “Battery Incident Chart - Federal Aviation Administration.” [Online]. Available: https://www.faa.gov/hazmat/resources/lithium_batteries/media/Battery_incident_chart.pdf. [Accessed: 28-Oct-2021].

[4] D. Jones, “A passenger's battery charger exploded on a united flight, forcing an emergency landing,” The Washington Post, 27-Feb-2020. [Online]. Available: https://www.washingtonpost.com/travel/2020/02/27/passengers-laptop-battery-exploded-united-flight-forcing-an-emergency-landing/. [Accessed: 28-Oct-2021].

[5] “IEEE 1156.1-1993 - IEEE standard microcomputer environmental specifications for computer modules,” IEEE SA - The IEEE Standards Association - Home, 17-Jun-1993. [Online]. Available: https://standards.ieee.org/standard/1156_1-1993.html. [Accessed: 28-Oct-2021].

[6] “MIL-STD-810,” Wikipedia, 18-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/MIL-STD-810. [Accessed: 28-Oct-2021].

[7] “IEC 62262,” Wikipedia, 18-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/IEC_62262. [Accessed: 28-Oct-2021].

[8] Smith, Ted, David Sonnenfeld, and David Pellow. Challenging the Chip. Philadelphia: Temple UP, 2008. Web.

[9] Gu, Wei, Di Liu, Chen Wang, Shufen Dai, and Donghui Zhang. "Direct and Indirect Impacts of High-tech Industry Development on CO2 Emissions: Empirical Evidence from China." Environmental Science and Pollution Research International 27.21 (2020): 27093-7110. Web.

[10] Person, “Air Cargo Carriers Battle competition from seas, passenger planes,” Reuters, 03-Jun-2014. [Online]. Available: https://www.reuters.com/article/airlines-iata-cargo/air-cargo-carriers-battle-competition-from-seas-passenger-planes-idUSL6N0OK1BY20140603. [Accessed: 29-Oct-2021].

[11] S. Lekach, “Self-flying planes could transport passengers one day-but first, packages,” Mashable, 24-Jul-2021. [Online]. Available: https://mashable.com/article/self-flying-airplanes. [Accessed: 29-Oct-2021].

[12] “System-leve ESD protection guide (rev. C) - ti.com,” Texas Instruments. [Online]. Available: https://www.ti.com/lit/sg/sszb130c/sszb130c.pdf. [Accessed: 05-Dec-2021].

[13] “API requester TLS client authentication to a restful API endpoint,” TLS client authentication to a RESTful API endpoint. [Online]. Available: https://www.ibm.com/docs/en/zosconnect/3.0?topic=options-tls-client-authentication-restful-api-endpoint. [Accessed: 28-Oct-2021].

[14] “Electrostatic discharge,” Wikipedia, 15-Nov-2021. [Online]. Available: https://en.wikipedia.org/wiki/Electrostatic_discharge. [Accessed: 05-Dec-2021].

[15] “Antistatic device,” Wikipedia, 22-Sep-2021. [Online]. Available: https://en.wikipedia.org/wiki/Antistatic_device. [Accessed: 05-Dec-2021].

2.4.2. File Links

N/A

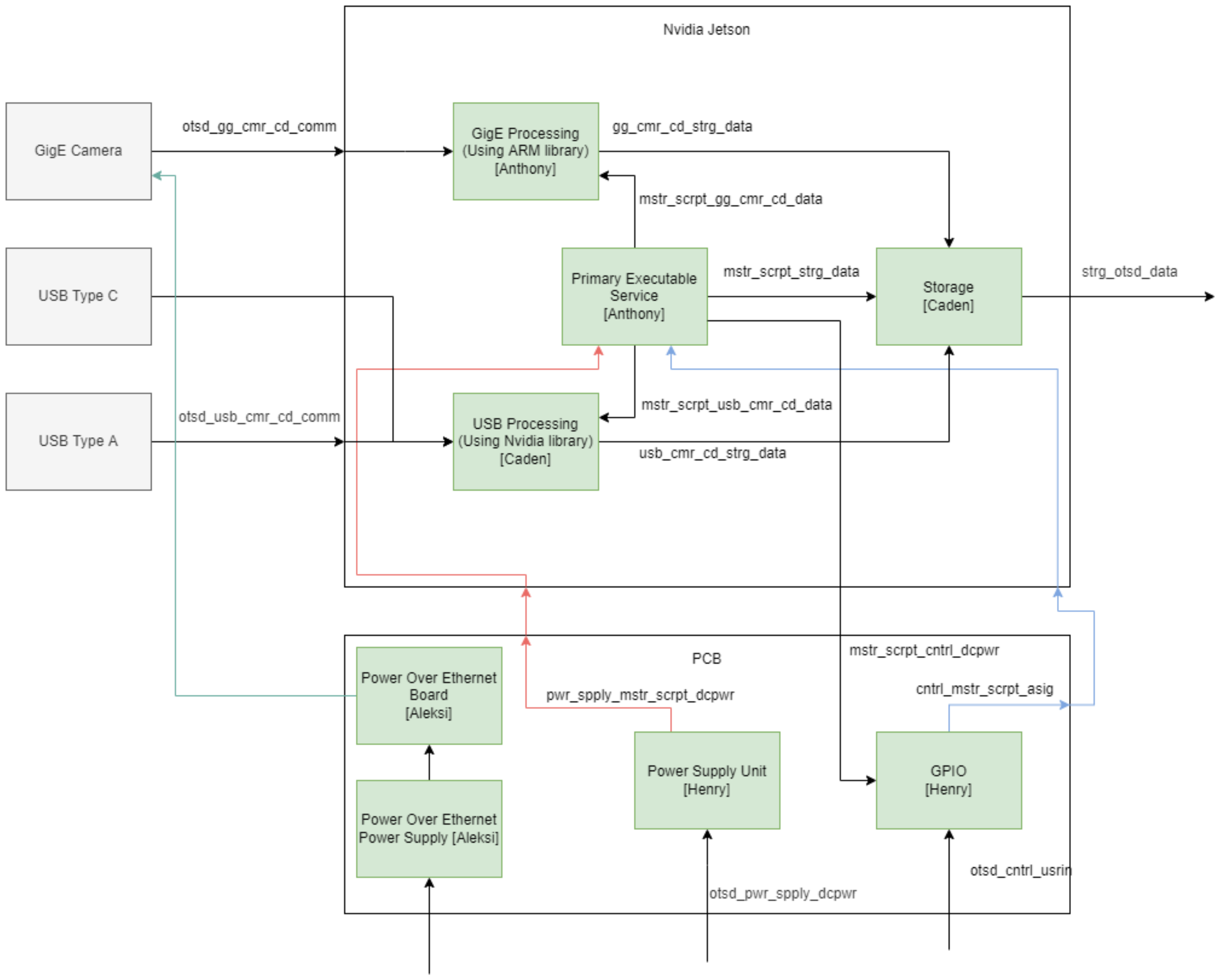
2.5. Revision Table

5/6/2022	Henry: Update each section in the section 2.2
5/6/2022	Anthony Kung - Added Section 2.2.1 - Added Section 2.2.2 - Added Section 2.2.6 - Added References Section 2.4.1
4/22/2022	Anthony Kung - Updated Requirement 2.3 - Added requirement and verification steps
3/13/2022	Caden Friesen: Edited requirements 1 and 7 verification steps
3/12/2022	Anthony Kung - Updated Section 2 Requirements
3/12/2022	Caden Friesen: Reformatted 2.3 Removed risk 9. Edited risk 2 and 3.

3/6/2022	Caden Friesen: Completely overhauled or removed all requirements. Added new requirements to reach 8.
3/6/2022	Anthony Kung - Updated Section 2 Requirements
1/29/2022	Caden Friesen: Wrote new project requirements.
12/3/21	Caden: Made edits to risk table. Overhauled and added content to all requirements
12/3/21	Aleksi: Additions to risk table
12/3/21	Henry: Additions to risk table
11/19/21	Caden Friesen: Added titles to requirements and edited some language
11/12/21	Caden Friesen: Reformatted 2.3 to the landscape. Added project partner requirements, one more requirement, and edited all eight requirements. Added responsible parties to 2.3. Added citations to requirements
10/29/21	Henry Chen: added to section 2.3
10/29/21	Anthony Kung - Added verification methods to 2.1 - Added additional requirements to 2.1
10/29/21	Aleksi Hieta: Added to 2.1 and 2.3
10/28/21	Caden Friesen: Added all sections. Started work on 2.1 and 2.3

3. Top-Level Architecture

3.1. Block Diagram



Nvidia Jetson
GigE Camera
otsd_gg_cmr_cd_comm
GigE Processing (Using ARM library) [Anthony]
gg_cmr_cd_strg_data
mstr_scrpt_gg_cmr_cd_data
USB Type C
Primary Executable Service [Anthony]
mstr_scrpt_strg_data
Storage [Caden]
strg_otsd_data
USB Type A
otsd_usb_cmr_cd_comm
USB Processing (Using Nvidia library) [Caden]
mstr_scrpt_usb_cmr_cd_data
usb_cmr_cd_strg_data
PCB
mstr_scrpt_cntrl_dcpwr
Power Over Ethernet Board [Aleksi]
pwr_spply_mstr_scrpt_dcpwr
cntrl_mstr_scrpt_asig
Power Supply Unit [Henry]
GPIO [Henry]
Power Over Ethernet Power Supply [Aleksi]
otsd_cntrl_usrin
otsd_pwr_spply_dcpwr

Figure 3.1: **Block diagram for the overall project**

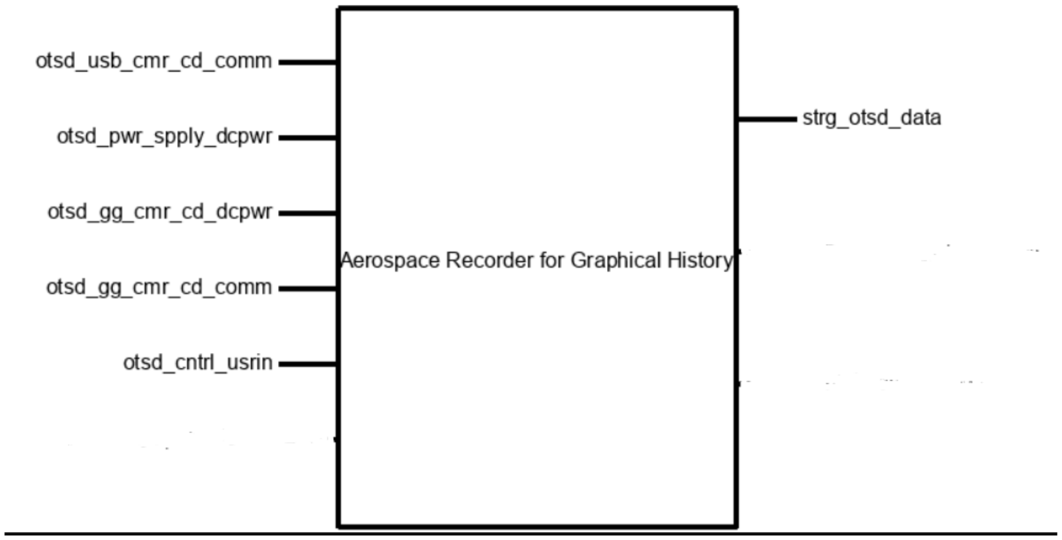


Figure 3.2: **Black box diagram of the system**

3.2. Block Descriptions

GigE Camera Research and Code

Block Champion: Anthony Kung

The GigE code block is responsible for interfacing with the GigE Cameras via an Ethernet connection. This camera operates at high speed taking advantage of Gigabit Ethernet and Power over Ethernet. The number of GigE cameras connected can vary, this means that this block has to be able to handle multiple cameras at once using forks. The block will take images from the two GigE cameras aiming for at least 15FPS on each. This block will also need to report the location of where the recorded files are, and the unique identifier for each camera to the Primary Executable. The way that this block would work is that when the Primary Executable calls its function it connects to the cameras and starts getting image streams from them. The Primary Executable will then control the recording by setting up a time to start and stop recording.

USB Camera Research and Code

Block Champion: Caden Friesen

The USB Camera block will work with a Boson320 Flir camera to record image streams. This camera will use a USB type C connection. These streams will be recorded at a minimum of 10FPS and stored on the local memory to be moved and renamed by the storage block. This block will be the code that performs all the above actions, but will be set up as a function so that it can be called upon by the primary executable.

Primary Executable

Block Champion: Anthony Kung

The Primary Executable is a Linux service that starts when the Nvidia Jetson block is powered on. This program will start recording protocols immediately but look to the GPIO input for a preset delay or to stop recording. It will drive many other parts of the project blocks, these include using the camera scripts to detect and record image streams, and running the storage script to move recorded images to a solid state drive.

Storage Code

Block Champion: Caden Friesen

The Storage Code block will be focused on moving files that are stored on local memory to a solid-state drive with far more available space. It will be made in a way where it could be called by the primary executable as a function. It will need to take in file paths, timestamps, and camera numbers so that it can move image files from local memory and rename and store them in an organized function. It will make folders for each camera so the images will be stored separately from each other to avoid confusion.

Control System

Block Champion: Henry Chen

This block will be in charge of all user input to the device and to the primary executable. The block will allow the user to start and stop recording at will as well as set a time delay before recording or a time that the device will record for. It will contain a display that shows whether it is currently recording or stopped as well as the current time on the timers. The code for this block will be covered by the primary executable while the hardware and a PCB will be the responsibility of this block. Different buttons will contain different functions, there are four buttons total.

Power Supply

Block Champion: Henry Chen

The power supply block will be in charge of using the 100 Watts 28 Volts Dc power provided by the drone and transforming it into the 10-12 Volt 5.5-7 Ampere requirement for the NVIDIA Jetson AGX Xavier that will be the core of the system.

3.3. Interface Definitions

Table 3.1: **Interface Definitions List**

Name	Properties
otsd_usb_cmr_cd_comm	• **Other:** The camera can be moved at least one foot in each direction (up, down, left, right, forward, back) without a disconnection • **Other:** Flir Boson320 camera will be connected as the input device • **Protocol:** USB Type C
otsd_pwr_spply_dcpwr	• **Inominal:** 1A • **Ipeak:** 1.5A • **Vmax:** 28V • **Vmin:** 26V
otsd_gg_cmr_cd_dcpwr	• **Inominal:** 1A • **Ipeak:** 1.5A • **Vmax:** 28V • **Vmin:** 26V
otsd_gg_cmr_cd_comm	• **Other:** Can handle 1 or 2 GigE cameras with the same level of functionality • **Other:** Cameras can be moved one foot in each direction without a disconnection • **Protocol:** GigE
otsd_cntrl_usrin	• **Other:** Buttons press down greater than or equal to 0.1cm distance • **Other:** Buttons have flat non-extruding top • **Type:** Four Push Buttons are Present

usb_cmr_cd_strg_data	● **Datarate:** At least 10 images will be saved per second from the camera ● **Messages:** Files saved will be images with 320x256 pixel resolution ● **Protocol:** Data will be saved on the eMMC
pwr_spply_prmry_xctbl_dcpwr	● **Inominal:** 5.5A ● **Ipeak:** 7A ● **Vmax:** 12V ● **Vmin:** 10V
gg_cmr_cd_strg_data	● **Datarate:** Files produced are at least 15FPS ● **Messages:** Each frame will be saved as Imperx RAW image file ● **Protocol:** Data will be saved on the eMMC
prmry_xctbl_usb_cmr_cd_data	● **Other:** Ability to set variable with camera number for storage function. (Boson320 designated camera 1) ● **Other:** Ability to call storage function on filepath of the saved images. ● **Protocol:** Script for running the camera can be called by a program using a system("./[scriptname]") execution
prmry_xctbl_gg_cmr_cd_data	● **Other:** Ability to return number of connected devices with unique identifier ● **Other:** Ability to return filepath for saved data ● **Protocol:** Function Call
prmry_xctbl_strg_data	● **Messages:** File name (char array), camera number (int), timestamp (char array) ● **Other:** Ability to provide status of if it is still in the process of storing data or is waiting ● **Protocol:** Function Call

prmry_xctbl_cntrl_dcpwr	● **Inominal:** 60mA ● **Ipeak:** 70mA ● **Vmax:** 3.35V ● **Vmin:** 3.25V
strg_otsd_data	● **Datarate:** Able to transfer 12 Gigabytes in under 5 minutes. Spread across 6 files. ● **Other:** Able to detect if files for storage are already present and create them if not ● **Other:** Data can be moved to a USB using a monitor and can be played back on another computer without corruption. Shutting the device off while recording or in storage mode will result in one corrupted file which does not count against this property
cntrl_prmry_xctbl_asig	● **Other:** 60-90 mA current (Button Active) ● **Other:** code that can read button inputs through GPIO pins ● **Vrange:** 3-3.3 volts (Button Active)

3.4. References and File Link

3.4.1. References

[1]Kangalow, W. Lucetti, Ícaro, Sam, Idella, Sagi, J, L. dae hee, David, J. Daniel, J. Daniel, Eirik, Arrakisun, Simran, Tegwyn☠Twmfatt, L. Steinbach, Sai, and T. T. Twmfatt, “GPIO interfacing – Nvidia Jetson TX1,” *JetsonHacks*, 02-Nov-2019. [Online]. Available: https://jetsonhacks.com/2015/12/29/gpio-interfacing-nvidia-jetson-tx1/. [Accessed: 21-Apr-2022].

3.4.2. File Links

N/A

3.5. Revision Table

4/21/2022	Henry Chen: edit the block descriptions and add some references link
3/12/2022	Caden Friesen: Wrote all block descriptions in section 3.2. Added interface table in 3.3. Added Black Box diagram to 3.1
3/6/2022	Anthony Kung - Updated Section 3
1/27/2022	Caden Friesen: Wrote new block description general requirements.
1/5/22	Henry: added update on the power supply
12/3/21	Caden Friesen: Added updated block diagram and black box diagram. Added new interface otsd_mbl_pplctn_usrin. Minor edits to block descriptions.
12/3/21	Aleksi Hieta: Interface Revision
11/19/21	Anthony Kung : - Block Diagram - Interface Definition
11/19/21	Caden: Added block descriptions and part of interface table. Worked on block diagram
11/19/21	Henry: add stuff in section 3
11/19/21	Aleksi Hieta: 3.1 Block Diagram, part of 3.2

4. Block Validations

4.1. Primary Executable

4.1.1. Description

Our project is to create a video recorder using Nvidia Jetson Development Board. The Automated Primary Executable Block (APE) also known as Master Script is a Linux systemd service that starts when the Nvidia Jetson block is powered on. This program will look for GPIO input for control and start recording after a set delay as well as controlling other parts of the project blocks, these include using the camera scripts to detect and record videos, using the storage script to move recorded video to another partition. The APE also controls the user interface which handles the user input (push buttons) and user output (E-Ink e-Paper Display).

4.1.2. Design

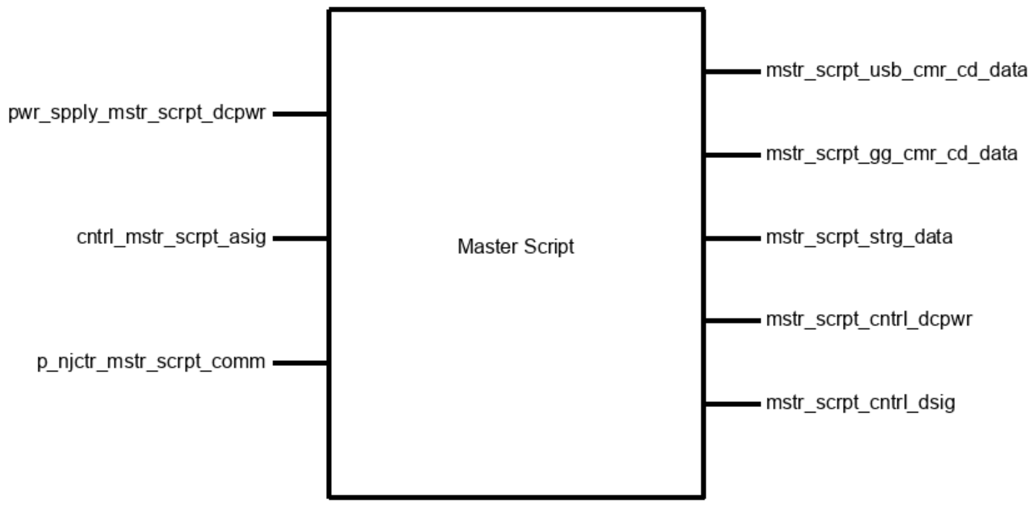


Figure 4.1.2.1: Primary Executable Block Illustration

Figure 4.1.2.1 shows the Primary Executable Block's block diagram, which includes 8 interfaces with 26 total properties. The APE will be running as a system service on power-up, it leverages Jetson's automation headers [1] to autoboot when power is supplied. Once the power is supplied, APE will start counting down a delay set beforehand. When the timer hits 0, APE will start the recording process with the help of the two camera blocks and the storage block. Once the recording started, any interruption to the recording process, e.g. crashes or power failure, will resume the recording process immediately. This is done by writing a file containing a delay, the delay is then decremented and written to another counter file.

4.1.3. General Validation

Our entire project is to be developed in C++ to take advantage of many camera library and Jetson built-in library which supports C++. By using a unified language for all processes, we can ensure that all of our programs are compatible with each other. And C++ is great for multithreading [2] which allows us to run multiple instances of a function simultienously. Although Python and Node.js also support multithreading as well as the Jetson libraries, the Aravis GigE library [3] would only support C++ and thus, C++ would be a better option than Python in this case. By using a counter, we can create a fully automated process that can still skip the unnecessary part of the recording (e.g. taking off) while requiring no user interaction of any sort during operation. This timer counter will be set as part of the preflight check of the drone, no user interaction is required or possible once the drone are started up. Having a single primary program to manage multiple parts of the block would helps keeping everything in sync as well as facilitating communication between services. To best reduce the circuitries needed for this project, GPIO will be managed by Jetson itself, no additional power supply nor external HAT system is required, Jetson even comes with its own level shifter as well.

4.1.4. Interface Validation

pwr_spply_mstr_scrpt_dcpwr : Input

Power Supply to Master Script (APE) DC Power Input

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Inominal: 3.5A	We have determined that we need at least 30 W of power for the Jetson for maximum performance [4].	The minimum of this setup would provide at least 35 W.
Ipeak: 4.0A	This is the maximum voltage we will supply to Jetson board to provide sufficient power for the board.	The Jetson has an absolute maximum of over 200 Watt using the MAXN power mode [4]. This current is way less than the absolute max that Jetson can handle while still providing more than enough power for the 8 ARM cores.
Vmax: 19V	This is the voltage we are targeting for our Jetson to provide sufficient power for the board.	The power supply that comes with the Jetson provides 19V, the power specification of Jetson supports 19V and it would be the maximum voltage we are willing to go for best module operating efficiency [5].
Vmin: 10V	The Jetson requires at least 9V to be in normal operation mode, anything below 9V will result in the Jetson entering low-power mode [5].	The power supply minimum is at least 1V above the absolute minimum.

mstr_scrpt_usb_cmr_cd_data : Output

Master Script (APE) to USB Camera Code Data Output

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Other: Ability to return filepath for saved data	A file path is required for the Storage block processes.	The APE is able to receive strings returned from a C++ function. This is how the filepath will be sent from the camera block.
Other: Ability to return number of connected devices with unique identifier	The number of connected devices are needed to determine the number of threads to create.	The APE is able to receive integers returned from a C++ function. This is how the numbe of cameras will be sent from the camera block.
Protocol: Function Call	The APE will call a function within the camera block to obtain these informations.	The APE is able to call any C or C++ functions as it is a C++ program. All codes are equipped with a header file to be included in the APE with the necessary extern variables.

mstr_scrpt_gg_cmr_cd_data : Output

Master Script (APE) to GigE Camera Code Data Output

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Other: Ability to return filepath for saved data	A file path is required for the Storage block processes.	The APE is able to receive strings returned from a C++ function. This is how the filepath will be sent from the camera block.
Other: Ability to return number of connected devices with unique identifier	The number of connected devices are needed to determine the number of threads to create.	The APE is able to receive integers returned from a C++ function. This is how the numbe of cameras will be sent from the camera block.

Protocol: Function Call	The APE will call a function within the camera block to obtain these pieces of information.	The APE is able to call any C or C++ functions as it is a C++ program. All codes are equipped with a header file to be included in the APE with the necessary `extern` variables.

mstr_scrpt_strg_data : Output

Master Script (APE) to Storage Data Output

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Messages: File name (char array), camera number (int), timestamp (char array)	This is to provide the necessary information for the Storage block to function.	The file name will be obtained by the APE from the camera blocks. The camera blocks will return the filename to APE once the file has been saved, this is managed by thread processes.
Other: Ability to provide status of if it is still in the process of storing data or is waiting	This is necessary to ensure the APE do not quit unexpectedly during file storing.	The APE will wait for all thread processes to exit before turning itself off.
Protocol: Function Call	The APE will be providing these information via functions as parameters.	The APE is able to call any C or C++ functions as it is a C++ program. All codes are equipped with a header file to be included in the APE with the necessary `extern` variables.

mstr_scrpt_cntrl_dcpwr : Output

Master Script (APE) to Control (GPIO) DC Power

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Inominal: 60mA	This is the estimated current required by the GPIO block to operate.	The Jetson is more than capable to supply 1A of current if needed [1], this is way less than the maximum.
Ipeak: 70mA	This is the estimated maximum current required by the GPIO block to operate.	The Jetson is more than capable to supply 1A of current if needed [1], this is way less than the maximum.
Vmax: 3.35V	This is the maximum voltage the GPIO block can handle.	Jetson is equipped with a 3.3V power supply pin [1].
Vmin: 3.25V	This is the minimum voltage the GPIO block can handle.	Jetson is equipped with a 3.3V power supply pin [1].

cntrl_mstr_scrpt_asig : Input

Control (GPIO) to Master Script (APE) Analog Signal Input

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Other: code that can read button inputs through GPIO pins	The Jetson GPIO will be checking the signal received from the GPIO pin from button press.	The Jetson GPIO pins are compatible with Raspberry Pi GPIO and thus is able to be programmed in C++ easily, in Python and even Node.js.
Other: 60-90 mA current (Button Active)	This is the current that the button would likely generate.	Jetson is more than capable to accept this current on specific pins [1].
Vrange: 3-3.3 volts (Button Active)	This is the voltage that the button might supply to the Jetson.	Jetson is more than capable to accept up to 5V on specific pins [1].

p_njctr_mstr_scrpt_comm : Input

PoE Injector to Master Script Communication Input

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Other: Parse input from multiple cameras	The PoE switch is needed to breakout the single Ethernet port on Jetson to support multiple cameras.	Our Jetson is setup in a way to accept up to 2^16 unique IP addresses, which would be capable of accepting more cameras than a PoE switch can provide.
Protocol: GigE	This is the protocol that the camera is going to use.	The Aravis vision library [3] is specifically designed in a way to work with GigE cameras.
Message: Video Stream	The cameras will encode the video stream in video format.	The Aravis visionlibrary [3] is capable of decoding various video format including Bayer colored format, Mono black/white format as well as some other popular video format.

4.1.5. Verification Process

The verification plan for this block would simply be verifying that the APE is able to perform the require actions. The APE comes with a thermal regulation function that would control the temperature of the Jetson board. This thermal regulation system is created in the same way as the camera blocks and the GPIO blocks would be created. The thermal regulation functions will be called as a thread and the temperature is constantly measured and reported. By checking if the fan on the Jetson is operational, we can tell the multithreading workflow are operating normally.

1. Power up Jetson
2. Verify APE started and running by checking the GPIO output for e-Paper select line
3. Verify APE timer is operational by checking the timer tracking file on Jetson
4. Verify APE able to create threads by checking thermal regulation system
5. Verify APE is able to call external functions by checking the thermal regulation system
6. Verify Jetson is able to supply the required power by measuring the GPIO pins
7. Verify Jetson is able to received the supply power by using an external power supply

4.1.6. References and File Link

4.1.6.1. References

[1] NVIDIA Jetson AGX Xavier Developer Kit Carrier Board Specification, https://developer.nvidia.com/embedded/dlc/Jetson_AGX_Xavier_Developer_Kit_Carrier_Board_Specification

[2] C++ Multithreading, https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm

[3] Aravis vision library, https://github.com/AravisProject/aravis

[4] Power Management for Jetson Xavier NX and Jetson AGX Xavier Series Devices, https://docs.nvidia.com/jetson/l4t/index.html#page/Tegra%20Linux%20Driver%20Package%20Development%20Guide/power_management_jetson_xavier.html

[5] Jetson AGX Xavier Series Thermal Design Guide, https://developer.nvidia.com/embedded/dlc/jetson-agx-xavier-series-thermal-design-guid

4.1.6.2. File Link

The project repository lives on GitHub at https://github.com/Optical-Interface and all files related to the project will be available on the repository. Note that not all files will be publicly available or if any file would be publicly available. These files are also available on request by contacting the project group at capstone@anth.dev.

4.1.7. Revision Table

Name	Time	Description
Anthony Kung	2/19/2022	Updated Primary Executable Block Validation
Anthony Kung	2/4/2022	Created Primary Executable Block Validation

4.2. GigE Camera Block

4.2.1. Description

The GigE code block is responsible for interfacing with the GigE Camera via an Ethernet connection. This camera operates at high seep taking advantage of Gigabit Ethernet and Power over Ethernet. The number of GigE cameras connected can vary, this means that this block has to be able to handle multiple cameras at once using forks. This block would also need to report the number of connected devices, the location of where the recorded file is, and the unique identifier for each camera to the master script. The way that this block would work is that when the master script would call its function and it looks for the connected Ethernet devices using Linux native services, connect to these devices, and starts getting video streams from it. The function will then return a ready message for the master script letting it know the camera is ready for recording. The master script will then control the recording by setting up a time to start and stop recording.

4.2.2. Design

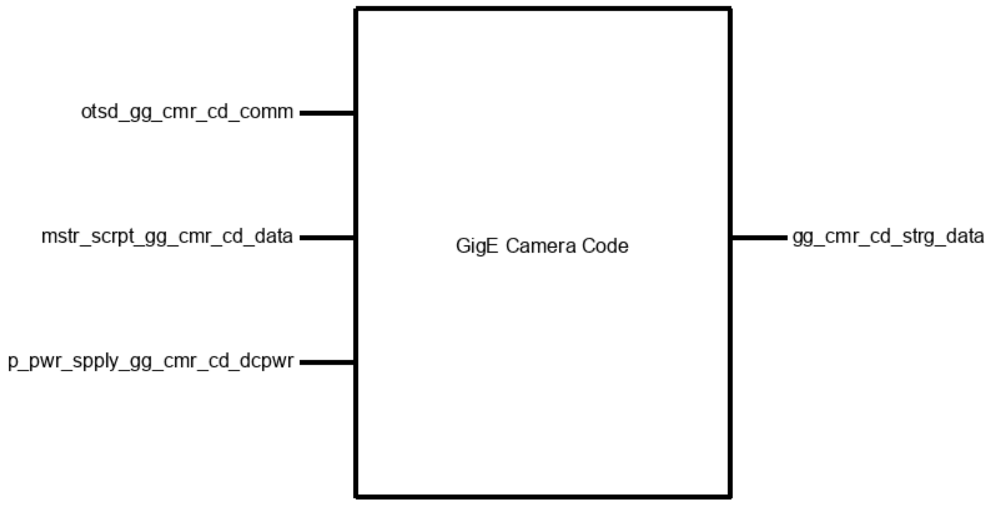


Figure 4.2.2.1: GigE Camera Block Illustration

Figure 4.2.2.1 shows the operation flow of the GigE Camera Block. The GigE Camera Block is a code block consisting of codes for the APE Primary Executable (Master Script) to use as an external library. The primary goal for this block is to provide the ability to detect cameras, connect to cameras, start recording and save the recording.

During setup, the APE will call a function from this block to get the number of connected cameras and if any cameras were unable to connect. It will then pass a timestamp of when to stop recording, this is to synchronize the recording time for all cameras from USB to GigE and when a recording was started midway.

Pseudo Code as follow:

```
function GetCameraStats() {
    if (EthernetPort == InUse) {
        let DevicesDetected = ScanForDevicesOnPort();
        let CameraObjectsArray = [ {} ];
        for (i = 0; i < DevicesDetected; i++) {
            let CameraObject = {};  // Initialize Camera Object
            try {
                CameraObject = {
                    id: GetDeviceID(),
                    type: "GigE",
                    status: "Connected",
                    device: ConnectToDevice()
                }
            }
            catch (error) {
                error.log(error);
                CameraObject = {
                    id: GetDeviceID(),
                    type: "GigE",
                    status: "Error"
                }
            }
            CameraObjectsArray.push(CameraObject);
        }
        // Return Array which indicate number of cameras connected
and camera status
        return CameraObjectsArray;
    }
    else {
        return null;  // Return nothing if no camera detected
    }
}
```

```
function StartRecording(DeviceID, Timestamp, StorageLocation) {
    let Duration = Timestamp - Duration;
    try {
        StartCameraRecording(DeviceID, Duration, StorageLocation);
    }
    catch (error) {
        error.log(error);
        return false;
    }
    // If no error, return true
    return true;
}

function EndRecording(DeviceID) {
    try {
        SendCameraRecordingInterrupt();
    }
    catch (error) {
        error.log(error);
        return false;
    }
    // If no error, return true
    return true;
}
```

4.2.3. General Validation

The project requires the use of GigE cameras which is not supported by the Jetson by default. A GigE camera uses the Gigabit Ethernet connection to stream video data and receives power. Since Jetson does not support Power over Ethernet, a separate block is created to power the camera.

Since there is no driver nor library available natively by Jetson, a third-party Linux library known as Aravis [1] will be used. However, the library is licensed under GNU Lesser General Public License v2.1 and to avoid licensing issues, under Condition 5 and Condition 6 of the GNU Lesser General Public License v2.1 any code that uses the library but not including the library called "work that uses the Library" is exempted from the license requirements. So this block will not be including the library and the library will have to be downloaded separately from this code.

This block will be created with the C++ programming language which will work with both the APE and the Library. The function will simply be in a header file to be called by the APE, no sockets required.

The library is recommended by Nvidia staff and has been widely used by the Linux community, being the only GigE library, this is the only library that we can use at this time besides the SDK that our project partner might be able to provide. There are many reasons to believe that this library can meet our use case:

1. GigE standard is universal, meaning there is no specific library to use for different GigE cameras, if one GigE camera works, others will work too.
2. This library has been proven to work with the company that makes the same cameras that we are going to use.
3. Jetson comes with all the Linux systems that Ubuntu OS comes with and this library supports these systems including GStreamer that we are using for our USB cameras
4. This library has been used by a camera company to support Nvidia Tegra systems which powers our Jetson AGX Xavier [2]

The block will store the streamed data to the local eMMC of the Jetson which allows high speed read/write. This will prevent bandwidth bottleneck at the SSD and act as a buffer as the eMMC has a higher bandwidth than the SSD can provide.

4.2.4. Interface Validation

otsd_gg_cmr_cd_comm : Input

Outside to GigE Camera communication, this is the GigE wired communication to the code block from the camera.

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Other: Cameras can be moved one foot in each direction without a disconnection	Since the GigE camera will be mounted outside of our system, we will need to allow the user to position the camera according to their preference.	Our design uses an Ethernet port and does not require the camera to be installed on the system. This will allow the user to use an Ethernet cable of any length and mount the camera wherever they see fit.

Other: Can handle 1 or 2 GigE cameras with the same level of functionality	Our project partner requires us to be able to provide a platform that can be plugged in and go without any setup and support an undisclosed amount of cameras.	Our system do not limit the number of cameras that can be connected and there is nowhere on our block that set a limit on how many connections can be made. The block will simply look for the number of cameras on the system and attempt to connect to them.
Protocol: GigE	This is the protocol that the camera will be using.	The library we chose has been proven to work with GigE cameras and support the Jetson platform, we will be calling functions within the library to connect to and obtained video streams from it. The library has extensive documentation on how to do that [3].

gg_cmr_cd_strg_data : Output

GigE Camera to Storage Block

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Datarate: Files produced are at least 15FPS	This is the FPS that our test camera can produce.	From our current tests, the FPS stated by the camera is usually the FPS we are getting for the files. We can also set the FPS we are obtaining should we need to as well.
Messages: Files saved will be in 5-minute pieces of video	This is to prevent file corruption when the recording is interrupted.	This same method has been used by dashcam recorders to prevent file corruption. By saving the recording in chunks, we can guarantee that the chunk completed will not be corrupted.
Protocol: Data will be saved on the eMMC	This is Jetson’s local storage.	Since this is the local storage, all files are saved here by default without us needing to do anything.

mstr_scrpt_gg_cmr_cd_data : Input

APE (Master Script) to GigE Camera

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Other: Ability to return file path for saved data	This is needed so the APE can let the Storage block know where the file is located.	The APE will be providing the location to be saved to, this ensures the saved file path is always known by the APE.
Other: Ability to return the number of connected devices with unique identifier	The APE would need to know the connected cameras in order to alert the user if any cameras were showing error and start recording for specific cameras.	The program will return an array of all connected devices with their ID and their status. By looking at the size of the array, the APE can determine the number of devices and the device status is included with the array.
Protocol: Function Call	The APE will be calling the function of the GigE block so that no IPC will be required.	The entire program is created in modular C++ functions which is the same language used by the APE and can be included easily.

p_pwr_spply_gg_cmr_cd_dcpwr : Input

Power Block to GigE camera

Note: This block is the responsibility of the PoE block, no work on the GigE block is to work with this. The GigE block will interface with the PoE block then the PoE block will deliver the data as well as the power to the GigE camera.

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Inominal: 400mA	This is the current required by the IEEE standards for PoE devices, the minimum of such is 350 mA and this is just a good margin above the minimum.	The camera would only need 4.8W to operate, this will provide amply.

Ipeak: 550mA	This is the anticipated extra power needed for the camera.	This is the maximum current we will allow the camera to have, given that the camera only needs 4.8 W. It is also right below the 600 mA maximum current allowed by a Type 2 PoE cable.
Vmax: 56V	This is the maximum voltage as stated by the IEEE standards for PoE devices.	This meets the maximum voltage allowed for the GigE camera.
Vmin: 48V	This is the minimum voltage as stated by the IEEE standards for PoE devices.	This meets the minimum voltage allowed for the GigE camera.

4.2.5. Verification Process

1. Plug two GigE Camera to system
2. Call connect function to verify cameras are detected and able to connect to
3. Call record function to verify recording can be done
4. Call interrupt function to stop recording
5. Open the recorded video to ensure it can be played.

4.2.6. References and File Link

4.2.6.1. References

[1] Aravis, https://github.com/AravisProject/aravis

[2] The Imaging Source USB and GigE Linux package, https://www.theimagingsource.com/products/software/linux/software-for-linux/

[3] Aravis Documentation, https://aravisproject.github.io/aravis/

4.2.7. Revision Table

Name	Time	Description
Anthony Kung	2/4/2022	- Removed FPGA Logic Block Validation - Created GigE Camera Block Validation
Anthony Kung	1/5/2022	Created FPGA Logic Block Validation

4.3. USB Camera Block - Caden Friesen

4.3.1. Description

This block which will be primarily composed of code will take in data from a USB2 type C Flir Boson320 thermal camera and record data to files. The files will be .tiff images recorded with a speed of at least 10 frames per second. The block will keep a general naming scheme for the files recorded to be able to call the storage block’s code efficiently and in an organized way. All files submitted to the storage block will be under the designation of “Camera number 1” for the purpose of file organization and naming.

4.3.2. Design

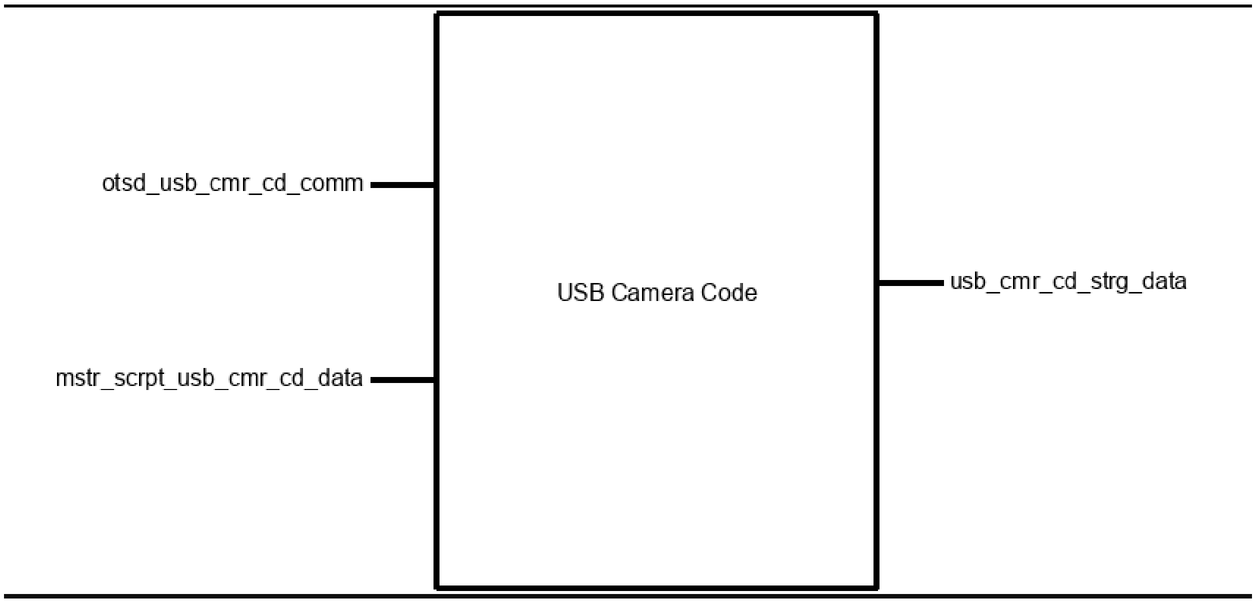


Figure 1: Black Box Diagram of USB Camera Code Block

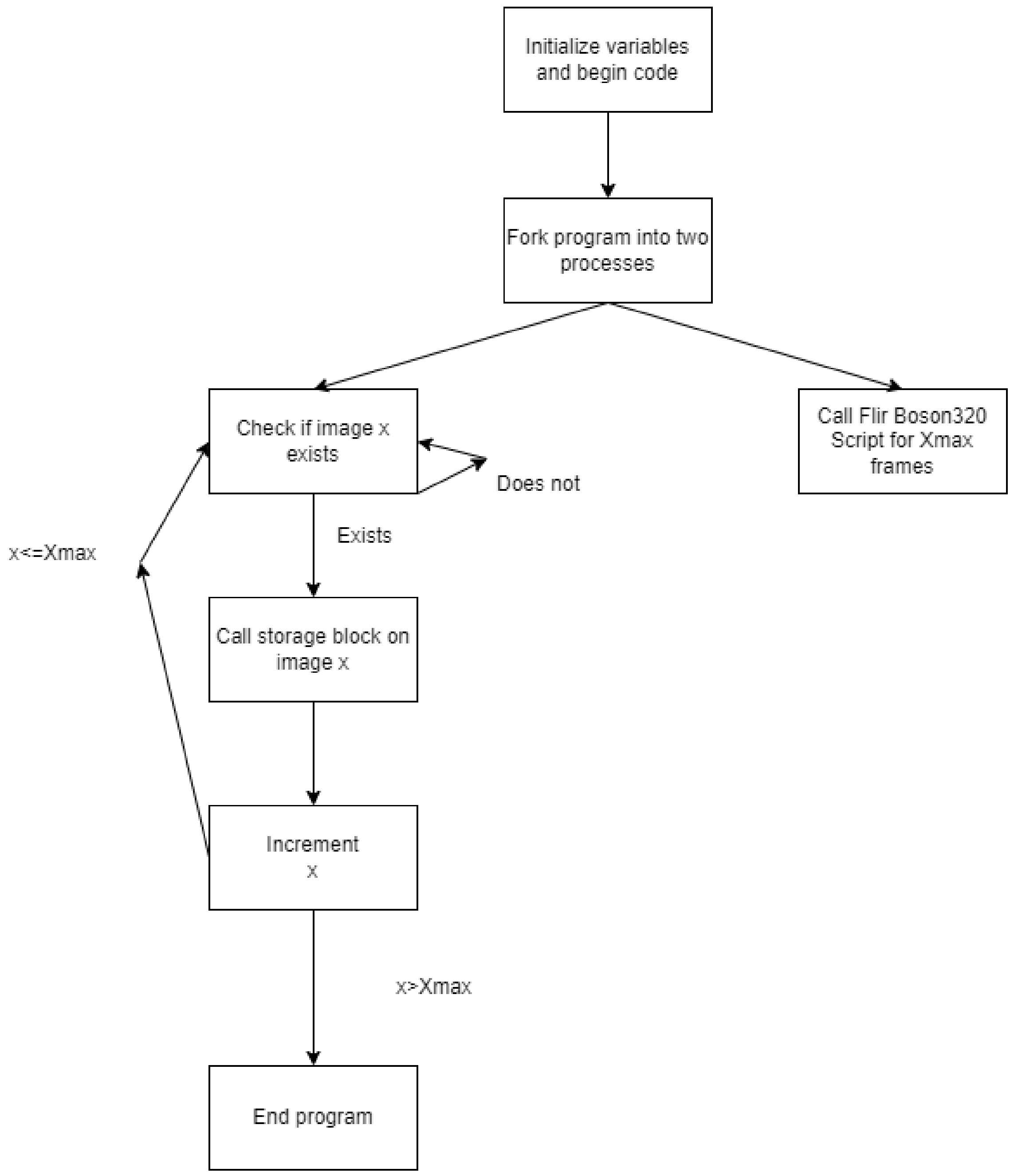

Initialize variables and begin code
Fork program into two processes
Check if image x exists
Call Flir Boson320 Script for Xmax frames
Does not
Exists
x<=Xmax
Call storage block on image x
Increment x
x>Xmax
End program

Figure 2: Camera Recording Flowchart

4.3.3. General Validation

The design above fills the needs of this project code wise thanks to the biggest resource for this block: the Flir Boson320 Script[1]. This script runs this specific camera and is able to record images at thirty frames per second and decode them into a viewable .tiff format. The main challenge is simply writing a program to run this script and control it as well as being able to interface with the storage block.

The largest concern right now is whether or not the NVIDIA Jetson will be able to take pictures on the GigE cameras and the Boson320 at once all at at least 10 pictures per second. According to the NVIDIA Jetson tutorials series this design is using a lot of knowledge from, the NVIDIA Jetson Nano is used extensively in rapid image processing and machine learning[2]. Since the NVIDIA Jetson Nano has far lower capabilities compared to the NVIDIA Jetson AGX Xavier being used the design should be fine. The design will use threads and forking to allow multiple cameras from other blocks as well as multiple storage processes to occur at the same time.

As a backup if we are unable to achieve the speeds desired then the option of long videos will be looked into instead of rapid picture taking. This was the original plan but it was decided against due to the unknown reliability of the drone’s power source. With the image taking process the only time lost will be time resetting up the cameras. This would be especially a challenge on this block as the Flir script is only able to take images.

One other thing to look into will be how long to record. One of our project partner’s recommendations was to record for a set amount of time. If we could accurately record time we could set this up although power failure could be a concern for this. Alternatively we could record indefinitely, although this may cause issues with storing all of the data before a shutdown occurs. This will be an important issue to consider when performing system integration.

4.3.3 Interface Validation

Table 4.3.1: **Interface Property Validation Table**

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
Otsd_usb_cmr_cd_comm: Input		
Other: Cameras can be moved one foot each direction without a disconnect.	This was chosen because this is about how far we are expecting cameras to be from the Jetson at a maximum for the project partner's implementation. Our project partner will be making an enclosure and it is only expected to need about a foot of wire for the cameras to be moved and placed within.	The design is not a physical one, this is just a testing of what wires we have bought to make sure they are solid for the connection. If this does not work the solution will be to simply get new wires to connect cameras.
Other: Flir Boson320 camera will be connected as the input device.	This camera was selected as it was one of the three cameras provided by our project partner. Originally the design would have been adaptable to more cameras, but this type of thermal camera needed more special attention so the scope was lowered.	The design above is made to work with specifically this camera. It uses code provided by Flir themselves to drive and actually take photos on the camera.
Connection Protocol: USB type C	This was chosen because the Flir Boson320 is a USB type C camera and the NVIDIA Jetson AGX Xavier has two available ports for it.	Since there are two open USB C ports on the NVIDIA Jetson and no conversion will need to happen between the camera and the Jetson, one USB C wire will accomplish this easily.

Usb_cmr_cd_strg_data: Output

Datarate: At least 10 images will be saved per second from one camera.	This value was chosen based on the project partner's goals. The project partner would like pictures taken of video frames with a goal of 10 frames per second. To achieve this we will need 10 pictures from each camera per second.	The tutorials being followed from Flir on using their recording code are able to run 30 frames per second on the camera. We have also in testing been able to record 30 frames per second using this code. The main challenge will be writing code that is fast enough to take full advantage of this ability, especially when full system integration happens.
Messages: Files saved will be images with a 320x256 pixel resolution.	This resolution is a common size for thermal cameras and is what the Boson320 should be producing.	The script provided by Flir has been shown in its demo to provide photos at this resolution. After running the script on the camera locally the photos were the proper resolution as well.
Protocol: Data will be saved on the eMMC.	This is the local data on the Jetson. The script we will use to save images automatically save to the local memory. We decided moving things to the SSD should be the storage block's job so this value was chosen to lighten the load of the camera block.	Data stores to the eMMC using the Flir script by default. This will not require any additional design and is more of an interface for the storage block to know about.

Mstr_scrpt_usb_cmr_cd_data: Output

Other: Ability to set variable with camera number (Boson320 Designated Camera 1)	This has been picked as a property because the storage block and master script will need to know what camera number is being used when they move files or call set up procedures.	With threads this should be able to be manipulated and used in other processes at the same time. It will also simply be passed as a variable to the storage block which is even simpler and has already been tested on the built storage block.
Other: Ability to call storage function on the filepath of the saved image.	This has been chosen as a property because it is important for the storage function to be able to find the saved images to actually move them.	The Flir code always saves images to the same file with a predictable naming scheme so the design above will be able to pass that location to the storage function.
Protocol: Script for running the camera can be called by a program using a system(“./[scriptname]”) execution.	It is important that the Flir program can be executed from other programs so that the modular design can be put together.	It has been found through testing that this is possible and easy to do. The Flir program can be called to record indefinitely or a specific number of frames.

4.3.5. Verification Process

1. Plug in Flir Boson320 Camera
2. Verify that the camera is plugged into a USB type C port connected to the NVIDIA Jetson.
3. Move the camera one foot left to right, up and down, and forwards and backwards. Verify that the blue light remains on at all times.
4. Run a program that calls the Boson320 script and the storage script for 1000 frames of footage.
5. While running, view files initially appearing on the eMMC local memory before they get moved.
6. Open directory with saved images on the solid state drive. Verify 1000 images are present in Camera1 folder.
7. Check the timestamp of the initial photo and the last photo. Find the difference between these and divide 1000 by this number to calculate the framerate.
8. Check that images are 320x256 pixels in their properties.
9. Show that code uses a System("./BosonUSB [arguments]") formatted call.

These nine steps will verify:

Otsd_usb_cmr_cd_comm: Uses Boson320, camera can be moved, USB type C
Usb_cmr_cd_strg_data: At least 10 images per sec per camera, Resolution requirements, data on eMMC
Mstr_scrpt_usb_cmr_cd_data: Able to run storage function, able to set camera number variable, runs Flir Boson Script

4.3.6. References and File Links

[1]A. Prieto-Moreno, "FLIR/BosonUSB: Tool to capture Boson USB video in linux," *GitHub*. [Online]. Available: https://github.com/FLIR/BosonUSB. [Accessed: 04-Mar-2022].

[2]D. Franklin, "Dusty-NV/Jetson-inference: Hello AI World Guide to deploying deep-learning inference networks and deep vision primitives with TENSORRT and Nvidia Jetson.," *GitHub*. [Online]. Available: https://github.com/dusty-nv/jetson-inference/. [Accessed: 19-Feb-2022].

4.3.7. Revision Table

3/12/2022	Caden Friesen: Reformatted and moved all of validation document to project document. Changed interface table to landscape orientation and resized all parts.

3/4/2022	Caden Friesen: Changed interfaces, verification, switched code flowchart, added more issues for consideration to 4.3, edited block description
2/18/2022	Caden Friesen: Made rough draft of all sections

4.4. Jetson Storage Process

4.4.1. Description

The purpose of this block is to take video data recorded by the two camera blocks and store it on a solid state drive attached to the NVIDIA Jetson AGX Xavier. It will be built in a way where it can be called as a function with the parameters of a file path, camera number, and timestamp. It will go to the file specified, move it over to the solid state drive's memory from the Jetson's memory, and then rename the file to "CameraX[Timestamp]".

The reason this block is needed is because the memory on the Jetson is only 32 Gigabytes which is not nearly enough for the 4k video footage we will be recording over an hour of. The one terabyte solid state drive has plenty of room for storing all the footage we record. The block will also have the ability to create sorting directories on the solid state drive if they have been removed.

4.4.2. Design

Figure 1: Storage Block Black Box Diagram

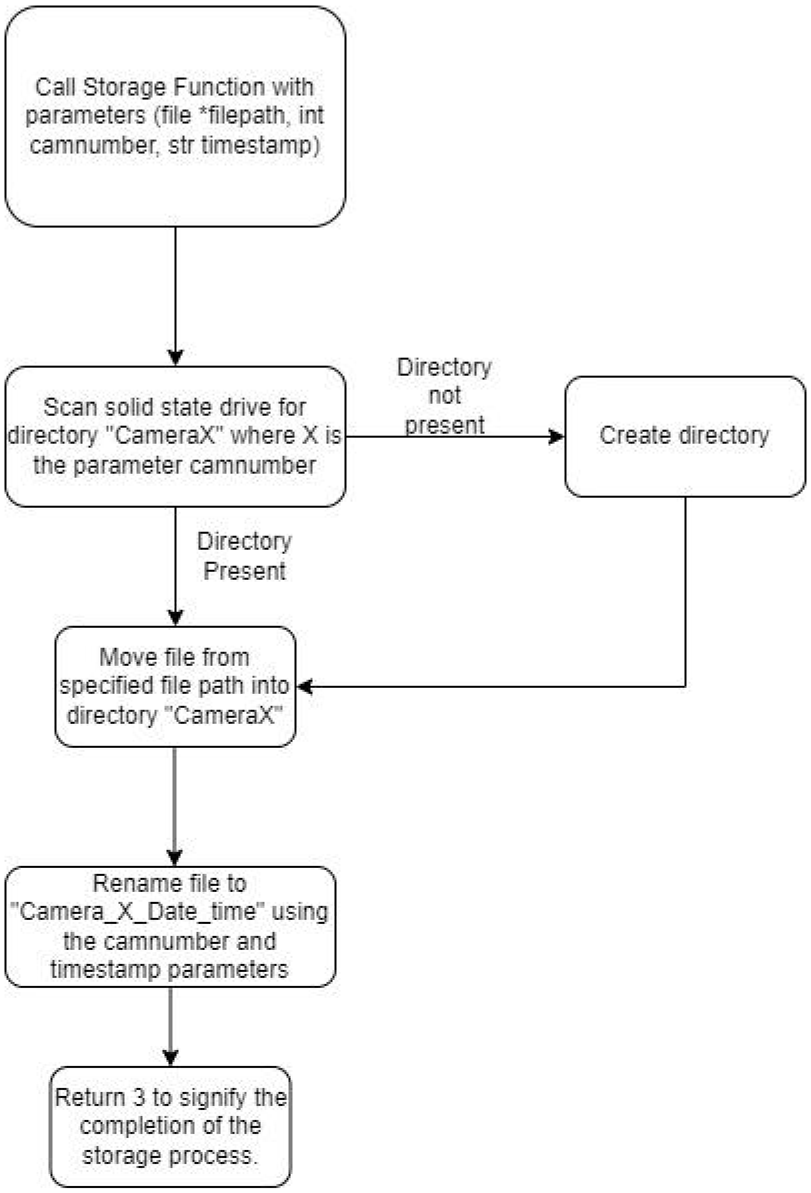

Call Storage Function with parameters (file *filepath, int camnumber, str timestamp)
Scan solid state drive for directory "CameraX" where X is the parameter camnumber
Directory not present
Create directory
Directory Present
Move file from specified file path into directory "CameraX"
Rename file to "Camera_X_Date_time" using the camnumber and timestamp parameters
Return 3 to signify the completion of the storage process.

Figure 2: The process of the code for the storage block. The process is straightforward other than the need to branch if the storage directory is not present. This process will be called once per file saved. This means it will be called multiple times at the end of every five minutes.

Pseudocode:
This pseudocode provides the names of the functions that will be used in this program. Links to their databases will be included in the file links section. Sections that use strcat will be slightly more involved to construct the proper string in the full coded version.

```
Int Storage(file *filename, int camnum, str timestamp){
        Str camstring = “ssd/camera”; //holds the start to the directory file path
        strcat(camstring,camnum); //creates the directory path with the camnum
        if(exists(“ssd/camera1”) != true){ //will check to see if directory exists
                mkdir(camstring); //creates directory if it does not exist
        }
        Boost library copy_file(filepath, camstring); //will copy file to SSD
        strcat(camstring,timestamp); //will create the name for the moved file
        rename(camstring,); //will rename file to “Camera_X_Date_Time”
        remove(filename); //will remove file from Jetson
        return 3; //signifies proper completion of the storage
}
```

4.4.3. General Validation

The black box diagram in Figure 1 shows the layout of the storage block in relation to the rest of the system. It will have three inputs from other blocks in the system and one output. Two of these inputs will come in the form of files from the USB and GigE camera blocks. These files will be 5 minute pieces of video but the format of the files will be inconsequential to the storage block. The files will be 30FPS video with definition of up to 4K meaning their maximum size will be 1.75 Gigabytes each[1]. The system is being built to handle up to six cameras so the total video data that will need to be transferred within 5 minutes is 10.5 Gigabytes at a maximum. The system should be able to handle this as the write speed of the solid state drive attached is 3.2 Gigabytes/sec[2].

The third input interface is the master script to storage interface. The master script will be calling the storage block as a function call. This will include parameters for the file path, the camera number, and the time stamp. These parameters will be used to determine where the file to be moved is and to set the name of the file after it is moved.

The final interface is an output to a solid state drive that will be considered external to our block. This interface will require files to be moved to the solid state drive. It will also need the ability to create directories for file sorting if they are not present.

The designed pseudocode above will work for moving the file which is the main concern of the block complexity wise. The original design for this block included using rename() to move the file but this function was not able to move files between storage devices. The copy_file() function from the boost library was suggested for a task like this and I verified with test code that it can cross storage devices[3]. This left a new task which was the removal of the file from the Jetson after. The remove() function was chosen for this as it is a well documented C++ function for this purpose. Rename() will be used after the file is moved to change the name of the video file. The three parameters provided at the start of the function will be used throughout the whole process to locate the original file and name directories and files on the solid state drive.

4.4.4 Interface Validation

Table 4.4.1: **Interface Property Validation Table**

usb_cmr_cr_strg_data: Input

Interface Property	**Why is this interface this value?**	**Why do you know that your design details <u>for this block</u> above meet or exceed each property?**
Data Rate: Files produced are at least 30fps	Thirty frames per second was decided by the USB camera block. This value was picked for what the USB block could handle recording with four cameras active using the NVIDIA Jetson AGX Xavier. This will be explained in more detail in the interface table for the USB camera block.	The design above will work fine with this because the highest resolution cameras we are expecting from our project partner are 4k cameras. At 30FPS the five minute files produced will be 1.75 Gigabytes each if they use 4k cameras. The solid state drive this program will be writing to has a write speed of 3.2 Gigabytes per second. This allows the design above to write about 548 files in the 5 minutes before the storage system will be called for its next set of files. The project is only being verified to work with 6 files at once so this block will be well within range of working with this.
Messages: Files saved will be in 5 minute pieces of video	This value will be controlled by the USB camera block. It was chosen because if a power failure or camera disconnection occurs, the current video file being recorded will be corrupted. Five minute video pieces will allow for protections to large amounts of data loss while only needing 20 files to represent an hour of data.	This requirement goes with the property above. It sets the 5 minute limit for this block to finish transferring all previous files. As stated above the write speed for the solid state drive will be many times more than enough for the expected six 5 minute files.

Protocol: Data will be saved on the eMMC	This was decided by the USB Camera block to be the easiest space to store data to. The eMMC is the local memory on the NVIDIA Jetson. This will allow the USB Camera to pick its own file path to save to.	This is the local memory on the NVIDIA Jetson. The design laid out above will work with this because the boost library's copy_file function works across storage devices allowing it to copy from the eMMC straight to the solid state drive.

gg_cmr_cd_strg_data: Input

Data Rate: Files produced are at least 15fps	Fifteen frames per second was decided by the GigE camera block. This value was based on the hardware we have available to test with. The system should be able to go higher than 15FPS and will be built to work at 30FPS but the cameras we have access to will only run at 15FPS so this is all we can verify.	The design above will work fine with this because the highest resolution cameras we are expecting from our project partner are 4k cameras. At 15FPS the five minute files produced will be 0.825 Gigabytes each if they use 4k cameras. The solid state drive this program will be writing to has a write speed of 3.2 Gigabytes per second. This allows the design above to write about 1096 files in the 5 minutes before the storage system will be called for its next set of files. The project is only being verified to work with 6 files at once so this block will be well within range of working with this.
Messages: Files saved will be in 5 minute pieces of video	This value will be controlled by the GigE camera block. It was chosen because if a power failure or camera disconnection occurs, the current video file being recorded will be corrupted. Five minute video pieces will allow for protections to large amounts of data loss while only needing 12 files to represent an hour of data.	This requirement goes with the property above. It sets the 5 minute limit for this block to finish transferring all previous files. As stated above the write speed for the solid state drive will be many times more than enough for the expected six 5 minute files.
Protocol: Data will be saved on the eMMC	This was decided by the GigE Camera block to be the easiest space to store data to. The eMMC is the local memory on the NVIDIA Jetson. This will allow the GigE Camera to pick its own file path to save to.	This is the local memory on the NVIDIA Jetson. The design laid out above will work with this because the boost library’s copy_file function works across storage devices allowing it to copy from the eMMC straight to the solid state drive.

mstr_scrpt_strg_data: Input/Output

Messages: File path, camera number (integer), timestamp (string)	These three values will be passed to the storage program. The file path will be used to locate the file that needs to be moved. The camera number and timestamp will be used to rename the files after they are moved to “CameraX[Timestamp]”.	These three parameters will be all that is needed as input to the design above. They will be used for finding the file that needs to be moved, creating the title to rename the file to using the strcat function, finding the directory to store the file in on the solid state drive, and removing the old file off the NVIDIA Jetson. The pseudocode above does not need any more external variables to function smoothly.
Other: Ability to provide status of if it is still in the process of storing data or is waiting	This property was chosen so that the master script can update the E-Ink display with information on whether a file is currently being saved or not. This is important so that the user does not disconnect the device while a video file is being transferred.	The master script will set a variable to 2 whenever this function is called indicating that storing is in progress. When this function finishes it will return a 3 which will indicate that the storage process is finished.
Protocol: Function Call	The protocol for linking these two blocks will be a function call. The master script will call upon the storage block as a function with the parameters being the information in the messages property. This allows us to make cohesive code but better modularize the functions we want to provide.	This code will work as a function call and a function call only since it requires parameters to work when called. It will return a value at the end as well to signify it has finished.

strg_otsd_data: Input

Datarate: Able to transfer 12 Gigabytes in under 5 minutes spread across 6 files	This amount of data was chosen because 5 minutes of 4k video data at 30 frames per second is 1.75 Gigabytes. We are testing with 6 cameras so if all 6 of them were 4k this would be 10.5 Gigabytes of data. Since the files come in 5 minute chunks the storage device will need to finish transferring all 6 files within that time.	The system will be able to accomplish this since the write rate of the solid state drive attached is 3.2 Gigabytes per second. This means that it will only take around 4 seconds to execute all of the transfers. The other lines of code contain no loops and due to the small size of the code (likely around 20 lines) there will be no issue with extra time on top of the 4 seconds.
Other: Able to detect if files for storage are already present on drive and create them if they are not	The solid state drive will be formatted to have some file directories created on it for storing all video footage. If these file directories are not present when called the device will need to create them for full functionality.	The design above will use the C++ function exists() with a preset file path and using the camera number parameter to verify if a directory is present or not.
Other: Data can be moved to a USB using a monitor and can be played back on another computer without corruption. Shutting the device off while recording or in storage mode will result in one corrupted file.	This is an important thing to verify. Collins will not want to open up the Jetson and remove the solid state drive every time they want to see the data. For this reason it should be verified that data can be removed onto a plugged in USB drive without issue.	This must just be assumed. If the code properly transfers to the solid state drive there is no reason that this should not work. This is a great end cycle verification test to see if the code is functioning as intended.

4.4.5. Verification Process

Interfaces: usb_cmr_cr_strg_data, gg_cmr_cd_strg_data, mstr_scrpt_strg_data, strg_otsd_data

1. Remove all files from the solid state drive.
2. Create a video file that is 30 frames per second and 5 minutes long. This file must be stored on the Jetson's native memory.
3. Use a C++ program to call the function of the storage program with the parameters set to (filepath of video file, 1, Feb_3_22_12:00).
4. While this is running it should print "Currently storing data". When it is finished it should print "Finished storing".
5. Verify that a directory called "Camera 1" has been added to the solid state drive.
6. Verify that the directory contains the video file renamed to "Camera_1_Feb_3_22_12:00".
7. Attach a USB to the NVIDIA Jetson.
8. Using an attached monitor screen, transfer the video file from the solid state drive to the USB.
9. Plug the USB into a laptop computer and play the video file.
10. If the video plays as normal, the title is unchanged, and the properties still lists it as 30FPS then the following interface properties have been verified:
    usb_cmr_cr_strg: 30FPS Video file, 5 minute videos, video stored on eMMC
    gg_cmr_cr_strg: 15FPS Video file, 5 minute videos, video stored on eMMC
    mstr_scrpt_strg_data: Program called with a function call, accepts file path timestamp and camera number parameters, available status of saving process
    strg_otsd_data: Data can be transferred to a USB and played normally, the device can detect

Interfaces: strg_otsd_data

1. Create 6 files each of size 2 Gigabytes (these do not need to be video files).
2. Call the storage function for each file using the file path as the first parameter and 1 for the second and third parameters. Start a timer from the beginning of execution.
3. Once the notification for the final file transfer is done, stop the timer.
4. If the timer reads less than 5 minutes the following interface is verified:
    strg_otsd_data: 12 Gigabytes can be transferred in under 5 minutes

4.4.6. References and File Links

References 4.4.6.1

[1]S. Caldwell, "How much storage space does 4K video take up on your iphone 8 or 8 plus?," *iMore*, 14-Sep-2017. [Online]. Available: https://www.imore.com/how-shoot-trim-edit-and-share-4k-video-iphone#:~:text=30%20seconds%20of%204K%20at,up%203.5GB%20(1.7GB). [Accessed: 05-Feb-2022].

[2] “WD Green™ SN350 nvme™ SSD,” *Western Digital*. [Online]. Available: https://www.westerndigital.com/products/internal-drives/wd-green-sn350-nvme-ssd#WDS240G2G0C. [Accessed: 05-Feb-2022].

[3]MariusMarius, Some programmer dude, syvexsyvex, and icabodicabod, “C++ how to move files and copy them from one disk to different without the usage of winapi?,” *Stack Overflow*, 01-Jan-1960. [Online]. Available: https://stackoverflow.com/questions/9081311/c-how-to-move-files-and-copy-them-from-one-disk-to-different-without-the-usage. [Accessed: 05-Feb-2022].

File Links 4.4.6.2

Copy_File() Reference:

https://www.boost.org/doc/libs/1_48_0/libs/filesystem/v3/doc/reference.html#copy_file

Remove() Reference:

https://www.cplusplus.com/reference/cstdio/remove/

Exists() Reference:

https://en.cppreference.com/w/cpp/filesystem/exists

Strcat() Reference:

https://www.cplusplus.com/reference/cstring/strcat/

Mkdir() Reference:

https://pubs.opengroup.org/onlinepubs/009695299/functions/mkdir.html

Rename() Reference:

https://www.cplusplus.com/reference/cstdio/rename/

4.4.7. Revision Table

3/12/2022	Caden Friesen: Reformatted and move into main project document.
1/21/2022	Caden Friesen: Project was rescoped. Whole document was restarted. All sections written again with new block, interfaces, and verification steps.
1/7/2022	Caden Friesen: Created document and made rough draft of all sections

4.5. Jetson GPIO

4.5.1. Description

This block is called control. The control block is for getting the user input and then signaling the Jetson. There will be one button for the user to turn on recording and another button to turn it off. The other 2 buttons will be for general purpose. This block will get power from the Jetson by using connectors. It will need to use the Jetson 3.3V pin.

4.5.2. Design

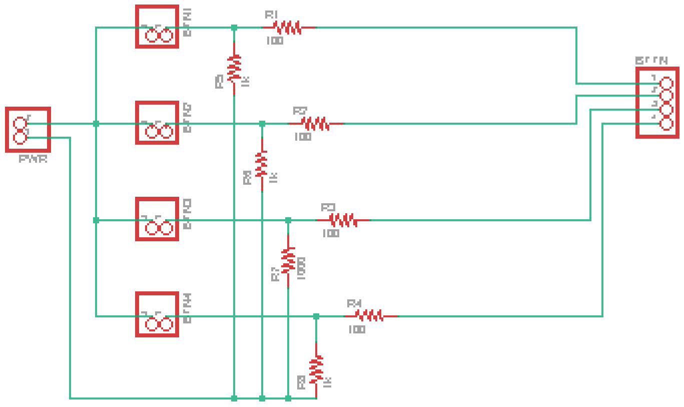


Figure 1: Circuit that connecting the button and the Jetson

4.5.3. General Validation

This design will match what the system needs. The block will be getting power from Jetson by using the pins from Jetson. The block will be supplying the user signal to turn on or turn off the program in the Jetson agx xavier. It will also be able to tell the user it is on or off. The buttons are also hard to push, it is flat at the top, in order to press the button, the user has to apply some force and push down into it, so it won't be touched or pressed by accident and turn the Jetson on and off. This met what the project partner needs for the user interface. The buttons and the resistors needed for the systems are cheap and reasonable to purchase. The parts are all available to purchase online easily on Amazon and other websites. The buttons are also of a high quality. The size of the total block is not big so it fits what the project partner wants. If at the end, the button broke or it does not send the signal to the Jetson, we can ask the project partner and they will buy the buttons and the solutions.

4.5.4. Interface Validation

otsd_cntrl_usrin: input

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Other: Buttons press down X DISTANCE NEED TO CHOOSE	In this example, this property was base on the user interface, it is important that prevent the button pressing too easy. Avoid some unnecessary acident press.	● According the button datasheet, it needs to press down for some distance in order to activate
Other: Buttons have flat non extruding top	In this example, we do not want somethings that cause accidentally pressed in the button. Having a flat non extruding top can reduce the chance of accidentally press.	● The button shows that there is nothing on the top, we can see that is flat non extruding top. ● In the amazon website we can see it is flat
Type: Four Push Buttons are Present	In this example, this property was chosen based on the design we plan to use in the control block.	● In the Amazon website, it has 6 buttons in one package

Table 1: Interface Property Validation for otsd_cntrl_usrin.

prmry_xctbl_cntrl_dcpwr: input

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Inominal: 60mA	This nominal current was chosen based on the expected current needs of the system overall	For Nvidia Jetson Carrier Board Specification[1] ● According to the data sheet, the maximum current is 1A, so the normal current will be around half of this
Ipeak: 70mA	This peak current was	For Nvidia Jetson Carrier

	chosen based on the expected current needs of the system overall	Board Specification[1] ● According to the data sheet, the maximum current is 1A, we do not want to be touch around the maximum, so lower the 1A into 0.9mA
Vmax: 3.35V	In this example, this property was chosen based on the design we plan to use in the control block.	For Nvidia Jetson Carrier Board Specification[1] ● According to the data sheet, the Voltage output from the pin is 3.3V, so the Vmax can be little bit higher in reality
Vmin: 3.25V	In this example, this property was chosen based on the design we plan to use in the control block.	For Nvidia Jetson Carrier Board Specification[1] ● According to the data sheet, the Voltage output from the pin is 3.3V, so the Vmin can be little bit lower in reality

Table 2: Interface Property Validation for mstr_scrpt_cntrl_dcpwr.

cntrl_prmry_xctbl_asig: output

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Other: code that can read button inputs through GPIO pins	In this example, this property is very important because that can show whether the connection is good or not, also if the GPIO is able to read or not	● The Jetson datasheet [1]shows it support the GPIO reading and function
Other: 60-90 mA current (Button Active)	Thisl current range was chosen based on the expected current needs of the system overall	● According to the circuit calculation, it will be outputting the current in the range
Vrange: 3-3.3 volts (Button	In this example, this property	● According to the

Active)	was chosen based on the design we plan to use in the control block.	circuit calculation, it will be outputting the voltage in the range

Table 3: Interface Property Validation for cntrl_mstr_scrpt_asig.

4.5.5. Verification Process

1. For the normal voltage measurement, put the positive in the input, and put the negative end into the ground.
2. For the current requirements, disconnect the button and put the measure in series with the button, and then measure the current with a multimeter.
3. For the maximum voltage and current requirement, we will send the maximum current and voltage to see if the system can handle it.
4. Run the program, start pressing the button 1 and see if the Jetson receives the signal or not. The screen will pop up a message that says which button is being pressed.
5. Do step 4 for button 2-4.
6. Measure the distance between the buttons and the top when we press down the buttons.

4.5.6. References and File Link

[1]“Jetson AGX Xavier - Arrow.” [Online]. Available: https://static5.arrow.com/pdfs/2018/12/12/12/22/1/565659/nvda_/manual/jetson_agx_xavier_thermal_design_guide_v1.0.pdf. [Accessed: 08-Jan-2022].

1. Nvidia Jetson Data sheet
2. Button information from Amazon
3. Nvidia Jetson Carrier Board Specification

4.5.7. Revision Table

4/22/2022	Henry: edit the session
3/12/2022	Henry: edit the table
2/4/2022	Henry: final revision
2/3/2022	Henry: update the each session
2/2/2022	Henry: delete everything and change the block

4.6. Jetson Power Supply

4.6.1. Description

This block is called power supply. The power supply block is for getting the power from the plug outlet of the airplane and then powering the Jetson. The power from the airplane is normally 28V DC. And My block is able to handle between 18V and 36V, it will convert into 48W, 12V and 5.5A to Jetson. The jetson will accept the power and only take the power that Jetson needs.

4.6.2. Design

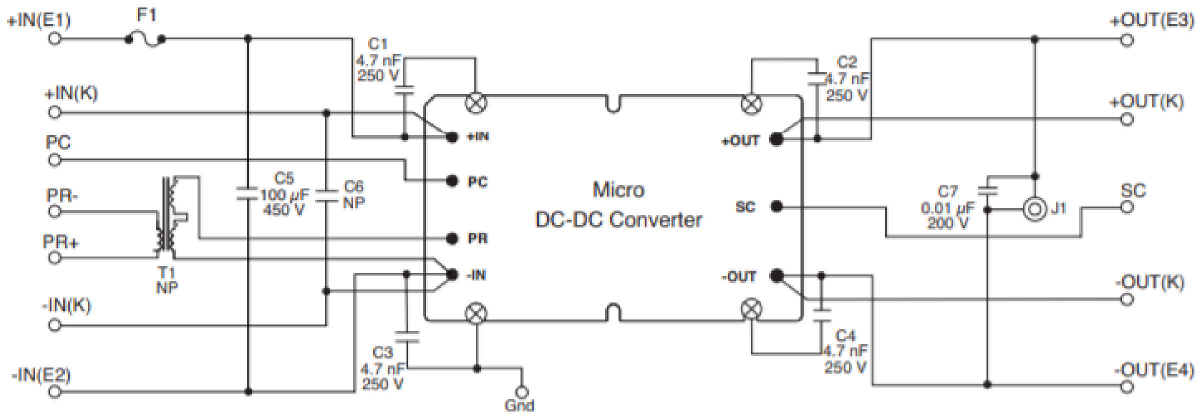


Figure: Verification and validation wiring diagram

4.6.3. General Validation

This design will match what the system needs. The block will be getting from the plug outlet of the airplane and then powering the Jetson. The block will be supplying the power needed for the main system. The parts are all available to purchase online easily in digikey. But for this time, our project partner already bought all the stuff we need so we do not have to pay anything for this block. The size of the total block is not big so it fit what the project partner wants. The power supply of this board is small and only using one chip, no external resistance. If at the end, somehow it does work, we can buy a maded power supply to power Jetson. But the chance will be low because our project partner is already providing the board and the chip.

4.6.4. Interface Validation

otsd_pwr_spply_dcpwr: input

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Inominal: 2.3A	In this example, this property	• According the

	was based on the information provided from the project partner.	V24C12T150BL datasheet, [3] it is able to handle this normal current
Ipeak: 2.5A	In this example, this property was based on the information provided from the project partner.	● According the V24C12T150BL datasheet, [3] it is able to handle this peak current
Vmax: 28V	In this example, this property was based on the information provided from the project partner.	● According the V24C12T150BL datasheet, [3] it is able to handle this maximum voltage
Vmin: 26V	In this example, this property was based on the information provided from the project partner.	● According the V24C12T150BL datasheet, [3]it is able to handle this minimum voltage

Table 1: Interface Property Validation for otsd_pwr_spply_dcpwr.

pwr_spply_mstr_scrpt_dcpwr: output

Interface Property	Why is this interface this value?	Why do you know that your design details for this block above meet or exceed each property?
Inominal: 5.5A	This nominal current was chosen based on the expected current needs of the system overall	● According the V24C12T150BL datasheet, [3] it is able to output this current 3.5A
Ipeak: 7.0A	This peak current was chosen based on the expected current needs of the system overall	● According the V24C12T150BL datasheet, [3] it is able to output 4.0A
Vmax: 12V	In this example, this property was chosen based on the design we plan to use in the control block.	● According the V24C12T150BL datasheet, [3] it is able to output this voltage

Vmin: 11V	In this example, this property was chosen based on the design we plan to use in the control block.	● According the V24C12T150BL datasheet, [3] it is able to output this minimum voltage

Table 2: Interface Property Validation for pwr_spply_mstr_scrpt_dcpwr.

4.6.5. Verification Process

1. For the voltage, put a testing input voltage into 28 V.
2. Then use a multimeter to connect the ground and the output voltage to measure the voltage in the output to test this requirement.
3. For the normal current output requirements. Put a load with 5.5A on the output of the circuit and put a testing input voltage into 28V
4. Use a multimeter to connect the output of the circuit to measure the current to test this requirement.
5. For the maximum voltage and current requirement, we will send the maximum current and voltage to see if the system can handle it. We will count for 30 seconds to see if it is working or not.
6. Check the multimeter if the voltage and current are correct then it is working. The output power is going to be about the same as input power. Power is going to be current times voltage.

4.6.6. References and File Link

[1]“Jetson AGX Xavier - Arrow.” [Online]. Available: https://static5.arrow.com/pdfs/2018/12/12/12/22/1/565659/nvda_/manual/jetson_agx_xavier_thermal_design_guide_v1.0.pdf. [Accessed: 08-Jan-2022].

[2]“Micro DC-DC Converter Evaluation Board user guide.” [Online]. Available: https://www.vicorpower.com/documents/user_guides/brick/UG_Micro_ElvBrd.pdf. [Accessed: 19-Feb-2022].

[3]“Micro DC-DC Converter Evaluation Board user guide.” [Online]. Available: https://www.vicorpower.com/documents/user_guides/brick/UG_Micro_ElvBrd.pdf. [Accessed: 19-Feb-2022].

1. Nvidia Jetson Data sheet
2. V24C12T150BL datasheet
3. https://www.vicorpower.com/documents/datasheets/ds_24vin-maxi-family.pdf
4. https://eecs.oregonstate.edu/capstone/ece/student/reports/blackboxdiagram.php?projectid=675

5. https://eecs.oregonstate.edu/capstone/ece/student/reports/interfacetablewithcomments.php?projectid=675

4.6.7. Revision Table

4/22/2022	Henry: update the section
3/12/2022	Henry: edit the block interface
3/4/2022	Henry: Revision on block interface
3/3/2022	Henry:Revision on each section
2/18/2022	Henry: revision
2/13/2022	Henry: update the each session
2/8/2022	Henry: delete everything and change the block

5. System Verification Evidence

5.1. Universal Constraints

5.1.1. The system may not include a breadboard

The only physical pieces of our design are the power supply and user input blocks. Neither of these is using a breadboard as the user input block is on a student-made PCB (figure 5.1.1) and the power supply is on a premade PCB board (figure 5.1.2). Images of the PCB design and built power supply have been attached below.

5.1.2. The final system must contain both of the following: a student-designed PCB and a custom Android/PC/Cloud application

Since the majority of the project is code-based, there are few physical parts that require a physical connection or additional physical components. Among the physical components, the power supply comes with a premade PCB for its design. Our one remaining physical block, the user input block, has included a student-designed PCB. This is less than 50% of the system but all the PCB that could reasonably be made for the system.

Our system includes a linux based application as the majority of the project. Nearly 90% of this project is coding which meets this requirement.

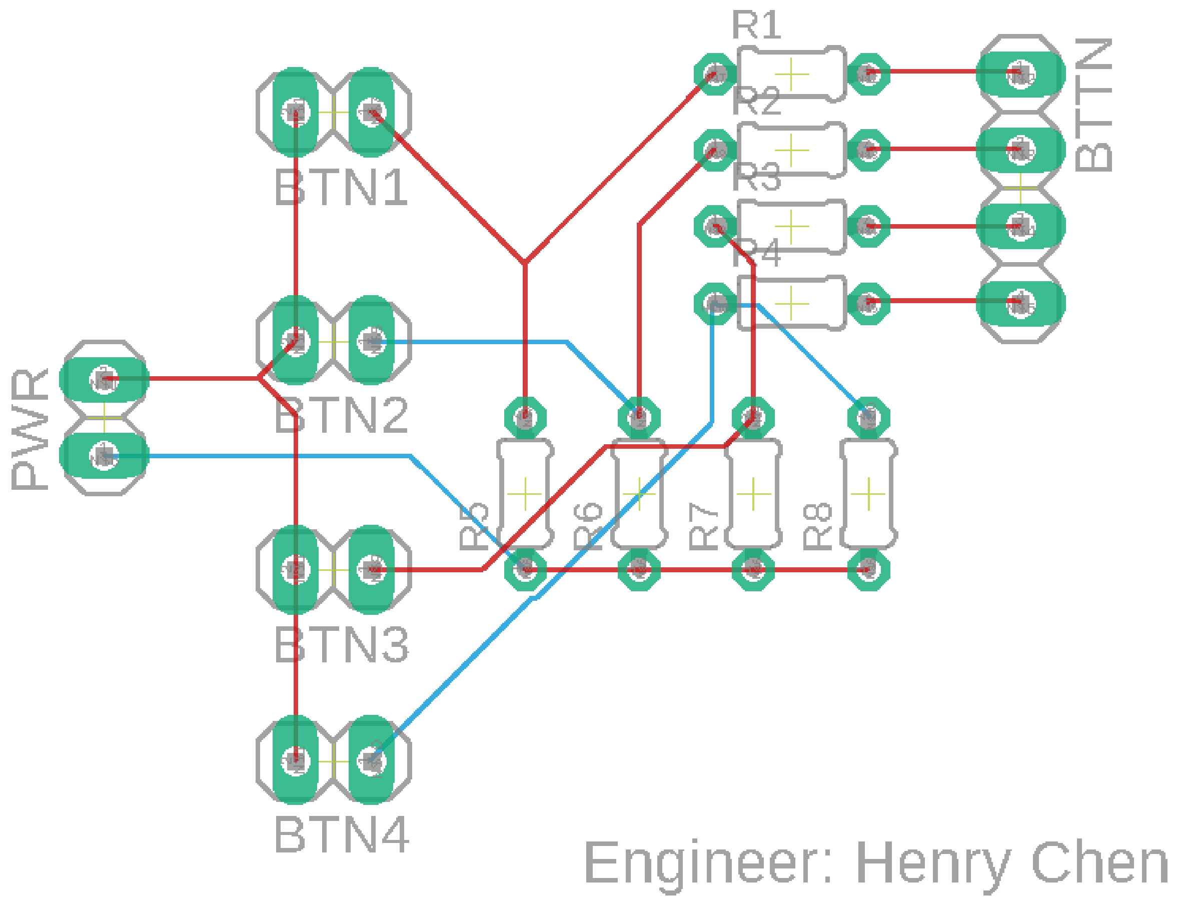


Figure 5.1.1: PCB layout

5.1.3. If an enclosure is present, the contents must be ruggedly enclosed/mounted as evaluated by the course instructor

The enclosure is provided by the project partner and our team has no involvement in designing, building, mounting, or installing the project hardware onto the enclosure itself, nor have we been provided information other than a weight limit. Therefore, this universal constraint does not apply to our project.

5.1.4. If present, all wire connections to PCBs and going through an enclosure (entering or leaving) must use connectors

All the connections will be able to use connectors in the system. Between the button and the PCB and the Jetson, we will be using connectors to connect the power in and signal out from the buttons. The buttons and pieces will all be connected to the PCB design in figure 5.1.1. Attached below is an image of the whole system.

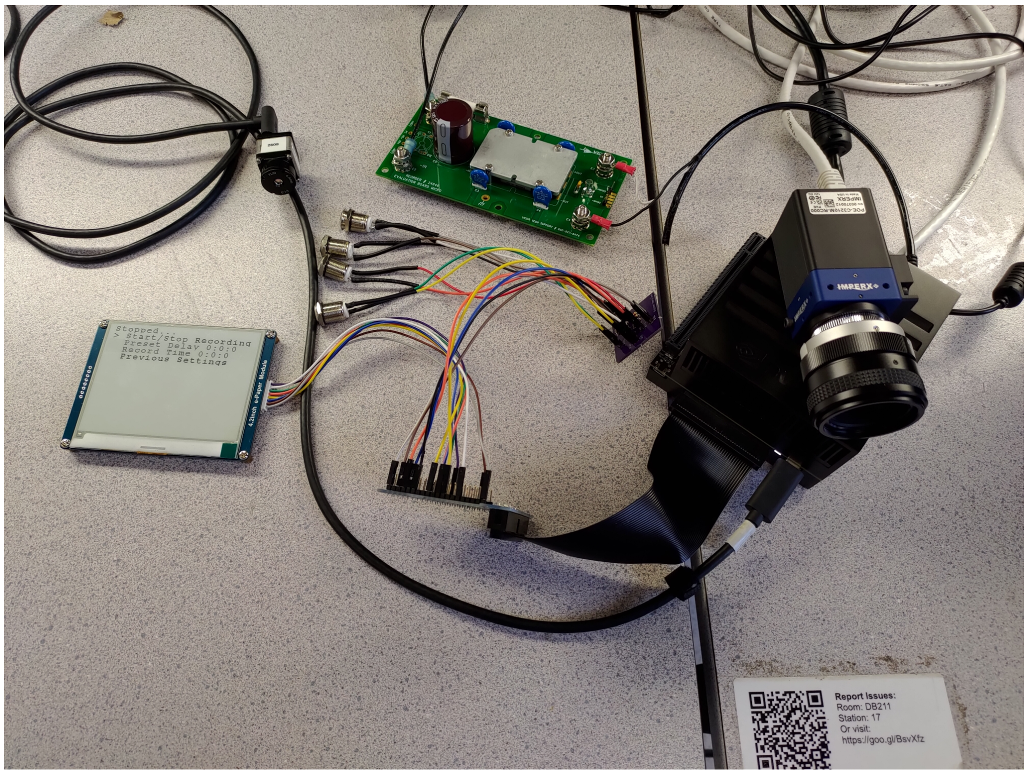


Figure 5.1.3 The entire system connected as one cohesive device

5.1.5. All power supplies in the system must be at least 65% efficient

Figure 5.1.2 shows the power supply taking in 18.1 Volts at 3.8 Amperes and outputting 11.26 Volts at 4.99 Amperes. This results in 68.78 Watts in (18.1 * 3.8) and 56.2 Watts out (4.99 * 11.26). This is an efficiency of 81.7% (56.2/68.78 * 100). This is above the 65% minimum efficiency.

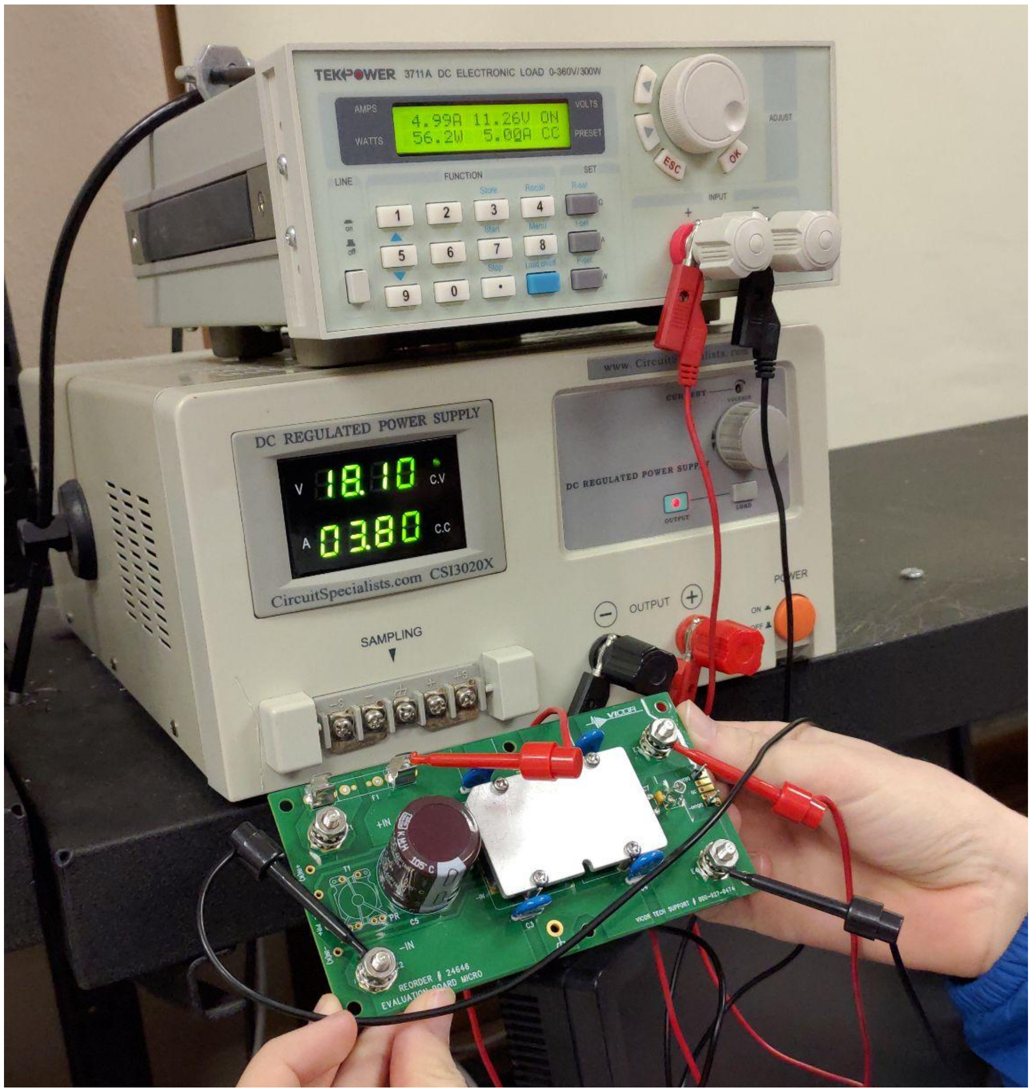

TEKPOWER 3711A DC ELECTRONIC LOAD 0-360V/300W
4.99A 11.26V ON
56.2W 5.00A CC
DC REGULATED POWER SUPPLY
V 18.10 C.V
A 03.80 C.C
CircuitSpecialists.com CSI3020X
SAMPLING
OUTPUT
POWER
VICOR

Figure 5.1.2: The power supply displaying efficiency

5.1.6. The system may be no more than 50% built from purchased modules

As far as blocks go, our user input block, USB and GigE camera blocks, storage block, and primary executable blocks are all designed and built by our team. The camera code blocks use some software provided by FLIR but still build programs around them. Our only prebuilt block is the power supply along with the NVIDIA Jetson AGX Xavier which is the core of our system. This means 5 of 6 blocks are designed and built by us which is more than 50%.

5.2. Flir USB Camera

5.2.1. Requirement

The Jetson will capture video frames from the Boson320 camera and the video frames will be stored as image files at a 10FPS minimum in 320x256 pixel images.

5.2.2. Testing Processes

1. Connect the Boson320 via USB C port on the Jetson along with the Imperx C3210 via the PoE switch.
2. Run APE service.
3. Set the recording time to 1 minute.
4. View data recorded on the Solid State Drive and verify there are at least 600 photos.
5. Check that saved images are 320x256 pixel images.

5.2.3. Testing Evidence

This video follows the steps outlined in 5.2.2 to verify that the Boson320 camera can record images that are the right size as well as at a rate faster than 10 frames per second.

Video From 3/13/2022:

https://www.youtube.com/watch?v=yzQLvrwamJk&ab_channel=Blokdude3456

https://youtu.be/BJ4kXFkERmM

5.3. Imperx GigE Camera

5.3.1. Requirement

The Jetson will capture image data from the Imperx camera and store it as an Imperx RAW image file with the original resolution of 3216 x 2208 pixels at a minimum of 10 images per second.

5.3.2. Testing Processes

1. Connect the camera to the PoE switch
2. Connect the NVIDIA Jetson to the PoE switch
3. Power up the Jetson board
4. Run APE service
5. Set recording time to 1 minute.

6. View data recorded on the Solid State Drive and verify there are at least 600 photos.
7. Check that saved images are 3216x2208 pixel images.

5.3.3. Testing Evidence

https://youtu.be/m1C5xK0qEK0

5.4. Saved Settings

5.4.1 Requirement

The Jetson will store the previously set recording time and preset delay time in a file. These should be able to be set with the user interface.

5.4.2 Testing Processes

1. Powering on the Jetson
2. Configure the recording time and preset delay time to 60 seconds
3. Verify the configuration has been written to the file by checking the .txt files for previous record and previous delay.
4. Power down the Jetson
5. Powering up the Jetson again
6. Verify the configuration is correctly restored when pressing the set to the previous button on the user interface.

5.3.3. Testing Evidence

This video follows the steps above to demonstrate that settings are not only saved for the preferred settings button but are also saved between power-ups of the Jetson.

Recorded 5/4/22:

https://youtu.be/C8T-9eeo2Ts

5.5. Jetson Power Supply

5.5.1. Requirement

The power supply block will take an input voltage of 28V with a 5V margin to produce a steady 12V supply for the Jetson.

5.5.2. Testing Processes.

1. Connect the DC voltage source to the input terminals of the Power Supply Unit.
2. Connect the PSU output wire to the NVIDIA Jetson.
3. Verify that Jetson runs the program and starts up with no noticeable abnormalities. This means being able to run the program and record the files.
4. Monitor the operating input voltage to see if the device is able to handle the variety of input voltage. The expected input from the aircraft is 28V DC. Vary the input voltage from 26V to 30V.

5.5.3. Testing Evidence

This video shows the power supply powering the Jetson as well as it receiving variable drone power and still functioning as intended.

Video from 5/4/22:

https://youtu.be/ZamSdmo7M4o

5.6. User Interface

5.6.1. Requirement

The system will allow 9 out of 10 users to perform the following functions: Manually starting/stopping recording, setting a preset delay for recording, setting a preset recording length, and resetting to previous settings.

5.6.2. Testing Processes

There will be four buttons: A menu button for switching between options, a select button for performing actions, a plus button for adding, and a minus button for subtracting. This step-by-step guide will not explain how to perform each action by each button press but will only use the buttons above.

This process will need to verify that users can use it. A small manual for the functions of buttons will be made and given to participants along with the device.

1. Attach the Boson320 and Imperx camera to the NVIDIA Jetson with the USB C port and PoE switch.
2. Set the delay time to 5 minutes and the record time to 10 seconds.

3. Run the Primary Executable allowing the cameras to record.
4. Hand the participant a list of small tasks they will pass if they are able to accomplish all of these tasks without help.
    a. Stop the recording
    b. Change the delay to 1 minute.
    c. Change the recording time to 2 minutes.
    d. Resume the recording.
    e. At any point after this return the system to previous settings.
6. This will be verified if 9/10 people can accomplish these tasks.

5.6.3. Testing Evidence

These videos demonstrate the user interface working and a test subject receiving the user guide and testing it out. The user is able to follow the directions and demonstrate all its functionality of it.

Videos from 5/4/22:

https://youtu.be/_CDQ8AFpG5I

https://youtu.be/EWjSMp0xCkY

5.7. Storage Organization

5.7.1. Requirement

The system will output image files to an SSD that will be organized by camera number.

5.7.2. Testing Processes

1. Attach the Boson320 and Imperx camera to the NVIDIA Jetson with the USB C port and PoE switch.
2. Run the Primary Executable allowing the cameras to record.
3. Stop the recording after 10 or more seconds
4. Check that files are stored in the folder on the solid state drive named “CameraX” where X is the number assigned in the primary executable included in the title (this number should be verified during a verification test).
5. Check that both cameras are stored properly in different folders.
6. Check that none of the files are still stored on the local memory.
7. Run this test one more time with this variation:

a. Run the test with the folder for the camera deleted and verify this folder is created on running.

5.7.3. Testing Evidence

This video follows all of the steps outlined in 5.7.2 to show files being moved from local memory to a solid-state drive that is organized by camera number.

Video From 3/13/2022:

https://www.youtube.com/watch?v=1YLHA3McY5U

This video repeats the experiment but with all of the systems integrated as one.

Video From 5/4/22:

https://youtu.be/eIOhud2yhPo

5.8. File timing

5.8.1. Requirement

The system will output images with the timestamps on the file name.

5.8.2. Testing Processes

1. Attach the Boson320 and Imperx camera to the NVIDIA Jetson with the USB C port and PoE switch.
2. Run the Primary Executable to record the cameras.
3. Stop the recording after one minute.
4. Check for the files stored in the Camera1 folder on the solid-state drive.
5. Verify that the files have a range of timestamps one minute long across all their names.

5.8.3. Testing Evidence

This video demonstrates both timestamps and folder organization in the solid-state drive.

Video recorded 5/4/22:

https://youtu.be/eIOhud2yhPo

5.9. Autonomous Operation

5.9.1. Requirement

The system will operate without user intervention when power is applied. If a preset delay or recording length was chosen this feature will continue from where it left off 10 out of 10 times.

5.9.2. Testing Processes

1. Attach the Boson320 to the NVIDIA Jetson with the USB C port.
2. Run the primary executable to record the camera.
3. Disconnect power from the NVIDIA Jetson.
4. Reconnect power to the NVIDIA Jetson.
5. Check that recording has begun again by viewing if new files are appearing in the Camera1 folder.
6. Stop recording and create a preset delay of 1 minute, and a recording length of 1 minute.
7. Restart the system with this preset delay applied.
8. View on the display that the delay counts down from 60 to 30 seconds.
9. Repeat steps 3 and 4.
10. View on the display that the delay continues counting down from 30 seconds before it begins recording. This can be verified in step 5.
11. View on the display that the preset recording length counts down from 60 to 30 seconds.
12. Repeat steps 3 and 4.
13. Verify that once the system resumes recording it records for only 30 more seconds.

5.9.3. Testing Evidence

This video demonstrates what we have currently working for this requirement. This requirement still needs fixes to its memory storage and automatic booting of the code.

Video recorded 5/6/22:

https://youtu.be/COjQmnUcZPo

5.10. References and File Link

5.10.1. References

5.10.2. File Links

5.11. Revision Table

4/22/2022	Caden: Adjusted grammar and updated steps to most testing sections to fit spring.
4/22/2022	Henry: update 5.5
3/13/2022	Adjusted verification steps for 2 and 7. Improved and added to all global requirements including pictures. Added evidence for 2 and 7. Added 5.9 Recorded and added videos for 2 and 7
3/12/2022	Henry: Edit the section 5
3/6/2022	Caden: Wrote starting evidence pieces
3/6/2022	Anthony Kung - Created Section 5

6. Project Closing

6.1. Future Recommendations

6.1.1. Technical Recommendations

For the project PCB, a 40-pin header is recommended for quick and easy connection between the Jetson AGX Xavier and the peripherals. Labeling is also recommended on the PCB to provide easier installation. A similar and better PCB design from the Doorgy Project [6.4.2.1] could serve as a reference design for a better PCB design, which includes a 40-pin header, and individually labeled peripheral connections.

The use of Waveshare e-Paper display is significantly more complex on the Jetson platform, since the library did not use the Jetson SPI library and create the SPI signals themselves, the refresh rate of the e-Paper display is significantly impaired. Some recommendations include creating a new e-Paper library using the Jetson native SPI support, using better hardware such as Raspberry Pi Single-Board Computer, or using a different display option such as an LCD display or even a mini HDMI display with a user interface built with Electron.

One major shortcoming of the project was getting the project to run the code effectively on boot up. We ran into issues with both our IPC not connecting and our solid-state drive not mounting immediately on boot. If there was one major spot, to begin with improving this project it would be here. Research into how to make sure a solid-state drive mounts without human interaction smoothly as well as verifying that IPC will connect. Over the process, it is possible that we had corrupted some pieces of our NVIDIA Jetson so the ability to start from scratch on this project could be a benefit to this. We made an assumption that things would work the same way running on boot as they did running manually and that was a major oversight in our project. A tutorial linked in 6.4.2.2 may be a good place to start better understanding IPC for this project.

If a team were to take this project and work another year on it, one major improvement would be to make it more adaptable to other cameras. We learned throughout the project that implementing a system that was adaptable to what cameras would be connected was not feasible. If we had more time, however, making smoother code that is more compartmentalized and allows for easier addition of cameras would be very beneficial. Allowing users to write code for a new camera and just call functions from the project, followed by just adding a simple call to their new camera in the primary executable would make this project much more reusable and useful overall. Researching and planning out in advance how to better modularize the codebase would be the best place to tackle this. A guide for this has been linked in 6.4.2.3 below.

6.1.2. Global Impact Recommendations

To help mitigate the issue of large computer chip companies sourcing work from underdeveloped countries and treating their workers very poorly we would recommend a future team focus on responsibly sourcing parts for this project. We think it is important to only support companies that care about ethical work situations and value their employees.

Our second recommendation is to make sure the project meets the IEEE 1156.1 standard [1] on microcomputers, and the MIL-STD-810 military standard [2] which includes test methods for various scenarios of potential risks. It is crucial that this device does not cause issues for a plane or drone especially when people are involved. Making sure the device meets proper safety standards is just as important as making sure it works functionally. We recommend following these specifications extremely closely to mitigate these risks.

6.1.3. Teamwork Recommendations

Over the course of the term our team sometimes struggled with staying organized and getting together to get things done. We would recommend two major things for future groups. Firstly semester 1 should be used to better unify your team. Doing assignments in person will not only result in assignments being done faster but will actually allow your team to bond and get to know each other in the process[3]. This will be key to better teamwork later. Our team did not take advantage of semester 1 well enough and it resulted in our team not having built chemistry by semester 2.

A second recommendation that goes along with the first is to make an excel sheet with the times everyone is regularly available. There will always be other homework so seeing the times people don't have courses and planning meetings in advance is key[4]. This spreadsheet will allow a leader to pick a time and declare a meeting rather than a team bouncing back and forth about what times people are available and never choosing. Once our team implemented this in semester 2 group meetings were far more regular and got a lot more work done.

One final bonus piece of advice to throw in is something Don told us at the start of semester two when planning other meetings. He said that having homework should not be an excuse to not get this work done because this is also homework but your teammates are also relying on you for it. This should take priority to get done. Once that was in perspective it was much easier to put this project first.

6.2. Project Artifact Summary with Links

The ARGH project includes a project showcase website highlighting the key features of the project as well as the project video updates throughout the development process.

https://argh.anth.dev

This link leads to our GitHub with all of the code we used in this assignment as well as some basic guides for setting it up on a device. The vast majority of our work is contained here.

https://github.com/Anthonykung/ARGH

This link leads to a user guide for the button interface of our system. It teaches what functionalities are available and which buttons to use to do them.

https://docs.google.com/document/d/18Z7XfFBTm8etnC6VymKIWAIs4GEBXvRME5v65HdPamI/edit?usp=sharing

6.3. Presentation Materials

The ARGH project includes a React.js and Markdown powered MDX Slide along with a project showcase poster.

Project Poster: https://anthos.link/docs/ARGH_Project_Poster.pdf

Project Slide Deck: https://github.com/Anthonykung/ARGH/tree/main/docs

6.4. References & File Links

6.4.1. References

[1] "IEEE 1156.1-1993 - IEEE standard microcomputer environmental specifications for computer modules," IEEE SA - The IEEE Standards Association - Home, 17-Jun-1993. [Online]. Available: https://standards.ieee.org/standard/1156_1-1993.html. [Accessed: 28-Oct-2021].

[2] "MIL-STD-810," Wikipedia, 18-Oct-2021. [Online]. Available: https://en.wikipedia.org/wiki/MIL-STD-810. [Accessed: 28-Oct-2021].

[3] The Mind Tools Content Team By the Mind Tools Content Team, "Working in a virtual team: Using technology to communicate and collaborate," *Career Development From MindTools.com*. [Online]. Available: https://www.mindtools.com/pages/article/working-virtual-team.htm. [Accessed: 06-May-2022].

[4] M. Woodward, "Here is a step-by-step how-to plan to set up effective meetings," *The Balance Small Business*, 21-Jan-2019. [Online]. Available: https://www.thebalancesmb.com/simple-steps-for-planning-meetings-4105855. [Accessed: 06-May-2022].

6.4.2. File Links

[6.4.2.1] Doorgy Project PCB: https://github.com/Anthonykung/Doorgy/tree/main/docs/pcb

[6.4.2.2] "Inter Process Communication Tutorial," *Inter Process Communication tutorial*. [Online]. Available: https://www.tutorialspoint.com/inter_process_communication/index.htm. [Accessed: 06-May-2022].

[6.4.2.3] “Single page apps in depth,” 2. Maintainability depends on modularity: Stop using namespaces! [Online]. Available: http://singlepageappbook.com/maintainability1.html. [Accessed: 06-May-2022].

6.5. Revision Table

5/6/2022	Henry: edit section 6.1.2 and other sections
5/6/2022	Anthony Kung - Added Section 6.1.1 - Added Project Website Section 6.2 - Added Section 6.3 - Added Section 6.4.2 - Added Section 6.5

A. Appendix

Intentionally left black for the time being.