Wireless Solar-Powered Thermal Imaging Camera

By:

Andy Bonk
Billy McVicker
Jacob Richardson

Sponsored By:

FLIR

Mechanical Engineering Department

California Polytechnic State University, San Luis Obispo

December 2011

Table of Contents

List of Tables

List of Figures

Nomenclature

DC	Direct Current
LFP or $LiFePO_4$	Lithium iron phosphate (battery)
Li-Ion/po	Lithium-Ion/Polymer (battery)
PWM	Pulse Width Modulation
QFD	Quality Function Deployment
SLA	Sealed Lead Acid (battery)
STC	Standard Test Conditions(solar)

Introduction

FLIR Corporation currently produces thermal imaging cameras for military, commercial and personal use. Surveillance is a high technology sector, with constantly evolving systems to match the evolving strategies of threats. Within the field of surveillance, large commercial customers typically have vast property they need to protect from theft and safety threats, forcing them to strategically place cameras to minimize cost and maximize the surveillance area.

Thermal imaging allows for easy detection of threats when visible light does not suffice. As you can see below in Figure 1, the infra-red spectrum allows for enhanced vision of threats even during daylight and camouflage conditions. This enhancement allows customers to greatly increase their security by providing a level of detection not available with normal security cameras.

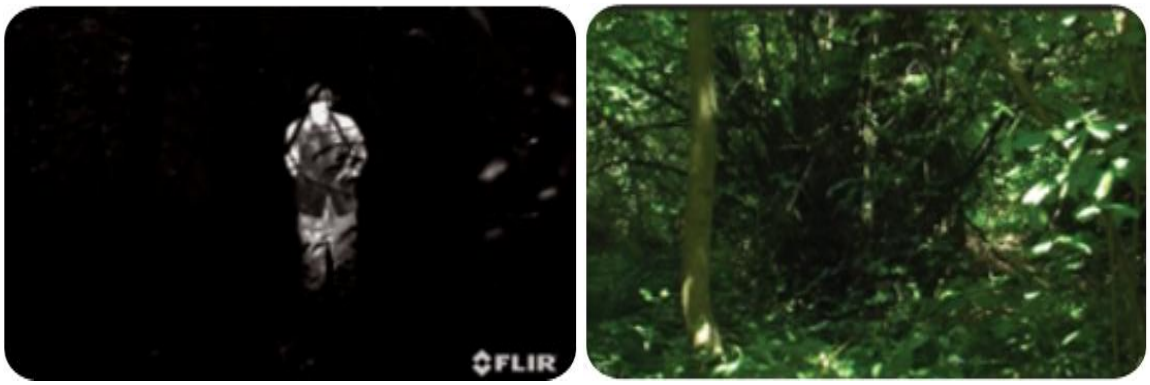


Figure 1: Comparison between thermal and visible spectrum

Often these cameras are placed on high posts or walls far from the central complex. A large cost in many security systems is trenching and wiring the system to the security hub. The main goal of this project is to mitigate this cost by developing a wireless communication system along with a solar-charged, battery stored, power supply. By developing this system, we will allow for a thermal security installation with no trenching or wiring required. This should significantly reduce cost of the installation of the system for the user.

A system will be designed to incorporate an existing camera system with a power generation and storage system as well as a means of wireless communication back to a security station. As with many security systems, the ability to continue to run when power is not available is very important. The camera must be able to sustain continuous operation in conditions such as at night, or after several days of bad weather. With this in mind, a large power storage capacity is necessary in conjunction with minimizing the power dissipation of the system, to allow the camera to run longer and use a smaller power storage system.

Background

The main intent of the project is to develop this system using existing technologies. This will result in a cheaper, more easily understood product. We should be able to utilize mainly off-the-shelf parts, with our mounting brackets for the camera and solar panel the only possible new part designs. This should allow for a quick lead time on production, as most components can be sourced through vendors.

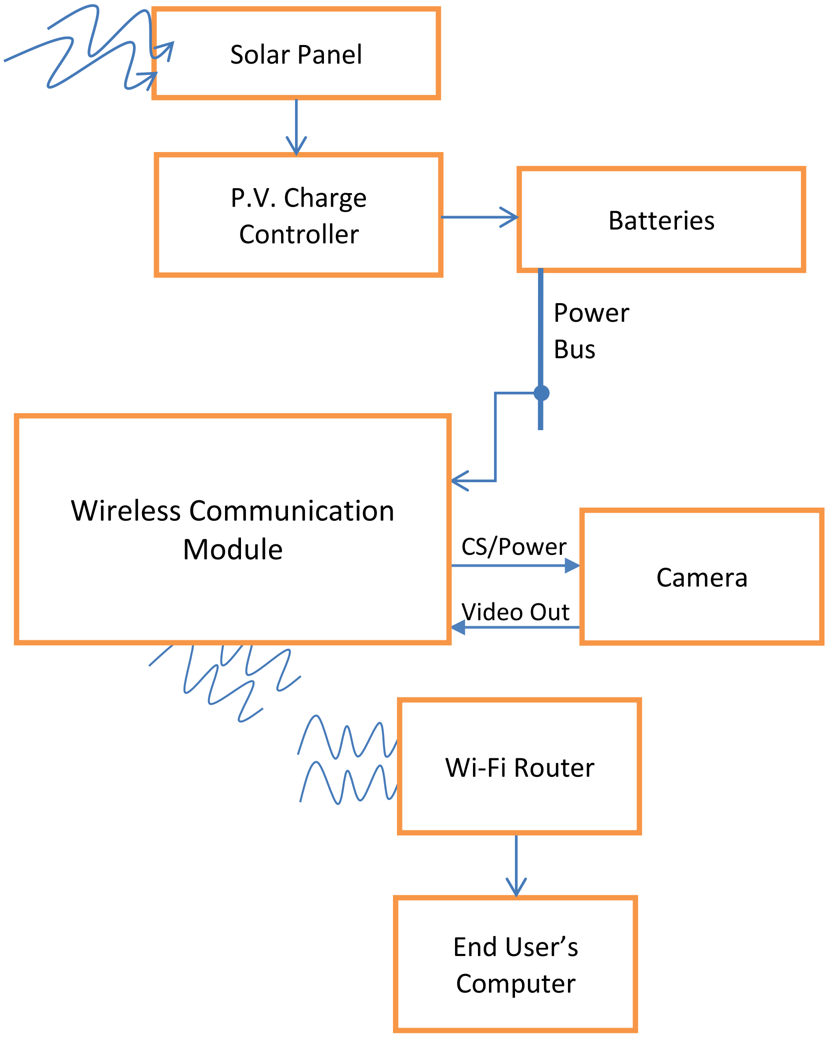


Figure 2: System Block Diagram

Solar

Solar insolation is the level of irradiance provided by the sun to a given surface on the earth. It is often measured in kWh/m^2/day, or in the number of direct hours of sunlight. NASA has conducted a large study of solar insolation levels. We have used this to find the levels for Pullman, Washington. The minimum is about 2.9 hours of direct sunlight per day. Using this information and the estimated power usage of our system, we can size the solar panel required for a 100% duty cycle.

Most solar panels are constructed from crystalline silicon modules. This semiconducting material absorbs the photons from the sunlight. The photons knock electrons loose from their atoms, causing an electricity flow through the material. The end product is a DC current produced purely from sunlight.

Most commonly these solar modules are mounted on a rigid substrate. This allows for the panels to undergo some forces without deflecting. They are also manufactured on a flexible substrate, which is often used to mount them directly to an object.

Solar panels are typically rated under a set of standard test conditions. These conditions include an irradiance of 1000 W/m^2, module temperature of 25°C, and solar spectrum of Air Mass 1.5. The air mass coefficient characterizes the solar spectrum after it has passed through the atmosphere. Direct sunlight at sea level would have an AM of 1.0. However, for most parts of the world the sun is at an angle to the surface, so it passes through roughly 1.5 times as many air molecules. The characteristics tested for are typically maximum power output, open circuit voltage (the maximum voltage available), and the short circuit current. These values are the maximum available at the STC conditions listed above, not the maximum possible.

The effects of dust and other particulate accumulation on solar output has been studied. It was found that the power output of the panels was reduced by less than 5% over time due to particulate buildup[1], using a short circuit test. This would be a higher effect for a different load on the circuit. It should also be noted from the study that rain had a good effect on cleaning off the glass, so with proper sizing it may be possible to not need to clean the panel every month, or to just spray it off with a hose in the dry months.

Charge Controller

A charge controller will need to be used to ensure that the power storage system is operating efficiently, does not damage the batteries and ensure the solar panel does not drain the batteries. Because different battery chemistries charge at an optimal rate under different conditions, there are a number of different types of charge controllers. The function of charge controlling can, with some minor hardware circuit design, be incorporated with a main microcontroller which would provide the necessary logic for PWM regulation and feedback of the charge status.

For optimal charging of lead-acid batteries a multi-phase charge cycle must be used. The initial phase is a constant current charge. This is followed by a constant voltage charge cycle which finishes charging the cell. Finally, if the battery is fully charged and power is still available to the charge circuit the voltage and current are allowed to float and match the state of the cell.

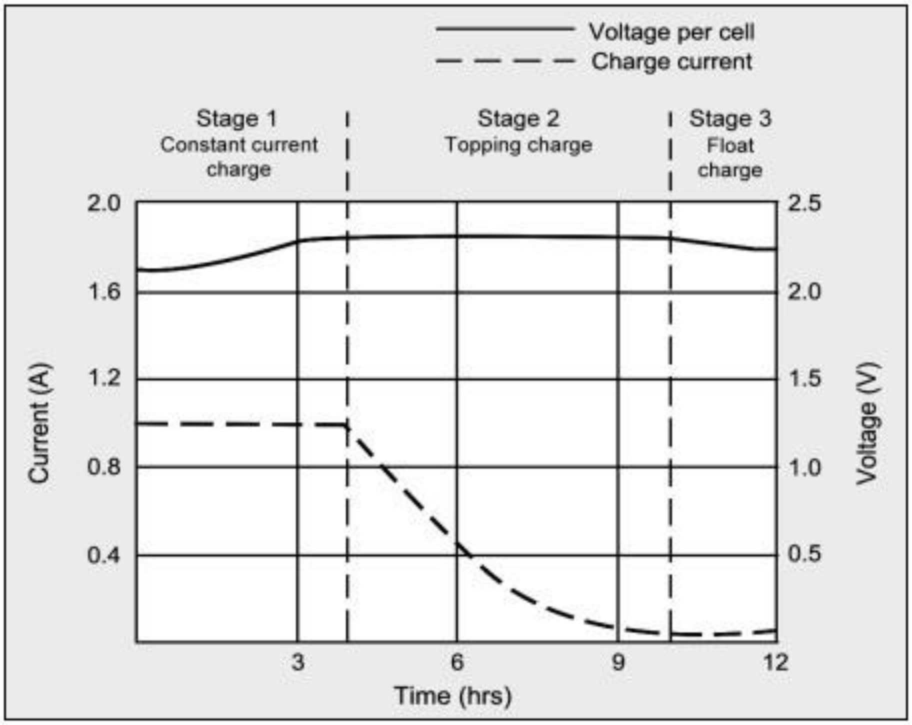


Figure 3: Typical Charge Profile of Lead-Acid Batteries[2]

In the case of lithium-ion chemistry batteries the charge cycle has similar traits to lead-acid but is not identical. The voltage linearly increases with a constantly applied current but above 4 V per cell the linear relationship falls apart. After the cell reaches the 4 V per cell limit the voltage is held constant at the maximum cell voltage until the current falls off to the manufacturer's specified current. Once the current reaches this cut off all charging of the cell stops at the risk of damaging the cell, even causing the cell to rupture and explode.

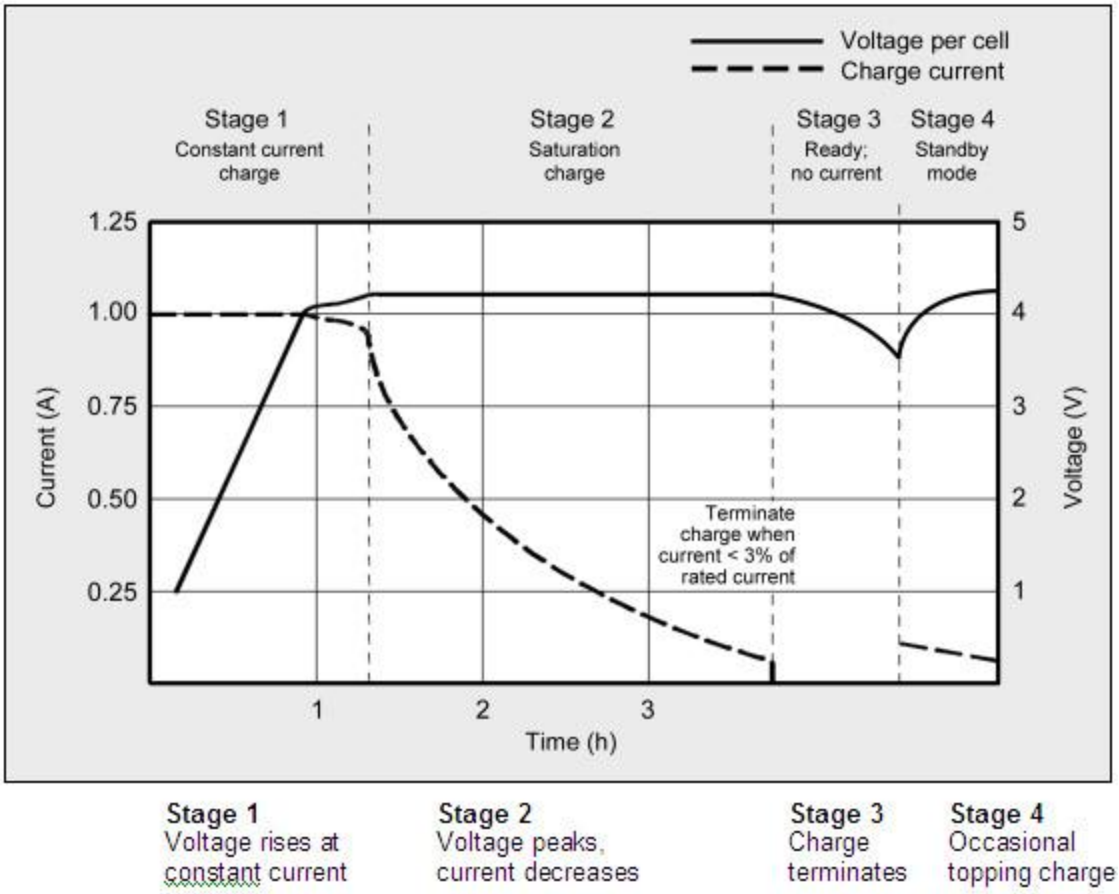


Figure 4: Typical Charge Profile of Li-Ion Batteries[3]

Batteries

Our system will require a secondary (rechargeable) battery, or some other storage device. Common battery types include lead-acid, nickel-cadmium, and lithium-ion. During charging these types of batteries, the positive active material is oxidized, producing electrons, and the negative active material is reduced, consuming electrons. During discharge, the opposite process happens.

Lifespan for these type of batteries varies by type. A lead-acid battery used in a car typically lasts 5-8 years, and can last up to 20 years in a stationary environment. Lithium Ion batteries tend to have a significantly shorter lifespan, at only 2-3 years, but have become increasingly popular because they have roughly 5 times the energy density.

The other consideration we had for energy storage was an ultracapacitor. These are a relatively new technology, and are currently being developed for widespread use in hybrid or electric cars. Smaller versions of these are available, but they are not available as widespread as lead-acid or lithium-ion batteries. This means that the consumer may have problems finding a quick replacement, and FLIR may find difficulty in locating multiple vendors.

Power Consumption

Minimizing the power dissipation of a system originally attached to a near infinite power supply is a large concern with this project. From the information provided by FLIR, the Tau 320 sensor core unit dissipates less than one watt. In order to communicate with the camera, to change the gain and to receive the video stream, an Ethernet module was also provided. The apparent power requirement for this unit was initially a concern, because the power supply is 9 Volts at 1.66 Amps with a power dissipation of 15 watts. A power draw this large would potentially require very large batteries as well as solar panels to charge them. After acquiring a Kill-A-Watt meter, and running the camera powered through the Photon 320 Ethernet Module for 75 hours, a total of 0.43 [kW h] was dissipated. This is 5.73Watts on average and when taking into account the potential 2% maximum error listed in the documentation for the meter 5.85 W can be used as an estimate of the steady-state power consumption of these elements. Though the continual power draw is less than the 15 W listed on the power supply of the Ethernet Module, the possible startup draw or other short-term spikes in draw will need to be taken into account for the design.

For the wireless communication aspect of the task the power consumption of various wireless access points were examined specifically those from Cisco as the product data sheets listed the power draw of their products. The greatest draw was listed as 13 watts maximum for Aironet- 1040, 1140, 1200, 1240, 1250, 1260, and 3500 Series wireless access points with other series dipping as low as 9.9 watts for the AP-541N, Aironet 1030G, and 521 Wireless Express Wireless Access Points. This gives a general guide of what can be expected for a wireless access point's power consumption. The total power consumption for the wireless could potentially be lowered with the creation of a custom board with the networking functionality built in with very low power integrated circuits but this would be at the expense of buying off the shelf components for the simplest possible sourcing and assembly.

Mounting and Housing

The main market for this system is consumers mounting systems at the far limits of their property. This means the camera will most likely be pole mounted, although it is possible that it could be wall mounted. The mounting system should allow for either type of mounting with some type of adapter.

FLIR currently makes both housing and a mounting solution for our camera core. If used, we would still need to create a mount for the solar panel. If we utilize some type of acid battery that requires venting, we will need to put it outside the housing in a separate box. This will leave us with plenty of room inside the housing for our wireless package, so we think it is likely that we will be able to utilize the existing housing.

Video Processing and Wireless Communication

One of the major wireless security camera providers is Swann Security. Their most comparable security camera they provide is the ADW-300 Digital Wireless Security Camera with Zero Interference, which is represented by Figure 5. This camera is not a thermal imaging camera, but does incorporate the wireless feature FLIR requires for this project. The wireless transmission range necessary for the project is 300ft, but this camera is only rated at 165 ft. Although this camera doesn't hold to the requirements of our projects, we can still use it as a model in order to learn how to incorporate wireless technology with the thermal imaging camera we are to build.

Figure 5: ADW-300 Digital Wireless Security Camera found at Swann.com

In order to achieve wireless communication, we will need a transceiver (a wireless receiver and transmitter) to handle the transportation of the video to and from the camera. To achieve maximum range and modularity, we will use the IEEE 802.11n wireless communication protocol. We can either use an off-the-shelf wireless access point that could attach to the Pleora Ethernet Module or we can use a microcontroller with a wireless module. After researching these two possibilities, we composed the following information.

Potential wireless access points are the TEW-654TR Mini 300Mbps Wireless Travel Router and the TEW-455APBO 14dBi High Power Wireless Outdoor PoE Access Point both made by TRENDNET.

Table 1: Wireless Access Points

	TEW-654TR Mini 300Mbps Wireless Travel Router	**TEW-455APBO 14dBi High Power Wireless Outdoor PoE Access Point**
Power Consumption	2.5 watts (max)	6 watts (max)
Wi-Fi Compliant	IEEE 802.11b/g/n	IEEE 802.11b/g XR
Coverage	200 meters (650 ft)	50-300 meters (164ft –984ft)
Temperature	0 to 40 degrees C	-30 to 60 degrees C
Waterproof	X	IP66/67 compliant
Price	~$50 USD	~$200 USD

The downside of using a wireless access point as the transceiver is the necessity for a device to convert the video signal from the camera to an ethernet signal. The device provided by FLIR that can do this is the Pleora Ethernet Module. Ideally, it would be best to integrate these devices into a single device that could handle all the processing of the unit using a microcontroller because of the power consumption ratings. We found by using a Kill-A-Watt meter, as noted above, the camera and Pleora Ethernet Module consumed 5.73 watts on average. If we add the wattage of the wireless module to this, then the total power consumptions would add up to, at minimum, 7.73 watts using the TEW-654TR Mini 300Mbps Wireless Travel Router.

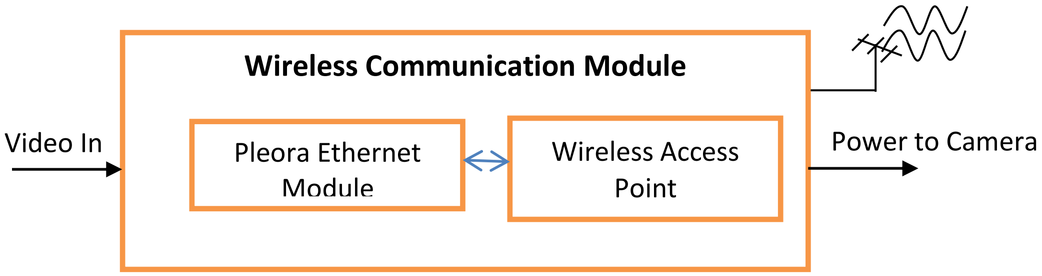


Figure 6:Wireless Communication Module with wireless access point & Pleora Ethernet Module

The TMS320DM355 digital media system-on-chip (DMSoC) processor is a commonly used processor for visual media applications. A white paper was written by TI analyzing power consumption for Common TMS320DM355 Application Usage Scenarios. A Live Preview application was setup where video was received from an image sensor and displayed to an LCD through the video port back end (VPBE). No video compression (usually used to reduce the size of the video) was used and the frame rate was 30 fps. Power consumption results can be found on the TI website at http://focus.ti.com/lit/an/spraaz5a/spraaz5a.pdf. To sum them up, the overall system used 232 mW. This application jointly coupled with a wireless module would be ideal in saving power and handling the video processing.

Through research we have found a manufacturer that implements the DM355 chip in a small power efficient package known as the Leopardboard 355 from Leopard Imaging, Inc.. This board utilizes the DM355 processor and contains the following communication protocols with which we would use: 10/100 Ethernet Port, SDIO, GPIOs, and UART. According to TI, when the Leopardboard 355, with a camera board attached, is streaming 720p video @ 30fps its power consumption averages at 2W1.

A wireless module that would easily interface with the Leopardboard 355 is the OWL222a OEM Wireless LAN Module from connectBlue. This device's power consumption averages at 612 mW. Using a wireless module with the Leopardboard, we can save our unit on average 5.1 W of power (Pleora Ethernet module uses 2.5 W and the Wireless AP uses 2.5W). This wireless module uses the IEEE 802.11n communication protocol for wireless transmission. To communicate with the board, the SDIO communication protocol can be used. This communication protocol is rated to transmit data using a maximum clock frequency of 50MHz, which works out to a transmission rate up to 25 MB/sec (200 Mbps). Refer to Appendix J:Tau-Leopardboard-OWL222a Pin Connection Diagram for the Architecture design of the wireless communication module using the Leopardboard 355 and the OWL-222a wireless module. Refer to Appendix K for the software flowchart to be implemented on the Leopardboard.

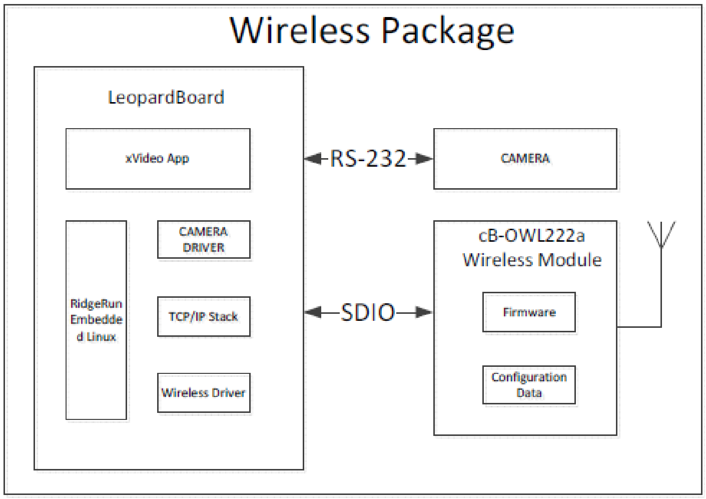


Figure 7: Wireless Package Architecture

Environment

FLIR's existing cameras (F-Series and SR-Series) that the wireless functionality will augment are designed to spend their lifetime outdoors exposed to various climates. In the presentation introducing the project the enclosure(s) were described as needing to be sealed to IP67 levels meaning they are dust tight and sealed for water immersion up to 1 m. The environmental operating temperature range is expected to be -20°C to +55°C. The region to use for determining the solar energy available will be Pullman Washington with a minimum solar insolation of 1.06 kWh/m^2/day which will need to charge the system and allow it to run for 24 hours.

[1] http://processors.wiki.ti.com/index.php/Leopard_Board_DM355_Hardware_Overview

Existing Products

The FLIR F-Series stationary outdoor camera weighs 9.5 lbs without the hardware to mount to a structure or pole. The camera’s dimensions are 18.1”x5.5”x6.3”. The unit draws, without heaters, 10 W. There are two output options, analog BNC and Ethernet both requiring a physical connection to the unit. The camera housing is extruded aluminum with a cast base to attach to mounting hardware and a sheet metal solar shield to help maintain a lower temperature inside the unit.

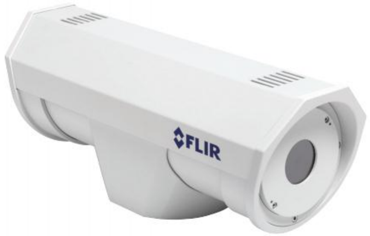


Figure 8: FLIR F-Series Camera

FLIR additionally produces a smaller SR-series which lacks the networking ability with only a video output. The size of the SR-series is 10.5”x5”x5.7” roughly half the length of the F-series cameras. These cameras consume a nominal 3 W but can consume up to 10 W unheated.

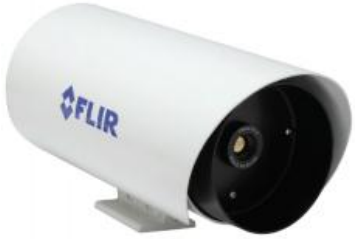


Figure 9: FLIR SR-Series Camera

Axis communications has produced 2 outdoor thermal cameras (Q1921-E and Q1910-E) that connect to existing wired networks. The casings of these cameras are IP66 rated (dust proof and withstands water jets). The dimensions of both the Q1921-E and Q1910-E cameras are 15.9”x6.4”x4.7” (without mounting brackets). The Axis cameras do benefit from having built-in two-way audio.

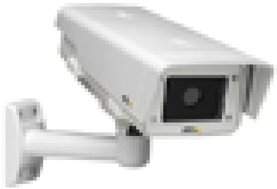


Figure 10: Axis Q1910-E Camera

Pelco produces one stationary thermal camera series:TI2500 Series. This camera lacks Ethernet connectivity and consumes 4 W. The operating temperature range is -32°C to 55°C. The casing meets IP66 specifications (sealed against dust and water jets).

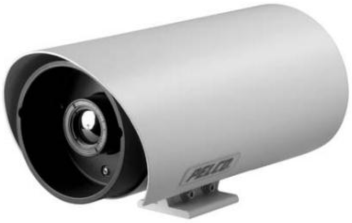


Figure 11: Pelco TI2500 Series Camer

In February 2011 FLIR has released a new lineup of handheld infrared cameras which feature an integrated touch screen interface and the ability to connect wirelessly to a Wi-Fi network and download images from the handheld device to an Apple mobile device such as iPhone, iPad, or iPod. These units are powered by a Li-ion battery and will run for up to 4 hours per charge. Though this unit features wireless connectivity the ability to continually monitor over the wireless connection seems not to exist, only the ability to transfer still images. These units are designed to be hand-held and mobile, not mounted for surveillance and as such lack their own charging system.

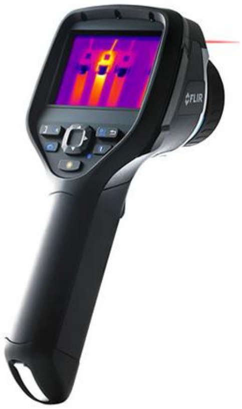


Figure 12: FLIR E-Series Camera

Objectives

The main objective of the project is to present a fully working prototype to FLIR. This prototype should be presented in a manner that would be commercially manufacturable by the company. The final design specifications should include materials to be used, CAD drawings of the design, and multiple potential sources for any parts bought.

The prototype should consist of a thermal imaging camera that is powered via a solar panel and is wirelessly accessible to the user from a maximum distance of 300 feet. The camera must weigh a maximum of 20 pounds, with dimensions 20”x8”x10”, meet operating temperature range of 55 to -20°C, meet FCC requirements in regard to wireless communication, and meet the weatherproof IP67 requirements. Refer to Table 2below for other specifications relating to number of screws, serviceability, modularity, overall cost, and operation time. These specifications were established by the sponsor, FLIR, and discussed at the first sponsor meeting. The specifications were originally targeted toward customers that need surveillance technology, but cannot afford or are not willing to cover the expenses of trenching power and communication lines a maximum distance of 300 feet. This product should be usable both commercially and residentially.

Table 2:Project Specifications

Spec. #	Parameter Description	Requirement or Target (units)	Tolerance	Risk	Compliance
1	Weight	20 [lb]	Max	M	A, T, S
2	Production Cost	500 [USD]	Max	M	A
3	Length	20 [inches]	Max	M	A
4	Height	8 [inches]	Max	M	A
5	Width	10 [inches]	Max	M	A
6	Wind Resistance	70 [mph]	Max	H	A, T
7	Operation time from 2.9 hour charge	24 [hours]	Min	H	A, T
8	Max operating temperature	55 [°C]	Max	M	A, T
9	Min operating temperature	-20 [°C]	Min	M	A, T
10	Communication range	300 [feet]	Min	L	T
11	Mean time between service	2 [years]	Min	M	A
12	Weatherproof	IP67 [rating]	Min	M	T
13	Customers who like the product	90 [%]	Min	M	T
14	Few screws needed to remove for servicing	7 [screws]	Max	M	A
15	Installers who like the product	90 [%]	Min	M	T
16	Modular independent systems for simple replacement	5 [systems]	± 1	M	A

17	Components purchased as installed	100 [%]	± 20%	M	A

Design Development

- In order to develop a wireless thermal camera system the first step will be to research the requirements and outputs of the individual powered subsystems: camera core, battery, solar as well as wireless communications.
- To get a better understanding of how the system performs as well as gaining insight to the operation and power requirements of the camera core a working desktop model will be constructed to use a wired Ethernet connection to a standard router.
- Once a working knowledge of the camera core and operation has been developed integration with a wireless transmission scheme can be added allowing us to find the range and power dissipation of the combined components.
- With the total power dissipation of the camera and communication systems known an appropriate power storage method can be selected and used to power the assembly once charged from a conventional charging method such as a wall charger.
- The final step of the system integration will be to charge the battery from a solar array.
- Once the concept has been demonstrated and the components selected the packaging solutions can be explored to allow the complete system to be mounted in desirable locations for remote security applications.

Management Plan

The research and design work for the project is broken down between each member of the team. Here are the research categories and the member that are responsible for that area:

- Batteries – Jacob Richardson
- Solar Panel – Jacob Richardson
- Wireless Communication
    - ○ Leopardboard 355 Software Dev – Billy McVicker (Lead), Andy Bonk (Supporting)
    - ○ Wireless AP & Pleora Ethernet Module – Andy Bonk
- System Power Consumption – Andy Bonk
- Enclosure – Jacob Richardson

The team has decided to rotate alphabetically among the members to write the agenda for the weekly meetings and also take the minutes for that corresponding meeting. This includes updating the documentation of the project progress through a shared Google^{TM} document.

Andy Bonk, the manager of the Wi-FLIR team, is responsible for the prototype fabrications. He will manage how the design will go from the design phase to the rapid prototype phase. This will allow the team to demonstrate multiple designs to the sponsor, FLIR, at a lost cost.

Jacob Richardson is responsible for the manufacturing considerations of the project. He will need to research the components of the enclosure to take in consideration cost, manufacturability, and backup distributors. This phase should take in consideration the costs of assembly, future serviceability, and commercial manufacturing. Along with the enclosure, Jacob will research how the camera will be mounted to a pole and how the solar panel integrates with the camera.

Billy McVicker is responsible for the wireless communications research and interfacing with the camera. This will include deciding what type of antenna to use, the type of wireless band to operate across, and the mode of communication, such as through a router or access point on both ends of communication or to use a wireless card to communicate with the base station router. This will also involve testing the ranges of the device and the power consumption used for wireless operation.

Concept Selection

Solar

For selecting the appropriate type of Solar Panel we had two options: a solid substrate (eg: rigid) panel, or a flexible substrate that can form to shapes. Because the panel needs to hold up under high winds and be positionable by the user independent of the camera (both in rotation and tilt), the obvious choice was a solid substrate.

The other specification necessary for the solar panel was the size (wattage) of the panel. With our research on solar insolation and the estimated power consumption of our device, we used this simple formula to determine the number of watts required:

Figure 13. Industrial Solar IS-40J Panel

$$P_{Panel} = P_{Device} * \frac{24\ [hours]}{T_{insolation}}$$

Where:

P_{Panel} is the required solar panel wattage
P_{Device} is the wattage used by the device
$T_{insolation}$ is the average daily insolation (in hours) for December in Pullman, Washington

Using the average daily insolation of 2.9 hours, and an average power usage of the device of 3–5 Watts, gave a required solar panel size of 25–41 Watts. A 40 Watt Panel was selected for usage for the system, both because it is a common size and to err on the high side of our range. Researching 40 Watt solar panels, Table 8below was created of their properties to help us reach a final decision.

Table 3:Solar Panel Comparison

Product	BP BP340J[4]	Kyocera KC40T[5]	Industrial Solar IS-40J[6]
Rated Power (watts)	40 ± 10%	43 ± 15%	40 ± 10%
Nominal Voltage (V)	12	12	12
Size (L x W x T) (in)	25.8 x 21.1 x 1.97	25.7 x 20.7 x 1.42	25.8 x 21.1 x 1.97
I_{SC} (amp)	2.5	2.65	2.5
V_{OC} (V)	21.8	21.7	21.8
$I_{max\ power}$ (amp)	2.3	2.48	2.3
$V_{max\ power}$ (V)	17.3	17.4	17.3
Cost	$228.24[7]	$195.00	$176.00
Hole Pattern Symmetry	Datum	NO	YES

Using this table, we selected the Industrial Solar IS-40J as the appropriate panel for this project. It was the cheapest panel available, and it shares its size and hole pattern with the BP panel. This will allow the two to become interchangeable, if it is difficult to source the items from Industrial Solar in the future.

After our meeting with FLIR, we were asked to take the effects of dust and other particle accumulation into account for the solar system. Then we used a safety factor of 3 with the previously mentioned 5% decrease in power, to determine we should account for an additional 15% of power usage. This raised our panel size to about 46 watts. This is not a common size, so 50 watt panels were researched.

Table 4: Solar Panel Comparison for 50W Panels

Product	BP BP350J	Kyocera KC50T	Industrial Solar IS-50J
Rated Power (watts)	50 ± 10%	54± 10%	50 ± 10%
Nominal Voltage (V)	12	12	12
Size (L x W x T) (in)	33 x 21.1 x 1.97	25.2 x 25.7 x 1.42	33.0 x 21.2 x 1.98
I_{SC} (amp)	3.2	3.31	3.23
V_{OC} (V)	21.8	21.7	21.6
$I_{max\ power}$ (amp)	2.9	3.11	2.91
$V_{max\ power}$ (V)	17.5	17.4	17.2
Cost	$237.90	$246.00	$264.00

With this information we decided to go with the BP350J panel. However, upon ordering the panel from the vendor, we were informed that the BP panel was no longer produced. The vendor offered an alternative panel they had in stock and ready to ship. After reviewing the specs we deemed it to be sufficient for this prodject, and in the interests of expediting the project purchased it.

Table 5: SunWize SW-S55P Solar Panel Specs

Product	SW-S55P
Rated Power (watts)	55 ± 10%
Nominal Voltage (V)	12
Size (L x W x T) (in)	26.06 x 26.06 x 1.97
I_{SC} (amp)	3.3
V_{OC} (V)	22.0
$I_{max\ power}$ (amp)	3.15
$V_{max\ power}$ (V)	17.4
Cost	$270.

Battery

After completing a Pugh matrix on battery selection, two options remained for potential battery types: Lead Acid or Lithium Ion. With our power usage of 5 Watts, we found the required battery size (in Ah) using the following equation for a 12 V battery:

$$Ah = \frac{P_{device} * (24\ hours) * (5days)}{(12\ V)} = 50\ Ah$$

This size battery will be able to last for 5 days when new, without receiving any solar support, which will allow for any bad weather to be averaged, obtaining a 100% duty cycle for our system. Another benefit of designing to 5 days is, as the battery capacity decreases during the life cycle of the product, it will still be sufficient to keep the camera powered full time.

This size or similar of Lithium Ion batteries were found to be both scarce and expensive. There are very few companies producing batteries in this size, and many are not able to keep up with demand. The decision was use a Lead Acid battery for this project, as it will be the cheapest and most widely available, both for us and the consumer.

Research on lead acid batteries showed there are many batteries available in this size. Table 6has been created below to show the specifications of the various options and our final choice.

Table 6: Sealed Lead Acid Battery Comparison

Product	Universal UB12500[8]	BB Battery EB50-12[9]	Dynasty WB1250LD-FR[10]
Rated Power (Ah)	50	50	50
Nominal Voltage (V)	12	12	12
Size (L x W x T) (in)	7.72 x 6.50 x 7.17	7.76 x6.50 x 6.73	5.5 x 9 x 8.8
Weight (lb)	32	36.38	40.1
Cost	$79.00[11]	$165.00	$176.00

From this selection, the clear choice for use was the Universal Battery UB12500, as it was by far the cheapest, and the dimensions are among the smallest.

Enclosures

For the camera enclosure, we are opting to utilize the current FLIR F-Series housing (as seen in Figure 8). This will allow for product consistency, and a smaller number of unique items that need to be stocked. This housing already meets all the environmental considerations we needed to take into account.

The battery enclosure was important as it will not fit in the FLIR enclosure, and must be vented. After researching we found there are many choices for battery enclosures, but most were too large for our project. It was determined that an Allied Enclosures AM1648L[12] would be the best choice, as the size is correct to house our battery with a charge controller, with room for wiring but without being oversized. The size of this enclosure is 16 x 14 x 8 inches (L x W x H), which will give us plenty of room to mount the two devices. The battery will likely be mounted on the bottom of the enclosure, and the Charge controller to the top, as it is relatively thin compared to the overall height of the enclosure. The enclosure will cost us $88[13].

Mounting

The camera enclosure will also be mounted using FLIR's Wall Mount[14] with the pole mount adapter[15]. This mounting allows for varying pole sizes from 1.5 to 8 inches and, like the purchased enclosure, will allow for less unique items to be manufactured specific to this product.

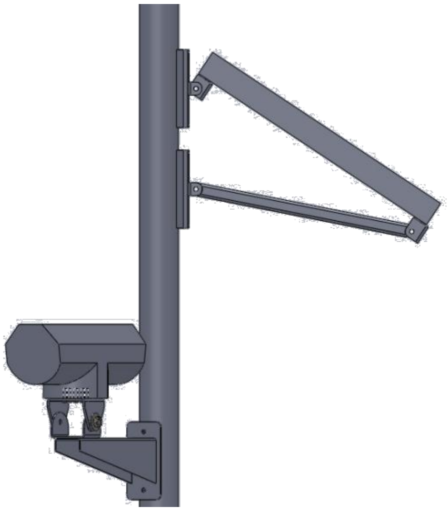


Figure 14: System Mounting on Pole

The Solar Panel will be mounted in the fashion of a 3-bar linkage. This linkage will allow the user to set the distance between the two pole mounts to get the angle desired on the panel. By being a separate mount from the enclosure, it will also allow for the panel to be positioned on the pole in the optimum direction for sunlight. A CAD model of the system can be seen below in Figure 14. The unique parts consist of 2 mounts to adapt to the pole bracket[16], 2 mounts on the solar panel, and the linkage arm[17,18].

The panel utilizes the same pivots top and bottom on both pole and frame sides. The Adapting plate to the pole is the same plate used to adapt the FLIR F-Series wall mount to a pole mount. Again, this was considered in a manner that will allow for the least number of unique parts for custom manufacturing.

The battery enclosure will be mounted by a pole mounting kit that is supplied by Allied Enclosures, the same company that manufactures the enclosure. This kit also uses steel band type mounts (like the FLIR pole mounts), so they will also be adjustable to many different pole sizes and types. The particular mount we are using is the AMPOLEMNT14[19], which will mount to up to a 15 inch pole, and will cost $59[20].

During our meeting with FLIR, some concern was express over the ability of the mount system to withstand side or uneven wind loads. It was decided to change the bottom mounting bar to a V-shape in order to stabilize side movement. It was also required to change the mounting brackets because the new solar panels had different mounting holes. A final design was reached and is shown below. Solidworks was used with this design to simulate the 3 different wind load types (even, uneven, and side), at 2 different angles (30 and 60 degrees from horizontal). The loads and deflections found in the simulations showed the new design to be sufficient in all these conditions. The results of the Simulations can be found in Appendix G.

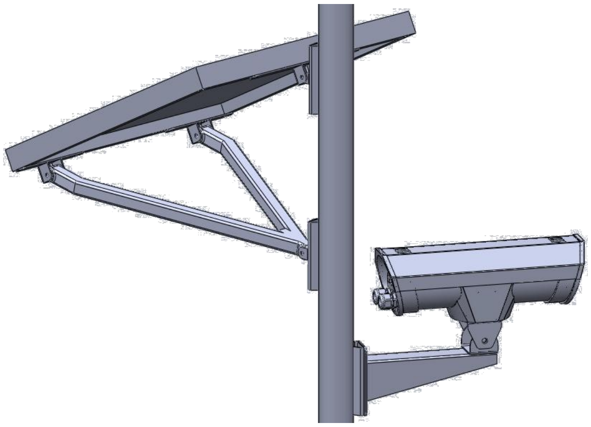


Figure 15: New V-Shaped Bottom Bar Design

Video Processing and Wireless Communication

For the wireless communication, there were two options to choose from. We could either use a wireless access point interfaced with the Pleora Ethernet Module or a microcontroller interfaced with a wireless module. Both of these methods would satisfy the communication requirements as far as minimum wireless range (300 feet) and capable of transmitting video. The following comparison and contrast exercise was done to help decided on which implementation should be taken.

Table 7: Video Processing and Wireless Communications Configurations

Product	TEW-654TR (Wireless AP) with Pleora iPort PT1000-CL IP Engine	TM320DM355 (Leopardboard 355) with OWL221a
Power Consumption	5 watts (2.5W for Wireless AP and 2.5W for IP Engine)	2.232W (2W for Leopardboard and 232 mW for OWL222a)
Coverage	200 meters (650 ft)	250 meters (with external antenna)
Temperature	0 to 70 °C (OEM)	0 to 85 °C , -40 °C to 100 °C
Cost	$2,026.99	$144.00

After analyzing this table, it became obvious using the DM355 microcontroller would save money and power consumption for our product; however, the time to market is extremely long due to custom design of the software. Since using the wireless access point and Pleora Ethernet Module can be easily implemented (these products were either provided by FLIR or currently owned), we concluded to temporarily implement the wireless communication using a wireless access point and the Pleora Ethernet Module while developing the software for the DM355 chip. This will allow for the project to continue making progress without the most energy-efficient solution. Ideally, if the software development for the DM355 chip is successful and the debugging process is not delayed, then we will swap out the temporary system with the microcontroller system. Refer to Appendix J:Tau-Leopardboard-OWL222a Pin Connection Diagramfor the wireless communications module block diagram using the Leopardboard.

Charge Controller

In order to create an initial prototype to demonstrate proof of concept an off the shelf charge controller will be purchased and integrated. While the development of a custom microcontroller board takes place the functionality of the charge controller will be implemented. This two-fold approach will allow progress to be made with testing and developing the initial prototype while working toward a physically smaller system with minimal total power consumption. This integration with the main board additionally will allow remote polling of the battery status.

Table 8: Charge Controller ICs

Product	TI: BQ24156[21]	TI: BQ24210[22]	TI: BQ24650[23]
Accepts VariableSolar Power	Yes	Yes	Yes
Battery Chemistry	Li-Ion/Li-Po	Li-Ion	Li-Ion/Po, $LiFePO_4$, Lead Acid
Chip Package	20DSBGA (SMC)	10WSON (SMC)	16VQFN (SMC)

Temperature Range	-40 to +85°C	-40 to +85°C	-40 to +85°C
Price	AVNET: 1 pc: $6.64 25 pcs: $4.65 50 pcs: $4.15 100+ pcs: $3.87 Digi-Key: 1 pc: $6.00 250 pcs: $4.255	(not yet released)	AVNET: 1 pc: $9.52 25 pcs: $6.67 50 pcs: $5.95 100+ pcs: $5.56 Digi-Key: 1 pc: $8.75 250 pcs: 5.61

Because we will be using a SLA battery for the power storage we will be using the Texas Instruments BQ24650 chip to integrate with the low power board. The BQ24156 and BQ24210 chips could be used for the system if it is changed to use and alternate battery chemistry.

For the purposes of creating a functional base prototype while a board is developed we will use an off the shelf solar charge controller to interface between the components.

Table 9: Complete Charge Controller Systems

Product	batteryspace.com: SC-10A[24]	batteryspace.com: SC-LI11V[25]	Chicago: 96728[26]
Watts	120	120	100
Battery Chemistry	SLA/LFP	Li-Ion	Unspecified

For this initial off the shelf charge controller we will use a SC-10A from batteryspace.com which has the ability to regulate the charge of the SLA battery we will use.

Product Realization

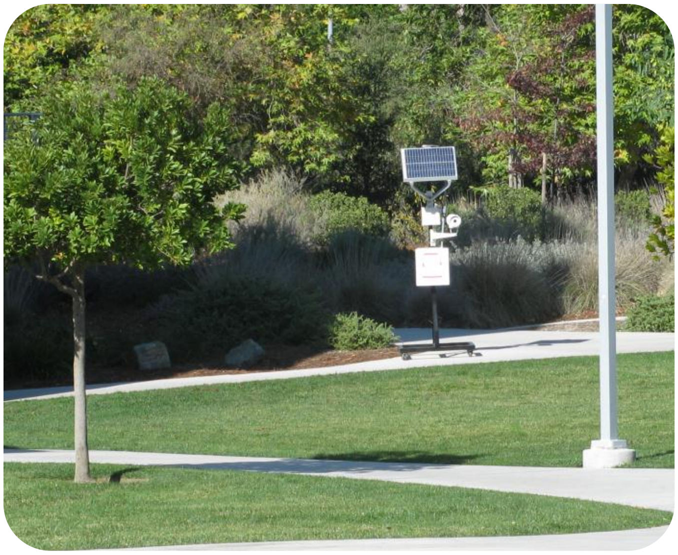


Figure 16: Wi-FLIR Displayed During Senior Project Design Expo

Video Processing and Wireless Communication

The Leopardboard is an Open Source hardware project developed by Leopard Imaging as a low power, low cost, and high performance development system. It uses TI's TMS320DM355 microprocessor, which is powered by an ARM core and a video processing subsystem known as the DaVinci processor. This development board is targeted for portable video applications. The goal of the video processing subsystem is to replace the Pleora Ethernet Module in order to save2.76watts of power. By reducing the video processing power of the system, we could extend the operating time of our battery-powered video surveillance system.

The Leopardboard was chosen because of its targeted application towards portable video applications and its advantage of interfacing with a low-powered wireless communication module. This would allow for our system to save power not only in the video processing subsystem but also in the wireless communication subsystem by eliminating the use of a wireless access point to transmit the video across a wireless network to the client machine.

The results of the design process yielded a subsystem using the Leopardboard355 development board and the OWL222a wireless module. With regards to video processing, this subsystem was designed to interface with the Tau320 camera core via the communication protocol RS-232 and the Legacy Digital Data Channel. The Leopardboard supports the RidgeRun Evaluation Software Development Kit (SDK), which consists of a Linux-based real-time operating system (RTOS) along with drivers for SPI, I^2C, UART (a type of RS-232), GPIO, and SDIO communication protocols. The Leopardboard can interface with the Tau 320 camera core using the RS-232 communication protocol and the Legacy LVDS digital channel or the BT-656 digital interface. The RidgeRun SDK does not support the LVDS protocol and therefore must be implemented using the GPIO pins; however, the SDK does support the YCbCr 422 format 8- and 16-bit resolution in its video processing sub-system (VPSS), which can be used as a BT-656 digital interface by the Tau 320 camera core. As for wireless communication, the Leopardboard can communicate with the OWL222a wireless module via the SDIO communication protocol, which is rated for communication speeds up to 50 MHz.

The following development process was created:

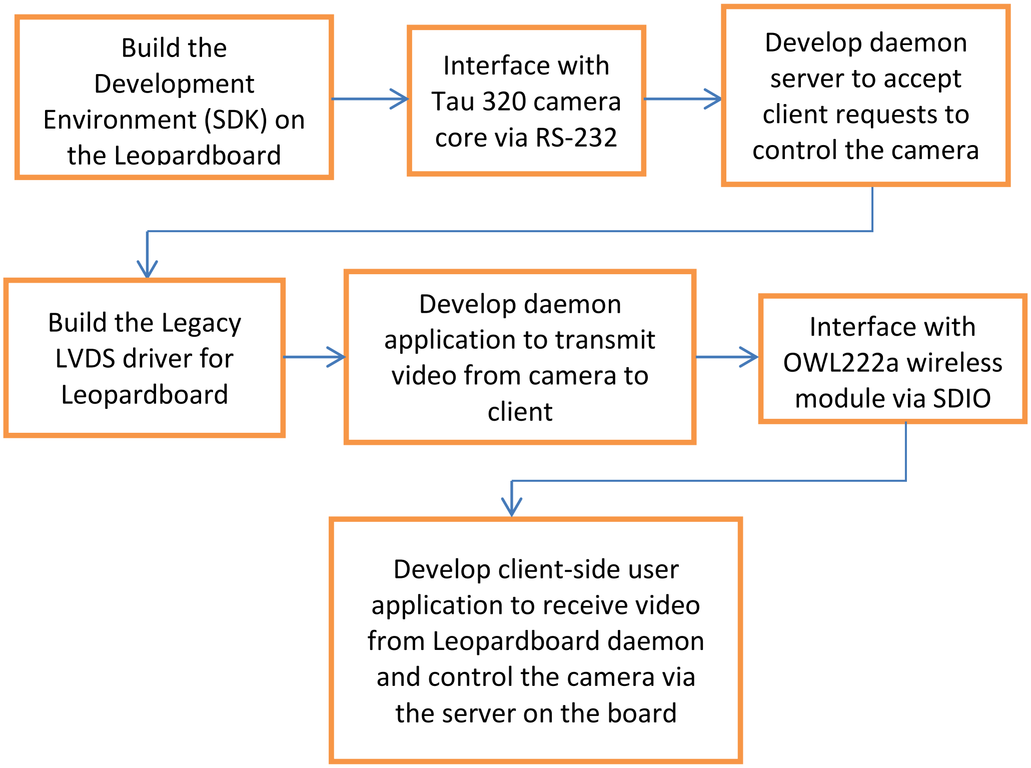

Build the Development Environment (SDK) on the Leopardboard
Interface with Tau 320 camera core via RS-232
Develop daemon server to accept client requests to control the camera
Build the Legacy LVDS driver for Leopardboard
Develop daemon application to transmit video from camera to client
Interface with OWL222a wireless module via SDIO
Develop client-side user application to receive video from Leopardboard daemon and control the camera via the server on the board

Figure 17: Leopardboard Development Process

Building the Development Environment

The development environment is provided by RidgeRun and is an evaluation SDK. It consists of a linux-based RTOS with a suite of communication drivers: SPI, I^2C, UART, GPIO, and SDIO. The process of loading the RTOS onto the Leopardboard consists of: uploading the bootloader, compiling the kernel with the desired drivers and file system, and uploading the compiled binaries. This process proved to be nontrivial due to the lack of support from RidgeRun. Issues encountered consisted of: uploading the bootloader, and transferring the binaries using a TFTP server.

To run any RTOS on a microcontroller a bootloader is needed. RidgeRun provides a bootloader with its SDK called Uboot. The necessary tools to upload the bootloader are provided by, but little documentation is provided on how to do it. If this process is interrupted, the Leopardboard becomes non-responsive to all communication due to a corruption of the NAND flash. (This state is called *bricked*). In many attempts to upload the bootloader to the Leopardboard, the process was interrupted due to the TFTP server's configuration, which is used to transfer the binaries across a Local Area Network (LAN), causing the board to become bricked. This state was not documented by RidgeRun and required me to search for a solution from other resources. After extensive browsing through online discussion boards, I found the paper *Bring up a Bricked Leopard Board using the RidgeRun's SDK, MonoDevelop and DM35x FlashAndBootUtils 1.00* by Yoksan Varela, et al. This paper outlines how to recover a bricked Leopardboard and was used to recover the board several times. After discovering the interruption was caused by an invalid TFTP server configuration, the bootloader was then uploaded using the serial UART protocol, which was not initially used because of its significantly lower transfer rate and RidgeRun's driver having transfer issues discovered when trying to send files over 100 kilobytes.

When attempting to transfer the binaries to the Leopardboard, the TFTP configuration issues re-occurred. The binary files were larger than 100 kilobytes and failed to transfer using the UART protocol. This forced me to investigate the TFTP configuration issues. Documentation of setting up a TFTP server in regards to the RidgeRun SDK was found on a DesignSomething.org forum posted by a RidgeRun software engineer. The steps provided did not solve the problem and further investigation was needed. Other steps provided by other resources, but proved insufficient. The problem lied in identifying which folder the TFTP server would look in to find the desired files to transfer. By default, the TFTP server uses the folder "/var/lib/tftpboot," but the RidgeRun SDK saves the binaries in the folder "/srv/tftp." This requires the developer to modify the TFTP server's configuration to search in the "/srv/tftp" folder instead of its default folder. It is documented across the internet that to make this change one must modify the "server_args" variable found in the file "/etc/xinetd.d/tftp." This did not work under Ubuntu 10.10, which I was developing under. I could not find a way to change the folder that the RidgeRun SDK saved the binaries to. So to solve this problem, I created the folder "/srv/tftp" as a symbolic link to the folder "/var/lib/tftpboot." This resolved the communication problems with regards to uploading files to the Leopardboard. The steps to follow in order to properly setup the TFTP server to work with the RidgeRun SDK are:

1. Install the following packages as follows:
   ```
   $ sudo apt-get install tftp tftpd xinetd
   ```
2. Create the TFTP directory used by the SDK as a symbolic link to the default TFTP directory
   ```
   $ sudo ln –s /var/lib/tftpboot /srv/tftp
   $ sudo chmod –R 777 /srv/tftp
   $ sudo chown –R nobody /srv/tftp
   ```
3. Configure the TFTP server by creating the file "/etc/xinetd.d/tftp" with the content:
   ```
   service tftp
   {
           protocol        = udp
           port            = 69
           socket_type     = dgram
           wait            = yes
           user            = nobody
           server          = /usr/sbin/in.tftpd
           server_args     = /srv/tftp
           disable         = no
   }
   ```
4. Lastly, restart the TFTP server:
   ```
   $ sudo /etc/init.d/xinetd restart
   ```

Once the TFTP server is configured, the developer needs to configure the RTOS and then compile the kernel and file system with the following commands: "make" then "make install."

Interface with the Tau 320 Camera Core

The camera core can be controlled by an external device using the RS-232 protocol with a 3.3V voltage swing from 0V to 3.3V. The RidgeRun SDK includes a UART driver, which is the same as an RS-232 driver with only an RX wire and a TX wire. (The standard RS-232 protocol supports flow control with additional wires and a 12V voltage swing). Table 10: Serial Port Settings includes the necessary port for communicating with the camera (this table can be found in the Tau 320 User Guide):

Table 10: Serial Port Settings

Parameter	Value
Baud rate:	57600
Data bits:	8
Parity:	None
Stop bits:	1
Flow control:	None

The serial packet protocol consists of a header describing the type of packet with the data appended to the end of the packet. It also contains a CCITT-16 checksum. The specifics of the packet and status bytes which define the type of packet can be found in the Tau 320 User Guide distributed by FLIR.

When developing the code to interface with the camera, I ran into communication problems. The generated packet was transmitted successfully and verified using an oscilloscope, but the camera would not respond with the appropriate packet. I debugged the packet by transmitting the example packet as described in the user guide. This eliminated the possibility of generating an invalid packet and verified I correctly implemented a checksum algorithm. To narrow down the possibilities of what the problem could be, I transmitted the SHUTTER_POSITION command and toggled between the open and close data argument. As indicated by the name of the command, this command tells the camera to open or close its shutter depending on the value of the packets data segment. By sending this command, I could audibly verify the camera received a valid packet. When sending this packet, the camera did not respond, which indicated the packet either did not transmit across the wire successfully or the packet was generated incorrectly. After contacting Nile Fairfield, a FLIR Systems Engineer, he informed me that the camera requires inverted logic, which is not documented in the camera's user guide. By feeding the RX and TX lines through a NAND gate, the signal was inverted and the camera began to respond. The circuit diagram in Figure 18: Serial Communication Circuit Diagram was needed to successfully communicate with the camera core.

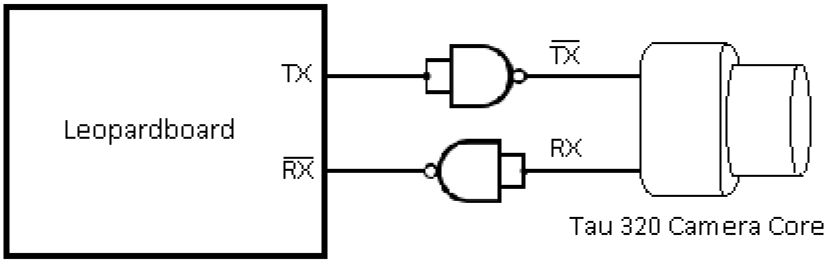


Figure 18: Serial Communication Circuit Diagram

The code for communicating with the camera can be found in Appendix L: Serial Camera Communication Code .

Develop the Legacy LVDS Driver

There are three different ways one can receive digital video from the Tau 320 camera core: the Legacy LVDS digital data channels, the BT-656 digital interface, and the CMOS digital interface. Using the LVDS digital data channels is the recommended method of retrieving digital data from the camera core. The interfacing port we received with the camera core called the Photon Replicator Kit is designed to transmit digital video using the LVDS channels, which is why we chose this protocol to retrieve digital video from the camera core. The problem with using the LVDS channels is that the RidgeRun SDK does support it, which means we needed to create our own driver to add support. Low-voltage differential signaling (LVDS) uses three wires: a clock, a composite sync (frame sync and data valid), and a serial data line. By using low-voltage differential signals (LVDS), the clock rate is set to 73.636 MHz. The timing diagram can be found in Figure 19: LVDS Timing Diagram.

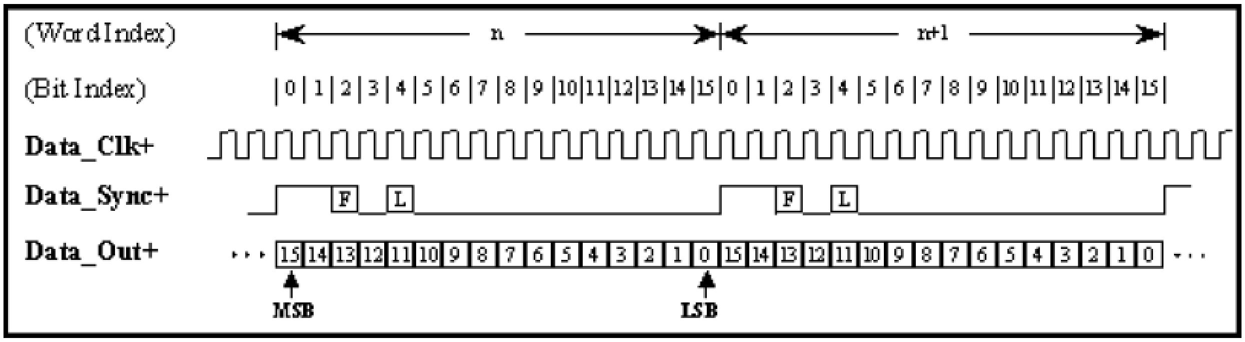


Figure 19: LVDS Timing Diagram

This high clock rate was a key factor in choosing what microcontroller to use. On first inspection, the TMS320DM355 microprocessor found in the Leopardboard is reported to have an internal oscillator operating at 216 MHz. When the time came to generate this clock, I discovered the microprocessor only provides a reference clock of 24 MHz. The internal oscillator cannot be directly divided down to a 73.636 MHz signal. This led to a redesign of our project's video processing unit. As indicated in section *3.5.3 Supported Clocking Configurations for DM355-216* of the TMS320DM355 datasheet, the microprocessor supports either a 24 MHz or 36 MHz reference clock. Further analysis showed the GPIO pins clocked by the SYSCLK2 signal, which is fed the 24 MHz reference clock. (Refer to Figure 3-2 Device Clocking Block Diagram in the TMS320DM355 datasheet for details on how peripheral clocking is handled.)

Due to this unfortunate mishap, our project's video processing unit needed to be redesigned. The options we had to choose from were either to retrieve the digital video via another interface – either the BT-656 or CMOS interface – or resort to using the Pleora Ethernet Module. After researching the possibility of using the BT-656 protocol, we discovered the Leopardboard does not utilize all the pins of its Video Processing Sub-System (VPSS) limiting the video resolution to 6-bits. This limitation reduces the video quality of the entire surveillance system. The final decision was to not compromise the video quality for better power consumption because this would reduce the accuracy of the systems surveillance abilities; therefore, using the Pleora Ether Module coupled with a wireless access point. The final realization of the Leopardboard was reduced to a networked battery voltage monitoring device.

Battery Voltage Monitoring System

One of the requirements for the project was to implement a way of monitoring the battery voltage level from the client machine. This would allow the owner of the camera to easily monitor the battery voltage without having to physically checking the voltage with a multimeter. Since the video is transmitted via a wireless access point, the voltage monitoring device could use the same wireless network to transmit the voltage data. The Leopardboard355 development board was used to interface with an analog-to-digital (A/D) converter and record the voltage level to a log file. A server/client application would then be used to transmit the log file across the network to the client machine. The software flow diagram in

Figure 21: Battery Monitoring Software Flow Chart and block diagram in Figure 20: A/D Interfacing Circuit Diagram was created during the design of this voltage monitoring system.

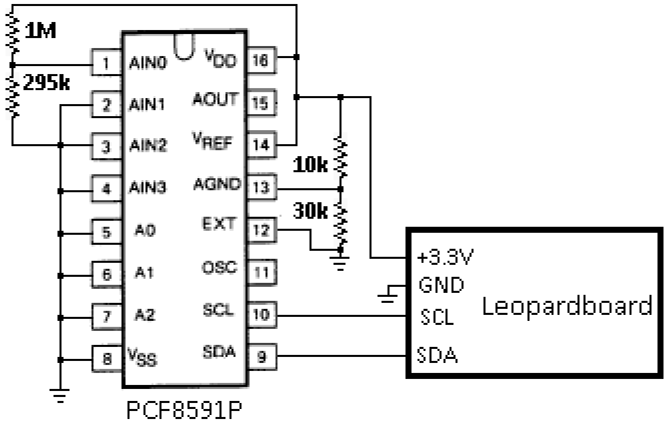
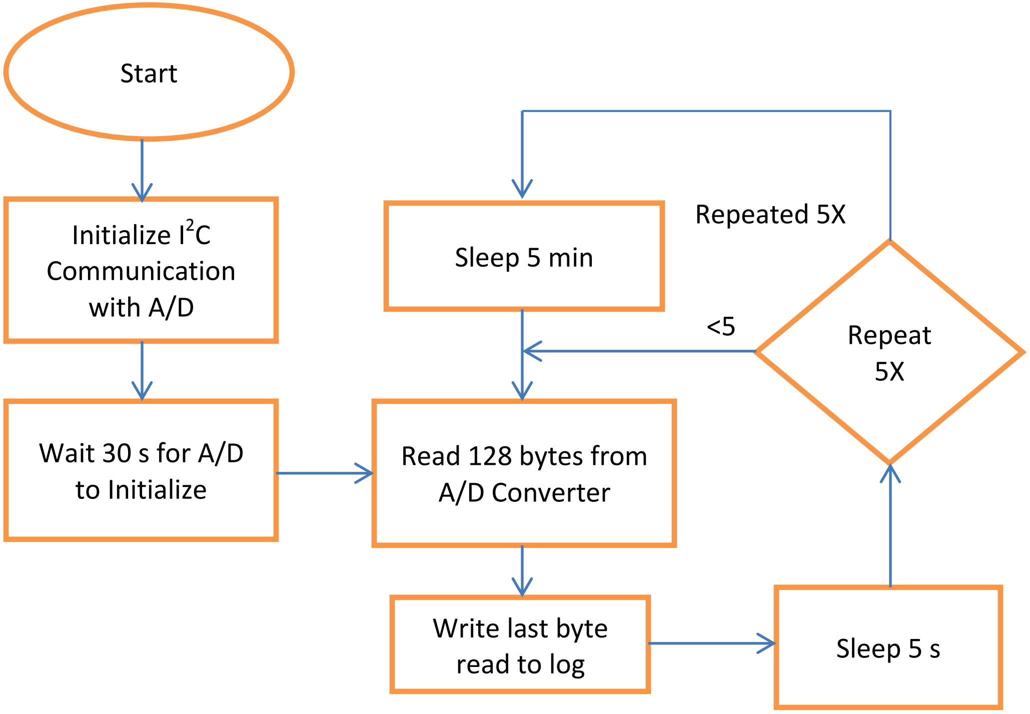


Figure 20: A/D Interfacing Circuit Diagram



Start
Initialize I2C Communication with A/D
Wait 30 s for A/D to Initialize
Sleep 5 min
Repeated 5X
<5
Repeat 5X
Read 128 bytes from A/D Converter
Write last byte read to log
Sleep 5 s

Figure 21: Battery Monitoring Software Flow Chart

The PCF8591P A/D converter was chosen to interface with the Leopardboard in order to retrieve a digital reading of the battery voltage. The A/D converter has an 8-bit resolution and uses the I^{2}C serial protocol to communicate with other devices. The RidgeRun SDK supports this protocol. We needed to divide the voltage down to match the I^{2}C signal levels of 3.3V because the maximum A_{IN} voltage is VDD and VDD's maximum voltage is 6V. To calculate the voltage divider resistor values we set the maximum battery voltage to 15V and chose a value of 1MΩ for the top resistor in the divider.

$$A_{in} = V_{battery} * \frac{R_2}{R_1 + R_2}$$

$Set:$

$$R_1 = 1\ M\Omega$$
$$A_{in} = I^2C\ Voltage\ Levels = 3.3V$$
$$V_{battery} = Theoretical\ Max\ Battery\ Voltage = 15V$$

$$R_2 = \frac{A_{in} * R1}{V_{battery} - A_{in}}$$
$$R_2 = \frac{3.3V * 1M\Omega}{15V - 3.3V} = 295k\Omega$$

Since the battery voltage does not drop below ~11.3V, we needed to raise the analog ground in order to utilize the full 8-bit sampling resolution. To calculate the maximum voltage we can use as the analog ground and still cover the full battery voltage swing, we plugged in the safest and lowest possible battery voltage into the above voltage divider equation with the appropriate resistor values and solved for A_{in}.

$$A_{in} = V_{battery} * \frac{R_2}{R_1 + R_2}$$
$$Set:$$
$$R_1 = 1\ M\Omega$$
$$R_2 = 295k\Omega$$
$$V_{battery} = Lowest\ Possible\ Safe\ Battery\ Voltage = 11.3V$$

$$A_{in} = 11.3 * \frac{295k\Omega}{1M\Omega + 295k\Omega} = 2.57V$$

Now using this voltage and the same method as above, we can calculate the necessary voltage divider to raise the analog ground to 2.57V.

$$A_{in} = V_{battery} * \frac{R_2}{R_1 + R_2}$$
$$Set:$$
$$R_1 = 10\ k\Omega$$
$$A_{in} = I^2C\ Voltage\ Levels = 2.57V$$
$$V_{I^2C} = I^2C\ Voltage\ Level = 3.3V$$

$$R_2 = \frac{A_{in} * R1}{V_{I^2C} - A_{in}}$$
$$R_2 = \frac{2.57V * 10k\Omega}{3.3V - 2.57V} = 35.2k\Omega$$

To compensate for the theoretical maximum and minimum voltage levels the following R_1 and R_2 resistor values were used: 10kΩ and 30kΩ respectively. This resulted in an analog voltage level of 2.45V. With this configuration, we can achieve a resolution of $(3.3V – 2.45V)/2^8$ = 3.33mV. Once the digital measurements are retrieved, they need to be converted to a value that represents the analog battery voltage. To generate the conversion equation, a variable power supply was used to sweep the voltage range while measuring the corresponding converted digital value. The data collected was graphed and a trend line was computed which is used as conversion equation. Table 11: A/D Calibration Measurements includes the measurements taken and Figure 22:A/D Conversion Calibration represents the graph of the measurements with a trend line.

Table 11: A/D Calibration Measurements

Analog Voltage (V)	A/D Conversion
14.59	250
13.98	205
13.00	130
12.42	85
11.69	29
11.43	9

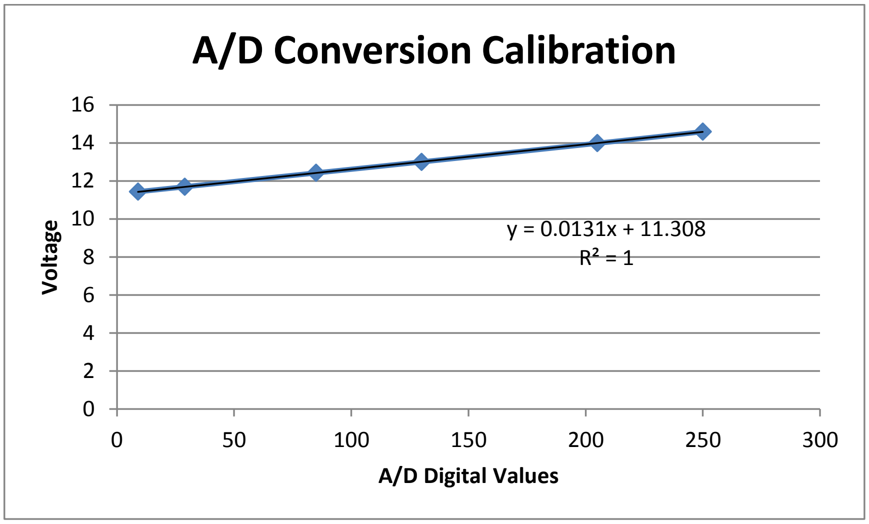


Figure 22:A/D Conversion Calibration

To log the voltage measurements from the A/D, a daemon application was developed to interface with the A/D and record the values every 5 minutes. Assuming the power system is working properly, the voltage should not drop a significant amount to loose relevant data. One difficultly in requesting the data from the A/D converter was reading enough data to get the latest reading. Once the A/D converter is configured to read values in it begins to store the digital values in its internal registers. These registers were found to persist until read out using the I^2C data line. To handle this issue, when requesting for data every five minutes, our application read in 128 values, appended the 128th value as the most current voltage reading to the log file, slept for five seconds, and then repeated five times before sleeping for five minutes. A time stamp is appended with the voltage value in the log file to indicate when the value was taken. Refer to Figure 21: Battery Monitoring Software Flow Chart for a flow diagram of this measurement retrieval process. The code for the A/D daemon application can be found in Appendix M: A/D Daemon Application Code.

Lastly, a server/client application was developed to transfer the log file from the Leopardboard to a client application for the user to view. The server/client application was developed by Billy McVicker in an Intro to Networks course at Cal Poly, San Luis Obispo. It was easily ported to the Leopardboard by compiling it with the ARM cross-compiler included in the RidgeRun SDK. This server/client application uses the UDP communication protocol with a custom Stop-and-Wait error handling algorithm to ensure successful transmission of data packets. On the client-side, a Windows application was developed to request for the log file from the server on the Leopardboard. Once the log file is retrieved, the latest voltage reading is displayed on the screen. This application was developed in C++ using Visual Studio 2010. The networking code used the client code from the server/client application. The software flow diagram can be found in Figure 23: Client-Side Battery Monitoring GUI. Appendix N: Server Daemon Code and Appendix O: Battery Monitoring Client Application Code contain the server code and client-side code respectively.

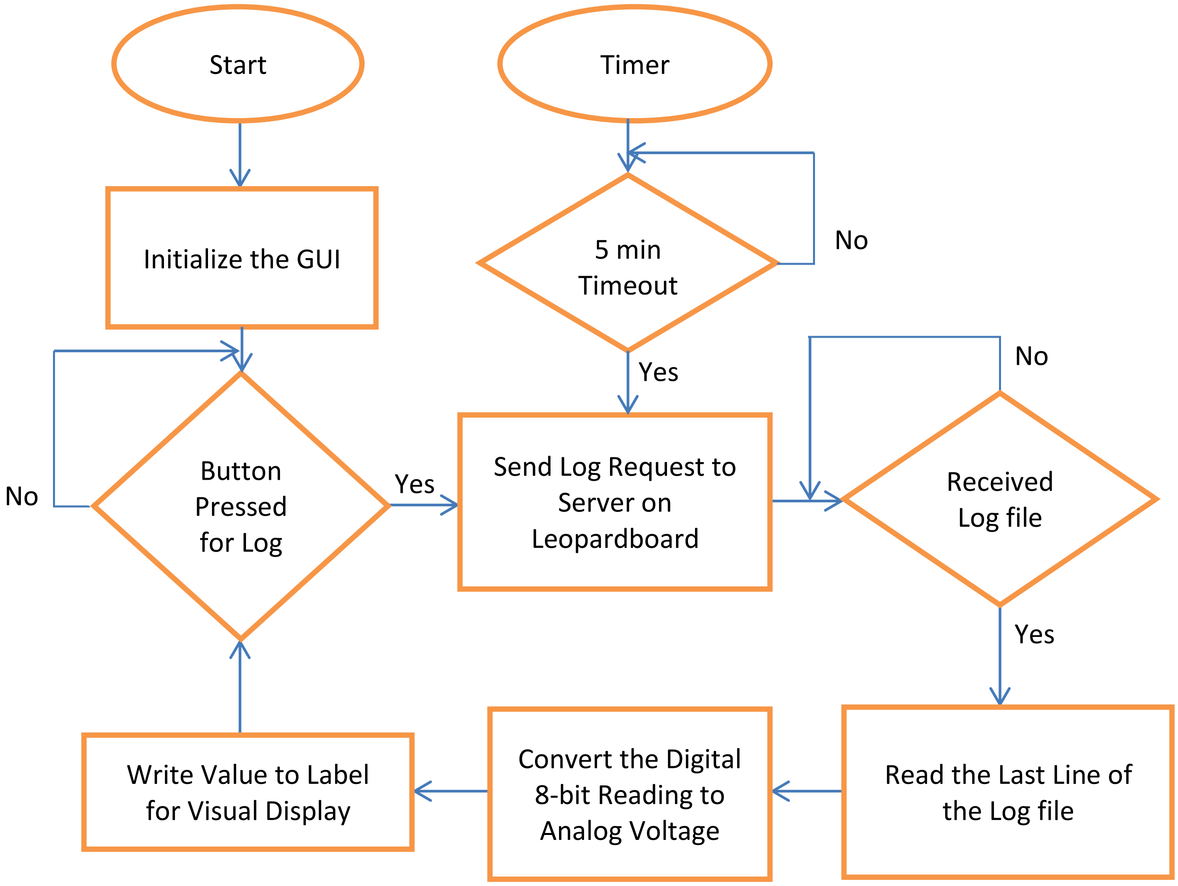


Figure 23: Client-Side Battery Monitoring GUI

Due to the unexpected problems that occurred when implementing the Leopardboard as the video processing unit, time to develop the battery monitoring system was shortened. Testing this application consisted of transferring several log files across the wireless network. Through testing, the application was further modified to periodically request for a new log file every five minutes as opposed to only requesting a new log file upon manual user request via a button on the GUI. Further testing would include: verifying the Leopardboard recovers from a network failure, verify the daemon server does not exit after failing to communicate with the A/D converter. Instead of exiting, further development should require the daemon to report the error in an error message log and then start a new instance of the daemon.

Charge Controller

The initial design specified that the Texas Instruments BQ24650 the design demonstrated used the initial stand-alone SC-10A charge controller from batteryspace.com. After ordering the evaluation module for the TI chip we discovered that the maximum charge voltage is specified by the selection of two resistors in the circuit. Because the board used fine pitch surface mount components we could not switch out these resistors. With the evaluation board limited to a maximum charge of 12 [V] the maximum charge achieved would be limited to only 20% of capacity. In a production environment the BQ24650 chip could be implemented in an appropriately designed charge circuit allowing the full battery charge rather than applying such a low limit. With this circuit board implementation the charge status monitoring Analog to Digital chip and a low power Atmel chip could be put in place to combine a full charge control/regulation/monitoring suite in one package.

Solar Mount Manufacturing

The mounting for the solar panel was all cut using a CNC plasma cutter. The plasma cutter uses AutoCAD .dxf file formats to cut patterns out of flat steel. Plasma cutting uses superheated high pressure air to cut the steel. It was advantageous to use a plasma cutting process because it was available and cheap; however, it would provide a better, more accurate cut to use a laser or water-jet cutter, or a high-definition plasma cutter. These all provide much lower kerf and blowout amounts.

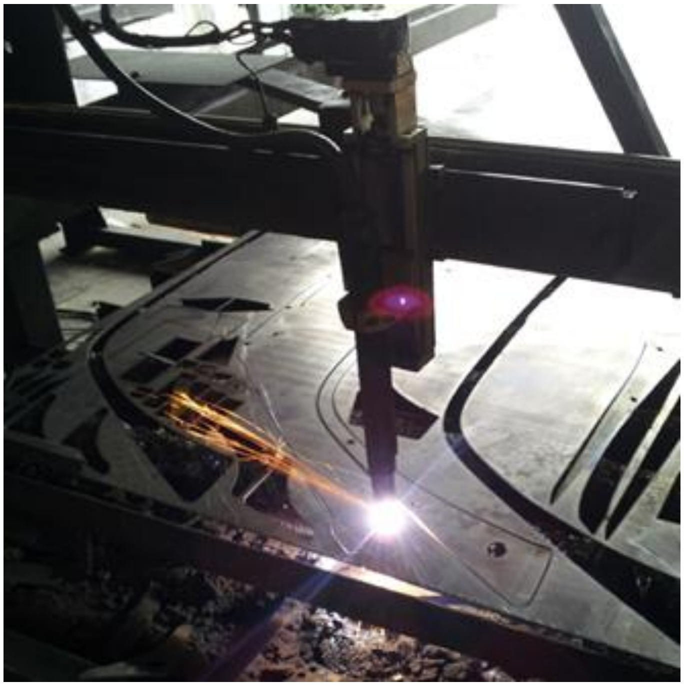


Figure 24: Plasma Cutting Unformed Brackets

The freshly cut brackets were cleaned of any slag, and the holes were deburred. They were then taken to the press brake. The press brake used was a 60 ton hydraulic commonly found in steel fabrication companies. We used a 1 inch bottom jaw and a window upper tooling with a 1/8" tool radius. The window tooling allows the tight channel to be formed by letting it bend inside the tooling.

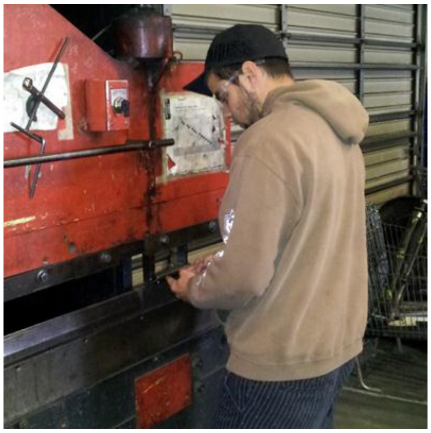


Figure 25: Press Brake Used to Form Brackets

The panel side brackets were then welded utilizing a wire feed MIG welder. MIG was used because it is a much easier process for an amateur welder to utilize. It allows the welder to focus solely on the weld and not on feeding the filler by hand. After all welding was finished, the brackets were put into a rock tumbler for 20 minutes, to clean them and leave a nice finish.

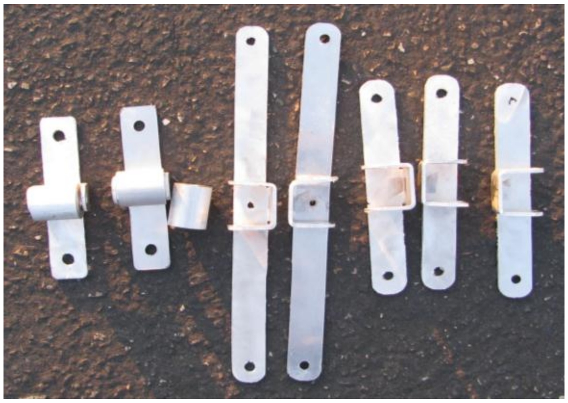


Figure 26: Finished Brackets

The pivot tubes for the lower V-bar were cut to rough length using an automated horizontal band saw out of aluminum solid stock, before being machined on a hand operated lathe. The machining consisted of facing the parts to length and drilling a 3/8" hole for the bolts.

The square tubing was cut to length using a chop saw. It was then marked and bent using a Jancy hydraulic tube bender with a square tooling. After bending the ends were angle cut with the chop saw, and coped to the pivot tubes using a tubing notcher.

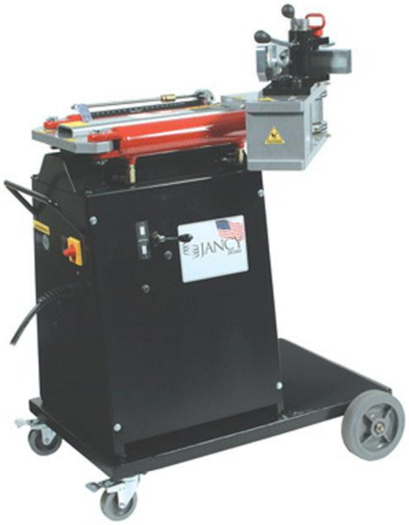


Figure 27: Hydraulic Ram Tube Bender Used

The V-bar was then also constructed using a wire feed MIG process. A jig was created on the work bench to keep the components squared with each other before it was tack welded together. After all welds were laid, the part was cleaned using a wire brush, and polished using an emery cloth.

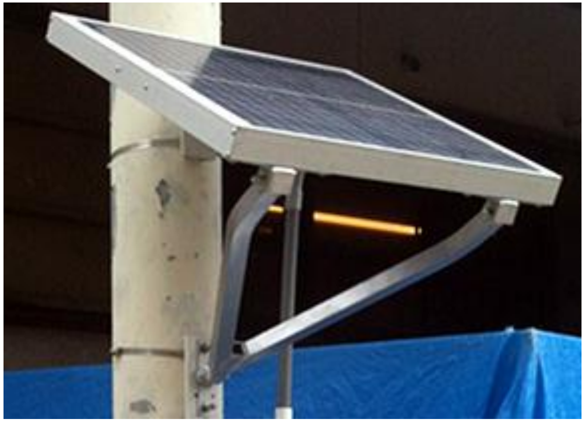


Figure 28: Finished Solar Mounting

All parts were painted using an aluminum metallic paint. This was done to help prevent corrosion and give a more industrial look to the parts. Ideally the parts would be plated or powder coated in order to provide a more long lasting resistance.

Conduit and Storage Box

In order to protect the external cables leading from the battery box to the solar panel, directional antenna, and camera housing 1 [in] diameter ridged PVC conduit was installed. Water tight fittings were used to secure the conduit directly to the battery box and to the solar panel by using a step-down fitting in conjunction with two 90 degree large radius bends. A tee fitting was used in the middle section of the vertical run of conduit to allow cabling to the directional antenna as well as the camera. We attached an additional large radius 90 degree bend fitting to this tee and pointed the end downward to prevent any possible leaked or condensed water from flowing into the battery box. At the end of this fitting we installed an end cap with a hole drilled in it to allow cables to pass through; this drilled hole was subsequently sealed with putty. Though the interconnections were not sealed with PVC pipe cement until it had been mounted onto the mobile stand there were no traces of water or other penetration of the battery housing even following heavy rains.

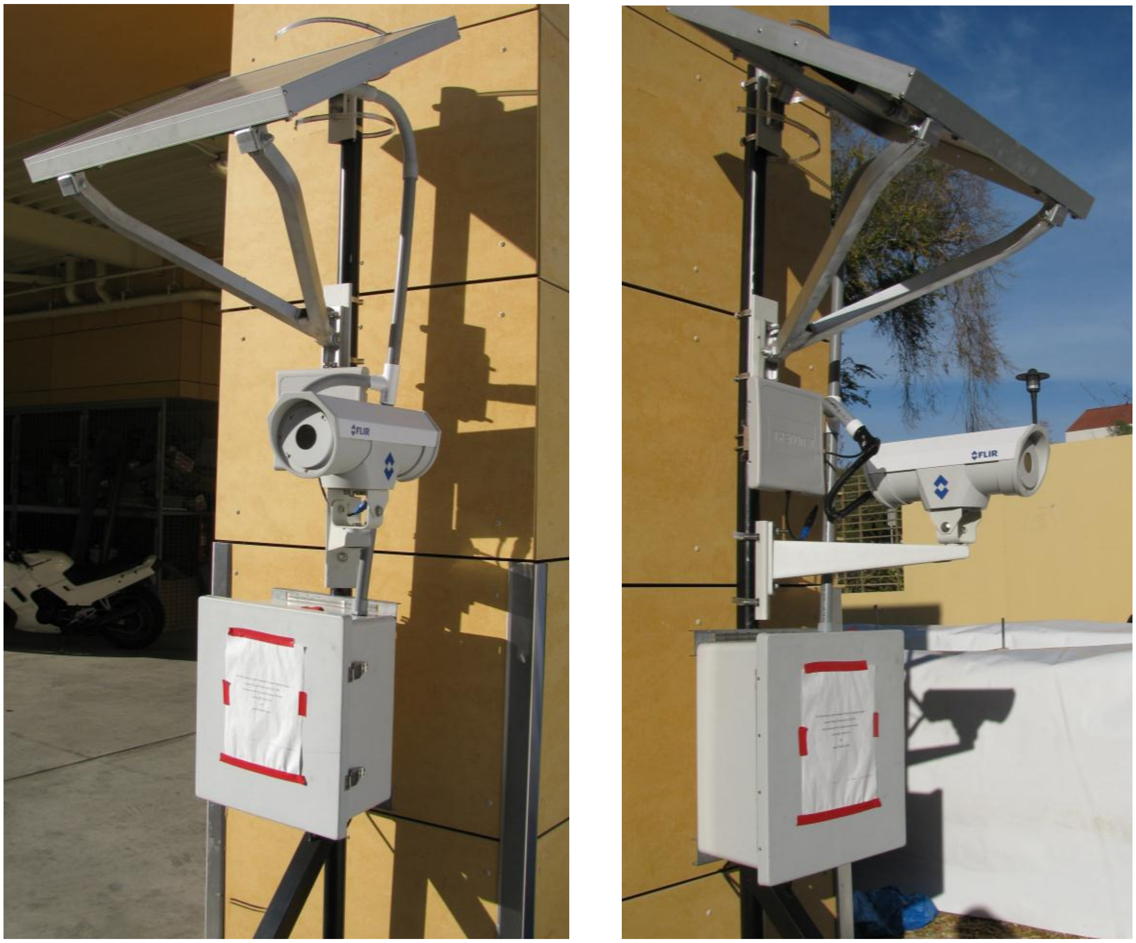


Figure 29: Left and Right View of Mounted System

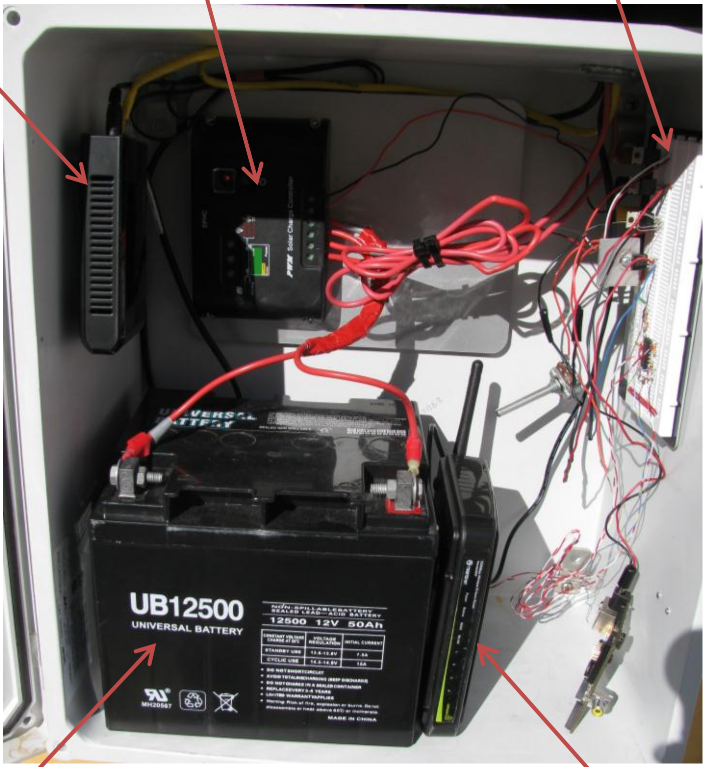

Wireless Access Point (TEW-638APB)
Charge Controller (SC-10A)
Breadboard for Voltage Regulators, A/D circuit and Leopardboard connection.
UB12500
UNIVERSAL BATTERY
Sealed Lead Acid Battery (UB12500)
Temporary Router to Connect to Leopardboard (TEW-652BRP)

Figure 30: Interior of Battery Box

Design Verification

Solar Panel Power Output

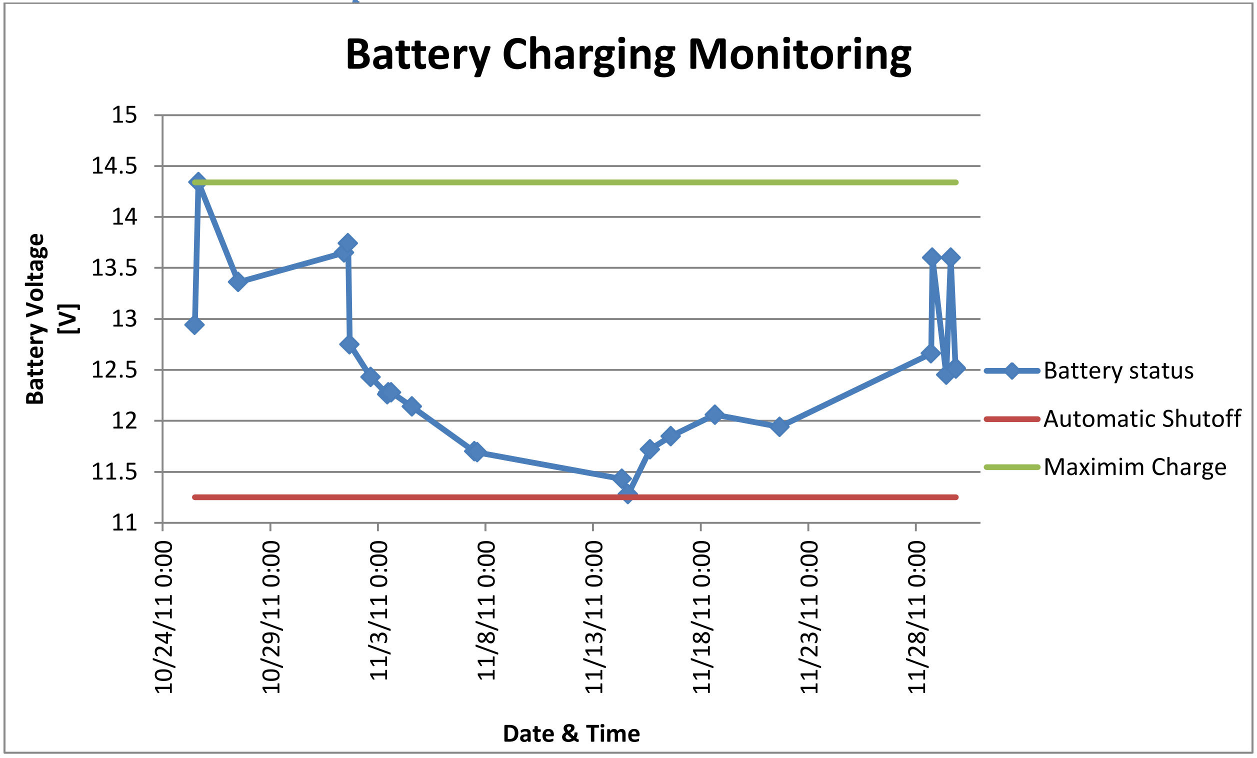


Figure 31: Battery status measured periodically with the system mounted at Cal Poly

The system was in a position such that there was a large shadow cast on the panel limiting the direct sunlight to a few hours a day starting in late October. On November 22nd the system was placed on the mobile demonstration platform and positioned such that it received more direct sunlight, this change can be seen with the increased average voltage of after the repositioning.

System Power Usage

Wireless Access Point

When the system was first mounted on the balcony of Building 13 at Cal Poly the Wireless Access Point was the only load applied due to the camera being used for the continuing development of the Leopard Board. The system was positioned with the solar panel facing to the south to maximize on the average sunlight throughout the day. The panel was angled to near 40° from horizontal as an initial setup. With this configuration, starting in late October there was a large shadow cast from the nearby Eucalyptus trees which only afforded the solar panel a few hours of sunlight throughout the day.

With this mounting and positioning setup the access point load was turned on and the partial system allowed to charge and discharge according to the available sunlight. During the course of this testing additional cables were routed though the conduit on November 3rd which resulted in the solar panel becoming detached from the charge circuit. With the battery at a charge state of 12.75 [V] the battery drained with the applied load of the access point until the charge regulator shut off the load on November 8th. With this incident the ability of the wireless access point to operate for 5 days on an incomplete charge was demonstrated.

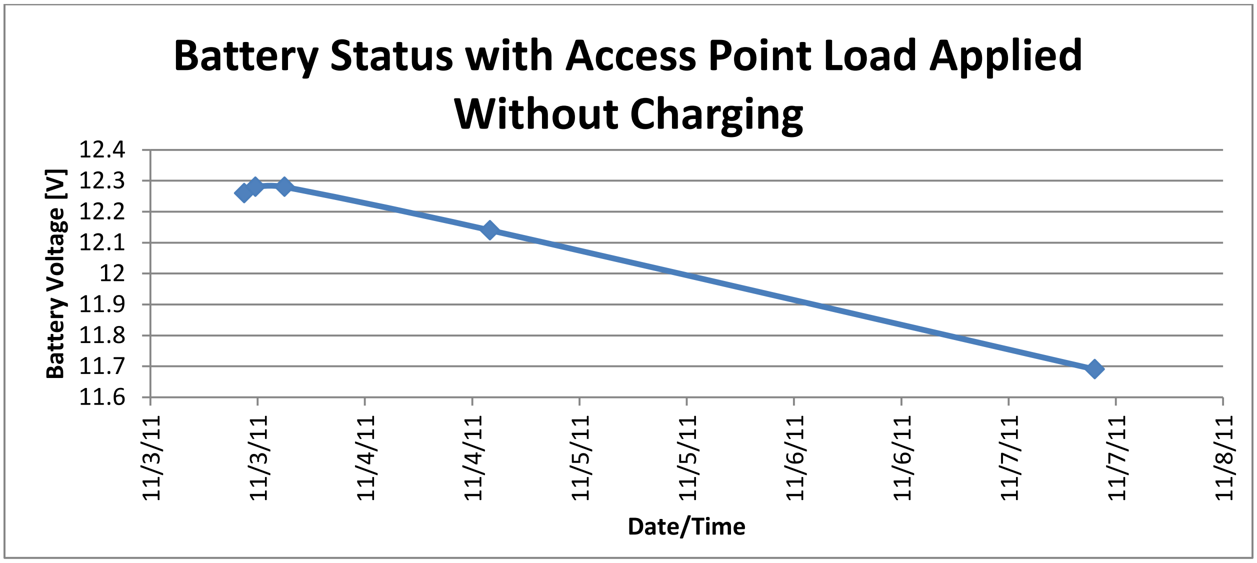


Figure 32: Battery status with no charging and access point load applied

On November 3rd the system was charging until the solar panel became disconnected in the late afternoon. After this the system discharged for 5 days until the charge regulator turned off the load on November 7th.

Camera Core with Pleora Video Processing Module

With the Tau 320 camera core plugged into the Pleora video processer we determined the power dissipation by measuring the actual voltage of the 9-Volt nominal voltage regulator as well as the current draw used to power the Pleora and attached Tau 320 core. The voltage regulator was measured to output 9.02 [V] and the draw was measured to be 0.325 [A] at steady state. With this data we can easily calculate the power dissipation to be 2.93 Watts when operated on a strictly DC power source. This power draw is 51% less power than the 5.73 Watts determined previously for the Pleora-Tau 320 combination when operating from AC power.

Charge Controller

TheSC-10A charge controller's efficiency was determined to be 94.6% by dividing the power into the charge controller by the power output of the charge controller. This was determined by measuring the open-loop voltage and current for the panel as well as for the battery connection from the charge controller.

Table 12: Open Loop Voltage, Current, and Power into Charge Controller

Solar Panel		
Open Loop Voltage	21.12	Volts
Open Loop Current	1.707	Amps
Open Loop Power	36.05	Watts
Charge Controller		
Open Loop Voltage	20.64	Volts
Open Loop Current	1.653	Amps
Open Loop Power	34.12	Watts

Solar Mounting Strength

Our initial design for the pole side of the solar mounts did not have enough stiffness to satisfy us that it would survive fatigue loading exerted by the wind. The issue was rectified by adding a bent rib along both sides of the brackets, and using mild steel; this dramatically increased the stiffness of the brackets. The discrepancy between the stiffness observed and our expected deflection values seen in the simulations was determined to be due to the type of fixture used in the simulation. Rather than fixing the brackets with bolts, the entire face of the bracket was fixed, which did not accurately predict the twisting deflections we were seeing in the face of the bracket. The rib removed these twisting deflections almost entirely, which brought the brackets back to our expected levels.

The actual strength of the entire system was initially tested by hand. We slowly increased the load on the solar panel until a 160lb team member was fully hanging from it. This load is more than 3 times our expected load from the wind, so we determined the brackets to be more than sufficiently strong. After this testing, the system also stood up to very strong Santa Ana winds, which had reported gusts of over 80 mph in the local newspapers.

Wireless System Range

Omni-Directional Antenna

The standard direction antennae that come on the wireless access point and the router were initially tested by the team. They were able to obtain range of 300 feet with a video feed, but the video got choppy at around 200 feet. From this testing we determined that the omnidirectional antenna would be a good solution for many applications of our system, but did not sufficiently meet the 300 foot requirement given to us. It was decided to attempt to integrate directional antennae into the system in order to get a good video feed at the full 300 feet as depicted below in Figure 33.

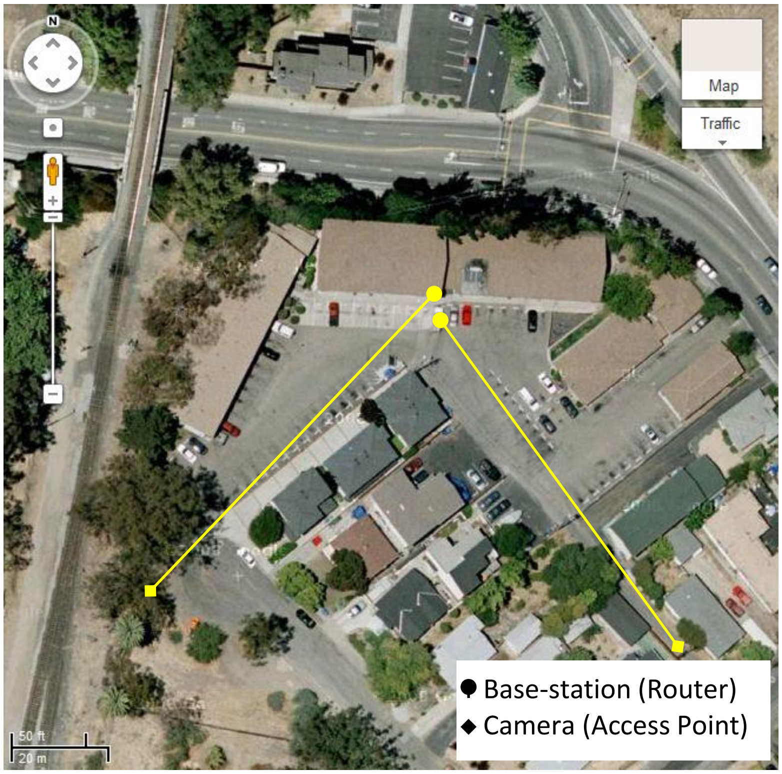


Figure 33: Range testing with omnidirectional antenna over 802.11n

Directional Antenna

The Trendnet TEW-A014D 14DBi high gain directional antenna was selected and implemented to improve the range and quality of the signal for wireless communication. This directional antenna is designed for indoor and outdoor use making inclement weather not a concern. The antenna was attached to the Trendnet TEW-638APB access point with no issues; however, when attaching this antenna to any other router we discovered that very few routers have removable antenna and legacy routers such as the Linksys WRT54GS which featured removable antenna have a different connection from those included with the TEW-A014D. Even with the problem of not being able to attach the second directional antenna to the base-station we found there was a dramatic improvement in the range and quality of the video, even when operation on 802.11g as opposed to 802.11n with the omnidirectional antenna. Furthermore, in this testing we discovered that having the camera wirelessly connected to the camera and any computer viewing the wireless feed from the camera using a wired connection the router dramatically improved the video quality. With these thoughts in mind we were able to achieve real-time streaming video at 375 feet purely outdoors and 300 feet through an exterior wall as depicted in Figure 34 below. The absolute maximum range was not tested due to a lack of available space that had power available for the “base station” router and terrain that could be crossed with the mobile stand significantly greater than 300 feet. In addition it should be noted that these tests were performed without a directional antenna on the “base station” end, the inclusion of the directional antenna would only aid the range and quality of the signal.

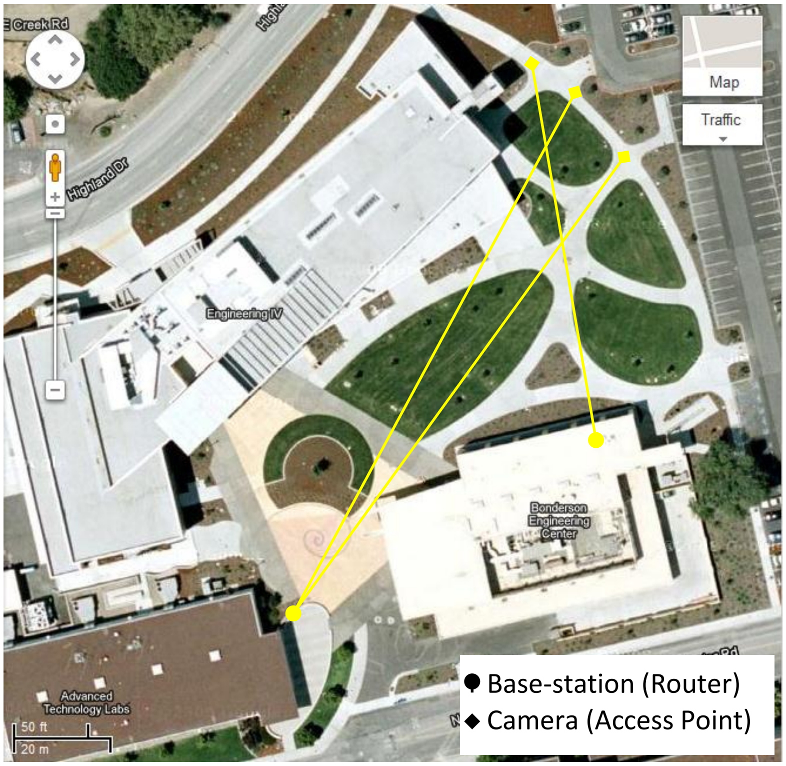


Figure 34: On-campus testing of single directional antenna over 802.11g

Battery Monitoring System

The battery monitoring system was tested across the wireless network via a switch connecting the camera and Leopardboard to the wireless access point inside the storage box. The conversion was originally tested in the lab by sweeping the battery range using a power supply. After installing the system in the storage box, the supply voltages to the A/D converter and Leopardboard were slightly higher. This required the conversion equation to be adjusted. The conversion equation was outputting a voltage of 0.1 volts above the voltage measured by a multimeter. This required a linear adjustment of the y-intercept in the conversion equation from 11.30383 to 11.20383. This voltage was verified by transmitting the log file from the Leopardboard and verifying the voltage. The conversion is done on the client-side and thus did not require any modifications to the server-side code.

Conclusions and Recommendations

Power Generation and Storage

The solar panel performed at the levels expected from the spec sheets. At the level of power generation of we were receiving we could easily operate the Leopardboard based system in our target area of Pullman, Washington. The panel did not have much room for error if there was no power savings with the internals of the system, and it would likely be a good idea to increase the size of the solar panel if that was the final solution.

The battery performed well over the course of 4 months of testing. It was cycled from high voltage to low voltage without issues, and the charge controller kept it from being damaged. We found the power usage on the charge controller to be very low, so it would not really be necessary to use the Texas

Instruments charge controlling chip instead of the integrated box. The battery was discharged without solar power and took 5 days with a 2.5 watt load to become too low of a voltage to continue draining. This is a good indicator that the Leopardboard system would work very well with our battery, as that is very close to the load we expected.

Power Usage

Ultimately the final product should include some type of video processor with wireless connection capabilities. This would allow for all the camera electronics to be consolidated into one component that would be much more power efficient. These power savings would remove any need for increases in the power system, which will keep the visual size and cost of our components down. The best solution would likely be to have Pleora create a new solution using the code they have to interface with the camera as the basis. They mentioned this was possible to us when we spoke with them, but they were not willing to contribute to the project without a potential purchase of a volume of components later.

Mounting

The mounting was strong and held up well. We used FLIR mounting for the camera, and the enclosure came with a mounting solution. This means that we only needed to produce mounting for the solar panel. The mounts were design in a way to be usable at multiple points, and as such there were only 4 new unique parts to create. The parts are mostly small, inexpensive brackets. We would recommend the brackets be laser cut for final production, as it will give a much better looking product for only a marginal cost. The V-bar support would be the most expensive item to produce due to the amount of welding, but the cost could be kept down buy molding it out of plastic or casting it. We found the strength to be above what is necessary for a final product, as the bar will never see loads that would cause it to fail before the brackets. We would also recommend the parts be coated to match the current FLIR mounting components for consistency in appearance.

Video Processing

The final product's video processing unit used the supplied Pleora Ethernet Module. We did not modify this unit since it was designed specifically for FLIR products. In testing the video processing unit, we found the video was clear when not transmitting across a wireless network (solely transmitting on a wired network through a router). There was a noticeable difference in using a wireless network, but the quality was still steaming in real-time at an acceptable rate. Ideally, modifying the Pleora specifically for wireless communication could help increase the quality. This would include optimizing the buffering of video to handle more interference introduced by the wireless protocol.

Wireless Communication

The system utilized a wireless access point with a bridge connecting it to a router on the client-side. Without directional antennas, testing proved to meet the 300 foot wireless transmitting specification. The addition of directional antennas, although not necessary, would help increase the quality of real-time video streaming. The Pleora Ethernet Module could be improved to use the TCP/IP protocol instead of the OSI model. This would help improve flow control and error handling when transmitting the video. Also, combining wireless communication with the video processing subsystem would improve power efficiency and transmitting throughput since it would eliminate an extra unit between the video processing subsystem and client-side router.

References

[1] http://www.gregorybkatz.com/Home/effect-of-dust-on-solar-panels/FullReport-DustonSolarPanelResearch.doc
[2] http://batteryuniversity.com/learn/article/charging_the_lead_acid_battery
[3] http://batteryuniversity.com/learn/article/charging_lithium_ion_batteries
[4] BP 340J 40 Watt Solar Panel Data Sheet http://www.bp.com/liveassets/bp_internet/solar/bp_solar_usa/STAGING/local_assets/downloads_pdfs/pq/BP340J_1-10.pdf
[5] Kyocera KC40T Lowest Price $237.07 http://www.wholesalesolar.com/products.folder/module-folder/kyocera/KC40T.html
[6] Industrial Solar IS-40J Cheapest Price $176.00http://www.amazon.com/Solar-BPSX340J-Bolt-Replacement-Panel/dp/B00467YG2G
[7] BP 340J 40 Watt Solar Panel Lowest Price $228.28 http://www.solarhome.org/bp40wattsolarpanelbp340j.aspx
[8] Universal Battery UB12500 12V 50Ah Lead Acid Battery: Size (LxWxH) is 7.72 x 6.50 x 7.17, Weight 32 lb http://upgi.com/Themes/leanandgreen/images/UPG/Downloads/UPG_low_res_catalog_VR122209.pdf
[9] BB Battery EB50-12 50Ah battery http://www.batteryprice.com/bbbatteryeb50-1212v50ahleadacidbattery.aspx
[10] Dynasty WB1250LD-FR 50Ah Battery http://www.pcuniverse.com/Dynasty-UPS-battery-1-x-lead-acid-50-Ah/WB1250LD-FR/pd/p3332571
[11] Universal Battery UB12500 cheapest price, $79 http://www.factoriesonline.com/ProductInfo.aspx?id=2442127
[12] AMPOLEMNT14 AM1648L Pole Mounting Kit Instructions http://www.alliedmoulded.com/UserFiles/My%20Media/Pole%20Mount%20Instructions%207-17-07.pdf
[13] Allied Enclosures AM1648L Cheapest Price $87.79 http://store.solutionsdirectonline.com/am1648l-nema-4x-fiberglass-enclosure-with-snap-latch-hinged-cover-16x14x8-p116.aspx
[14] FLIR F-Series Wall Mount http://www.surveillent.com/flir-systems-500046200-f-series-wall-mount.aspx
[15] FLIR F-Series Pole Mount Adapter http://www.surveillent.com/flir-systems-4119507-pole-adapter-for-f-series-wall-mount.aspx
[16] Aluminum .080" Sheet Pricing http://www.onlinemetals.com/merchant.cfm?pid=1243&step=4&showunits=inches&id=76&top_cat=60
[17] Aluminum 1 x 1.5 x .125 Rectangle Tubing Pricing http://www.onlinemetals.com/merchant.cfm?pid=18109&step=4&showunits=inches&id=1269&top_cat=60
[18] Aluminum 1" Solid Rounds http://www.onlinemetals.com/merchant.cfm?pid=18115&step=4&showunits=inches&id=1278&top_cat=60
[19] AMPOLEMNT14 AM1648L Pole Mounting Kit Instructions http://www.alliedmoulded.com/UserFiles/My%20Media/Pole%20Mount%20Instructions%207-17-07.pdf
[20] AMPOLEMNT14 Cheapest Price$59.23 http://store.solutionsdirectonline.com/ampolemnt14-pole-mount-kit-for-14x12-and-16x14-fiberglass-enclosures-p76.aspx
[21] http://focus.ti.com/docs/prod/folders/print/bq24156.html
[22] http://focus.ti.com/docs/prod/folders/print/bq24210.html
[23] http://focus.ti.com/docs/prod/folders/print/bq24650.html
[24] 12V SLA/LFP Charge Controller $27.95 http://www.batteryspace.com/solardcchargecontroller120w10aratefor12vslaorlfpbattery.aspx
[25] 11.1V Li-Ion Charge controller $29.98 http://www.batteryspace.com/solardcchargecontroller120w10aratefor111vli-ionbatterypack.aspx
[26] P/N:#96728 7A Solar Charge Regulator $29.99 http://www.harborfreight.com/7-amp-solar-charge-regulator-96728.html

Appendix A: Contact Info

FLIR Contacts

FLIR Systems, Inc.
70 Castilian Drive
Goleta, CA 93117

Main Contact:

Don Shrum
Project Manager - Maritime and Airborne Products
don.shrum@flir.com
805.690.5009 office
805.690.7155 fax
805.689.5712 mobile

Technical Contacts:

David.Risdall
Systems Engineer
david.risdall@flir.com

Marcel Tremblay
Director of Mechanical Engineering – Commercial Systems
marcel.tremblay@flir.com
805.690.6749

Wi-FLIR Team Contacts

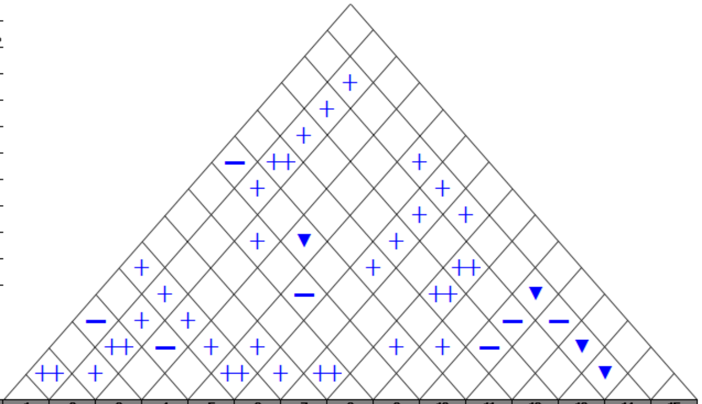
Andy Bonk
Wi-FLIR Point of Contact
andycb123@gmail.com
760.855.2352

Billy McVicker
anothermcvicker@gmail.com
530.913.1035

Jacob Richardson
jacob.richardson11@gmail.com
661.246.6763

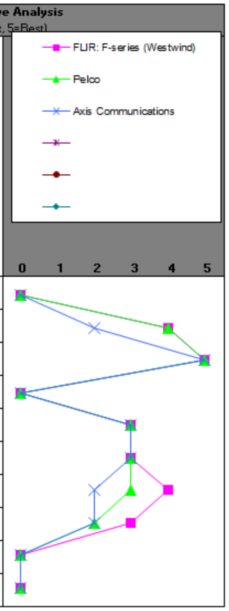
Appendix B: QFD

Title:	Wi-FLIR
Author:	Andy Bonk, Nicholas Luzuriaga, Billy McVickor, Jacob Richardso
Date:	1/28/2011
Notes:	



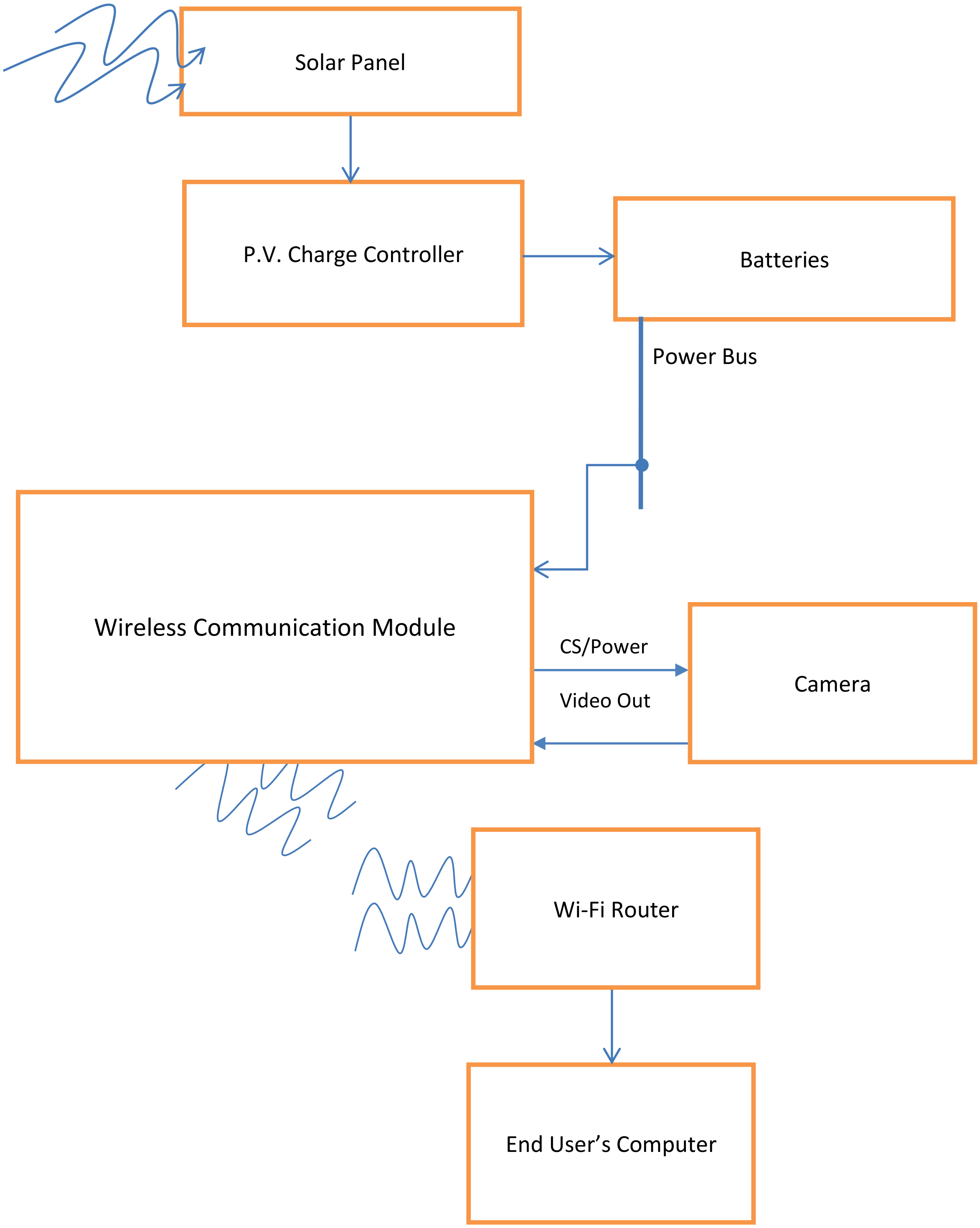
Legend		
⊙	Strong Relationship	9
○	Moderate Relationship	3
▲	Weak Relationship	1
++	Strong Positive Correlation	
+	Positive Correlation	
—	Negative Correlation	
▼	Strong Negative Correlation	
▼	Objective Is To Minimize	
▲	Objective Is To Maximize	
X	Objective Is To Hit Target	

				Column #	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Competitive Analysis** (0=Worst, 5=Best)					
				Direction of Improvement: Minimize (▼), Maximize (▲), or Target (x)	X	▲	X	X	▼	▲	X	▲	X	X	X	▼	▲	▲							
Row #	**Max Relationship Value in Row**	**Relative Weight**	**Weight / Importance**	**Quality Characteristics** (a.k.a. "Functional Requirements" or "Hows") **Demanded Quality** (a.k.a. "Customer Requirements" or "Whats")	communication range	% of people who like it	IP Code Rating	% operating during day from 2.9 hrs sunlinght	Numer of screws to remove for service	Installers who like it	Number of independent systems	Percent of product purchased as installed	Windspeed it can withstand	Cost of components	Size	Minimum operating temperature	Maximum operating temperature	Mean Time Between Service		FLIR: F-series (Westwind)	Pelco	Axis Communications			
1	9	13.0	50.0	Wireless Communication	⊙			▲		○		○		▲		▲	▲	▲		0	0	0			
2	9	2.6	10.0	Aestetics		⊙	▲			▲	○		▲	▲	⊙		○			4	4	2			
3	9	13.0	50.0	Weatherproof		▲	⊙		▲				⊙			▲	▲	⊙		5	5	5			
4	9	13.0	50.0	Self-powered	▲	○		⊙		○	▲			○		▲	▲	○		0	0	0			
5	9	9.1	35.0	Easily servicable & accessable componets		○	○		⊙	○	▲				▲					3	3	3			
6	9	3.9	15.0	1-Person for making repairs (not needing a team to do maintainance on 1 unit)		▲			○	⊙	○									3	3	3			
7	9	9.1	35.0	Modular components (swapping parts)	○	▲			▲	○	⊙	⊙								4	3	2			
8	9	10.4	40.0	Off-the-shelf components	▲						○	⊙		○	▲					3	2	2			
9	9	13.0	50.0	Inexpensive addition to current component cost	▲							⊙		⊙						0	0	0			
10	3	13.0	50.0	Positionable solar array and camera		▲		○		○										0	0	0			
				Target or Limit Value	300 ft	90%	IP67	100%	7 screws	90%	5 independent subsystems	100%	70 mph	$500	8x10x14 inches	-20°C	+55°C	2 years							
				Difficulty (0=Easy to Accomplish, 10=Extremely Difficult)	3	7	7	8	5	5	6	6	5	6	7	5	6	7							
				Max Relationship Value in Column	9	9	9	9	9	9	9	9	9	9	9	1	3	9							
				Weight / Importance	180.5	128.6	146.8	168.8	115.6	209.1	154.5	331.2	119.5	202.6	42.9	39.0	46.8	168.8							
				Relative Weight	8.8	6.3	7.1	8.2	5.6	10.2	7.5	16.1	5.8	9.9	2.1	1.9	2.3	8.2							



Powered by QFD Online (http://www.QFDOnline.com)

Appendix C: System Block Diagram



Appendix D: Decision Matrices

Pugh Matrix: Wireless				
Selection Criteria	Pleora Ethernet with AP (Reference)	Microcontroller with wireless (802.11) module	Satellite Communication	Laser Communication
Range	0	0	+	-
Power usage	0	+	-	+
Reliability	0	-	-	-
Serviceability	0	0	-	-
Hardware Cost	0	+	-	-
Manufacturing Cost	0	-	-	-
Weight	0	+	-	-
Latency	0	0	-	+
Sum '+'	0	3	1	2
Sum '0'	8	3	0	0
Sum '-'	0	2	7	6
Net Score	0	1	-6	-4
Rank	2	1	4	3
Continue?	yes	yes	no	no

Pugh Matrix: Energy Storage						
Selection Criteria	Lead Acid (Reference)	Li-Ion	Li-Po	Ni-Cd	Gel	Ultra-Capacitor
Energy Density	0	+	+	+	0	-
Lifetime	0	-	-	+	+	+
Cost	0	-	-	-	-	-
Oxidation	0	0	0	0	+	+
Volatility	0	-	-	0	+	+
Availability	0	0	-	-	-	-
Leakage Rate	0	-	-	-	0	-
Sum '+'	0	1	1	2	3	3
Sum '0'	7	2	1	2	2	0
Sum '-'	0	4	5	3	2	4
Net Score	0	-3	-4	-1	1	-1
Rank	2	4	5	3	1	3
Continue?	yes	no	no	yes	yes	no

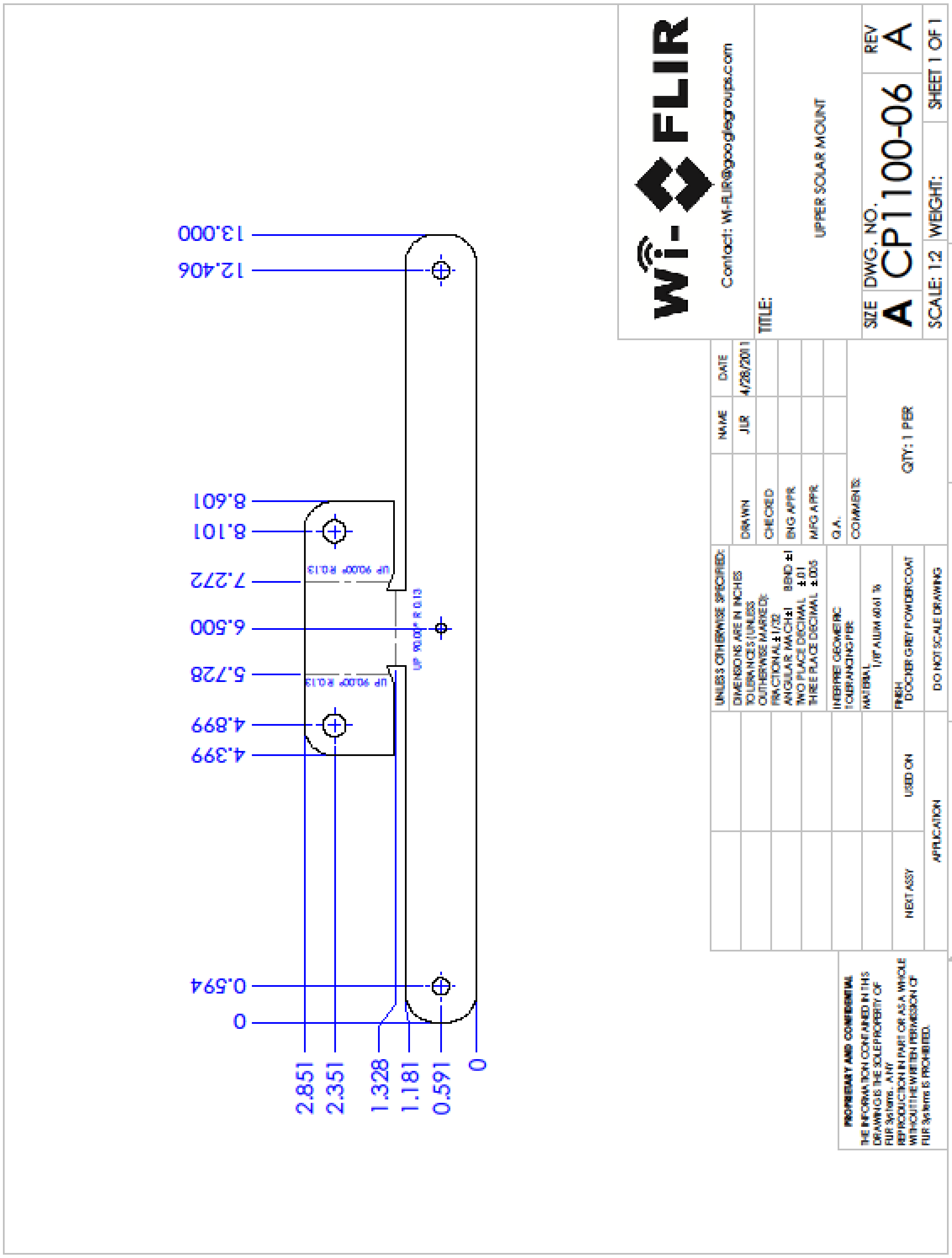
Pugh Matrix: Energy Generation

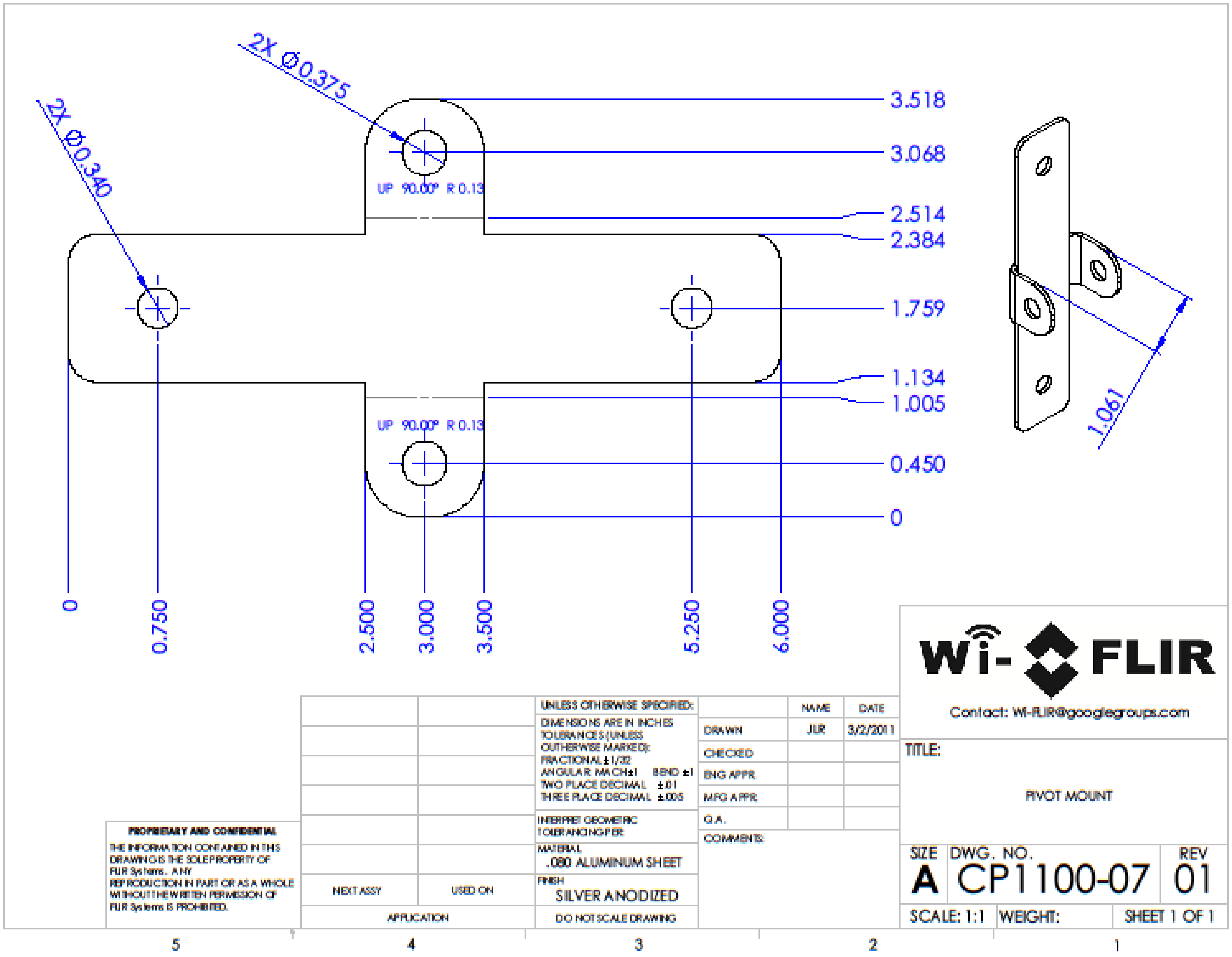
Selection Criteria	Solar (Reference)	Wind	Nuclear	Fossil Fuel
Cost	0	0	-	-
Durability	0	-	-	-
Volatility	0	0	-	-
Sustainability	0	-	-	-
Aesthetics	0	0	-	0
Availability	0	-	-	+
Size	0	-	-	0
Sum '+'	0	0	0	1
Sum '0'	7	3	0	2
Sum '-'	0	4	7	4
Net Score	0	-4	-7	-3
Rank	1	3	4	2
Continue?	yes	no	no	no

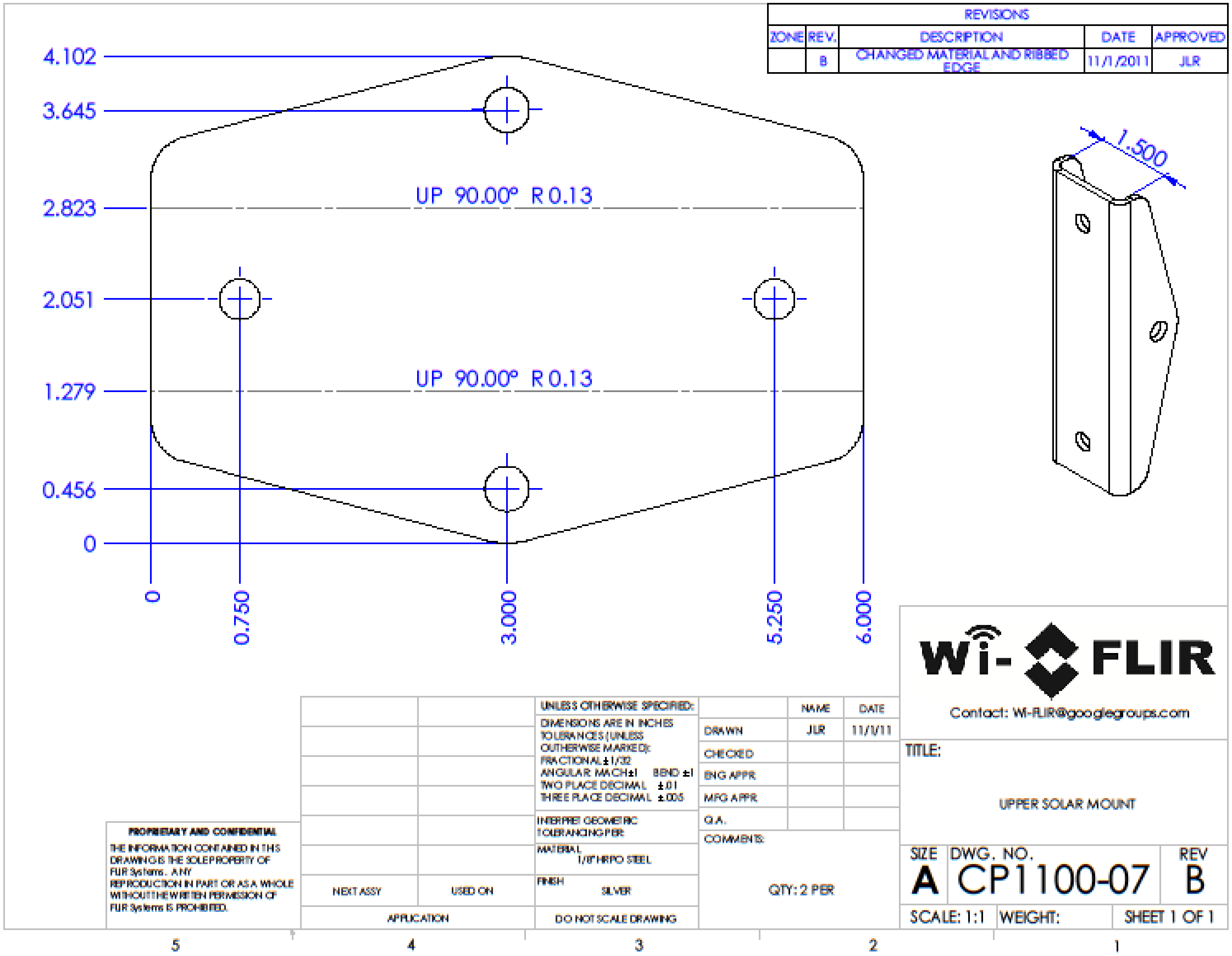
Pugh Matrix: Enclosure

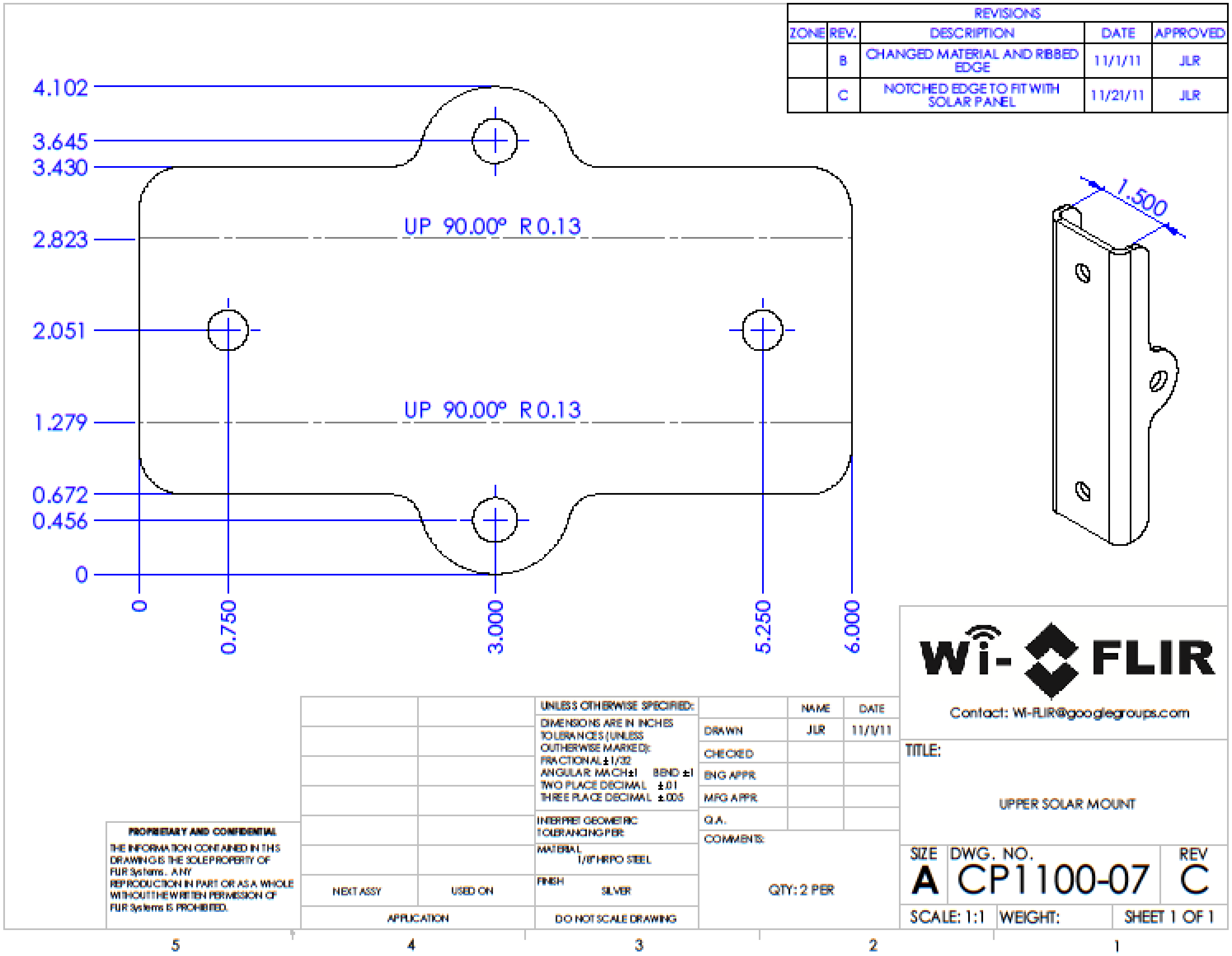
Selection Criteria	FLIR: F-Series (Reference)	FLIR: SR-Series	Custom Plastic	Custom Aluminum
Cost	0	+	+	0
Availability	0	0	-	-
Durability	0	-	-	0
Weight	0	+	+	0
Sealing	0	0	0	0
Aesthetics	0	0	-	0
Size	0	-	0	0
Sum '+'	0	2	2	0
Sum '0'	7	3	2	6
Sum '-'	0	2	3	1
Net Score	0	0	-1	-1
Rank	1	1	2	2
Continue?	yes	yes	yes	yes

Appendix E: Final Drawings




2X Ø0.375
2X Ø0.340
UP 90.00° R 0.13
UP 90.00° R 0.13
3.518
3.068
2.514
2.384
1.759
1.134
1.005
0.450
0
0
0.750
2.500
3.000
3.500
5.250
6.000
1.061
Wi-FLIR
Contact: Wi-FLIR@googlegroups.com
TITLE:
PIVOT MOUNT
SIZE
A
DWG. NO.
CP1100-07
REV
01
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
NAME
DATE
DRAWN
JLR
3/2/2011
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES (UNLESS OUTHERWISE MARKED):
FRACTIONAL ± 1/32
ANGULAR: MACH±1 BEND ±1
TWO PLACE DECIMAL ±.01
THREE PLACE DECIMAL ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
.080 ALUMINUM SHEET
FINISH
SILVER ANODIZED
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF FLIR Systems. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF FLIR Systems IS PROHIBITED.
5
4
3
2
1


REVISIONS
ZONE
REV.
DESCRIPTION
DATE
APPROVED
B
CHANGED MATERIAL AND RIBBED EDGE
11/1/2011
JLR
4.102
3.645
2.823
2.051
1.279
0.456
0
0
0.750
3.000
5.250
6.000
UP 90.00° R 0.13
UP 90.00° R 0.13
1.500
Wi-FLIR
Contact: Wi-FLIR@googlegroups.com
TITLE:
UPPER SOLAR MOUNT
SIZE
A
DWG. NO.
CP1100-07
REV
B
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
NAME
DATE
DRAWN
JLR
11/1/11
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
QTY: 2 PER
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES (UNLESS OUTHERWISE MARKED):
FRACTIONAL ± 1/32
ANGULAR MACH ± 1 BEND ± 1
TWO PLACE DECIMAL ± .01
THREE PLACE DECIMAL ± .005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
1/8" HRPO STEEL
FINISH
SILVER
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF FLIR Systems. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF FLIR Systems IS PROHIBITED.
5
4
3
2
1



REVISIONS				
ZONE	REV.	DESCRIPTION	DATE	APPROVED
	B	CHANGED MATERIAL AND RIBBED EDGE	11/1/11	JLR
	C	NOTCHED EDGE TO FIT WITH SOLAR PANEL	11/21/11	JLR

Wi-FLIR

Contact: Wi-FLIR@googlegroups.com

TITLE:

UPPER SOLAR MOUNT

SIZE	DWG. NO.	REV
A	CP1100-07	C

SCALE: 1:1 WEIGHT: SHEET 1 OF 1

	NAME	DATE
DRAWN	JLR	11/1/11
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:

QTY: 2 PER

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES (UNLESS OUTHERWISE MARKED):
FRACTIONAL ± 1/32
ANGULAR MACH ±1 BEND ±1
TWO PLACE DECIMAL ±.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC TOLERANCING PER:

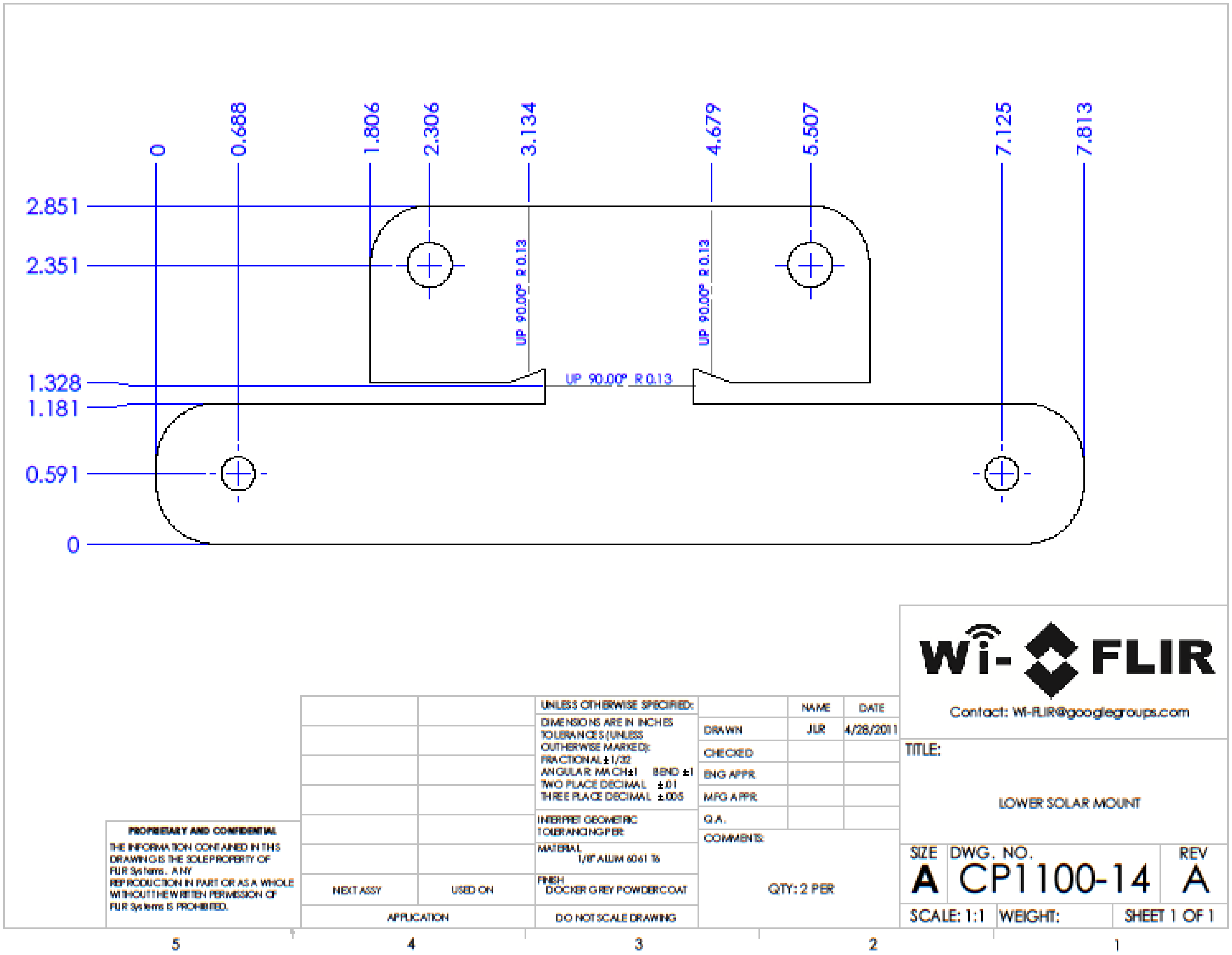
MATERIAL 1/8" HRPO STEEL

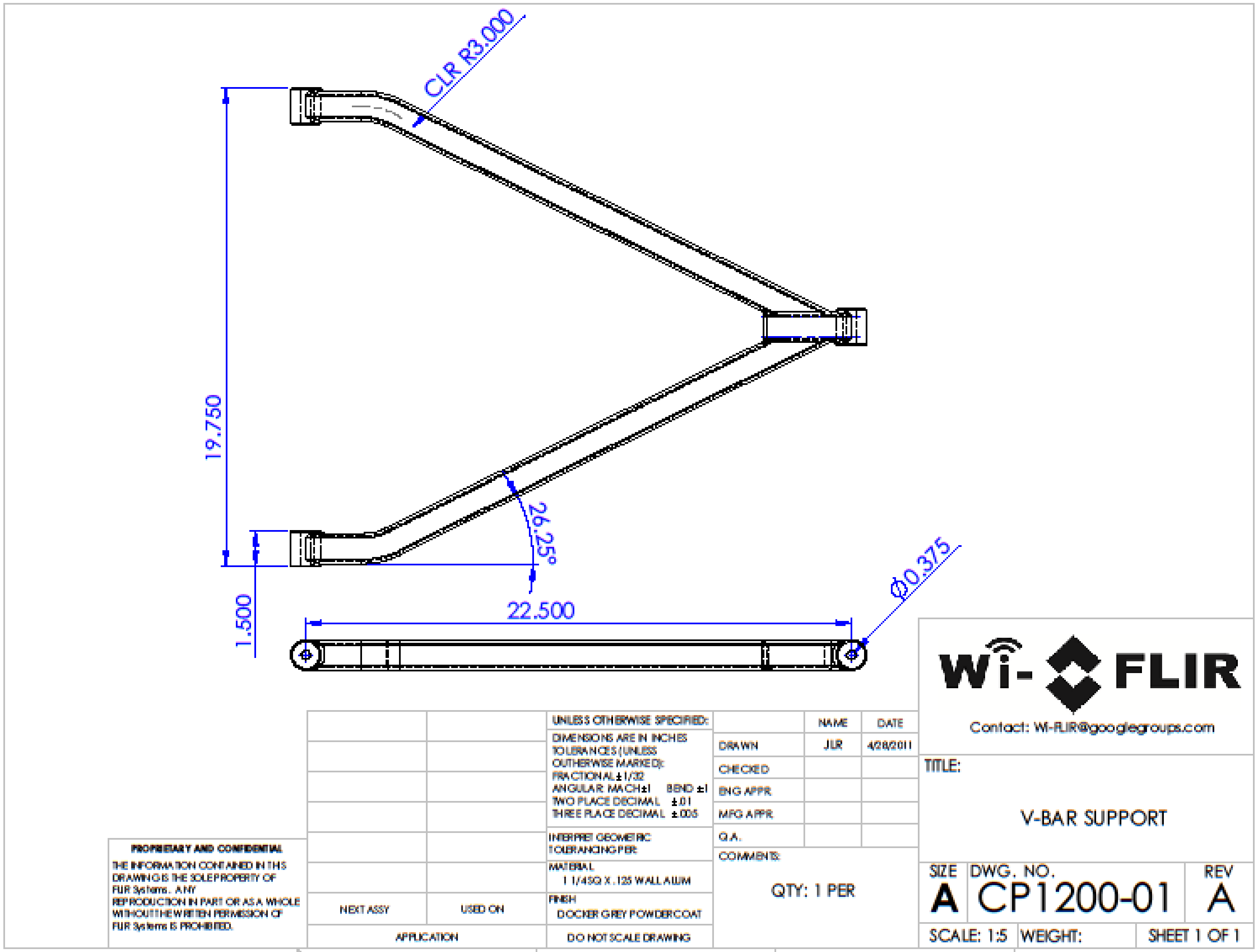
FINISH SILVER

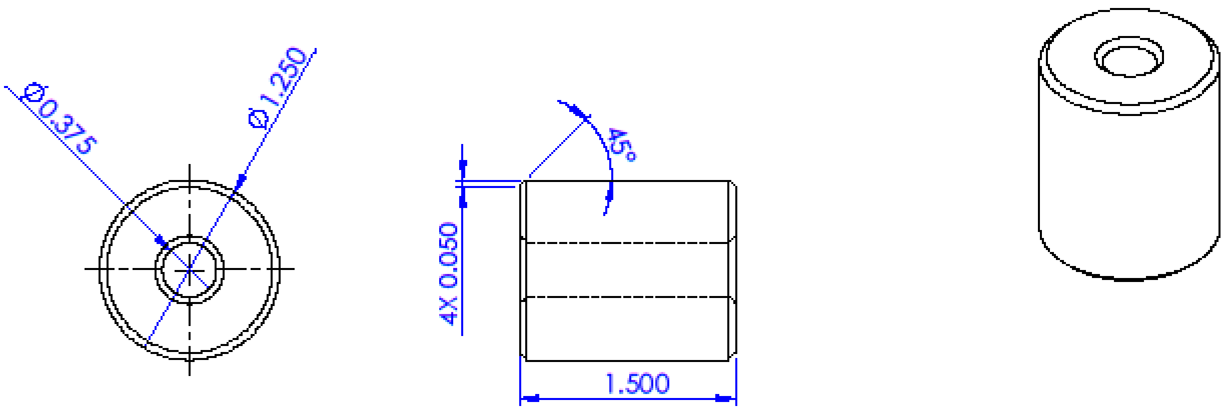
DO NOT SCALE DRAWING

NEXT ASSY	USED ON
APPLICATION	

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF FLIR Systems. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF FLIR Systems IS PROHIBITED.

0
0.688
1.806
2.306
3.134
4.679
5.507
7.125
7.813
2.851
2.351
1.328
1.181
0.591
0
UP 90.00° R 0.13
UP 90.00° R 0.13
UP 90.00° R 0.13
Wi-FLIR
Contact: Wi-FLIR@googlegroups.com
TITLE:
LOWER SOLAR MOUNT
SIZE A
DWG. NO. CP1100-14
REV A
SCALE: 1:1
WEIGHT:
SHEET 1 OF 1
NAME
DATE
DRAWN JLR 4/28/2011
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
QTY: 2 PER
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES (UNLESS OUTHERWISE MARKED):
FRACTIONAL ± 1/32
ANGULAR MACH ±1 BEND ±1
TWO PLACE DECIMAL ±.01
THREE PLACE DECIMAL ±.005
INTERPRET GEOMETRIC TOLERANCING PER:
MATERIAL
1/8" ALUM 6061 T6
FINISH
DOCKER GREY POWDERCOAT
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF FLIR Systems. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF FLIR Systems IS PROHIBITED.
5
4
3
2
1


CLR R3.000
19.750
26.25°
1.500
22.500
Ø0.375
Wi-FLIR
Contact: Wi-FLIR@googlegroups.com
TITLE:
V-BAR SUPPORT
SIZE
A
DWG. NO.
CP1200-01
REV
A
SCALE: 1:5
WEIGHT:
SHEET 1 OF 1
NAME
DATE
DRAWN
JLR
4/28/2011
CHECKED
ENG APPR.
MFG APPR.
Q.A.
COMMENTS:
QTY: 1 PER
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES (UNLESS
OUTHERWISE MARKED):
FRACTIONAL ± 1/32
ANGULAR: MACH ±1 BEND ±1
TWO PLACE DECIMAL ±.01
THREE PLACE DECIMAL ±.005
INTERPRET GEOMETRIC
TOLERANCING PER:
MATERIAL
1 1/4SQ X .125 WALL ALUM
FINISH
DOCKER GREY POWDERCOAT
DO NOT SCALE DRAWING
NEXT ASSY
USED ON
APPLICATION
PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS
DRAWING IS THE SOLE PROPERTY OF
FLIR Systems. ANY
REPRODUCTION IN PART OR AS A WHOLE
WITHOUT THE WRITTEN PERMISSION OF
FLIR Systems IS PROHIBITED.
5
4
3
2
1



Wi-FLIR

Contact: Wi-FLIR@googlegroups.com

TITLE:

LINKAGE ARM END PIVOT TUBE

SIZE	DWG. NO.	REV
A	CP1200	01
SCALE: 1:1	WEIGHT:	SHEET 3 OF 3

	NAME	DATE
DRAWN	JLR	3/2/2011
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		
COMMENTS:		

UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN INCHES
TOLERANCES (UNLESS
OUTHERWISE MARKED):
FRACTIONAL ± 1/32
ANGULAR: MACH ±1 BEND ±1
TWO PLACE DECIMAL ±.01
THREE PLACE DECIMAL ±.005

INTERPRET GEOMETRIC
TOLERANCING PER:

MATERIAL
1 1/4" 6063 T53 ALUMINUM ROUNDS

FINISH
SILVER ANODIZED

DO NOT SCALE DRAWING

NEXT ASSY	USED ON
APPLICATION	

PROPRIETARY AND CONFIDENTIAL
THE INFORMATION CONTAINED IN THIS DRAWING IS THE SOLE PROPERTY OF FLIR Systems. ANY REPRODUCTION IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF FLIR Systems IS PROHIBITED.

5 4 3 2 1

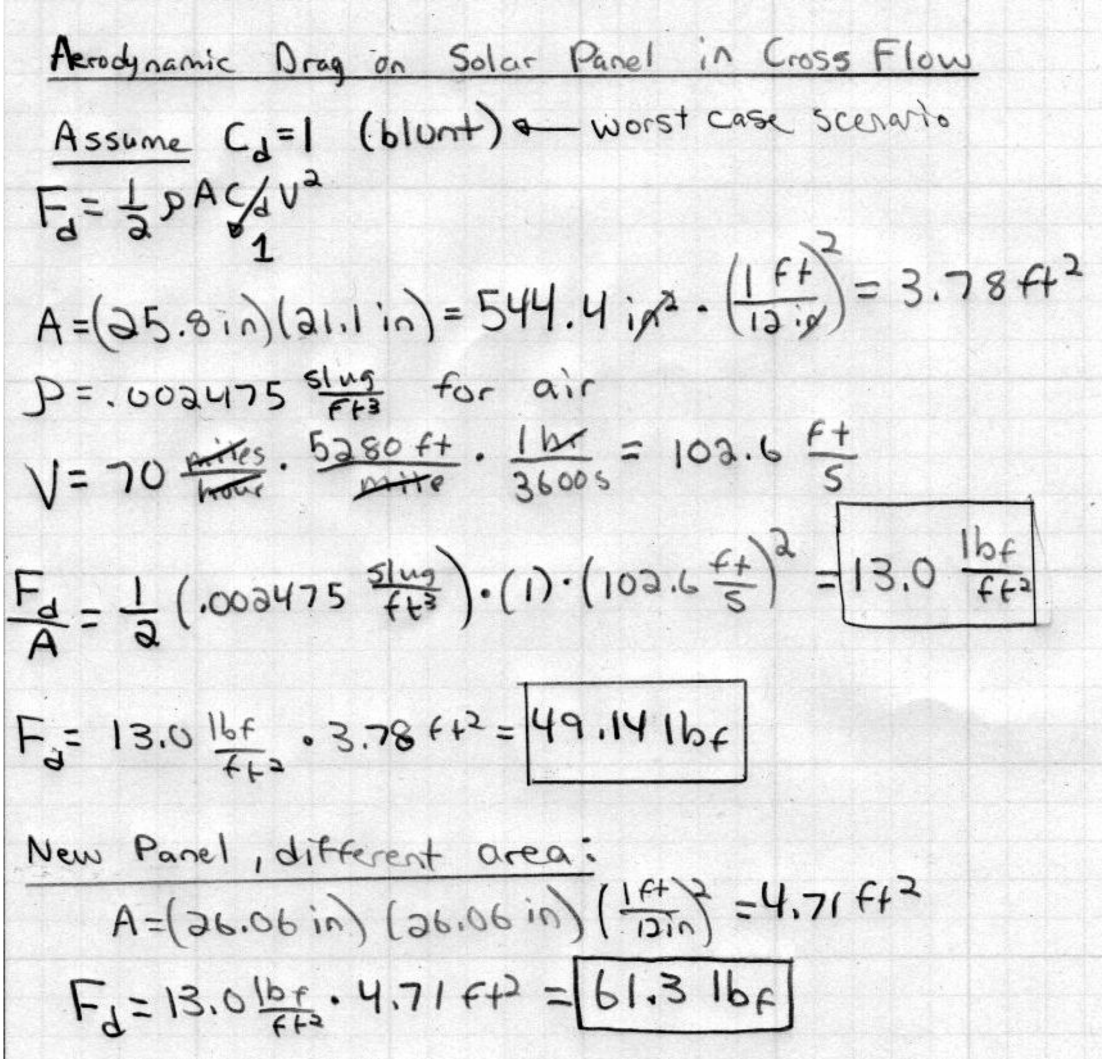
Appendix F: Bill of Materials

Part Number	Description	Quantity	Piece Price	Extended Price	Reference #
IS-40J	Solar Panel	1	$176.00	$176.00	5
4119507	FLIR Pole-Adapter	3	$130.00	$390.00	12
500046200	FLIR Wall Mount	1	$269.00	$269.00	13
CP1100-07	Solar Panel Pole Adapter	2	$7.00	$14.00	15, App E
CP1100-06	Solar Panel Pivot Mount	2	$12.00	$24.00	15, App E
CP1200	Solar Panel Linkage Arm	1	$18.00	$18.00	16, 17, App E
UB12500	50 Ah Lead Acid Battery	1	$79.00	$79.00	10
AM1648L	Battery Enclosure	1	$87.79	$87.79	12
AMPOLEMNT14	Battery Enclosure Pole Mounting Kit	1	$59.23	$59.23	19
SC-10A	DC Charge Controller SLA >10W	1	$27.95	$27.95	23
BP24650	Charge Controller Chip	1	$9.52	$9.52	22
LI-LB01	Leopardboard 355	1	$84.00	$84.00	
LI-SER-01	Serial Adapter from Leopard Imaging	1	$13.00	$13.00	
cB-OWL222ax-02	OEM Wireless LAN Module	1	$94.00	$94.00	
TEW-638APB	Trendnet Wireless N Acesss Point	1	$68.23	$68.23	
TEW-658BRM	Trendnet Wireless N Router	1	$54.99	$54.99	
TEW-AO14D	Trendnet 14 dB Gain Antenna	2	$46.99	$93.98	
			Total:	$1,562.69	

Appendix G: Simulation Results

Table 13: Wind Load Simulation Results Summary

	30 Degrees		60 Degrees	
Load Type	Maximum Von Mises Stress (Mpa)	Maximum Deflection (mm)	Maximum Von Mises Stress (Mpa)	Maximum Deflection (mm)
Even	39.7	0.49	33.4	0.35
Uneven	9	0.3	41.6	0.27
Side	11.7	0.13	24.3	0.18



Aerodynamic Drag on Solar Panel in Cross Flow

Assume $C_d=1$ (blunt) ← worst case scenario

$$F_d=\frac{1}{2}\rho A \underset{1}{\cancel{C_d}} V^2$$

$$A=(25.8\,in)(21.1\,in)=544.4\,\cancel{in^2}\cdot\left(\frac{1\,ft}{12\,\cancel{in}}\right)^2=3.78\,ft^2$$

$\rho=.002475\,\frac{slug}{ft^3}$ for air

$$V=70\,\frac{\cancel{miles}}{\cancel{hour}}\cdot\frac{5280\,ft}{\cancel{mile}}\cdot\frac{1\,\cancel{hr}}{3600\,s}=102.6\,\frac{ft}{s}$$

$$\frac{F_d}{A}=\frac{1}{2}\left(.002475\,\frac{slug}{ft^3}\right)\cdot(1)\cdot\left(102.6\,\frac{ft}{s}\right)^2=\boxed{13.0\,\frac{lbf}{ft^2}}$$

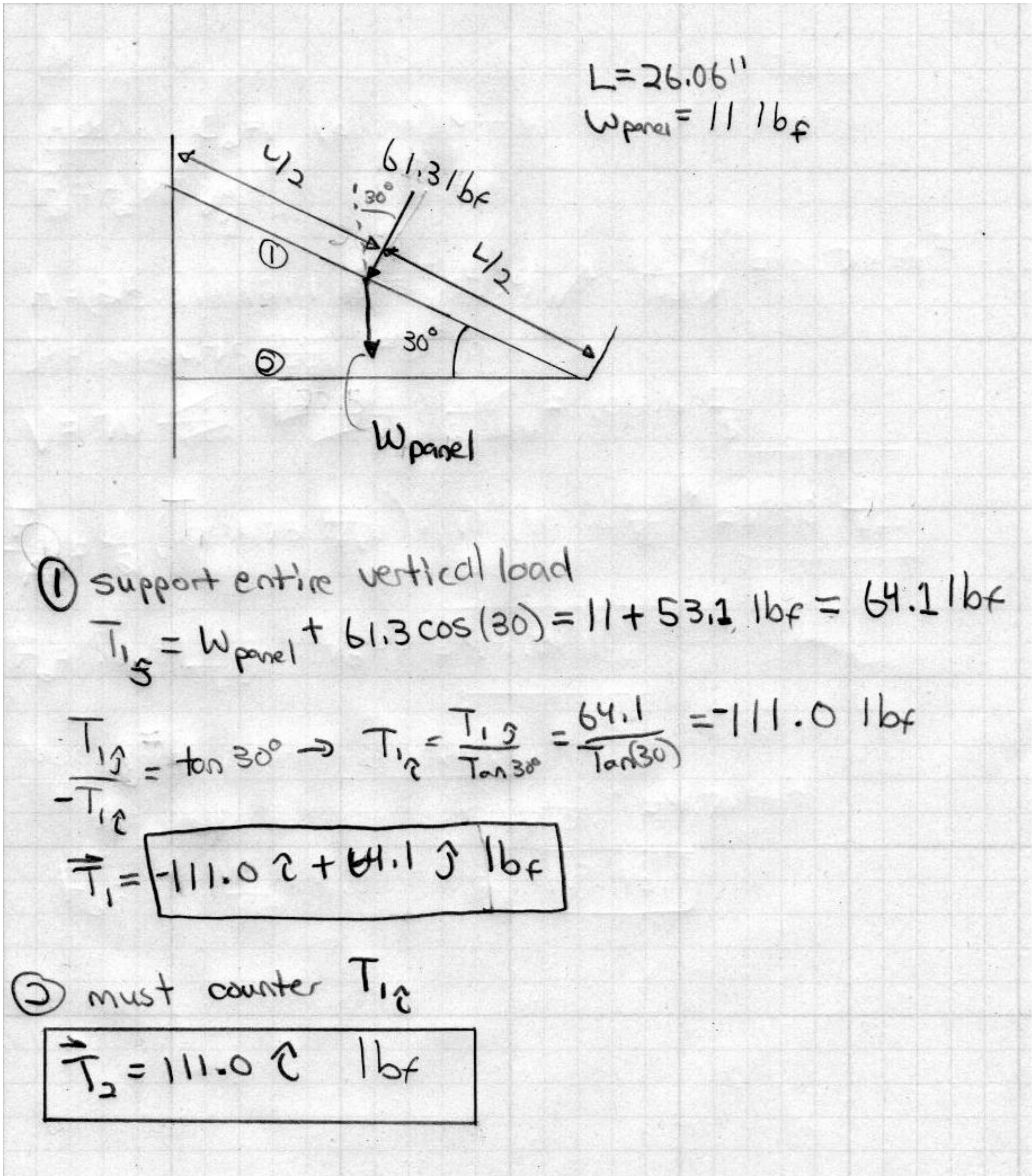
$$F_d=13.0\,\frac{lbf}{ft^2}\cdot 3.78\,ft^2=\boxed{49.14\,lbf}$$

New Panel, different area:

$$A=(26.06\,in)(26.06\,in)\left(\frac{1\,ft}{12\,in}\right)^2=4.71\,ft^2$$

$$F_d=13.0\,\frac{lbf}{ft^2}\cdot 4.71\,ft^2=\boxed{61.3\,lbf}$$

Figure 35: Aeodynamic Analysis of Panel in Cross Flow



① support entire vertical load

$$T_{1\hat{j}} = W_{panel} + 61.3\cos(30) = 11 + 53.1\ \text{lbf} = 64.1\ \text{lbf}$$

$$\frac{T_{1\hat{j}}}{-T_{1\hat{i}}} = \tan 30° \rightarrow T_{1\hat{i}} = \frac{T_{1\hat{j}}}{\tan 30°} = \frac{64.1}{\tan(30)} = -111.0\ \text{lbf}$$

$$\vec{T}_1 = \boxed{-111.0\,\hat{i} + 64.1\,\hat{j}\ \text{lbf}}$$

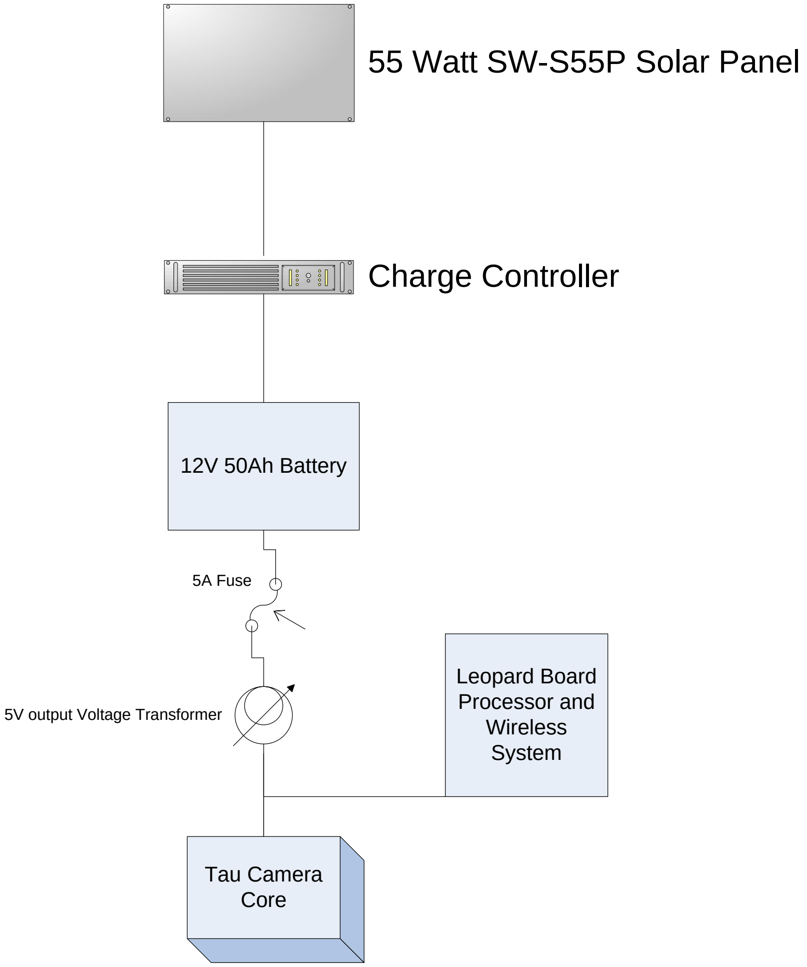
⑤ must counter $T_{1\hat{i}}$

$$\boxed{\vec{T}_2 = 111.0\,\hat{i}\ \ \text{lbf}}$$

Figure 36: Basic Statics Analysis for forces in Members

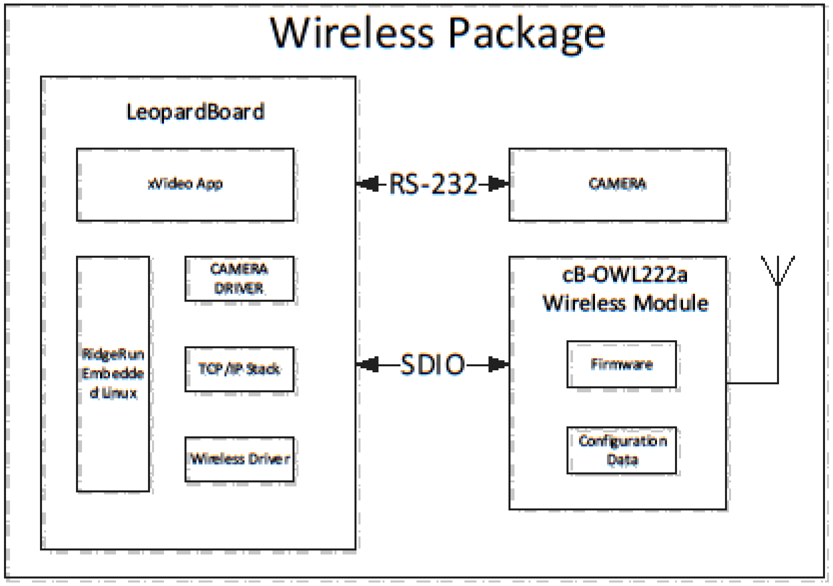
Appendix H: Expanded Power System

The below graphic has been developed to better show how the entire power system will piece together.

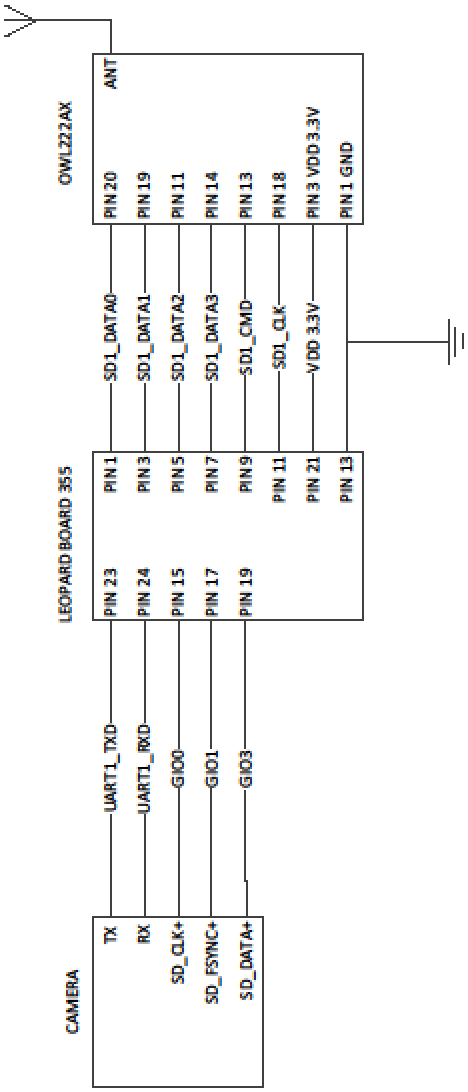

55 Watt SW-S55P Solar Panel
Charge Controller
12V 50Ah Battery
5A Fuse
5V output Voltage Transformer
Leopard Board Processor and Wireless System
Tau Camera Core

Appendix I: Wireless-Video Architecture

Architecture

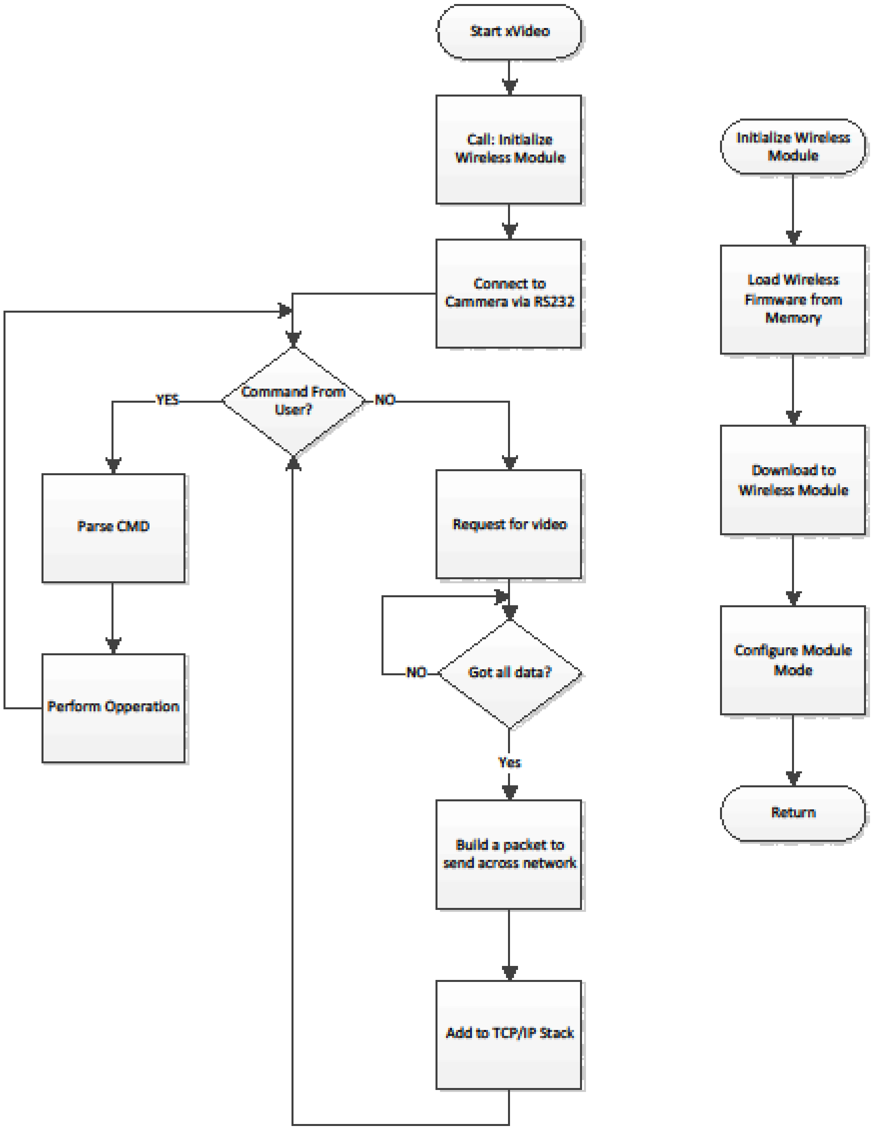


Appendix J:Tau-Leopardboard-OWL222a Pin Connection Diagram



Hardware Block Diagram

Appendix K: Software Model


Start xVideo
Call: Initialize Wireless Module
Connect to Cammera via RS232
Command From User?
YES
NO
Parse CMD
Perform Opperation
Request for video
Got all data?
NO
Yes
Build a packet to send across network
Add to TCP/IP Stack
Initialize Wireless Module
Load Wireless Firmware from Memory
Download to Wireless Module
Configure Module Mode
Return

Appendix L: Serial Camera Communication Code

Makefile:

```
#
# myapps/serial_com/Makefile
#

.PHONY: build chkconfig preconfig buildfs install clean

ifeq ($(DEVDIR),)
$(error ====== DEVDIR variable is empty, invoke this Makefile from the BSP root, or
provide the path to it =====)
endif
include $(DEVDIR)/bsp/mach/Make.conf

BIN                  = serial_com

SRCS                 = serial_com.c
HDRS                 = serial_packets.h
OBJS                 = $(SRCS:.c=.o)

CFLAGS += $(EXTRA_CFLAGS) -Wall -g

ifeq ($(CONFIG_USER_APPS_SERIALCOM),y)
build:
       $(V)$(CC) -c $(CFLAGS) -O2 $(SRCS) -o $(OBJS) $(QOUT)
       $(V)$(CC) $(LDFLAGS) -O2 -o $(BIN) $(OBJS) $(QOUT)

sim:

chkconfig:

preconfig:

buildfs:

install:
       $(V)install -D -m 755 $(BIN) $(FSROOT)/examples/$(BIN) $(QOUT)

clean:
       $(V)rm -f $(BIN) *.debug *.o core *~ $(QOUT)
else
build:
sim:
chkconfig:
preconfig:
buildfs:
install:
clean:
endif

serial_com: serial_com.o
       $(V)$(CC) -o $@

serial_com.o: $(SRCS) $(HDRS)
       $(V)$(CC) $(CFLAGS) -c $<
```

serial_packets.h:

```
/* This Header contains information on the serial packet protocol
 * used to communicate with the Tau 320 Camera core.
 *
 * Created Tue May 24, 2011
 * @author Billy McVicker
 */

/****** Commands to the Camera ********/
/* Serial Packet Protocol *
 * Byte # |  Upper Byte  | Comments
 * --------------------------------------------
 *    1    | Process Code      | Set to 0x6E on all valid incoming and
 *         |                   | outgoing messages
 *    2    | Status            |
 *    3    | Reserved          |
 *    4    | Function          | The function code Table B-4 in User Manual
 *    5    | Byte Count (MSB) |
 *    6    | Byte Count (LSB) |
 *    7    | CRC1 (MSB)        |
 *    8    | CRC1 (LSB)        |
 *         | Data              | See argument data types in Table B-4 in manual
 *   ...   | Data              |
 *    N    | Data              |
 *   N+1   | CRC2 (MSB)        |
 *   N+2   | CRC2 (LSB)        |
 */

typedef struct {
    char function;      /* Function code                                 */
    short byte_count;   /* number of data bytes                          */
    short CRC1;         /* CCITT-16 initialized to 0 on the first 6 bytes */
    char *data;         /* the data to be sent                           */
    short CRC2;         /* CCIT-16 initialized to 0 on all previous bytes */
} serial_packet;

#define PROCESS_CODE    0x6E
#define STATUS_CODE     0x00
#define RESERVED_CODE   0x00

/* Function Codes */
#define NO_OP           0x00
#define SET_DEFAULTS    0x01
#define CAMERA_RESET    0x02
#define RESET_FACTORY_DEFUALTS 0x03
#define SERIAL_NUMBER   0x04
#define GET_REVISION    0x05
#define GAIN_MODE       0x0A
#define FFC_MODE_SELECT 0x0B

#define SHUTTER_POSITION 0x79
#define DO_FFC           0x0C
```

serial_com.c

```
/*****************************************************************************
 * This application is used to access the GPIO pins using the user space gpio
 * driver.  In this example, we are powering on USB VBUS on DM6467 using GPIO22.
 * To do that, first we need to set the PINUMUX0 register to indicate that
 * GPIO22 is used to power on VBUS then write 1 on GPIO22 to power On the bus.
 *
 * Original Author: Brijesh Singh, Texas Instruments Inc
 *****************************************************************************/

/* Include the standard linux header files */
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>

#include <stdint.h>

#include <string.h>
#include "serial_packets.h"

#define BAUDRATE B57600
#define DATABITS CS8                /* CS5, CS6, CS7, or CS8 */
#define DEV_PORT "/dev/ttyS1"
#define TRUE 1
#define FALSE 0

/* function prototypes */
static int uart1_write(int fd, uint8_t *string, int len);
int initialize_port(struct termios *new_term, struct termios *old_term);
int sendCmd(int fd, uint8_t function, short byte_count, uint8_t *data);
short crc_16(uint8_t *data, int len);

/** Reads from the file descriptor fd, which should be a file pointer
 * to a serial UART port.  It assumes the data is read in as Big-Endian
 * and converts it to Little-Endian.
 * @param fd the file descriptor of the serial port to read from
 * @param string a pointer to a spot in memory to store the data
 * @param len the number of bytes to read from the file descriptor
 * @return the number of bytes read (len) if successful, else -1
 */
static int uart1_read(int fd, uint8_t *string, int len) {
   int rdata = -1;

   rdata = read(fd, string, len);
   /* check if an error occurred */
   if (rdata < 0) {
      /* error occurred */
      perror("read");

      return -1;
   }
   return rdata;
}
```

```
/** Write len bytes of the data pointed at by string to the file descriptor fd
 * @param fd the file descriptor to write the data to
 * @param string the data to write to the file descriptor
 * @param len the number of bytes to write
 * @return the number of bytes written on success, else -1
 */
static int uart1_write (int fd, uint8_t *string, int len) {
   int wdata=-1; //, i;

   if ((wdata = write(fd, string, len)) < 0) {
      perror("write");
      return -1;
   }

   return wdata;
}

/** The main thing. Initializes the UART1 serial communication protocol
 * to read and write data.  Then it builds a packet to send to the camera.
 * Then, it sends the command.  Lastly, it waits for a response.  The
 * following is then repeated.
 * @param argc number of input arguments
 * @param argv an array of the input arguments
 * @return 0 on success, else 1-255 on error
 */
int main (int argc, char *argv[]) {
   int uart1_fd;
   int rdata = -1;
   uint8_t data[1024];
   uint8_t shutter_pos = 1;
   struct termios old_term, new_term;    /* old and new serial settings */
   /*struct sigaction siga_io;*/

   if (argc != 2) {
      fprintf(stderr, "usage: %s requires two argument\n", argv[0]);
      exit (EXIT_FAILURE);
   }

   /* initialize the port to the proper settings, i.e. baudrate */
   uart1_fd = initialize_port(&new_term, &old_term);

   /* do important stuff here */
   for (; ;) {
      int j;
      data[1] = 0;
      data[0] = shutter_pos;

      /* Write to the UART serial port */
      if (1 == sendCmd(uart1_fd, SHUTTER_POSITION, 2, data)) {
         fprintf(stderr, "Failed to send command %x\n", SHUTTER_POSITION);
      } else {
         printf("successfully sent data to port\n");
      }

      /* Read from the UART serial port */
      if ( (rdata = uart1_read(uart1_fd, data, 1024)) < 0) {
         /* failed to read data from the UART1 serial port */
         fprintf(stderr, "Failed to read data from the "
               "uart1 serial port\n");
      } else {
         /* print out the data that was retrieved */
         fprintf(stderr, "Read %d bytes from the uart1 serial port\n",
               rdata);

         fprintf(stderr, "\n******* Data Read from UART ********\n");
         for (j = 0; j < rdata; j++) {
            if ((j > 0) && ((j % 16) == 0)) {
               fprintf(stderr, "\n");
            }

            fprintf(stderr, "%x ", (unsigned int) data[j]);
         }
         fprintf(stderr, "\n******** End of Read Data *********\n");
      }

      sleep(5); /* wait for the port to become ready */
      shutter_pos ^= 1;
   }

   /* restore old port settings before exiting */
   tcsetattr(uart1_fd, TCSANOW, &old_term);
   close(uart1_fd);  /* close the serial port */

   return EXIT_SUCCESS;
}

/** Sends the command to the camera using the proper serial protocol
 * that is defined in serial_packets.h
 * @param fd the file descriptor to write to
 * @param function the function code of the command
 * @param byte_count the number of data bytes to send
 * @param data the data to send
 * @return 0 on success, else 1
 */
int sendCmd(int fd, uint8_t function, short byte_count, uint8_t *data) {
   int i;
   int w = 0;
   short crc1, crc2;
   uint8_t *crc_data;

   /* uses the all previous bytes in packet to calculate crc */
   crc_data = (uint8_t *) malloc (10 + byte_count);
   if (!crc_data) {
      /* failed to malloc data for crc2 calculation */
      perror("malloc crc data");
      return 1;
   }
   crc_data[0] = PROCESS_CODE; crc_data[1] = STATUS_CODE;
   crc_data[2] = RESERVED_CODE; crc_data[3] = function;
   crc_data[4] = (uint8_t) (byte_count >> 8);
   crc_data[5] = (uint8_t) (byte_count & 0xff);
   crc1 = crc_16(crc_data, 6); /* calculate the crc1 value */
   crc_data[6] = (uint8_t) (crc1 >> 8);
   crc_data[7] = (uint8_t) (crc1 & 0xff);
   for (i = 0; i < byte_count; i++) {
      /* highest index refers to Most Significant Byte, so make sure
       * you send the most significant byte first */
      crc_data[8+i] = data[(byte_count-1)-i];
   }
   crc2 = crc_16(crc_data, 8+byte_count); /* calculate the crc2 value */
   crc_data[8+byte_count] = (uint8_t) (crc2 >> 8);
   crc_data[9+byte_count] = (uint8_t) (crc2 & 0xff);

   /* send the entire packet */
   if ((10+byte_count) != (w = uart1_write(fd, crc_data, 10+byte_count))) {
      /* failed to write the Process Code byte */
      fprintf(stderr, "Failed to send packet across UART1\n");
      return 1;
   }

   /* free the allocated crc2 data */
   if (crc_data) free(crc_data);

   return 0;
}

/** Initialize the serial port to write to and returns a file
 * descriptor to it.
 * @param new_term a struct to save the new settings to
 * @param old_term a struct to save the old settings to
 * @return the file descriptor of the opened serial port
 */
int initialize_port(struct termios *new_term, struct termios *old_term) {

   int fd;

   /* allow the process to receive SIGIO */
   /*fcntl(fd, F_SETOWN, getpid());*/
   /* Make the serial device asyncronous */
   /*    fcntl(fd, F_SETFL, O_ASYNC);*/

   /* save the current port settings before modifying them */
   memset(new_term, 0, sizeof (new_term));
   new_term->c_cflag = BAUDRATE | DATABITS | CLOCAL | CREAD;
   new_term->c_iflag = IGNPAR | ICRNL;  /* ignore framing and parity errors */
   new_term->c_oflag = 0;
   new_term->c_lflag = 0;
   new_term->c_cc[VQUIT]    = 0;     /* Ctrl-\ */
   new_term->c_cc[VERASE]   = 0;     /* del */
   new_term->c_cc[VKILL]    = 0;     /* @ */
   new_term->c_cc[VEOF]     = 4;     /* Ctrl-d */
   new_term->c_cc[VTIME]    = 0;     /* inter-character timer unused */
   new_term->c_cc[VMIN]     = 1;     /* blocking read until 1 character arrives */
   new_term->c_cc[VSWTC]    = 0;     /* '\0' */
   new_term->c_cc[VSTART]   = 0;     /* Ctrl-q */
   new_term->c_cc[VSTOP]    = 0;     /* Ctrl-s */
   new_term->c_cc[VSUSP]    = 0;     /* Ctrl-z */
   new_term->c_cc[VEOL]     = 0;     /* '\0' */
   new_term->c_cc[VREPRINT] = 0;     /* Ctrl-r */
   new_term->c_cc[VDISCARD] = 0;     /* Ctrl-u */
   new_term->c_cc[VWERASE]  = 0;     /* Ctrl-w */
   new_term->c_cc[VLNEXT]   = 0;     /* Ctrl-v */
   new_term->c_cc[VEOL2]    = 0;     /* '\0' */
   /* open the serial port as non-blocking */
   fd = open(DEV_PORT, O_RDWR | O_NOCTTY | O_NONBLOCK);
   if (fd < 0) {
      perror(DEV_PORT);
      exit (EXIT_FAILURE);
   }

   tcgetattr(fd, old_term);
   tcflush(fd, TCIFLUSH);
   tcsetattr(fd, TCSANOW, new_term);

   return fd;
}

/** Calculates the CCITT-16 or CRC-16 value of the given
 * data.  This is used for error checking when transmitting the
 * data to the camera. It uses the polynomial x^16 + x^12 + x^5 + 1
 * to calculate a 16-bit CRC value. Algorithm taken from wikipedia.org.
 * @param data the data to check for errors
 * @param len the length of the data in bytes
 * @return the crc value
 */
short crc_16(uint8_t *data, int len) {
   short rem = 0;
   int i,j;

   for (i=0; i<len; i++) {
      rem = rem ^ (*(data+i) << 8);

      for (j=0; j < 8; j++) {
         if (rem & 0x8000) {
            rem = (rem << 1) ^ 0x1021;
         } else {
            rem = (rem << 1);
         }

         rem = rem & 0xffff;
      }
   }

   return rem;
}
```

Appendix M: A/D Daemon Application Code

Makefile:

```
#
# myapps/i2c_com/Makefile
#

.PHONY: build chkconfig preconfig buildfs install clean

ifeq ($(DEVDIR),)
$(error ====== DEVDIR variable is empty, invoke this Makefile from the BSP root, or
provide the path to it =====)
endif
include $(DEVDIR)/bsp/mach/Make.conf

BIN                  = i2c_com

SRCS                 = i2c_com.c
HDRS        = i2c_com.h
OBJS                 = $(SRCS:.c=.o)

CFLAGS += $(EXTRA_CFLAGS) #-Wall -g

ifeq ($(CONFIG_USER_APPS_I2CCOM),y)
build:
       $(V)$(CC) -c $(CFLAGS) -O2 $(SRCS) -o $(OBJS) $(QOUT)
       $(V)$(CC) $(LDFLAGS) -O2 -o $(BIN) $(OBJS) $(QOUT)

sim:

chkconfig:

preconfig:

buildfs:

install:
       $(V)install -D -m 755 $(BIN) $(FSROOT)/examples/$(BIN) $(QOUT)

clean:
       $(V)rm -f $(BIN) *.debug *.o core *~ $(QOUT)
else
build:
sim:
chkconfig:
preconfig:
buildfs:
install:
clean:
endif

i2c_com: i2c_com.o
       $(V)$(CC) -o $@

i2c_com.o: $(SRCS) $(HDRS)
       $(V)$(CC) $(CFLAGS) -c $<
```

i2c_com.h

```
/* This Header contains information on the i2c protocol
 * used to communicate with the PCF8591P A/D converter.
 *
 * Created Fri Nov 18, 2011
 * @author William McVicker
 */

#ifndef I2C_COM_H
#define I2C_COM_H

#define DEV_PORT "/dev/i2c-1"
#define I2C_ADDR 0x48
#define TRUE 1
#define FALSE 0

#define A_TO_D_CTRL 0x00

/* function prototypes */
int i2c_write(int fd, char *data, int len);
int i2c_read(int fd, char *data, uint16_t len);
int initialize_port();
int openLog(char *filename);
void saveToLog(int file, uint8_t *data, int len);

#endif
```

i2c_com.c

```
/*****************************************************************************
 * This application is used to access the i2c pins using the i2c device
 * "/dev/i2c-1".
 *
 * @author William McVicker
 *****************************************************************************/

#include <termios.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/signal.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <stdint.h>
#include <string.h>
#include <time.h>

#include "i2c-dev.h"
#include "i2c_com.h"

/** Reads from the file descriptor fd, which should be a file pointer
 * to a i2c port.
 * @param fd the file descriptor of the serial port to read from
 * @param data a pointer to a spot in memory to store the data
 * @param len the number of bytes to read from the file descriptor
 * @return the number of bytes read (len) if successful, else -1
 */
int i2c_read(int fd, char *data, uint16_t len) {
   struct i2c_rdwr_ioctl_data msgset;
   struct i2c_msg messages[1];

   messages[0].addr  = I2C_ADDR;
   messages[0].flags = I2C_M_RD;
   messages[0].len   = len;
   messages[0].buf   = data;

   msgset.nmsgs = 1;
   msgset.msgs  = messages;

   if (ioctl(fd, I2C_RDWR, &msgset) < 0) {
      perror("ioctl read");
      exit(EXIT_FAILURE);
   }

   return len;
}

/** Write len bytes of the data pointed at by data to the file descriptor fd,
 * which should be an open i2c port.
 * @param fd the file descriptor to write the data to
 * @param data the data to write to the file descriptor
 * @param len the number of bytes to write
 * @return the number of bytes written on success, else -1
 */
int i2c_write (int fd, char *data, int len) {
   struct i2c_rdwr_ioctl_data msgset;
   struct i2c_msg messages[1];

   messages[0].addr  = I2C_ADDR;
   messages[0].flags = 0; // write
   messages[0].len   = len;
   messages[0].buf   = data;

   msgset.nmsgs = 1;
   msgset.msgs  = messages;

   if (ioctl(fd, I2C_RDWR, &msgset) < 0) {
      perror("ioctl read");
      exit(EXIT_FAILURE);
   }

   return len;
}

/** The main thing. Initializes the I2C communication protocol
 * to read and write data from an A/D convert.  The data is logged in a
 * text to be read from another application.
 * @param argc number of input arguments
 * @param argv an array of the input arguments
 * @return 0 on success, else 1-255 on error
 */
int main (int argc, char *argv[]) {
   int i2c_fd, i;
   char data[1];
   int logfile;
   int counter = 0;

   /* open the i2c device for reading and writing */
   i2c_fd = initialize_port();
   logfile = openLog("voltage_log\0");
   sleep(30);

   while (1) {

      for (i = 0; i < 128; i++) {
         memset(data, 0, 1);

         /* Read from the i2c port */
         if (i2c_read(i2c_fd, data, 1) < 0) {
            /* failed to read data from the i2c port */
            fprintf(stderr, "Failed to read data from the "
                  "i2c port\n");
            exit(EXIT_FAILURE);
         }
      }

      // Write data to log file
      saveToLog(logfile, (uint8_t *) data, 1);

      if (++counter <= 5) {
         sleep(5);
      } else {
         counter = 0;
         sleep(300);
      }
   }

   close(i2c_fd);
   close(logfile);
   return EXIT_SUCCESS;
}

void saveToLog(int file, uint8_t *data, int len) {
   int wlen, i, wtot;
   char string[6];
   char *temp;
   char mtime[128];
   time_t currTime;

   for (i = 0; i < len; i++) {
      wtot = 0;
      memset(mtime, 0, 128);
      memset(string, 0, 6);

      // Get the current time
      currTime = time(NULL);
      if ((temp = ctime(&currTime)) == NULL) {
         perror("ctime");
         exit(EXIT_FAILURE);
      }
      strncpy(mtime, temp, strlen(temp)-1);
      mtime[strlen(temp)-1] = ',';
      if (snprintf(string, 5, "%hhu", data[i]) < 0) {
         perror("snprintf");
         exit(EXIT_FAILURE);
      }
      string[strlen(string)] = '\n';
      strcat(mtime, string);

      while ((wlen = write(file, &mtime[wtot], strlen(mtime)-wtot)) > 0) {
         wtot += wlen;
         if (wtot == strlen(mtime))
            break;
      }

      if (wlen == -1) {
         perror("write to log");
         exit(EXIT_FAILURE);
      }
   }
}

/** Opens the log file and returns its file descriptor
 * @param filename the name of the log file
 * @return the file descriptor of the log file
 */
int openLog(char *filename) {
   int file;

   file = open(filename, O_WRONLY | O_APPEND | O_CREAT);
   if (file < 0) {
      perror("open log");
      exit(EXIT_FAILURE);
   }

   return file;
}

/** Initialize the serial port to write to and returns a file
 * descriptor to it.
 * @return the file descriptor of the opened serial port
 */
int initialize_port() {
   int fd;
   char ctrl_byte = 0x00;

   /* open the serial port as non-blocking */
   fd = open(DEV_PORT, O_RDWR);
   if (fd < 0) {
      perror(DEV_PORT);
      exit(EXIT_FAILURE);
   }

   // Initialize the A/D Conveter to read from its analog input
   if (i2c_write(fd, &ctrl_byte, 1) < 0) {
      fprintf(stderr, "Failed to write the control byte to the A/D\n");
      exit(EXIT_FAILURE);
   }
   return fd;
}
```

i2c-dev.h

```
/*
    i2c-dev.h - i2c-bus driver, char device interface

    Copyright (C) 1995-97 Simon G. Vogl
    Copyright (C) 1998-99 Frodo Looijaard <frodol@dds.nl>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
*/

/* $Id$ */

#ifndef LIB_I2CDEV_H
#define LIB_I2CDEV_H

#include <linux/types.h>
#include <sys/ioctl.h>


/* -- i2c.h -- */


/*
 * I2C Message - used for pure i2c transaction, also from /dev interface
 */
struct i2c_msg {
       __u16 addr;   /* slave address                   */
       unsigned short flags;
#define I2C_M_TEN    0x10   /* we have a ten bit chip address  */
#define I2C_M_RD     0x01
#define I2C_M_NOSTART       0x4000
#define I2C_M_REV_DIR_ADDR  0x2000
#define I2C_M_IGNORE_NAK    0x1000
#define I2C_M_NO_RD_ACK           0x0800
       short len;           /* msg length                      */
       char *buf;           /* pointer to msg data                    */
};

/* To determine what functionality is present */

#define I2C_FUNC_I2C              0x00000001
#define I2C_FUNC_10BIT_ADDR       0x00000002
```

```
#define I2C_FUNC_PROTOCOL_MANGLING 0x00000004 /* I2C_M_{REV_DIR_ADDR,NOSTART,..} */
#define I2C_FUNC_SMBUS_PEC         0x00000008
#define I2C_FUNC_SMBUS_BLOCK_PROC_CALL    0x00008000 /* SMBus 2.0 */
#define I2C_FUNC_SMBUS_QUICK              0x00010000
#define I2C_FUNC_SMBUS_READ_BYTE   0x00020000
#define I2C_FUNC_SMBUS_WRITE_BYTE  0x00040000
#define I2C_FUNC_SMBUS_READ_BYTE_DATA     0x00080000
#define I2C_FUNC_SMBUS_WRITE_BYTE_DATA    0x00100000
#define I2C_FUNC_SMBUS_READ_WORD_DATA     0x00200000
#define I2C_FUNC_SMBUS_WRITE_WORD_DATA    0x00400000
#define I2C_FUNC_SMBUS_PROC_CALL   0x00800000
#define I2C_FUNC_SMBUS_READ_BLOCK_DATA    0x01000000
#define I2C_FUNC_SMBUS_WRITE_BLOCK_DATA 0x02000000
#define I2C_FUNC_SMBUS_READ_I2C_BLOCK     0x04000000 /* I2C-like block xfer  */
#define I2C_FUNC_SMBUS_WRITE_I2C_BLOCK    0x08000000 /* w/ 1-byte reg. addr. */
#define I2C_FUNC_SMBUS_READ_I2C_BLOCK_2    0x10000000 /* I2C-like block xfer  */
#define I2C_FUNC_SMBUS_WRITE_I2C_BLOCK_2 0x20000000 /* w/ 2-byte reg. addr. */

#define I2C_FUNC_SMBUS_BYTE (I2C_FUNC_SMBUS_READ_BYTE | \
                             I2C_FUNC_SMBUS_WRITE_BYTE)
#define I2C_FUNC_SMBUS_BYTE_DATA (I2C_FUNC_SMBUS_READ_BYTE_DATA | \
                                  I2C_FUNC_SMBUS_WRITE_BYTE_DATA)
#define I2C_FUNC_SMBUS_WORD_DATA (I2C_FUNC_SMBUS_READ_WORD_DATA | \
                                  I2C_FUNC_SMBUS_WRITE_WORD_DATA)
#define I2C_FUNC_SMBUS_BLOCK_DATA (I2C_FUNC_SMBUS_READ_BLOCK_DATA | \
                                   I2C_FUNC_SMBUS_WRITE_BLOCK_DATA)
#define I2C_FUNC_SMBUS_I2C_BLOCK (I2C_FUNC_SMBUS_READ_I2C_BLOCK | \
                                  I2C_FUNC_SMBUS_WRITE_I2C_BLOCK)
#define I2C_FUNC_SMBUS_I2C_BLOCK_2 (I2C_FUNC_SMBUS_READ_I2C_BLOCK_2 | \
                                    I2C_FUNC_SMBUS_WRITE_I2C_BLOCK_2)

/* Old name, for compatibility */
#define I2C_FUNC_SMBUS_HWPEC_CALC  I2C_FUNC_SMBUS_PEC

/*
 * Data for SMBus Messages
 */
#define I2C_SMBUS_BLOCK_MAX 32     /* As specified in SMBus standard */
#define I2C_SMBUS_I2C_BLOCK_MAX    32     /* Not specified but we use same structure */
union i2c_smbus_data {
       __u8 byte;
       __u16 word;
       __u8 block[I2C_SMBUS_BLOCK_MAX + 2]; /* block[0] is used for length */
                                                   /* and one more for PEC */
};

/* smbus_access read or write markers */
#define I2C_SMBUS_READ      1
#define I2C_SMBUS_WRITE     0

/* SMBus transaction types (size parameter in the above functions)
   Note: these no longer correspond to the (arbitrary) PIIX4 internal codes! */
#define I2C_SMBUS_QUICK                0
#define I2C_SMBUS_BYTE                 1
#define I2C_SMBUS_BYTE_DATA     2
#define I2C_SMBUS_WORD_DATA     3
#define I2C_SMBUS_PROC_CALL     4
#define I2C_SMBUS_BLOCK_DATA           5
```

```
#define I2C_SMBUS_I2C_BLOCK_BROKEN  6
#define I2C_SMBUS_BLOCK_PROC_CALL   7           /* SMBus 2.0 */
#define I2C_SMBUS_I2C_BLOCK_DATA    8


/* ----- commands for the ioctl like i2c_command call:
 * note that additional calls are defined in the algorithm and hw
 *     dependent layers - these can be listed here, or see the
 *     corresponding header files.
 */
                            /* -> bit-adapter specific ioctls  */
#define I2C_RETRIES  0x0701 /* number of times a device address      */
                            /* should be polled when not            */
                                /* acknowledging                      */
#define I2C_TIMEOUT  0x0702 /* set timeout - call with int            */


/* this is for i2c-dev.c    */
#define I2C_SLAVE    0x0703 /* Change slave address                   */
                            /* Attn.: Slave address is 7 or 10 bits */
#define I2C_SLAVE_FORCE     0x0706 /* Change slave address                  */
                            /* Attn.: Slave address is 7 or 10 bits */
                            /* This changes the address, even if it */
                            /* is already taken!              */
#define I2C_TENBIT   0x0704 /* 0 for 7 bit addrs, != 0 for 10 bit     */

#define I2C_FUNCS    0x0705 /* Get the adapter functionality */
#define I2C_RDWR     0x0707 /* Combined R/W transfer (one stop only)*/
#define I2C_PEC             0x0708 /* != 0 for SMBus PEC                   */

#define I2C_SMBUS    0x0720 /* SMBus-level access */

/* -- i2c.h -- */


/* Note: 10-bit addresses are NOT supported! */

/* This is the structure as used in the I2C_SMBUS ioctl call */
struct i2c_smbus_ioctl_data {
       char read_write;
       __u8 command;
       int size;
       union i2c_smbus_data *data;
};

/* This is the structure as used in the I2C_RDWR ioctl call */
struct i2c_rdwr_ioctl_data {
       struct i2c_msg *msgs;       /* pointers to i2c_msgs */
       int nmsgs;           /* number of i2c_msgs */
};


static inline __s32 i2c_smbus_access(int file, char read_write, __u8 command,
                                     int size, union i2c_smbus_data *data)
{
       struct i2c_smbus_ioctl_data args;

       args.read_write = read_write;
```

```
        args.command = command;
        args.size = size;
        args.data = data;
        return ioctl(file,I2C_SMBUS,&args);
}


static inline __s32 i2c_smbus_write_quick(int file, __u8 value)
{
        return i2c_smbus_access(file,value,0,I2C_SMBUS_QUICK,NULL);
}

static inline __s32 i2c_smbus_read_byte(int file)
{
        union i2c_smbus_data data;
        if (i2c_smbus_access(file,I2C_SMBUS_READ,0,I2C_SMBUS_BYTE,&data))
                return -1;
        else
                return 0x0FF & data.byte;
}

static inline __s32 i2c_smbus_write_byte(int file, __u8 value)
{
        return i2c_smbus_access(file,I2C_SMBUS_WRITE,value,
                               I2C_SMBUS_BYTE,NULL);
}

static inline __s32 i2c_smbus_read_byte_data(int file, __u8 command)
{
        union i2c_smbus_data data;
        if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
                            I2C_SMBUS_BYTE_DATA,&data))
                return -1;
        else
                return 0x0FF & data.byte;
}

static inline __s32 i2c_smbus_write_byte_data(int file, __u8 command,
                                              __u8 value)
{
        union i2c_smbus_data data;
        data.byte = value;
        return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                               I2C_SMBUS_BYTE_DATA, &data);
}

static inline __s32 i2c_smbus_read_word_data(int file, __u8 command)
{
        union i2c_smbus_data data;
        if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
                            I2C_SMBUS_WORD_DATA,&data))
                return -1;
        else
                return 0x0FFFF & data.word;
}

static inline __s32 i2c_smbus_write_word_data(int file, __u8 command,
                                              __u16 value)
```

```
{
       union i2c_smbus_data data;
       data.word = value;
       return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                               I2C_SMBUS_WORD_DATA, &data);
}

static inline __s32 i2c_smbus_process_call(int file, __u8 command, __u16 value)
{
       union i2c_smbus_data data;
       data.word = value;
       if (i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                            I2C_SMBUS_PROC_CALL,&data))
              return -1;
       else
              return 0x0FFFF & data.word;
}


/* Returns the number of read bytes */
static inline __s32 i2c_smbus_read_block_data(int file, __u8 command,
                                              __u8 *values)
{
       union i2c_smbus_data data;
       int i;
       if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
                            I2C_SMBUS_BLOCK_DATA,&data))
              return -1;
       else {
      printf("%d blocks\n", data.block[0]);
              for (i = 1; i <= data.block[0]; i++)
                     values[i-1] = data.block[i];
              return data.block[0];
       }
}

static inline __s32 i2c_smbus_write_block_data(int file, __u8 command,
                                               __u8 length, __u8 *values)
{
       union i2c_smbus_data data;
       int i;
       if (length > 32)
              length = 32;
       for (i = 1; i <= length; i++)
              data.block[i] = values[i-1];
       data.block[0] = length;
       return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                               I2C_SMBUS_BLOCK_DATA, &data);
}

/* Returns the number of read bytes */
/* Until kernel 2.6.22, the length is hardcoded to 32 bytes. If you
   ask for less than 32 bytes, your code will only work with kernels
   2.6.23 and later. */
static inline __s32 i2c_smbus_read_i2c_block_data(int file, __u8 command,
                                                  __u8 length, __u8 *values)
{
       union i2c_smbus_data data;
```

```
        int i;

        if (length > 32)
                length = 32;
        data.block[0] = length;
        if (i2c_smbus_access(file,I2C_SMBUS_READ,command,
                                 length == 32 ? I2C_SMBUS_I2C_BLOCK_BROKEN :
                                  I2C_SMBUS_I2C_BLOCK_DATA,&data))
                return -1;
        else {
                for (i = 1; i <= data.block[0]; i++)
                        values[i-1] = data.block[i];
                return data.block[0];
        }
}

static inline __s32 i2c_smbus_write_i2c_block_data(int file, __u8 command,
                                                       __u8 length, __u8 *values)
{
        union i2c_smbus_data data;
        int i;
        if (length > 32)
                length = 32;
        for (i = 1; i <= length; i++)
                data.block[i] = values[i-1];
        data.block[0] = length;
        return i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                                    I2C_SMBUS_I2C_BLOCK_BROKEN, &data);
}

/* Returns the number of read bytes */
static inline __s32 i2c_smbus_block_process_call(int file, __u8 command,
                                                         __u8 length, __u8 *values)
{
        union i2c_smbus_data data;
        int i;
        if (length > 32)
                length = 32;
        for (i = 1; i <= length; i++)
                data.block[i] = values[i-1];
        data.block[0] = length;
        if (i2c_smbus_access(file,I2C_SMBUS_WRITE,command,
                                 I2C_SMBUS_BLOCK_PROC_CALL,&data))
                return -1;
        else {
                for (i = 1; i <= data.block[0]; i++)
                        values[i-1] = data.block[i];
                return data.block[0];
        }
}


#endif /* LIB_I2CDEV_H */
```

Appendix N: Server Daemon Code

Makefile:

```
#
# myapps/fserver/Makefile
#

.PHONY: build chkconfig preconfig buildfs install clean


ifeq ($(DEVDIR),)
$(error ====== DEVDIR variable is empty, invoke this Makefile from the BSP root, or
provide the path to it =====)
endif
include $(DEVDIR)/bsp/mach/Make.conf

BIN                  = fserver

SRCS                 = fserver.c util.c checksum.c
HDRS        = fserver.h common.h checksum.h
OBJS                 = $(SRCS:.c=.o)

CFLAGS += $(EXTRA_CFLAGS) -O2 -Wall -g

ifeq ($(CONFIG_USER_APPS_FSERVER),y)
build:
       $(V)$(CC) -c $(CFLAGS) -O2 $(SRCS) -o $(OBJS) $(QOUT)
       $(V)$(CC) $(LDFLAGS) -O2 -o $(BIN) $(OBJS) $(QOUT)

sim:

chkconfig:

preconfig:

buildfs:

install:
       $(V)install -D -m 755 $(BIN) $(FSROOT)/examples/$(BIN) $(QOUT)

clean:
       $(V)rm -f $(BIN) *.debug *.o core *~ $(QOUT)
else
build:
sim:
chkconfig:
preconfig:
buildfs:
install:
clean:
endif

fserver: fserver.o util.o checksum.o
       $(V)$(CC) $(LDFLAGS)  -o $@ $^

fserver.o: fserver.c util.c $(HDRS)
```

```
	$(V)$(CC) $(CFLAGS) -c $<

util.o: util.c
	$(V)$(CC) $(CFLAGS) -c $<

checksum.o: checksum.c checksum.h
	$(V)$(CC) $(CFLAGS) -c $<
```

common.h

```
#ifndef _COMMON_H
#define _COMMON_H

#include <stdlib.h>
#include <stdint.h>

#define BUF_SIZE 1400

typedef enum {dpacket, cpacket} packetTypes;

#pragma pack(push, 1) /* set current alignment to 1 bytes */
struct data_packet {
   uint32_t sequenceNum;
   uint16_t ichecksum;
   int32_t errFlag;        /* if this is not 0, then error msg */
   uint16_t data_size;     /* actual amount of data in packet */
   uint8_t data[BUF_SIZE]; /* UDP header and data */
};

struct client_packet {
   uint32_t sequenceNum;
   uint16_t ichecksum;
   size_t packet_size;
   uint8_t ack;            /* 1->Reg ACK, 2->Error Msg ACK  3->last ACK */
   char filename[1024];
};

#pragma pack(pop)  /* restore original alignment from stack */

#endif
```

fserver.h

```
/** This is a server application that uses UDP to remotely transfer files.
 * The server waits for incoming requests by clients and transmits the
 * requested file back to the client.
 * Error handling was implemented using the Stop and Wait algorithm
 *
 * CPE 464 -- Programming Assignment #2
 * @author William McVicker wmcvicke@calpoly.edu
 */

#ifndef _SERVER_H
#define _SERVER_H

#include <netinet/in.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>
#include <stdint.h>

#include "common.h"

#define MAX_SRV_TRIES   5

struct initChild {
   int socket_fd;
   int file_fd;
   int errFlag;
};

void server_child(struct client_packet *clientPacket,
      struct sockaddr *client_addr, socklen_t client_addr_len);
int init_server();
struct initChild initializeChild(char *filename);
int sendDataPacket(int socket_fd, struct data_packet dataPacket,
      uint16_t packetSize, struct sockaddr *client_addr,
      socklen_t client_addr_len);
void buildDataPacket(int fd, struct data_packet *thePacket,
      uint16_t packetSize, uint32_t seqNum, int errFlag);
int getPacket(struct client_packet *clientPacket,
      struct sockaddr *client_addr, socklen_t *client_addr_len,
      int socket_num, int file_fd);
int handleChildren(pid_t *children, int numChildren);

/* Util functions */
extern int select_call(int socket, int seconds, int useconds);

#endif
```

fserver.c:

```
/** This is a server application that uses UDP to remotely transfer files.
 * The server waits for incoming requests by clients and transmits the
 * requested file back to the client.
 *
 * Error handling was implemented using the Stop and Wait algorithm
 *
 * CPE 464 -- Programming Assignment #2
 * @author William McVicker wmcvicke@calpoly.edu
 */

#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>

#include "fserver.h"
#include "checksum.h"

#define EXIT_SAFELY(X, Y, Z) {close(X); close(Y); exit(Z);}

/** The main server routine that accepts all the new incoming requests.
 *
 * @param argc the number of input arguments
 * @param argv an array of input arguments as char *'s
 * @return EXIT_FAILURE
 */
int main(int argc, char *argv[]) {
   int socket_num, numChildren = 0;
   pid_t childPIDs[1024];
   struct client_packet clientPacket;
   struct sockaddr mySocket, client_addr;
   socklen_t mySocket_len = sizeof(struct sockaddr);
   socklen_t client_addr_len = sizeof(struct sockaddr);

   /* Setup the socket and get the port address */
   if ((socket_num = init_server()) < 0) {
      fprintf(stderr, "Failed to setup socket\n");
      exit(EXIT_FAILURE);
   }
   if (getsockname(socket_num, &mySocket, &mySocket_len) < 0) {
      perror("getsockname call");
      close(socket_num);
      exit(EXIT_FAILURE);
   }
   printf("Server is using port %d\n", ntohs(((struct sockaddr_in *)
       &mySocket)->sin_port));

   while (1) {
      for (;;) {
         if (select_call(socket_num, 1, 0)) {
            if(getPacket(&clientPacket, &client_addr, &client_addr_len,
                  socket_num, -1))
               break;
         }
         numChildren = handleChildren(childPIDs, numChildren);
      }

      if (ntohl(clientPacket.sequenceNum) == 0 && clientPacket.ack == 0) {
         /* Fork a child to handle the request */
         if ((childPIDs[numChildren++] = fork()) == -1) {
            perror("forking child");
            exit(EXIT_FAILURE);
         } else if (childPIDs[numChildren-1] == 0) { /* child process */
            /* handle the transmission of data to client */
            close(socket_num);

            server_child(&clientPacket, &client_addr,
                  client_addr_len);
            exit(EXIT_SUCCESS);
         }
      }
   }

   return EXIT_FAILURE;  /* server should never exit */
}

/** This is a child process which will handle all data transmission between
 * the server and the client.  It opens its own socket to handle the
 * communication with the client.
 *
 * @param clientPacket the packet from the client
 * @param client_addr the client's address
 * @param client_addr_len the length of the client's address
 */
void server_child(struct client_packet *clientPacket,
      struct sockaddr *client_addr, socklen_t client_addr_len) {
   int socket_fd, file_fd, errFlag = 0;
   struct data_packet dataPacket;
   struct initChild initParams;
   uint16_t dataPacketSize, currentSeqNum = 0;

   /* setup the child's socket and open the file to be copied */
   initParams = initializeChild(clientPacket->filename);
   socket_fd = initParams.socket_fd;
   file_fd   = initParams.file_fd;
   errFlag   = initParams.errFlag;

   /* build the first packet */
   dataPacketSize = clientPacket->packet_size;
   buildDataPacket(file_fd, &dataPacket, dataPacketSize, currentSeqNum, errFlag);

   while (1) {
      /* send a data packet and wait till client is ready to receive data */
      int results = sendDataPacket(socket_fd, dataPacket, dataPacketSize,
            client_addr, client_addr_len);
      if (results != 0) {
         if (results == -1) /* system call error */
            EXIT_SAFELY(file_fd, socket_fd, EXIT_FAILURE);
         EXIT_SAFELY(file_fd, socket_fd, EXIT_SUCCESS);
      }

      if(!getPacket(clientPacket, client_addr,
               &client_addr_len, socket_fd, file_fd))
         continue;

      if ((ntohl(clientPacket->sequenceNum) == currentSeqNum) &&
            (clientPacket->ack == 1)) { /* found valid ACK */
         currentSeqNum++;

         if (!dataPacket.errFlag && dataPacket.data_size == 0) {
            /* no more data to transmit */
            EXIT_SAFELY(file_fd, socket_fd, EXIT_SUCCESS);
         } else {
            buildDataPacket(file_fd, &dataPacket, dataPacketSize, currentSeqNum, 0);
         }
      } else if ((ntohl(clientPacket->sequenceNum) == 0) &&
            (clientPacket->ack == 2)) {
         close(socket_fd); /* found ACK to the error message */
         return;
      }
   }
}

/** Initializes the server by opening a socket and naming it.
 *
 * @return the port number of the bound socket, or -1 on error
 */
int init_server() {
   int socket_num;
   struct sockaddr_in address;

   /* open the socket */
   if ((socket_num = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
      perror("socket call");
      return -1;
   }

   int opt = 1; /* reuse the port number */
   if (setsockopt(socket_num, SOL_SOCKET, SO_REUSEADDR,
         (char *) &opt, sizeof(opt)) < 0) {
      perror("setsockopt");
      return -1;
   }

   /* name the socket */
   address.sin_family = AF_INET;           /* Internet */
   address.sin_addr.s_addr = INADDR_ANY;   /* Use local IP Address */
   address.sin_port = htons(7777);

   /* bind the socket to a port */
   if (bind(socket_num, (struct sockaddr *) &address, sizeof(address)) < 0) {
      perror("bind call");
      close(socket_num);
      return -1;
   }

   return socket_num;
}

/** Builds a packet to be transmitted.  If the value of errFlag is
 * something other than 0, then it builds the packet as an error message to the
 * client.  Otherwise, the packet is built with the reqested data to the size
 * indicated by packetSize.
 *
 * @param fd the file descriptor to get the data from
 * @param packetSize the size of the entire packet
 * @param seqNum the sequence number of this packet
 * @param errFlag the errno if an error occurred when opening the file
 */
void buildDataPacket(int fd, struct data_packet *thePacket,
      uint16_t dataSize, uint32_t seqNum, int errFlag) {
   ssize_t rdata, rdata_tot = 0;
   uint8_t buf[BUF_SIZE];

   memset(thePacket, 0, sizeof(struct data_packet));
   memset(buf, 0, BUF_SIZE);

   if (errFlag) {
      /* need to send an error flag to the client */
      thePacket->errFlag = errFlag;
   } else {
      /* build regular data packet */
      while ((rdata = read(fd, &buf[rdata_tot], dataSize)) >= 0) {
         if (rdata == 0) {
            /* reached eof */
            break;
         }
         dataSize -= rdata;
         rdata_tot += rdata;
      }

      if (rdata < 0) {
         /* error reading from the file */
         thePacket->errFlag = errno;
         perror("read from file");
      } else {
         thePacket->sequenceNum = htonl(seqNum);
         thePacket->data_size   = rdata_tot;
         memcpy(thePacket->data, buf, rdata_tot);
      }
   }

   /* Calculate the checksum for the packet */
   thePacket->ichecksum = in_cksum((uint16_t *) thePacket,
         sizeof(struct data_packet));
}

/** Send a data packet to the client
 *
 * @param socket_fd the socket file descriptor
 * @param dataPacket the packet to send
 * @param packetSize the size of the packet
 * @param client_addr the client to send the packet to
 * @param client_addr_len the size of the client source address
 * @return 0 on success, 1 when client doesn't respond after 5 tries,
 * and -1 on error.
 */
int sendDataPacket(int socket_fd, struct data_packet dataPacket,
      uint16_t packetSize, struct sockaddr *client_addr,
      socklen_t client_addr_len) {
   int i;
   int transmitInterval = 1;

   for (i = 0 ;; i++) {
      if (sendto(socket_fd, (void *) &dataPacket,
               sizeof(struct data_packet), 0, client_addr,
               client_addr_len) != sizeof(struct data_packet)) {
         perror("server child: Error sending packet");
         return -1;
      }

      if (i >= 5) {
         /* client not responding anymore */
         return 1;
      } else if (select_call(socket_fd, transmitInterval, 0)) {
         break; /* ready for reading */
      }
   }

   return 0;
}

/** Opens a socket for the child and the file to be transferred
 * to the client.  If the file is unable to be opened, the errno
```

```c
 * is saved in order to be transferred to the client in error
 * message packet.
 *
 * @param filename the name of the file to open
 * @return a struct which contains the socket fd, the file fd,
 * and the errno related to the open command.
 */
struct initChild initializeChild(char *filename) {
   int socket_fd, file_fd, errFlag = 0;
   struct initChild initParams;

   /* Open a new socket to communicate with the client */
   if ((socket_fd = init_server()) < 0) {
      fprintf(stderr, "Failed to setup socket\n");
      exit(EXIT_FAILURE);
   }
   errno = 0;

   file_fd = open(filename, O_RDONLY);
   if (file_fd == -1) {
      /* error with file I/O */
      errFlag = errno;
   }

   initParams.socket_fd = socket_fd;
   initParams.file_fd   = file_fd;
   initParams.errFlag   = errFlag;
   return initParams;
}

/** Get a packet from the client
 *
 * @param clientPacket the address to save the packet at
 * @param client_addr the address of the client
 * @param client_addr_len the length of the address
 * @param socket_num the socket to listen on
 * @param file_fd the file fd to close if there is an error
 * @return 1 if a valid packet was received, else 0
 */
int getPacket(struct client_packet *clientPacket,
      struct sockaddr *client_addr, socklen_t *client_addr_len,
      int socket_num, int file_fd) {

   if (recvfrom(socket_num, (void *) clientPacket,
            sizeof(struct client_packet), 0,  client_addr,
            client_addr_len) < 0) {
      perror("recvfrom call");
      close(socket_num);
      if (file_fd >= 0)
         close(file_fd);
      exit(EXIT_FAILURE);
   }

   if (in_cksum((uint16_t *) clientPacket, sizeof(struct client_packet))) {
      /* This packet is corrupt */
      return 0;
   }
   return 1;
}

/** Checks if there are any children waiting for the parent
 *
 * @param children an array of chilren to check
 * @param numChildren size of array
 */
int handleChildren(pid_t *children, int numChildren) {
   int status, j=0, i;
   int foundChildren = 0;
   pid_t newChildren[1024];

   if (numChildren == 0)
      return foundChildren;

   for (i = 0; i < numChildren; i++) {
      int retval = waitpid(children[i], &status, WNOHANG);
      if (retval == -1) {
         perror("waitpid");
         exit(EXIT_FAILURE);
      } else if (retval > 0) {
         /* found a child */
         foundChildren++;
      } else {
         newChildren[j++] = children[i];
      }
   }
   memcpy(children, newChildren, 1024);

   return j;
}
```

util.c:

```
/** These are utility function used by both the server and client.
 *
 * CPE 464 - Programming Assignment #2
 * @author William McVicker wmcvicke@calpoly.edu
 */

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>

#include "common.h"

int select_call(int socket, int seconds, int useconds);

/** This function calls select(...) which monitors the requested socket
 * waiting till it is ready to be read from.  It waits for the requested amount
 * of time and returns a value indicated if the socket is ready to be read.
 *
 * @param socket the socket to be read from
 * @param seconds the number of seconds to probe the socket
 * @param useconds the number of microseconds to probe the socket
 * @return 1 if the socket is ready for read, else 0
 */
```

```c
int select_call(int socket, int seconds, int useconds) {

   fd_set rfds;
   struct timeval tv;
   int retval;

   /* watch the socket to see when to read in a packet */
   FD_ZERO(&rfds);
   FD_SET(socket, &rfds);

   tv.tv_sec = seconds;
   tv.tv_usec = useconds;

   retval = select(socket+1, &rfds, NULL, NULL, &tv);

   if (retval == -1) {
      perror("select");
      return 0;
   }

   return retval;
}
```

checksum.h:

```c
/* Checksum declaration
 * shadows@whitefang.com
 */

unsigned short in_cksum(unsigned short *addr,int len);
```

checksum.c:

```c
/*
 * Copyright (c) 1989, 1993
 *      The Regents of the University of California.  All rights reserved.
 *
 * This code is derived from software contributed to Berkeley by
 * Mike Muuss.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *      This product includes software developed by the University of
 *      California, Berkeley and its contributors.
 * 4. Neither the name of the University nor the names of its contributors
 *    may be used to endorse or promote products derived from this software
 *    without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 */


#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>

/*
 * in_cksum --
 *      Checksum routine for Internet Protocol family headers (C Version)
 */
unsigned short in_cksum(unsigned short *addr,int len)
{
        register int sum = 0;
        u_short answer = 0;
        register u_short *w = addr;
        register int nleft = len;

        /*
         * Our algorithm is simple, using a 32 bit accumulator (sum), we add
         * sequential 16 bit words to it, and at the end, fold back all the
         * carry bits from the top 16 bits into the lower 16 bits.
         */
        while (nleft > 1)  {
                sum += *w++;
                nleft -= 2;
        }

        /* mop up an odd byte, if necessary */
        if (nleft == 1) {
                *(u_char *)(&answer) = *(u_char *)w ;
                sum += answer;
        }

        /* add back carry outs from top 16 bits to low 16 bits */
        sum = (sum >> 16) + (sum & 0xffff);     /* add hi 16 to low 16 */
        sum += (sum >> 16);                     /* add carry */
        answer = ~sum;                          /* truncate to 16 bits */
        return(answer);
}
```

Appendix O: Battery Monitoring Client Application Code

ClientGUI.cpp:

```
// ClientGUI.cpp : main project file.

#include "stdafx.h"
#include <iostream>

#include "Form1.h"

#include "client.h"

using namespace System::IO;
using namespace ClientGUI;

[STAThreadAttribute]
int main(array<String ^> ^args)
{
        // Enabling Windows XP visual effects before any controls are created
        Application::EnableVisualStyles();
        Application::SetCompatibleTextRenderingDefault(false);

        // Create the main window and run it
        Application::Run(gcnew Form1());
        return 0;
}
```

client.h:

```
/** This is a client application which communicates to a server through the
 * UDP protocol.  This application adds error handling to the UDP protocol
 * in order to guarrantee successful transmittion of all the data.
 *
 * Error handling is implemented using the Stop and Wait algorithm.
 *
 * CPE 464 - Programming Assignment #2
 * @author William McVicker wmcvicke@calpoly.edu
 */

#ifndef _CLIENT_H
#define _CLIENT_H

#include "stdafx.h"

using namespace System;
using namespace System::IO;

#define WIN32_LEAN_AND_MEAN

#include <Windows.h>

#include <WinSock2.h>
#include <Ws2tcpip.h>

#include <stdlib.h>
#include <stdio.h>
#include <fstream>
```

```
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdint.h>
#include <iostream>

using namespace System;
using namespace System::Runtime::InteropServices; // for class Marshal
using namespace std;

#pragma comment(lib, "Ws2_32.lib")

#define PACKET_SIZE 1400

typedef enum {dpacket, cpacket} packetTypes;

#pragma pack(push, 1) /* set current alignment to 1 bytes */
struct data_packet {
   uint32_t sequenceNum;
   uint16_t ichecksum;
   int32_t errFlag;        /* if this is not 0, then error msg */
   uint16_t data_size;     /* actual amount of data in packet */
   uint8_t data[PACKET_SIZE]; /* UDP header and data */
};

struct client_packet {
   uint32_t sequenceNum;
   uint16_t ichecksum;
   size_t packet_size;
   uint8_t ack;            /* 1->Reg ACK, 2->Error Msg ACK  3->last ACK */
   char filename[1024];
};

#pragma pack(pop)  /* restore original alignment from stack */

bool getFile(String^ remote_machine, int serverPort);
SOCKET init_client();
struct client_packet *buildPacket(struct client_packet *thePacket,
      char *remote_filename, uint32_t sequenceNum, uint8_t ack);
int sendClientPacket(SOCKET socket_fd, struct client_packet *thePacket,
      int packetSize, struct sockaddr *address, int address_len);
int receivePackets(SOCKET socket_fd, char *local_filename, char *remote_filename);
int waitOnServer(int sec, int usec, SOCKET socket_fd, FileStream^ myfile);
int saveToFile(BinaryWriter^ myfile, uint8_t *data, uint16_t len);
int getPacket(struct data_packet *thePacket, struct sockaddr *server_addr,
      int *server_addr_len, SOCKET socket_fd, FileStream^ myfile,
      uint16_t currentSeqNum);

int select_call(int socket, int seconds, int useconds);
unsigned short in_cksum(unsigned short *addr,int len);

double updateVoltage(String ^filename);

#endif
```

client_win.cpp:

```
/** This is a client application which communicates to a server through the
 * UDP protocol.  This application adds error handling to the UDP protocol
 * in order to guarrantee successful transmittion of all the data.
 *
 * Error handling is implemented using the Stop and Wait algorithm.
 *
 * CPE 464 - Programming Assignment #2
 * @author William McVicker wmcvicke@calpoly.edu
 */
#include "stdafx.h"

#include "client.h"

using namespace System;
using namespace System::IO;

bool getFile(String^ remote_machine, int serverPort) {
   SOCKET socket_num;
   char *server_ip;
   char local_filename[] = "voltage_log\0";
   char remote_filename[] = "/voltage_log\0";
   struct client_packet thePacket;
   struct sockaddr_in address;
   struct hostent *theHost;
   int retval;

   socket_num = init_client();

   if (buildPacket(&thePacket, remote_filename, 0, 0) == NULL) {
         closesocket(socket_num);
      WSACleanup();
         Windows::Forms::MessageBox::Show("Failed to build initial packet!");
         return false;
   }

   server_ip = (char *)(void *) Marshal::StringToHGlobalAnsi(remote_machine);
   theHost = gethostbyname(server_ip);
   Marshal::FreeHGlobal((System::IntPtr)(void *) server_ip);
   if (!theHost) {
          int error_number = WSAGetLastError();
          printf("Error at socket(): %ld\n", error_number);
          return false;
   }
   memcpy(&address.sin_addr, theHost->h_addr, theHost->h_length);

   address.sin_family = AF_INET;
   address.sin_port = htons(serverPort);
   if (sendClientPacket(socket_num, &thePacket, sizeof(thePacket),
            (struct sockaddr *) &address, sizeof(address)) != 0) {
         closesocket(socket_num);
      WSACleanup();
         Windows::Forms::MessageBox::Show("Failed to send initial client packet!");
         return false;
   }

   retval = receivePackets(socket_num, local_filename, remote_filename);

   if (retval == 0) {
          //Windows::Forms::MessageBox::Show("Successfully downloaded log file");
          return true;
   } else if (retval == 1) {
          Windows::Forms::MessageBox::Show("Failed to download log!");
   } else {
          char error_string[1024];

          strerror_s(error_string, 1024, retval-1);
          Windows::Forms::MessageBox::Show(gcnew String(error_string));
   }

   return false;
}

/** Recieves all the data packets from the server
 *
 * @param socket_fd the socket to receive the packets from
 * @return a value of 0 on success, else 1
 */
int receivePackets(SOCKET socket_fd, char *local_filename, char *remote_filename) {
   struct data_packet thePacket;
   struct client_packet ackPacket;
   struct sockaddr server_addr;
   uint16_t currentSeqNum = 0;
   int server_addr_len = sizeof(struct sockaddr);
   FileStream^ myfile;
   BinaryWriter^ bw;
   int retval = 0;

   while (1) {
      if (!getPacket(&thePacket, &server_addr, &server_addr_len, socket_fd,
               myfile, currentSeqNum))
         continue; /* invalid packet */

      if (ntohl(thePacket.sequenceNum) == currentSeqNum) {
         uint8_t ackVal = 1;

         /* check if its an error message from the server */
         if (thePacket.errFlag) {
            currentSeqNum = 0;
            ackVal = 2; /* error message ack */
         } else {
                     if (!myfile) {
                        myfile = gcnew FileStream(gcnew String(local_filename),
FileMode::Create);
                        bw = gcnew BinaryWriter(myfile);
                     }
            if (saveToFile(bw, thePacket.data, thePacket.data_size) < 0) {
                        retval = 1;
                        break;
                     }
         }

         if (buildPacket(&ackPacket, remote_filename, currentSeqNum++,
             ackVal) == NULL) {
            retval = 1;
                     break;
               }
      }

      if (thePacket.data_size == 0 && !thePacket.errFlag)
         ackPacket.ack = 3; /* last packet */

      /* send ACK and wait till client is ready to receive data */
      int serverResponse;
      if ((serverResponse = sendClientPacket(socket_fd, &ackPacket,
                  sizeof(ackPacket), &server_addr, server_addr_len)) != 0) {
         if (serverResponse < 0) {
            retval = 1;
                     break;
               } else if (thePacket.errFlag) {
                      retval = thePacket.errFlag + 1;
                      break;
         } else {
            break;
         }
      }
   }

   if (myfile)
          myfile->Close();
   closesocket(socket_fd);
   WSACleanup();
   return retval;
}

/** Probes the supplied socket for the requested amount of time or until
 * the socket is ready to be read from.
 *
 * @param sec the number of seconds to probe the socket for
 * @param usec the number of microseconds to probe the socket for
 * @param socket_fd the socket fd to probe
 * @param myfile output file stream (closed if select_call fails)
 * @return 0 on success, else 1
 */
int waitOnServer(int sec, int usec, SOCKET socket_fd, FileStream^ myfile) {

   if (!select_call(socket_fd, sec, usec)) {
         closesocket(socket_fd);
         myfile->Close();
      WSACleanup();
         return 1;
   }

   return 0;
}

/** Writes the first len bytes of "data" to the requested file
 *
 * @param myfile output file stream to write to
 * @param data the data to write
 * @param the number of bytes to write to the file
 * @return the number of bytes written to the file, and -1 on error
 */
int saveToFile(BinaryWriter^ myfile, uint8_t *data, uint16_t len) {
   unsigned int data_len = len;
   array<unsigned char>^ data_array = gcnew array<unsigned char>((int) data_len);

   for (int i = 0; i < (int) data_len; i++)
          data_array[i] = data[i];

   myfile->Write(data_array, 0, (int) data_len);

   return data_len;
}

/** Initializes the client by setting up a socket and connecting to the
 * server through the requested port number.
 *
 * @return the socket fd on success, else -1
 */
SOCKET init_client() {
   SOCKET socket_num = INVALID_SOCKET;
   int iResult;
   WSADATA wsaData;

   iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
   if (iResult != NO_ERROR) {
          printf("WSAStartup failed with error: %d\n", iResult);
          exit(EXIT_FAILURE);
   }

   /* open the socket */
   socket_num = socket(AF_INET, SOCK_DGRAM, 0);
   if (socket_num == INVALID_SOCKET) {
      printf("Error at socket(): %ld\n", WSAGetLastError());
         WSACleanup();
         exit(EXIT_FAILURE);
   }

   return socket_num;
}

/** Build the packet to send to the server
 *
 * @param thePacket the packet to send
 * @param remote_filename the name of the requested file on the server
 * @param sequenceNum the sequence number of this packet
 * @param ack the type of ACK for this packet (if it's an ack)
 * @return the packet to send, NULL on error
 */
struct client_packet *buildPacket(struct client_packet *thePacket,
      char *remote_filename, uint32_t sequenceNum, uint8_t ack) {

   thePacket->sequenceNum = htonl(sequenceNum);
   thePacket->ack         = ack;
   thePacket->packet_size = PACKET_SIZE;
   thePacket->ichecksum   = 0;
   memset(thePacket->filename, 0, 1024);
   if (strcpy(thePacket->filename, remote_filename) == NULL) {
      fprintf(stderr, "Unrecognized filename format\n");
      return NULL;
   }
   /* calculate the internet checksum */
   thePacket->ichecksum = in_cksum((unsigned short *) thePacket,
         sizeof(struct client_packet));

   return thePacket;
}

/** Sends a client packet to the specified server.
 *
 * @param socket_fd the socket to send the packet across
 * @param thePacket the packet to send
 * @param packetSize the size of the packet
 * @param address the socket address
 * @param address_len the size of the socket address
 * @return 0 on success, 1 if the server is done, else -1 on error
 */
int sendClientPacket(SOCKET socket_fd, struct client_packet *thePacket,
      int packetSize, struct sockaddr *address, int address_len) {
   int i, sendOnce = 0;

   if (thePacket->ack > 1) {
      sendOnce = 1;
      if (thePacket->ack == 3) /* this is the last packet */
         thePacket->ack = 1;
   }

   for (i = 0;; i++) {
      if (sendto(socket_fd, (char *) thePacket, packetSize, 0, address,
               address_len) != packetSize) {
         printf("sendto(...) failed with error: %d\n", WSAGetLastError());
               closesocket(socket_fd);
               WSACleanup();
         return -1;
      }

      /* if the packet is an ACK to an eror message or the last packet,
       * then just send once */
      if (sendOnce)
         return 1;

      if (i == 10) {
         /* server not responding */
         return 1;
      } else if (select_call(socket_fd, 1, 0)) {
         break; /* ready for reading */
      }
   }

   return 0;
}

/** Gets a packet from the server and verifies that it is a valid packet.
 *
 * @param thePacket the address to save the packet at
 * @param server_addr the address of the server to get the packet from
 * @param server_addr_len the length of the server address
 * @param socket_fd the socket fd to communicate across
 * @param myfile the local file to close if an error occurs
 * @return 1 if the packet is valid and in sequence or older than expected,
 *  else return 0 if the packet is newer than expected or from unknown server
 */
int getPacket(struct data_packet *thePacket, struct sockaddr *server_addr,
      int *server_addr_len, SOCKET socket_fd, FileStream^ myfile,
      uint16_t currentSeqNum) {
   static uint16_t server_port = 0;

   if (recvfrom(socket_fd, (char *) thePacket, sizeof(struct data_packet),
            0, server_addr, server_addr_len) < 0) {
      printf("recvfrom(...) failed with error %d\n", WSAGetLastError());
         closesocket(socket_fd);
         myfile->Close();
      WSACleanup();
         return 0;
   }

   if (currentSeqNum == 0) {
      /* only talk to this server */
      server_port = ((struct sockaddr_in *) server_addr)->sin_port;
   } else {
      /* make sure this packet is from the correct server */
      if (server_port != ((struct sockaddr_in *) server_addr)->sin_port) {
         waitOnServer(10, 0, socket_fd, myfile);
         return 0;
      }
   }

   if (in_cksum((unsigned short *) thePacket, sizeof(struct data_packet)) != 0) {
      /* corrupt packet -- don't acknowledge packet */
      waitOnServer(10, 0, socket_fd, myfile);
      return 0;
   }

   if (ntohl(thePacket->sequenceNum) > currentSeqNum) {
      waitOnServer(10, 0, socket_fd, myfile);
      return 0;
   }

   return 1;
}

/** This function calls select(...) which monitors the requested socket
 * waiting till it is ready to be read from.  It waits for the requested amount
 * of time and returns a value indicated if the socket is ready to be read.
 *
 * @param socket the socket to be read from
 * @param seconds the number of seconds to probe the socket
 * @param useconds the number of microseconds to probe the socket
 * @return 1 if the socket is ready for read, else 0
 */
int select_call(int socket, int seconds, int useconds) {

   fd_set rfds;
   struct timeval tv;
   int retval;

   /* watch the socket to see when to read in a packet */
   FD_ZERO(&rfds);
   FD_SET(socket, &rfds);

   tv.tv_sec = seconds;
   tv.tv_usec = useconds;

   retval = select(socket+1, &rfds, NULL, NULL, &tv);

   if (retval == -1) {
      perror("select");
      return 0;
   }

   return retval;
}

/*
 * in_cksum --
 *      Checksum routine for Internet Protocol family headers (C Version)
 */
unsigned short in_cksum(unsigned short *addr,int len) {
    register int sum = 0;
    unsigned short answer = 0;
    register unsigned short *w = addr;
    register int nleft = len;

    /*
        * Our algorithm is simple, using a 32 bit accumulator (sum), we add
        * sequential 16 bit words to it, and at the end, fold back all the
        * carry bits from the top 16 bits into the lower 16 bits.
        */
    while (nleft > 1)  {
            sum += *w++;
            nleft -= 2;
    }

    /* mop up an odd byte, if necessary */
    if (nleft == 1) {
            *(unsigned char *)(&answer) = *(unsigned char *)w ;
            sum += answer;
    }

    /* add back carry outs from top 16 bits to low 16 bits */
    sum = (sum >> 16) + (sum & 0xffff);     /* add hi 16 to low 16 */
    sum += (sum >> 16);                      /* add carry */
    answer = ~sum;                           /* truncate to 16 bits */
    return(answer);
}

double updateVoltage(String^ filename) {
       String^ voltageValue;
       StreamReader^ sr;
       int dataValue;
       double actualVoltage = -1.0;

       try {
              FileStream^ fs = gcnew FileStream(L"voltage_log", FileMode::Open);
              fs->Seek(-10, SeekOrigin::End);
              sr = gcnew StreamReader(fs);
              String^ temp = sr->ReadLine();

              voltageValue = temp->Substring(temp->LastIndexOf(',')+1);

              fs->Close();
              dataValue = Convert::ToInt32(voltageValue);
              actualVoltage = 0.01308*(double) dataValue + 11.20383;
       } catch (Exception^ e) {
              voltageValue = "-1";
       }

       return actualVoltage;
}
```

Form1.h:

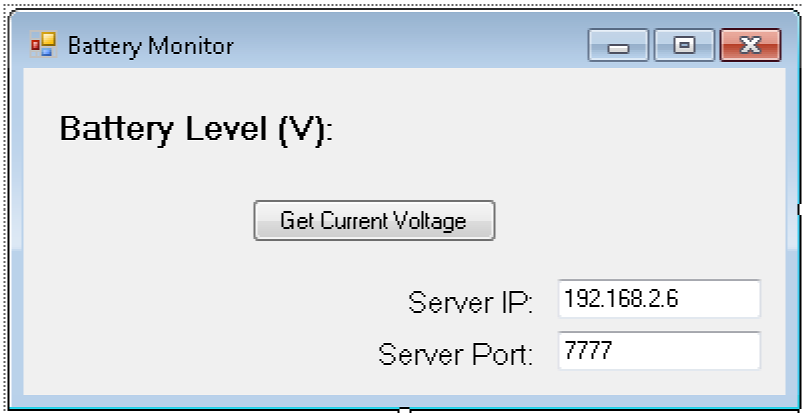
```
#pragma once

#include "client.h"
#include "myTimer.h"
using namespace System::Runtime::InteropServices; // for class Marshal

namespace ClientGUI {

    using namespace System;
    using namespace System::ComponentModel;
    using namespace System::Collections;
    using namespace System::Windows::Forms;
    using namespace System::Data;
    using namespace System::Drawing;

    /// <summary>
    /// Summary for Form1
    /// </summary>
    public ref class Form1 : public System::Windows::Forms::Form
    {
    private:
        static System::Timers::Timer^ aTimer;
    public:
        Form1(void)
        {
            InitializeComponent();

            // Create a new Timer with Interval set to 300 seconds.
            aTimer = gcnew System::Timers::Timer( 300000 );

            // Hook up the Elapsed event for the timer.
            aTimer->Elapsed += gcnew ElapsedEventHandler( &Form1::OnTimedEvent
);
            aTimer->Interval = 300000;
            aTimer->Enabled = true;

            GC::KeepAlive(aTimer);
        }

    protected:
        /// <summary>
```

```
            /// Clean up any resources being used.
            /// </summary>
            ~Form1()
            {
                if (components)
                {
                    delete components;
                }
            }
        private: System::Windows::Forms::TextBox^  txtServerPort;
        protected:
        private: System::Windows::Forms::Label^  lblTitle;
        private: System::Windows::Forms::Button^  btnConnect;
        private: System::Windows::Forms::Label^  label1;
        private: System::Windows::Forms::Label^  label2;
        private: System::Windows::Forms::TextBox^  txtServerIP;
        private: System::Windows::Forms::Label^  lblVoltageValue;
        private: System::Windows::Forms::Timer^  timer1;

        private: System::ComponentModel::IContainer^  components;

        private:
            /// <summary>
            /// Required designer variable.
            /// </summary>


#pragma region Windows Form Designer generated code
            /// <summary>
            /// Required method for Designer support - do not modify
            /// the contents of this method with the code editor.
            /// </summary>
            void InitializeComponent(void)
            {
                this->components = (gcnew System::ComponentModel::Container());
                this->txtServerPort = (gcnew System::Windows::Forms::TextBox());
                this->lblTitle = (gcnew System::Windows::Forms::Label());
                this->btnConnect = (gcnew System::Windows::Forms::Button());
                this->label1 = (gcnew System::Windows::Forms::Label());
                this->label2 = (gcnew System::Windows::Forms::Label());
                this->txtServerIP = (gcnew System::Windows::Forms::TextBox());
                this->lblVoltageValue = (gcnew System::Windows::Forms::Label());
                this->timer1 = (gcnew System::Windows::Forms::Timer(this-
>components));
                this->SuspendLayout();
                //
                // txtServerPort
                //
                this->txtServerPort->Location = System::Drawing::Point(267, 131);
                this->txtServerPort->Name = L"txtServerPort";
                this->txtServerPort->Size = System::Drawing::Size(102, 20);
                this->txtServerPort->TabIndex = 0;
                this->txtServerPort->Text = L"7777";
                this->txtServerPort->KeyPress += gcnew 
System::Windows::Forms::KeyPressEventHandler(this, &Form1::txtServerPort_KeyPress);
                //
                // lblTitle
                //
```

```
                     this->lblTitle->AutoSize = true;
                     this->lblTitle->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 12, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(0)));
                     this->lblTitle->Location = System::Drawing::Point(13, 20);
                     this->lblTitle->Name = L"lblTitle";
                     this->lblTitle->Size = System::Drawing::Size(148, 20);
                     this->lblTitle->TabIndex = 1;
                     this->lblTitle->Text = L"Battery Level (V):";
                     // 
                     // btnConnect
                     // 
                     this->btnConnect->Location = System::Drawing::Point(114, 65);
                     this->btnConnect->Name = L"btnConnect";
                     this->btnConnect->Size = System::Drawing::Size(123, 22);
                     this->btnConnect->TabIndex = 2;
                     this->btnConnect->Text = L"Get Current Voltage";
                     this->btnConnect->UseVisualStyleBackColor = true;
                     this->btnConnect->Click += gcnew System::EventHandler(this,
&Form1::btnConnect_Click);
                     // 
                     // label1
                     // 
                     this->label1->AutoSize = true;
                     this->label1->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 11.25F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(0)));
                     this->label1->Location = System::Drawing::Point(174, 133);
                     this->label1->Name = L"label1";
                     this->label1->Size = System::Drawing::Size(87, 18);
                     this->label1->TabIndex = 3;
                     this->label1->Text = L"Server Port:";
                     // 
                     // label2
                     // 
                     this->label2->AutoSize = true;
                     this->label2->Font = (gcnew System::Drawing::Font(L"Microsoft Sans
Serif", 11.25F, System::Drawing::FontStyle::Regular,
System::Drawing::GraphicsUnit::Point,
                            static_cast<System::Byte>(0)));
                     this->label2->Location = System::Drawing::Point(189, 107);
                     this->label2->Name = L"label2";
                     this->label2->Size = System::Drawing::Size(72, 18);
                     this->label2->TabIndex = 4;
                     this->label2->Text = L"Server IP:";
                     // 
                     // txtServerIP
                     // 
                     this->txtServerIP->Location = System::Drawing::Point(267, 105);
                     this->txtServerIP->Name = L"txtServerIP";
                     this->txtServerIP->Size = System::Drawing::Size(102, 20);
                     this->txtServerIP->TabIndex = 0;
                     this->txtServerIP->Text = L"192.168.2.6";
                     // 
                     // lblVoltageValue
                     // 
                     this->lblVoltageValue->AutoSize = true;
                     this->lblVoltageValue->Font = (gcnew 
System::Drawing::Font(L"Microsoft Sans Serif", 12, System::Drawing::FontStyle::Bold, 
System::Drawing::GraphicsUnit::Point, 
                            static_cast<System::Byte>(0)));
                     this->lblVoltageValue->Location = System::Drawing::Point(171, 20);
                     this->lblVoltageValue->Name = L"lblVoltageValue";
                     this->lblVoltageValue->Size = System::Drawing::Size(0, 20);
                     this->lblVoltageValue->TabIndex = 5;
                     // 
                     // timer1
                     // 
                     this->timer1->Enabled = true;
                     this->timer1->Interval = 60000;
                     this->timer1->Tick += gcnew System::EventHandler(this, 
&Form1::timer1_Tick);
                     // 
                     // Form1
                     // 
                     this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                     this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;
                     this->ClientSize = System::Drawing::Size(381, 163);
                     this->Controls->Add(this->lblVoltageValue);
                     this->Controls->Add(this->label2);
                     this->Controls->Add(this->label1);
                     this->Controls->Add(this->btnConnect);
                     this->Controls->Add(this->lblTitle);
                     this->Controls->Add(this->txtServerIP);
                     this->Controls->Add(this->txtServerPort);
                     this->Name = L"Form1";
                     this->Text = L"Battery Monitor";
                     this->ResumeLayout(false);
                     this->PerformLayout();

              }
#pragma endregion


private: System::Void btnConnect_Click(System::Object^  sender, System::EventArgs^  e) {
                     int serverPort;
                     double voltageValue;

                     serverPort = Convert::ToInt32(this->txtServerPort->Text);

                     if (getFile(txtServerIP->Text, serverPort)) {
                            voltageValue = updateVoltage("voltage_log\0");
                            if (voltageValue >= 0)  {
                                   this->lblVoltageValue->Text = 
Convert::ToString(voltageValue);
                            } else {
                                   MessageBox::Show("Failed to read the log file");
                            }
                     }
               }

private: static System::Void OnTimedEvent( Object^ source, Timers::ElapsedEventArgs^ e )
{

               }
```

```
private: System::Void txtServerPort_KeyPress(System::Object^  sender,
System::Windows::Forms::KeyPressEventArgs^  e) {
                      if (!Char::IsDigit(e->KeyChar) && e->KeyChar != 0x08)
                             e->Handled = true;
               }
private: System::Void timer1_Tick(System::Object^  sender, System::EventArgs^  e) {
                     int serverPort;
                     double voltageValue;

                     serverPort = Convert::ToInt32(this->txtServerPort->Text);

                     if (getFile(txtServerIP->Text, serverPort)) {
                            voltageValue = updateVoltage("voltage_log\0");
                            if (voltageValue > 0) {
                                   this->lblVoltageValue->Text =
Convert::ToString(voltageValue);
                            }
                     }
               }
};
}
```

Form GUI:



Server IP: 192.168.2.6

Server Port: 7777

Figure 37: Screen Capture of Battery Voltage GUI Window