Rocket Cam

Low Frequency Analog Transmission of Digital Video

by

Thomas J. Higdon

Advisor: Wayne Pilkington

Senior Project

ELECTRICAL ENGINEERING DEPARTMENT

California Polytechnic State University

San Luis Obispo

2015

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

Tables *Page*

Figures

Abstract

The camera module provides data for improving models of dynamic events on Orbital ATK Corp. rockets and aids in troubleshooting, if necessary. Video images provide a valuable addition to the strain, vibration, shock, and acoustic data used for modeling dynamic events, such as stage separations. The cameras can record a duration of video data suitable for capturing a dynamic event and of high enough quality to aid in its modeling. The module readily integrates into the rocket's current analog data collection systems. The project has further relevance to any other application that necessitates video data transmission over similar limited-bandwidth, analog data channels.

Though errorless data transmission was not achieve, over 99% of the digital by bytes transmitted where recovered to within 99% accuracy. This level of error is not suitable for compressed data. However, the primary sources of error can potentially be resolved by adaption to a more permanent prototype platform.

I. Introduction

The camera module records dynamic events on Orbital ATK Incorporated rockets. Examples of dynamic events include stage separations and payload (satellite) deployments. Video data provides a valuable addition to previously existing sensors such as strain gages, accelerometers (vibration and shock), and microphones (for acoustic information). Teams then use the information from these sensors to improve models of the events, and to troubleshoot any undesired outcomes [1].

Groups internal to Orbital ATK have done some work on developing an inexpensive camera module for this task, however, their versions do not currently satisfy all of the needs of other departments in the corporation. Unfortunately, Orbital ATK has not approved for public release the details of the status of current development. However, this project focuses on the need to utilize an existing data channel that has minimal impact on the resources of the larger rocket system (the same type of data channel which the accelerometers use) [2]. The data channel consists of an analog data line, with relatively low bandwidth when compared to typical video data channels. The primary constraint to the bandwidth comes from the receiver unit consisting of an ADC with a maximum sampling rate of 10 kilo-samples per second, with a maximum of 12 bits of resolution [2]. Fortunately, the video data utilizes buffering before transmission and does not need to achieve real time transmission. However, the severely limited bandwidth still places significant limitations and consequences on the design. Though no need exists for real time transmission, the module must still transmit the video sample within a widow constrained by the minimum flight time of the rocket after the dynamic event. Other groups have done some work on "low-bandwidth" video data transmission [3]. But it remains an under-developed area, because of the limited number of applications that necessitate it.

Because of the data bandwidth limitations, effective compression before its transmission must also occur, in order to minimize the necessary transmission time. In contrast to the "low-bandwidth" concerns, video compression remains an area of significant and active development. The MPEG-4 AVC/H.264 lossy compression standard illustrates a method with widespread use [4]. In addition, groups have also researched the implementation of compression algorithms in embedded applications, both using microprocessors [5] and using FPGAs [6].

However, the author believes that this project typifies one of very few that require the compression of video data, and its transmission through such a narrow bandwidth analog data channel.

II. Production Design Engineering Requirements

Customer Needs Assessment

Orbital ATK Incorporated needs a low-cost module to provide video data for improving models of dynamic events and for troubleshooting. It needs to be relatively stand-alone, and require minimal effort to integrate into their existing rocket system; and it should have minimal impact on their existing resources [1, 2].

Requirements and Specifications

The marketing requirements and engineering specifications listed in Table I below, primarily come from the need to integrate the module into the existing systems of the rocket. However, marketing requirements 1 & 2 and the first three specifications relate to the need to obtain sufficient data for useful study of dynamic events.

Orbital ATK deems that 0.5 to 3 seconds of video, with a resolution of 640x480 pixels, with 8-bit depth grayscale pixels, and a rate of 30 frames per second, is sufficient for aiding in their modeling and troubleshooting of typical dynamic events [1, 2]. The module must be able to transmit the data through of the previously existing data channel (the analog data line), and the data must be recoverable after receipt by their previously existing hardware (the ADC). In order to make better use of the limited bandwidth, the data must be compressed before transmission. The

transmission of the video sample must take less than 10 minutes. The module must also store the original, followed by the compressed, data, before being ordered by a dedicated 1-bit "Transmit Command" line send it. Recording and compression should start after the receipt of a signal from a dedicated 1-bit "Record Command" line. The device's addition to the larger rocket system should not constitute a significant increase to the cost of the overall system [1, 2]. The module must be safe and minimize its environmental impacts.

TABLE I

ROCKET CAM MODULE REQUIREMENTS AND SPECIFICATIONS

Marketing Requirements	**Engineering Specifications**	**Justification**
[1], [2]	Capable of recording at between 0.5 and 3 seconds of video	Sufficient time to observe event (i.e. stage separation)
[4]	At least 640x480 pixel image resolution, at 8 bits per pixel	Sufficient for useful image of event.
[1], [2], [3]	Captures at between 30 and 60 frames per second	Fast enough to track dynamics, but slow enough to not produce excessive data.
[2], [3], [6]	Sufficient memory to store captured data and to perform compression operations on the data	Necessary to hold data and send it only when requested.
[4]	Output data over analog line	Existing data interface that has the minimum system impact.
[4]	Data recoverable after analog signal is sampled by an ADC with 12 bit maximum resolution, at 10,000 samples per second	This describes the existing system that receives the transmitted data.
[5]	Inputs 2 separate, 1-bit command lines. One signals the module to begin recording; the other to transmit recorded data.	Minimal interfacing between an otherwise stand-alone module and the larger system.
[6]	Sufficient video data compression such that sample transmission requires less than 10 minutes (minimum compression factor of 2 to 1)	Due to the limited bandwidth of the necessary data channel, it's critical to reduce the size of the data, so that transmission fits within available time
[7]	Finished production units should cost no more than $200.	Desire a relatively inexpensive module to complement existing sensors.
[8]	Use at least 70% ROHS compliant components.	ROHS provides a good general standard for minimizing health and environmental impacts of component materials

Marketing Requirements

1. Provide short duration video data of dynamic events
2. Video data has sufficient quality to aid in event modeling and troubleshooting
3. Internally store the captured data until transmission
4. Use existing data interfaces on the rocket
5. Require minimal command interfacing
6. Transmit data in less than 10 minutes
7. Be "low-cost"
8. Be safe and relatively environmentally-friendly

Black Box Diagram (Level 0)

As its overall function, the design inputs light data, and then outputs an analog voltage representation of that data. The module also accepts two 1-bit command signals, the record and transmit commands, and it inputs power. Figure 1 shows this overall Level 0 functionality within the context of the larger rocket system, and Table III elaborates on each facet of it.

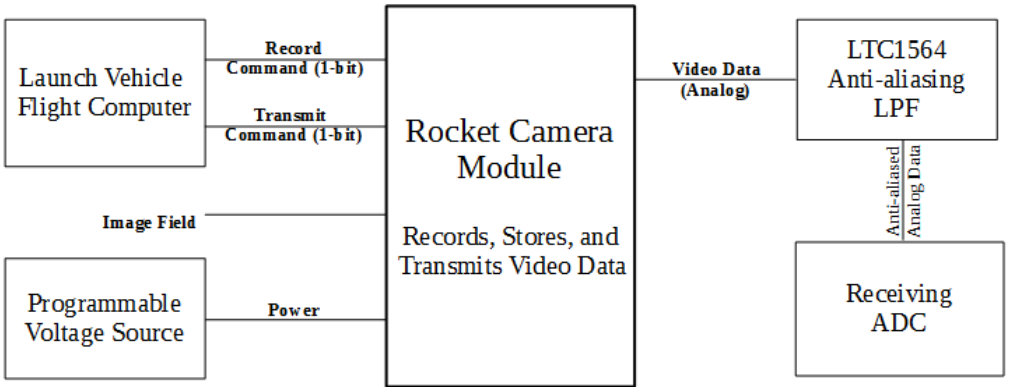


FIGURE I
ROCKET CAM – LEVEL 0 BLOCK DIAGRAM

TABLE II
ROCKET CAM – FUNCTIONAL REQUIREMENTS

Module	Rocket Camera
Inputs –	Record Command: Bi-level optical signal or 1-bit TTL level signal Transmit Command: Bi-level optical signal or 1-bit TTL level signal Image Field: Incident light from the surrounding environment Power: Sufficient power for the module to function on
Outputs –	Video Data: Analog signal, 8 to 12 bit resolution, sampled at 10,000 samples per second
Functionality –	Upon receipt of the Record Command, it records and stores 0.5 to 3 seconds of video data. Then, upon receipt of the Transmit Command, it outputs the analog Video Data

Bi-level optical signal and 1-bit TTL level signals represent the existing command line infrastructure available in the rocket system. The bi-level optical signals have the advantage of providing optical isolation between the camera module, and rockets more critical control systems. The 1-bit TTL signals have the advantage of being faster, cheaper, and easier to implement and integrate. The analog data line represents the available, and lowest impact, data channel [1,2].

IV: System Design – Functional Decomposition (Level 1)

Functional Block Diagrams

Figure II illustrates the primary subcomponents of the module in hierarchical decomposition format. Figure III shows them in function structure format. In words, the control unit receives the Record and Transmit Commands from the outside world and controls the other sub-modules' operations so that the device performs these tasks. Upon receipt of the Record Command, the video camera activates, takes in visual data and outputs an uncompressed digital representation of this data, which the memory access peripheral routes into memory. The control & data processing unit then communicates with memory, in order to perform compression and encoding operations on the video data. The compressed and encoded version of the data gets stored back in memory. Upon receipt of the Transmit Command, the compressed and encoded data gets read from memory, routed by the peripheral, and sent to the digital-to-analog converter, which converts it to its final analog form and outputs it to the outside world. During all of these operations, the power regulation unit provides appropriate power to all of the other sub-modules. The Control Unit contains the Memory Access Peripheral shown in the functional structure diagram, however, the peripheral can operate largely autonomously from the rest of the Control Unit, and is functionally distinct and significant enough to warrant separate mention at this level.

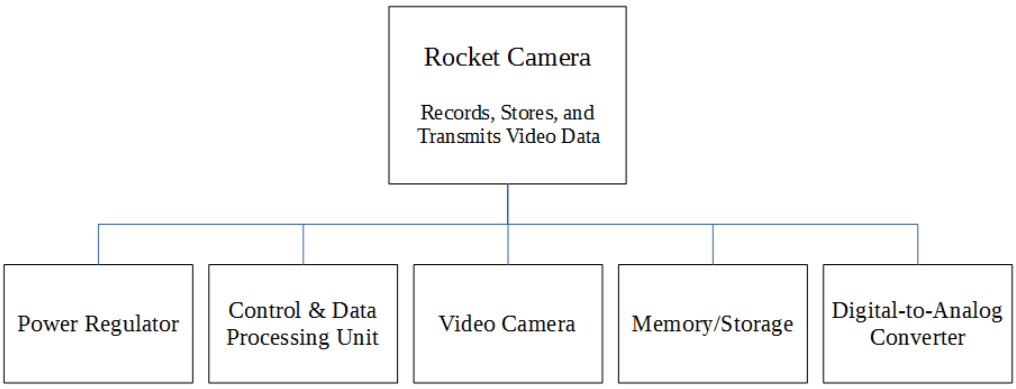


FIGURE II

ROCKET CAM – LEVEL 1 HIERARCHICAL DECOMPOSITION

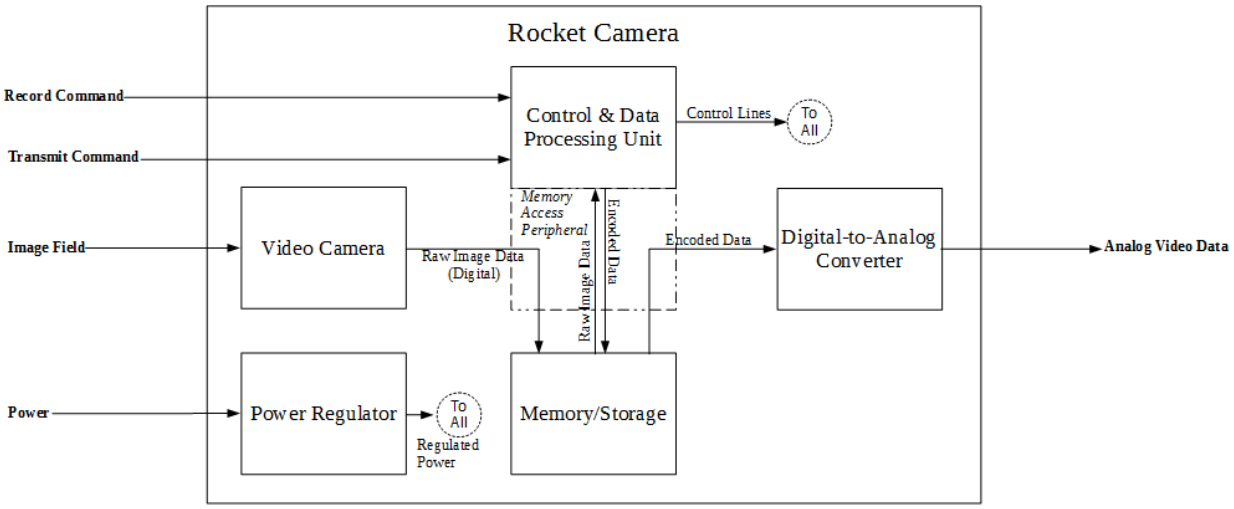


FIGURE III

ROCKET CAM – LEVEL 1 BLOCK DIAGRAM

Table IV provides a breakdown of the inputs, outputs, and functional requirements of each sub-module.

TABLE III

ROCKET CAM – LEVEL 1 FUNCTIONAL REQUIREMENTS

Module	**Power Regulator**
Inputs –	External Power: 3 to 13V, constant voltage source at up to 100mA [1] Or programmable 28V (max) voltage source, with unspecified current capability
Outputs –	Regulated Power: Power at voltage levels acceptable to all of the other components
Functionality –	Regulates externally supplied power to a level safe and usable by the other sub-modules
Module	**Video Camera**
Inputs –	Camera Control Lines: Digital. Controls operation and settings. Image Field: Optical. Incident light from the surrounding environment. Regulated Power: DC voltage source.
Outputs –	Raw Image Data: Digital. Uncompressed (raw) video image data.
Functionality –	After receiving instruction from the Control Unit, it inputs optical information from the surroundings and converts it into an uncompressed digital representation, which it outputs.
Module	**Control & Data Processing Unit**
Inputs –	Record Command: 1-Bit external control signal Transmit Command: 1-Bit external control signal Memory Data Bus: Digital. Accesses data stored in the Memory sub-module. Regulated Power: DC voltage source.
Outputs –	Camera Control Lines: Digital. Instructs the Video Camera sub-module when to activate and controls its settings. DAC Control Lines: Digital. Controls settings of the DAC sub-module. Memory Control Lines: Digital. Control setting of the Memory sub-module. Memory Data Bus: Digital. Send data to the Memory sub-module for storage.
Functionality –	Inputs the Transmit and Receive Commands from the outside world. Controls the other sub-modules and overall behavior of the device. Takes raw video data from memory, compresses and encodes it, then stores it back in memory.
Module	*Memory Access Peripheral*
Inputs –	Address Information: Digital. From the main part of the Control Unit module. Data In: Digital. Regulated Power: DC voltage source.
Outputs –	Memory Control Lines: Digital. Controls address access and settings of Memory sub-module. Memory Data Bus: Digital. Sends data to the Memory sub-module for storage.
Functionality –	Controls and formats data access from and storage in the Memory Sub-module. Physically integrated into the Control Unit sub-module, but can operate largely autonomously and considered functionally distinct and significant enough to warrant separate mention.

TABLE III (continued)
ROCKET CAM – LEVEL 1 FUNCTIONAL REQUIREMENTS

Module	**Memory**
Inputs –	Control Lines: Controls operation and settings. Address Line: Controls accessed address for read or write operations. Data Bus In: Digital. Regulated Power: DC voltage source.
Outputs –	Data Bus Out: Digital.
Functionality –	Stores values placed in it by the Control Unit or the Memory Access Peripheral. Allows retrieval of these stored values.
Module	**Digital-to-Analog Converter**
Inputs –	Control Lines: Controls operation and settings. Data Bus: Digital. Regulated Power: DC voltage source.
Outputs –	Analog Data Line: Analog. Transmitted to outside world. Converted at a resolution of higher than 8-bits, and a rate of 10,000 samples per second.
Functionality –	Takes digital data sent to it and converts it into an analog representation.

Implementation Concept Block Diagram

Figure IV below displays the implementation concept block diagram, indicating the physical component breakdown of the system. The control/processing unit consists of two processors, one which primarily controls the video capture and processing, and one which primarily controls the analog data transmission. The video data is captured by a digital camera unit. The non-volatile memory consists of a microSD card inserteded into the video processors board (on which the video processor's operating system is also stored). The digital-to-analog converter also exists on a separate chip, and steps are taken to reduce the effect of noise generated by the digital circuitry it. The power regulation system is actually comprised of multiple regulators associated with each submodule in order to provide the voltage levels necessary for all of the components and subcomponents.

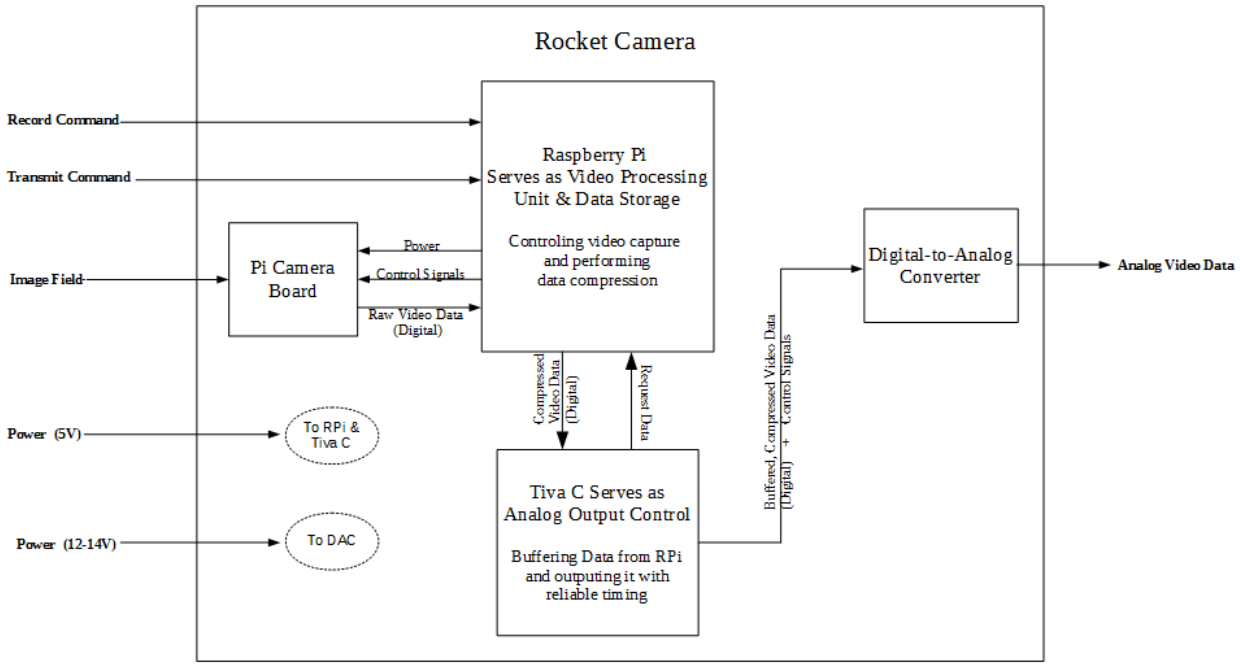


FIGURE IV
ROCKET CAM – IMPLEMENTATION CONCEPT BLOCK DIAGRAM

Table IV describes in greater detail, important data flow signals shown in Figure IV.

TABLE IV
ROCKET CAM – DATA FLOW SIGNALS

Source	Signal	Format/Function
Outside World	Record Command – Transmit Command – Image Field – Power –	1-Bit external control signal 1-Bit external control signal Physical light striking the sensor DC 3 to 13V, constant voltage source at up to 100mA or programmable 28V (max) voltage source, with unspecified current capability (higher than 100mA)
Image Sensor	Raw Image Data –	Camera Serial Interface (CSI-2), 15 pin interface [7]
Rasp. Pi w/ Cam	Raw Image Data/Camera Control –	Camera Serial Interface (CSI-2), 15 pin interface
Rasp. Pi	Compressed Data –	8-pins parallel data + 2-pins control (data ready & more data available)
Tiva C.	Buffered Data – Request Data –	8-pins parallel data + 1-pins control (write enable) 1-pin control
DAC	Analog Video Data –	1-pin. 0-10 V analog value with minimum of 8-bit accuracy.

Additional, parameters necessary for functional components in the design are summarized below.

Control/Processing Unit –

- Capable of inputting image data at real time video speeds

- ≥ 27.648 * 10^6 Bps $\left(640 * 480 \frac{pixels}{frame} * 3 \frac{Byte}{pixel} * 30\ fps\right)$ [YCbCr format]
- Capable of storing image data to memory at real time video speeds
  - ≥ 9.216 * 10^6 Bps $\left(640 * 480 \frac{pixels}{frame} * 1 \frac{Byte}{pixel} * 30\ fps\right)$ [Grayscale format]
- Capable of outputting a byte to the DAC at consistent frequencies up to 5 kHz
  - Maximum hypothetical unsynchronized transmission rate, the Nyquist Rate
  - Maximum allowable timing deviation, 20 μs (10%)
- Compression takes less than 2 minutes to complete
  - Any processing time beyond this mark must either be more than compensated in reduced transmission time, or must occur concurrently with transmission of earlier data
- Cost less than $60

Image Sensor –

- Capable of VGA quality resolution at 30 frames per second
  - 640x480 pixels @ 30 fps
- Minimum of 8-bit luma depth
  - 8-bit grayscale can be extracted

Memory –

- Capable of inputting image data at real time video speeds
  - ≥ 9.216 * 10^6 Bps $\left(640 * 480 \frac{pixels}{frame} * 1 \frac{Byte}{pixel} * 30\ fps\right)$ [Grayscale format]
- Sufficient memory to store both raw and compressed video data
  - ≥ 9.216 * 10^6 Bytes $2 * \left(640 * 480 \frac{pixels}{frame} * 1 \frac{Byte}{pixel} * 30\ fps * 0.5\ s\right)$

Digital-to-analog converter –

- Output levels of 0-10 V (unipolar operation)
  - The values read by the existing ADC
- Slew + settling time (to within 10/256 V) should be less than 100 μs
  - Operating at 3.33 kHz this means that at least 2/3 of the period should be at the correct level, and with the external ADC sampling 3 times for each period, two out of three samples should contain the correct value.
- Monotonic
  - An increase in value always equates to an increase in voltage
- Total Unadjusted Error should be less than 10/256 V and INL < the 8^{th} bit (1 LSB if a 8-bit DAC) [8]
  - The importance of these requirement can be reduced by sending a calibration signal over the line, prior to any data transmission, and accounting for existing offsets.

V. Technology Choices and Design Approach Alternatives Considered

The primary technology choice central to this design, was how to implement the control/processing unit. Several possibilities were investigated. The two primary alternatives categories were: a low-cost ARM microcontroller with direct C programing and real time operation (typically targeted at controls applications), or a more expensive ARM microprocessor built for mobile applications and running a full operating system. Below are listed some of the pros and cons of each option.

ARM "Controls" family –

Pros:

- Low-cost (with development boards as low as $13) [9]

- C environment gives designer a large degree of control over the behavior of the processor
- Interrupt driven, real time operation is possible
- Existing code for serial interfaces available for specific devices
- Existing C code for video compression could potentially be adapted for device

Cons:

- Small on-chip memory means video data must be stored off-chip
- Clock speeds and data transfer rates are on the edge of not being able to handle the live video data, especially with the need to immediately send it to external memory (though DMA's could help with this, that removes the possibility of performing any processing to the data before storage (i.e. convert to grayscale to save memory space))
- Compression algorithms would likely take significant time, and could be challenging to run simultaneously with data transmission

ARM "Mobile" family –

Pros:

- Sufficient data rates to handle video
- Active and helpful development communities and documentation (esp. for BeagleBone and Raspberry Pi)
- Existing projects and software for camera integration and video compression
- Either larger on chip memory, or relatively inexpensive development board with memory onboard

Cons:

- Non-real time operating systems make timings unreliable

FPGA –

Pros:

- Can be designed to handle capturing and moving video data into memory in real time, even with some preprocessing
- Various operations can be performed in parallel
- Hardware based compression algorithms could potentially run very fast
- Open source hobbyist community [10]
- Most flexible option

Cons:

- Most expensive option
- Requires external memory (or very large, and expense chip)
- HDL coding and implementation are potentially more challenging than software implementation

In the end, I choose a hybrid approach, utilizing both a low-cost "controls" ARM and a higher performance "mobile" ARM. This solution takes advantage of the benefits of both families. The higher performance processor can handle video capture and storage, and perform the computationally intensive compression operations; and it can make use of the large development communities, a higher level of abstraction, and available open source software, to streamline the development process. While the lower-performance controls ARM can control the analog data output with consistent, reliable timing, for little additional cost. This solution is still significantly less expensive than using an FPGA, and is also less expensive than high performance, real-time operation (single) ARM processors. The approach does have higher complexity than a single processor and requires the two to communicate with each other, but it also allows a greater degree of design flexibility.

A color image sensor was chosen, despite only needing grayscale data, because color image sensors are more prevalent, more supported, and actually cheaper. In specific an image sensor already supported by the "mobile" ARM processor was chosen.

A digital-to-analog converter that met all of the performance parameters discussed above was chosen.

Specifically the Raspberry Pi Model B+ development platform serves as the video capture, processing, and storage control role. The Raspberry Pi's low cost, large open-source development community, specifically supported low-cost camera unit, and existing software for performing H.264 video compression (included with the Raspbian operating system), caused its selection platform. The Raspberry Pi features a Broadcom BCM2835 system-on-a-chip, containing an ARM11 floating point CPU and Videocore 4 GPU [11]. For the analog data output control, the design uses the Tiva C Launchpad development platform, which features a Texas Instruments TM4C123GH6PM microcontroller with an ARM Cortex-M4 processor core and 32 KB of on-chip SRAM [12]. The Tiva C was chosen for its sufficient memory to buffer the video data, sufficient processing capabilities to control the data output and retrieval from the Raspberry Pi, and low-cost. The Raspberry Pi Camera Module's compatibility and support with Raspberry Pi determined its selection. The AD7245A digital-to-analog converter's compliance with the minimum static and dynamic performance parameters determined in the previous section, and its 0-10V unipolar operating mode allowed its consideration [13]. Its relatively simple parallel digital interface, and its availability in a PDIP package conducive to early prototyping furthered its selection.

VI. Project Design Description

Component Integration

The Level 2 Block Diagram of the final design with additional signal and interconnection detail is shown in Figure V. It contains all functional submodule connections.

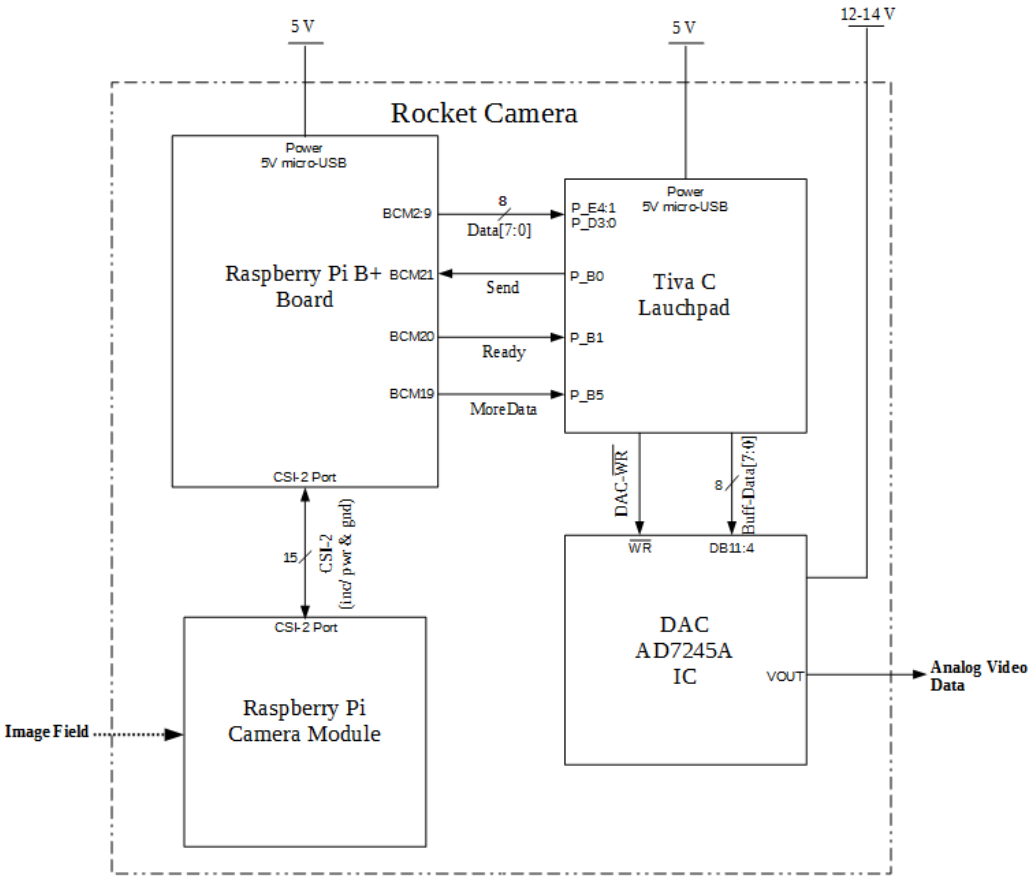


FIGURE V

ROCKET CAM – IMPLEMENTATION CONCEPT BLOCK DIAGRAM LEVEL 2

The connection between the Raspberry Pi B+ and the Raspberry Pi Camera Module follows the MIPI standard and is controlled using existing drivers. The communication between the Raspberry Pi and the Tiva C consists of a custom parallel data interface (coded by the project designer). The Tiva C controls the AD7245A via a single-latched, parallel data loading structure (the AD7245A contains a double latched input, however, the design is configure so that one latch is left transparent).

For simplicity the schematic wiring diagrams showing specific pinouts are broken into 3 diagrams, centered around the 3 main subcomponents. This prevents crossing wires in the diagrams. Signal names correspond to those shown in Figure V.

Figure VI shows the pin connections for the AD7245A DAC. The DAC is configured in unipolar-mode, with the LDAC latch held transparent, the chip select engage, and latch clearing disabled. The design only makes use the upper 8 data bits and the lower 4 are tied to ground; this allows the design to make use of the higher precision the AD7245A has over most 8-bit DACs, without pushing the resolution to its limit and incurring less noise and drift resilience. The device configuration is primarily based on the unipolar configuration and microprocessor interfacing application notes contained within its datasheet [13].

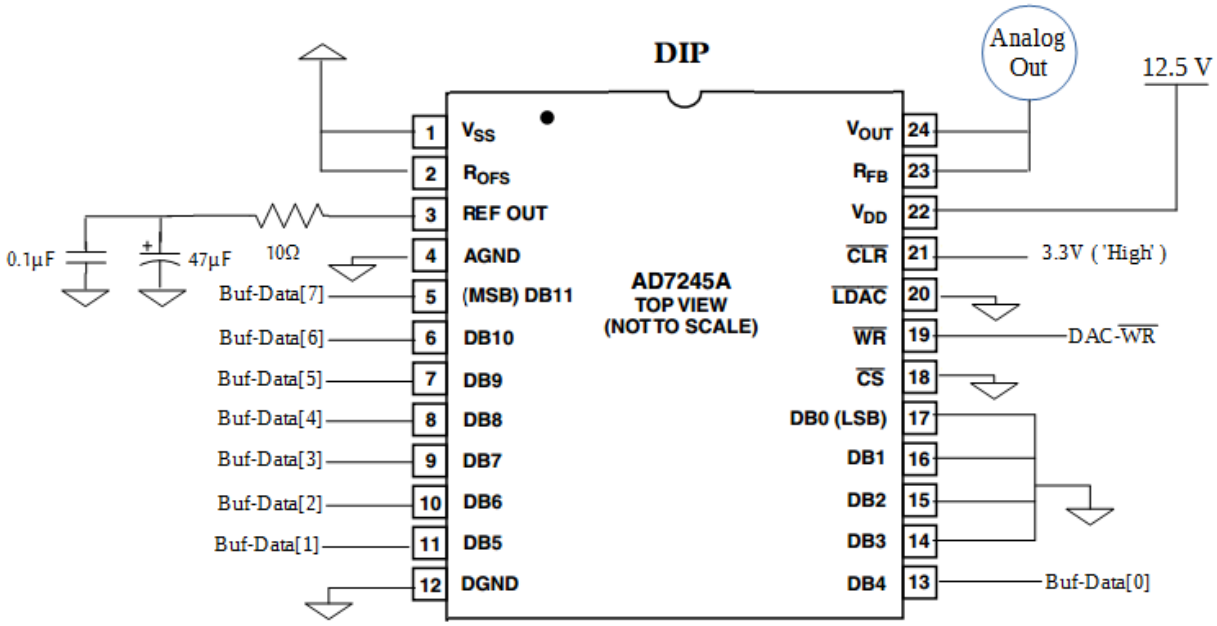


FIGURE VI
AD7245A PINOUT AND WIRING SCHEMATIC
(IMAGE BASED ON DATASHEET PINOUT [13])

Figure VII shows the pin connections for the Tiva C Launchpad evaluation board. In the design, the Tiva C serves the function of controlling the analog data output with reliable timing. After being informed that the Raspberry Pi has data ready for it, it buffers data from the RPi and sends it to the DAC at consistent time intervals. When the amount of data stored in the buffer falls below a minimum value, it requests the next byte from the RPi.

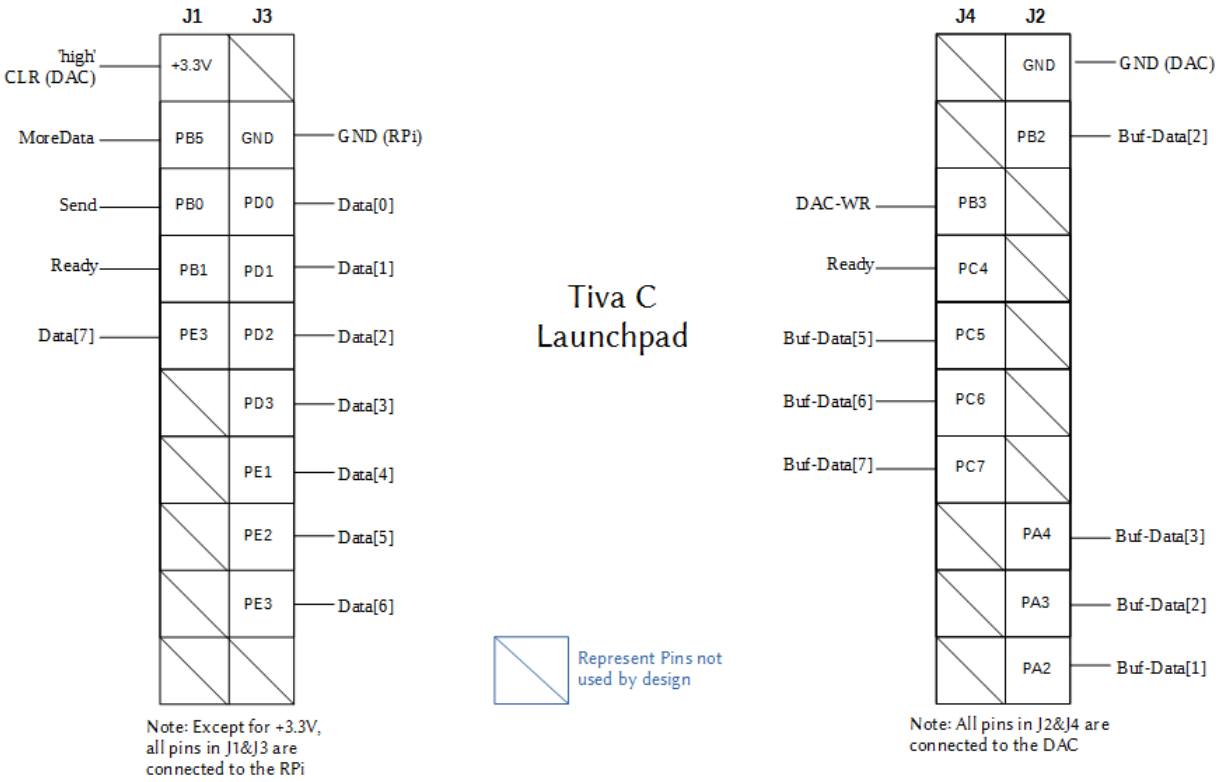


FIGURE VII
TIVA C LAUNCHPAD PINOUT AND WIRING SCHEMATIC

Figure VIII shows the pin connections on the Raspberry Pi B+. The RPi's function is to receive record and transmit commands, control video data collection, perform video compression, store the video data, and send it to the Tiva C when requested. For the purposes of testing and debugging, the receive and transmit commands are instructions sent over an SSH connection from the designer's computer, rather than the specified 1-bit lines. However, switching to 1-bit TTL-level commands takes minimal effort (the RPi's gpio ports are already compliant with TTL-levels).

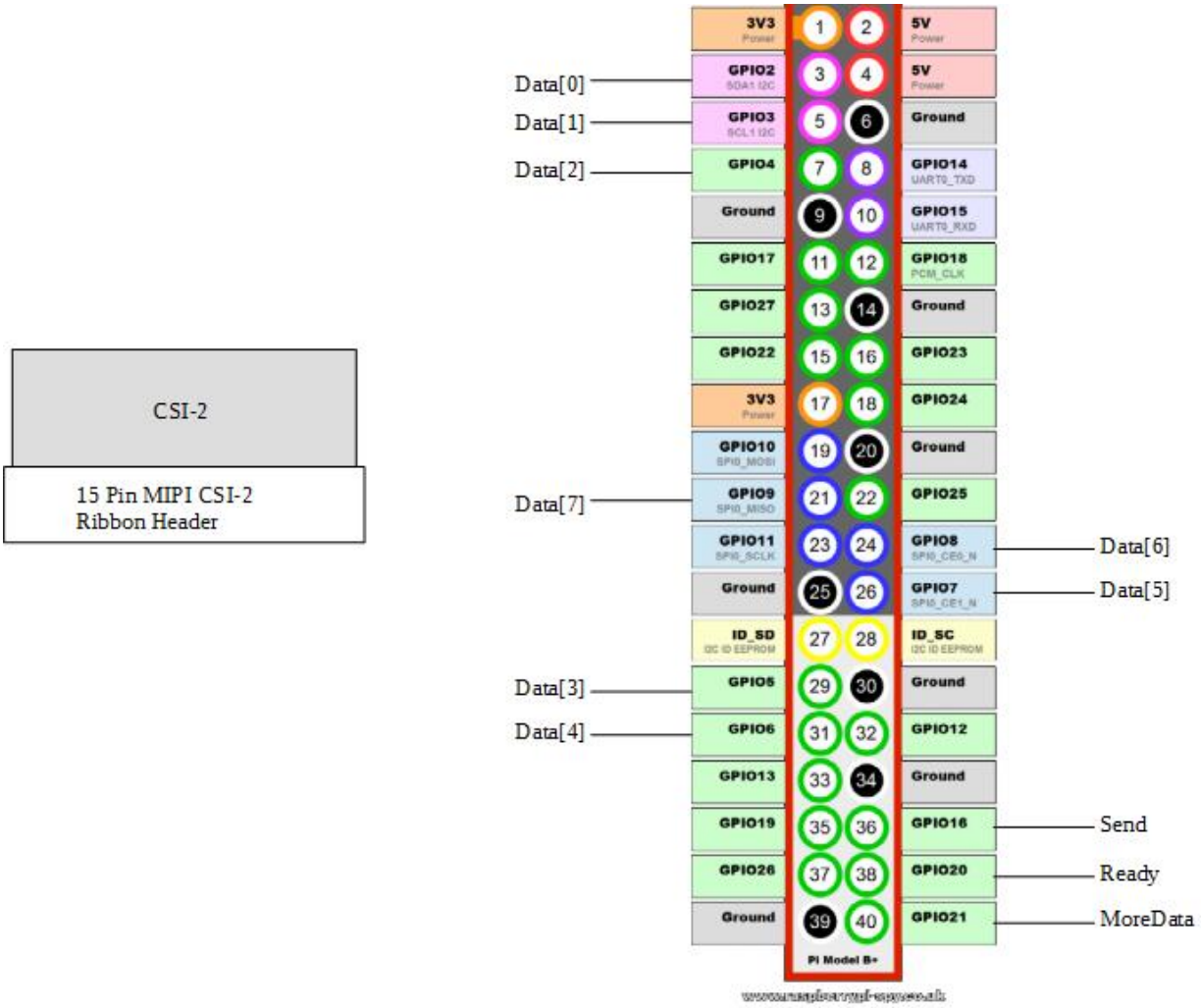


FIGURE VIII
RASPBERRY PI B+ PINOUT AND WIRING SCHEMATIC
(IMAGE BASED ON RASPBERRYPI-SPY.CO.UK IMAGE [14])

Design Operation

The final analog output of the design is at 256 discrete levels, corresponding to one full byte of data. Because of the inability to synchronize transmission and reception, the design faces Nyquist rate limitations. The DAC outputs the next byte at intervals of 300 µs (or slightly longer), so that the receiving end will have at least 3 samples for every new voltage level (every new byte). If transition (slew) and settling times are below 100 µs total, then (assuming negligible noise) there will be 2 settled samples with voltages correctly representing the byte of data, for each byte sent. Thus the analog data decoding algorithms on the receiving end can look for two consecutive readings at the same voltage level, to determine the byte value. The output interval is controlled by the Tiva C's timers, which are set to trigger at 3330 Hz. This necessity for settling in under 100 µs, was a primary reason behind the choice of the AD7245A, as mentioned in Section V. Before actual data is sent, a calibration signal containing 2 full 300 µs periods of every voltage level (byte value) is output, so that the decoding algorithm can adjust and calibrate for any offsets and persistent non-idealities of the DAC's output levels.

The Tiva C uses a 512-byte array as a circular type buffer. A byte received from the Raspberry Pi is stored into the array at one index location, while bytes sent to the DAC are read from a different index location. The program attempts to keep the receive index 500 elements in the lead of the send index, with the indices wrapping around when they reach the end of the array. The 500 element buffer length was chosen, because, during a test of the impact of the Linux's background multithreading on gpio output performance, the largest undesired delay (during a 3 minute measurement period) was approximately 5.3 ms with multiple others on the order of 4 ms. The 500 element buffering length, combined with the output interval of 300 μs, means that the buffer stores enough information for 150 ms, or roughly 30 times the maximum observed delay. This better than order of magnitude safety margin, which is desirable to protect against the possible danger if multiple delays are stacked before the buffer can be fully replenished. In the Tiva C's code, the receive and store process is run in the main program, while the read and send process occurs within an interrupt service routine (triggered by the timers).

The communication between the Raspberry Pi and the Tiva C uses the following pattern:

```
Begin:   RPi: MoreData = 1; //To tell the TC there is data available
Repeat:  TC: If the buffer is not full, set Send=1, else wait; //TC requests next byte from RP
         RPi: Load the byte onto the Data[7:0] pins;
         RPi: Set Ready=1; //Tell TC that the byte is ready
         TC: Read byte
         TC: Send = 0; //Acknowledge receipt of data
         RPi: If no more data (file end reached), MoreData = 0;
         RPi: Ready=0; //Prepare for the next loop iteration
         TC: If MoreData==1, repeat, else break loop.
```

This communication method was specifically designed to be resilient to interrupts, both from the Tiva C's timers and from background threads in the Raspberry Pi's non-real time operating system (Raspbian Linux), and to neither lose its place in the process nor miss data that was sent, due to timing errors. Thus its asynchronous nature. This provides a reliable communication scheme, which can still be quite fast when compared to the 300 μs per byte that it must exceed, because (when not interrupted) the next step runs as soon as the previous step is completed (and the outputs are stable), at the clock speeds the two processors are configured for.

The code for the Tiva C was rewritten in C using Texas Instruments' Code Composer Studio IDE. During compiling, the optimization level for set to 1, while optimize-for-speed was set 5 (maximum), because sufficiently fast operation is critical to the buffer's reliability. The gpio interactions of the Raspberry Pi were written in C in a simple text editor, before being compiled in Linux terminal commands using the GCC compiler with standard compilation and optimization options. The code made use of the WiringPi library for gpio control [15]. Running the exiting video capture and compression code, as well as higher level program flow control, takes place in Linux shell scripts. The compressed H.264 video data is wrapped in .mp4 format using the MP4Box application [16]. The analog data capture for testing purposes used an Analog Discovery's oscilloscope, which was being controlled by a Python script provided with the beta 2.8.13 Version of its software development kit (WaveformsSDK)[17][18]. Decoding of analog data and reconstruction into a digital file is performed in Matlab. Samples of all the source code developed for the project are included in Appendix D.

VII. Physical Construction and Integration

For prototyping purposes the design consists of the Tiva C Launchpad and Raspberry Pi B+ development boards, and an AD7245A in a PDIP package on a breadboard, all interconnected via wire-leads and jumpers, as shown in Figure IX.

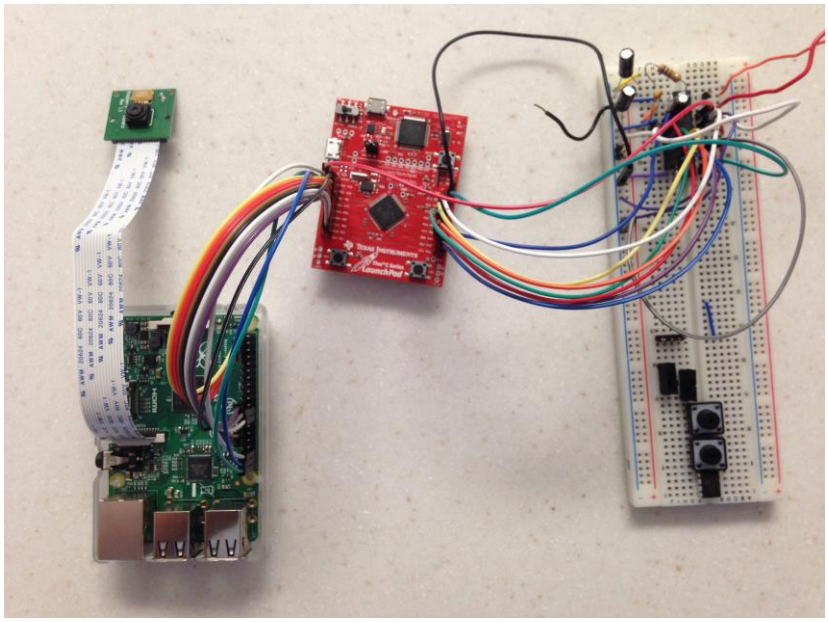


FIGURE IX
ROCKET CAM: FINAL PROTOTYPE

On the breadboard, all ground connections (especially for analog references) are made as physically close together as practical, and capacitive coupling is added between all nodes that are supposed to have fixed DC values. Figure X is a closer image of the bread board section.

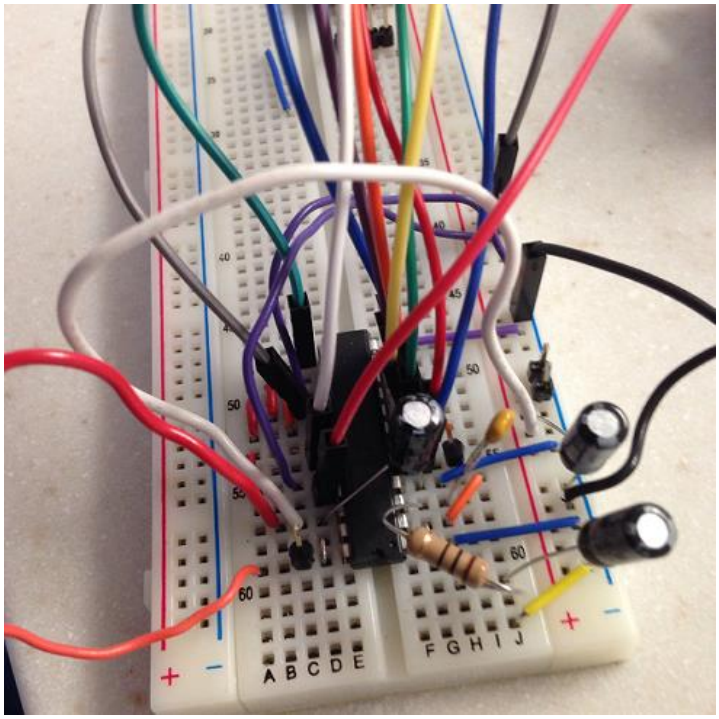


FIGURE X
ROCKET CAM: FINAL PROTOTYPE - DAC

For the prototype, power for the AD7245A is provided by a voltage source set to 12.5 V. Power (5V) for the Tiva C Launchpad is provided over a micro-USB connection. When debugging this is typically connected to the designer's computer. Power (5V) for the Raspberry Pi B+ is also provided via a micro-USB connection. This connection is typically to a wall outlet adapter.

VIII. Integrated System Tests and Results

For testing purposes the recovered, processed, and reconstructed video data from the analog output of the module is compared with the raw video data stored in the modules memory (accessed via an Ethernet connection). The reconstructed byte-array in Matlab is compared to the original file viewed as a byte-array. The following numbers are based off of a specific test of the video capture and transmission.

The first statistic to consider is the number of times the Matlab code detected that it had failed to read a byte, these points are stored in the byte array as NaN's (Not-a-Number's). This occurs when Matlab does not detect a settled value within 4 samples of the previous byte. The resulting count was:

NaN Count: 1025

The next relevant statistic is the length of the reconstructed data array, vs the actual data array. This gives a general starting point for comparison of the too samples.

Actual Video Data Length: 959038 bytes

Reconstructed Data Length: 959034 bytes

Which means that the decoding algorithm failed to read at least 4 bytes, which it did not explicitly detect as failed reads and use a NaN as a placeholder for. Also note that this also allows the calculation of the percentage of detected unreadable bytes, 1025/959038 = 0.107%.

Table V shows a comparison of the first 12 and the last 12 bytes of the actual and the reconstructed data. The sets clearly correlate, however there are several single bit errors present, most likely due either to inaccurate calibration of the decoding, or to noise on the analog line.

TABLE V

ACTUAL AND RECONSTRUCT VIDEO DATA – BEGINNING AND END COMPARISON

Index	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
Actual	0	0	0	24	102	116	121	112	105	115	111	109
Reconstructed	0	0	0	24	102	116	121	112	104	115	111	109
Index	*end-11*	*end-10*	*end-9*	*end-8*	*end-7*	*end-6*	*end-5*	*end-4*	*end-3*	*end-2*	*end-1*	*end*
Actual	67	32	48	46	53	46	48	45	114	101	118	0
Reconstructed	68	32	48	47	53	47	48	45	114	101	119	0

A plot of the between the first 50,000 bytes of the two arrays shows that up to the 24,138th byte, there was at most single, least significant bit errors. Figure XI shows this.

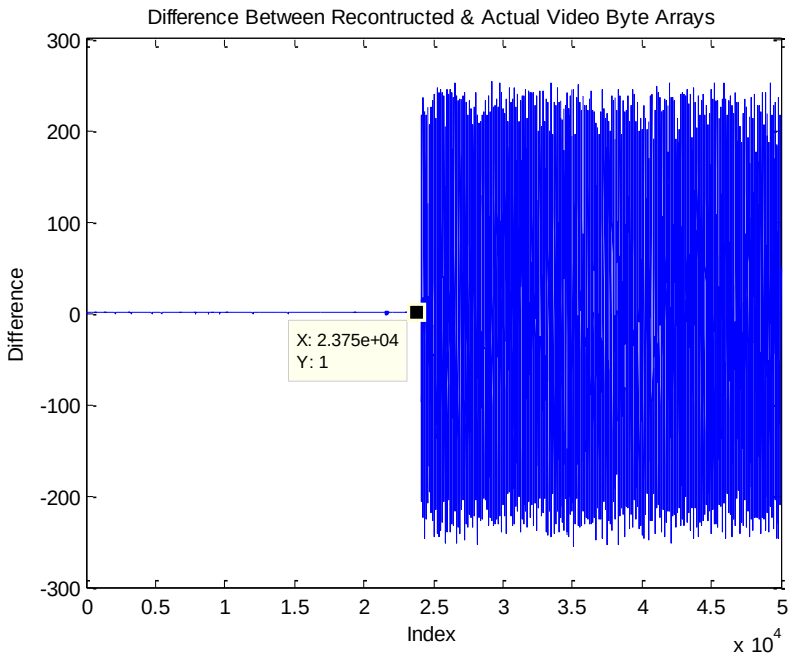


FIGURE XI

DIFFERENCE BETWEEN RECONSTRUCTED & ACTUAL BYTES

The 24,139th byte represents the appearance of an index offset, after which the array values no longer match up.

The primary cause of unreadable bytes is parasitic RC delays that increase the settling time of large code transitions to the point that two settled sample values are not detected before the output transition to a new value. Using the test file whose analog output is shown in Figure XII, it was determined that at code transitions of greater than about 50, byte detection became unreliable.

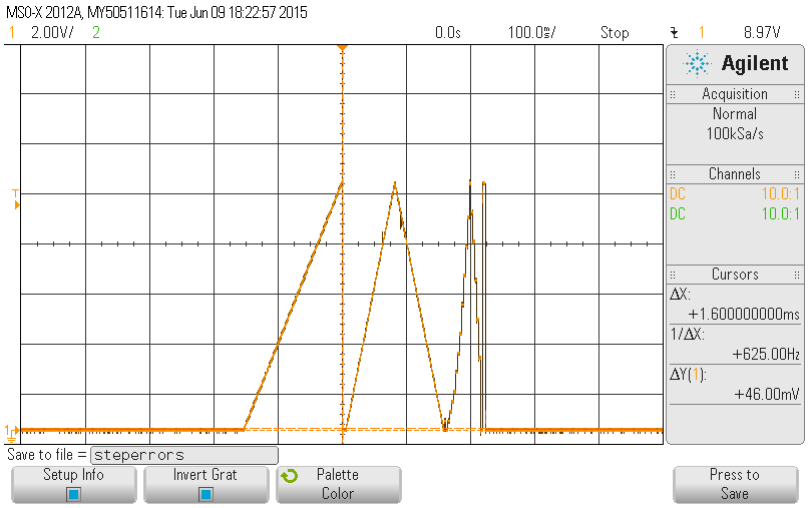


FIGURE XII

OSCILLOSCOPE CAPTURE OF TEST DATA USED TO DETERMINE SETTLING ERROR THRESHOLD

Because of the various errors present, the reconstructed file could not be opened by normal .mp4 players.

Summarizing, over 99% of the digital by bytes transmitted were recovered to within 99% accuracy; however, this is not sufficient for corruption free transmission of .mp4 files.

IX. Conclusions

The complete specifications were not met. Because of the noise and delay errors, the reconstructed, compressed video data file was corrupted and unreadable. In part this is due to the unforgiving nature of H264 compression, and compression in general, to errors. Compression involves the reduction of statistical redundancy in data, and reduction of redundancy also means a reduction in resilience to errors [19]. In its current state, the channel and reconstruction likely has sufficient accuracy for uncompressed video data to be sent and recovered with adequate quality. However, uncompressed video cannot have the specified video duration and resolution and still be transmitted in the specified time. In addition, although documentation to the contrary exists, the drivers for the Raspberry Pi Camera Module do not currently support 30 frames per second capture, the videos are actually captured at 25 fps (83% the specified speed) [20].

Despite its inability (in its current form) to meet all of the specifications simultaneously, this project serves as a solid proof of concept for the technology. In the trials discussed above, less than 0.2% of all bytes transmitted were unreadable, and of those that were readable almost all were within one least significant bit of the correct value. Moving forward with this project to a PCB design would most likely resolve the primary issues preventing the project from fully meeting all of the specifications. A well designed PCB will have significantly lower parasitic capacitances than a breadboard circuit, allowing the time until a settled value, to be decreased, potentially to less than 100 μs for a full 10 V swing. Furthermore, implementation of noise suppression techniques and methods for isolating analog components from digital noise, should greatly reduce the number of least significant bit errors. This combined with improved reconstruction algorithms could potentially remove them altogether, allowing for complete, uncorrupted reconstruction of the compressed video data and achievement of all of the specifications.

Specifications that the design currently fully meets are: capable of recording at between 0.5 and 3 seconds of video; at least 640x480 pixel image resolution, at 8 bits per pixel; sufficient memory to store captured data and to perform compression operations on the data; output data over analog line; sufficient video data compression such that sample transmission requires less than 10 minutes (minimum compression factor of 2 to 1); finished production units should cost no more than $200 (Note: the component costs of the final prototype are at less than $100); and use at least 70% ROHS compliant components. The 1-bit TTL-level record and transmit were not implemented in the final design for debug and testing reasons, but would be trivial to add to it.

Overall the project meets the majority of its specifications, and meets each of the remaining specifications by better than 80%, and there are clear routes that can improve the design until all of the specifications are met.

References

[1] Orbital Sciences Corporation, "Rocket Camera Proposal," *Orbital Sciences Corporation*, Unknown Date. [Online]. Available: Cal Poly Polylearn: EE460. [Accessed: Aug. 27, 2014].

[2] Orbital Sciences Corporation, "Rocket Camera Proposal – Revised 10/21/14," *Orbital Sciences Corporation*, Oct. 2014. Unpublished.

[3] M. Noonen, K. Deierling, K. Barraclough, and B. Martin, "Video compression and decompression arrangement having reconfigurable camera and low-bandwidth transmission capability," U.S. Patent 5 926 208, July 20, 1999.

[4] G.J. Sullivan and T. Wiegand , "Video Compression - From Concepts to the H.264/AVC Standard," Proceedings of the IEEE, vol.93, no.1, pp.18,31, Jan. 2005. *A*vailable: IEEE Xplore, http://www.ieee.org [Accessed: 10/20/14]

[5] Kyusam Lim; Jeonghun Kim; Hwangyoung So; Sejin Kang; Suki Kim, "A Low Power QXGA Camera Signal Processor for Mobile Camera Applications," International Conference on Consumer Electronics, 2007. ICCE 2007. Digest of Technical Papers., vol., no., pp.1,2, 10-14 Jan. 2007. Available: IEEE Xplore, http://www.ieee.org [Accessed: 10/20/14]

[6] M.L. Kaddachi, L. Makkaoui, A. Soudani, V. Lecuire, and J. Moureaux, "FPGA-based image compression for low-power Wireless Camera Sensor Networks," 3rd International Conference on Next Generation Networks and Services (NGNS), 2011, vol., no., pp.68,71, 18-20 Dec. 2011. Available: IEEE Xplore, http://www.ieee.org [Accessed: 10/20/14]

[7] D. Braun, "EE 460 Gantt Chart – Fall 2014," Cal Poly Courseware – EE. [Online]. Available: https://courseware.ee.calpoly.edu/~dbraun/courses/ee460/EE460Gantt.html. [Acessed: 10/27/14]

[8] I. Wahidah, A.B. Suksmono, Hendrawan, and T.L.R. Mengko, "Compressive sampling for digital video signal compression involving dynamic sparsity," 7th International Conference on Telecommunication Systems, Services, and Applications (TSSA), 2012, vol., no., pp.46,50, 30-31 Oct. 2012. Available: IEEE Xplore, http://www.ieee.org [Accessed: 10/20/14]

[9] Aptina Imaging, "1/4-Inch VGA Digital Sensor - MT9V011 Data Sheet," Aptina Imaging. [Online]. Available: Aptina, www.aptina.com . [Accessed: 11/15/14]

[10] Cypress Perform, "4-Mbit (256 K x 16) Static Ram - CY7C1041DV33 Data Sheet," Cypress Perform. [Online]. Available: Cypress, www.cypress.com. [Accessed: 11/15/14]

[11] Cypress Perform, "4-Mbit (256 K x 16) Static Ram," CY7C1041DV33 datasheet [Online]. Available: Cypress, www.cypress.com. [Accessed: 11/15/14]

[12] Analog Devices, "LC2MOS 12-Bit DACPORTs," AD7245A/AD7248A datasheet [Online] Available: Analog, www.analog.com. [Accessed: 6/12/15]

[13] Analog Devices, "LC2MOS 12-Bit Serial DACPORTs," AD7243 datasheet [Online] Available: Analog, www.analog.com. [Accessed: 6/12/15]

[14] Analog Devices, "12-Bit 100 kSPS A/D Converter," AD7243 datasheet [Online] Available: Analog, www.analog.com. [Accessed: 6/12/15]

[15] Linear Technology, "10kHz to 150kHz Digitally Controlled Antialiasing Filter and 4-Bit P.G.A," LTC1564 datasheet [Online] Available: Linear, www.linear.com. [Accessed: 6/12/15]

[16] Digilent, "PmodSF2 Reference Manual," [Online] Available http://www.digilentinc.com. [Accessed: 6/12/15]

[17] Numonyx, "Numonyx® OmneoTM P5Q PCM," P5Q PCM datasheet, [Online] Available, Digilent Inc, http://www.digilentinc.com. [Accessed: 6/12/15]

[18] E2e.ti.com, “SPI Speed on ARM microprocessors (Tiva for example) - TM4C Microcontrollers Forum - TM4C Microcontrollers - TI E2E Community,” 2015. [Online]. Available: http://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/p/274452/958695. [Accessed: 6/12/15].

[19] Elinux.org, “RPi Low-level peripherals - eLinux.org,” 2015. [Online]. Available: http://elinux.org/RPi_Low-level_peripherals#GPIO_Code_examples. [Accessed: 6/12/15].

[20] Wiringpi.com, 'WiringPi', 2015. [Online]. Available: http://wiringpi.com/. [Accessed: 6/12/15].

[21] Raspberrypi.org, “Raspberry Pi Forums - DMA and SPI and Interrupts - cant get it to work,” 2015. [Online]. Available: https://www.raspberrypi.org/forums/viewtopic.php?f=71&t=19797. [Accessed: 6/12/15].

[22] Raspberrypi.org, “Raspberry Pi Forums - DMA on GPIO,” 2015. [Online]. Available: https://www.raspberrypi.org/forums/viewtopic.php?f=44&t=8376. [Accessed: 6/12/15].

[23] Wallacoloo, “Wallacoloo/Raspberry-Pi-DMA-Example,” GitHub, 2014. [Online]. Available: https://github.com/Wallacoloo/Raspberry-Pi-DMA-Example/blob/master/dma-gpio.c. [Accessed: 6/12/15].

[24] J. Pihlajamaa, 'Benchmarking Raspberry Pi GPIO Speed | Code and Life', Codeandlife.com, 2012. [Online]. Available: http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/. [Accessed: 6/12/15].

[25] Raspberrypi.stackexchange.com, 'How to compile C files in terminal', 2015. [Online]. Available: http://raspberrypi.stackexchange.com/questions/5599/how-to-compile-c-files-in-terminal. [Accessed: 6/12/15].

[26] Raspberrypi.org, “Raspberry Pi Camera Module -Applications,” Raspberry Pi Documentation 2015. [Online]. Available: https://www.raspberrypi.org/documentation/raspbian/applications/camera.md. [Accessed: 6/12/15].

[27] Raspberrypi.org, “Camera - Hardware,” Raspberry Pi Documentation 2015. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera.md. [Accessed: 6/12/15].

[28]A. Allain, '”C File I/O Tutorial - Cprogramming.com.” Cprogramming.com, 2015. [Online]. Available: http://www.cprogramming.com/. [Accessed: 6/12/15].

[29] Codingunit.com, “C Tutorial – Binary File I/O | CodingUnit Programming Tutorials,” 2015. [Online]. Available: http://www.codingunit.com/c-tutorial-binary-file-io. [Accessed: 6/12/15].

[30] M. Samek, “Are We Shooting Ourselves in the Foot with Stack Overflow?”, Embeddedgurus.com, 2015. [Online]. Available: http://embeddedgurus.com/state-space/2014/02/are-we-shooting-ourselves-in-the-foot-with-stack-overflow. [Accessed: 6/12/15].

[31] N. Jones, “Efficient C Tip #13 – use the modulus (%) operator with caution,” Embeddedgurus.com, 2015. [Online]. Available: http://embeddedgurus.com/stack-overflow/2011/02/efficient-c-tip-13-use-the-modulus-operator-with-caution/. [Accessed: 6/12/15].

[32] Oshpark, “OSH Park ~ Pricing and Specifications,” 2015. [Online]. Available: https://oshpark.com/pricing. [Accessed: 6/12/15].

[33] R. Gonzalez, and R Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River: Prentice Hall, 2007.

Appendix A – Analysis of Senior Project Design

Project Title: Rocket Cam: Low Frequency Analog Transmission of Digital Video

Student's Name: Thomas Higdon

Advisor's Name: Professor Wayne Pilkington

1. Summary of Functional Requirements

The design's overall function inputs visible light video image data, and then to outputs an analog voltage representation of that data. The module also accepts two commands, record and transmit. Upon receipt of the record command, the module captures and stores 3 seconds of video data. Upon receipt of the transmit command, the module sends that data over the analog data line [1],[2].

2. Primary Constraints

The primary challenges came from the need to transmit the video data over a narrow bandwidth analog data channel. The available channel is sampled at 10,000 samples per second (at most), with 12 bit resolution. Thus, the primary challenges to overcome in order to meet product specifications involved: devising an effective analog encoding scheme within this bandwidth; compressing the data so that it takes a short enough duration to transmit (sending entire sample within mean flight time; keeping the noise from the module components off of the analog channel; maximizing the signals resilience to noise; and handling bit errors that occur. A supplemental challenge was establishing reliable communications between components.

3. Economic

The product's design and creation utilized multiple forms of capital. In human terms, a variety of individuals have contributed to the project's success (discussed further in terms of direct and indirect stakeholders in the Social and Political section). In addition, the project made use of third party services such as major components vendors, component manufacturers, and delivery infrastructure. In this way, the project utilizes the existing infrastructure of these organizations, rather than necessitating the construction of entirely new facilities. Lastly, the materials used to manufacture the product come from the planet itself, as does the energy used in the manufacturing process and in operation.

The direct monetary investment in the project, for physical component and equipment purchases, rounds to approximately $200. However, the equivalent monetary value of the labor provided in the development process approaches $8,000 dollars. Tables A1 and A2 below, detail the preliminary planning estimates of costs for components and labor (estimated using $30/hour). The formula, *estimated = (min + 4*typical + max)/6,* yields the PERT values listed. Further below, Tables A3 and A4 detail the actual final costs for part and labor.

TABLE A1

ESTIMATED PARTS AND EQUIPMENT COSTS

Estimated Development Costs (Equipment)				
Final Prototype				
System	*Min*	*Typ*	*Max*	*PERT*
Camera	$5	$10	$40	$ 14.17
Controller	$10	$25	$80	$ 31.67
Data Storage	$2	$8	$30	$ 10.67
DAC	$2	$10	$20	$ 10.33
Misc. Passive	$1	$5	$20	$ 6.83
Power Systems	$4	$6	$25	$ 8.83
PCB	$40	$75	$150	$ 81.67
Total -	$64	$139	$365	$164
Development Cycle's 1 and 2 (Not PCB Version)				
System	*Min*	*Typ*	*Max*	*PERT*
Total	$24	$64	$215	$ 82.50
Development and Test Equipment				
System	*Min*	*Typ*	*Max*	*PERT*
Total	$20	$35	$120	$ 82.50
Equip. Total	*Min*	*Typ*	*Max*	*PERT*
	$108	$238	$700	$329

TABLE A2

ESTIMATED LABOR COSTS

Estimated Development Costs (Labor)				
Labor (Weekly)				
Person	*Min*	*Typ*	*Max*	*PERT*
Myself (hrs)	8	18	30	18.3
Myself ($)	$ 240.00	$ 540.00	$ 900.00	$ 550.00
Labor (Total)				
Person	*Min*	*Typ*	*Max*	*PERT*
Myself (hrs)	160	360	600	366.7
Myself ($)	$4,800	$10,800	$18,000	$11,000
Others (hrs)	10	15	40	18.3
Myself ($)	$300	$450	$1,200	$549
Grand Total	*Min*	*Typ*	*Max*	*PERT*
	$5,100	$11,250	$19,200	$11,549

These estimated costs corresponded to the planned development cycle illustrated in the Gantt charts in Tables A5 (a), A5 (b), & A5 (c).

Tables A3 and A4 detail the actual final costs for part and labor (approximated using an updated $35/hour) for the project. The parts whose costs are listed at N/A, are ones which were already owned before the start of the project, but are important enough to the final prototype, to warrant explicit mentioning in the table. Components and devices used for early prototyping, and for development and testing, but not purchased specifically for the project are not listed.

TABLE A3

ACTUAL PARTS AND EQUIPMENT COSTS

Actual Development Costs (Equipment)					
Final Prototype					
System	*Part Name/Number*	*Vender*	*Manufacturer*	*Base Price*	*Cost (w/ Tax)*
Camera	Raspeberry Pi 5MP Camera	Amazon	Raspberry Pi Foun.	$ 26.95	$ 29.38
Video Data Processor	Raspberry Pi - Model B+	MCM Elec.	Raspberry Pi Foun.	$ 25.00	N/A
Memory	16 GB Micro SD Card	Amazon	Kingston Digital	$ 6.95	N/A
DAC	AD7245AANZ	DigiKey	Analog Devices	$ 19.37	$ 20.92
D/A Control Unit	Tiva C TM4C123GXL Eval	DigiKey	Texas Instruments	$ 16.68	$ 18.01
Misc. Passive	Caps/Resistors/Leads	N/A	N/A	$ 2.00	N/A
			Total -	$ 96.95	$68
Early Prototype					
System	*Part Name*	*Vender*	*Manufacturer*	*Base Price*	*Cost (w/ Tax)*
Camera	OV7670 300KP VGA Module	Amazon	Atomic Market	$ 10.99	$ 10.99
Memory	PmodSF2 - Serial PCM	Amazon	Digilent	$ 14.95	$ 14.95
DAC	AD7243ANZ	DigiKey	Analog Devices	$ 16.68	$ 16.68
			Total:	$ 42.62	$ 42.62
Development and Test Equipment					
Function	*Part Name*	*Vender*	*Manufacturer*	*Base Price*	*Cost (w/ Tax)*
ADC	AD1674JNZ	DigiKey	Analog Devices	$ 29.17	$ 31.50
USB-UART Cable	768-1204	DigiKey	FTDI	$ 15.00	$ 16.20
Antialiasing Filter	LTC1564IG#PBF	DigiKey	Linear Technologies	$ 17.59	$ 19.00
Breakout BNC	FEMALE-20AWG LEADS	DigiKey	Pomona Electronics	$ 17.10	$ 18.47
			Total:	$ 78.86	$ 85.17
	Shipping and Handing:				
Total	$16.50		**Equip. Total**	*Base Price*	*Actual Cost*
			S&H: $16.50	$ 218.43	$ 196.10

TABLE A4

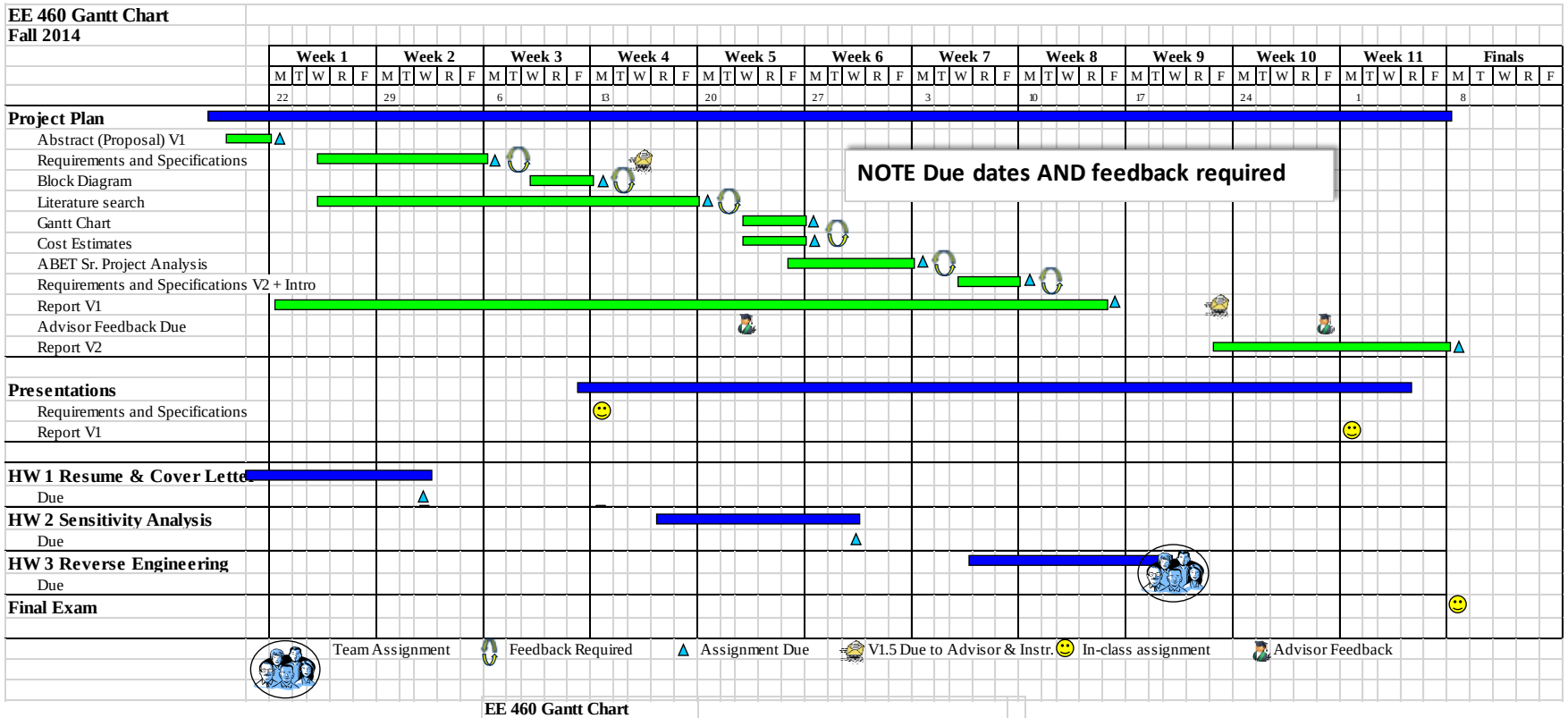
ACTUAL (APPROX.) LABOR COSTS

Approximant Development Costs (Labor)				
Labor (Weekly Averaged)				
Person	*Min*	*Typ*	*Max*	*PERT*
Myself (hrs)	8	10	15	10.5
Myself ($)	$ 280.00	$ 350.00	$ 525.00	$ 367.50
Labor (Total)				
Person	*Min*	*Typ*	*Max*	*PERT*
Myself (hrs)	160	200	300	210.0
Myself ($)	$5,600	$7,000	$10,500	$7,350
Others (hrs)	12	18	25	18.2
Others ($)	$420	$630	$875	$636
Grand Total	*Min*	*Typ*	*Max*	*PERT*
	$6,020	$7,630	$11,375	$7,986

The initial planning timeline for project development is shown in Tables A5 (a), A5 (b), & A5 (c) below.

TABLE A5 (a)

PLANNING GANTT CHART – FALL QUARTER AND WINTER BREAK 2014 [21]



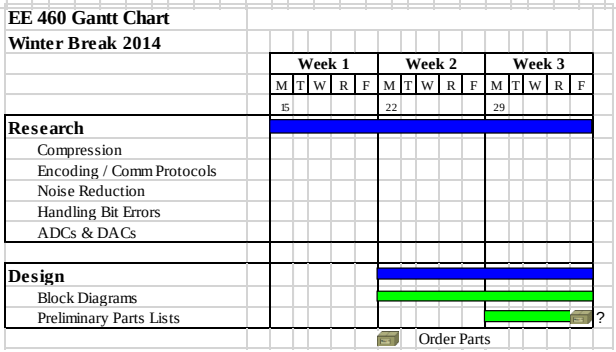


TABLE A5 (b)

PLANNING GANTT CHART –WINTER 2015

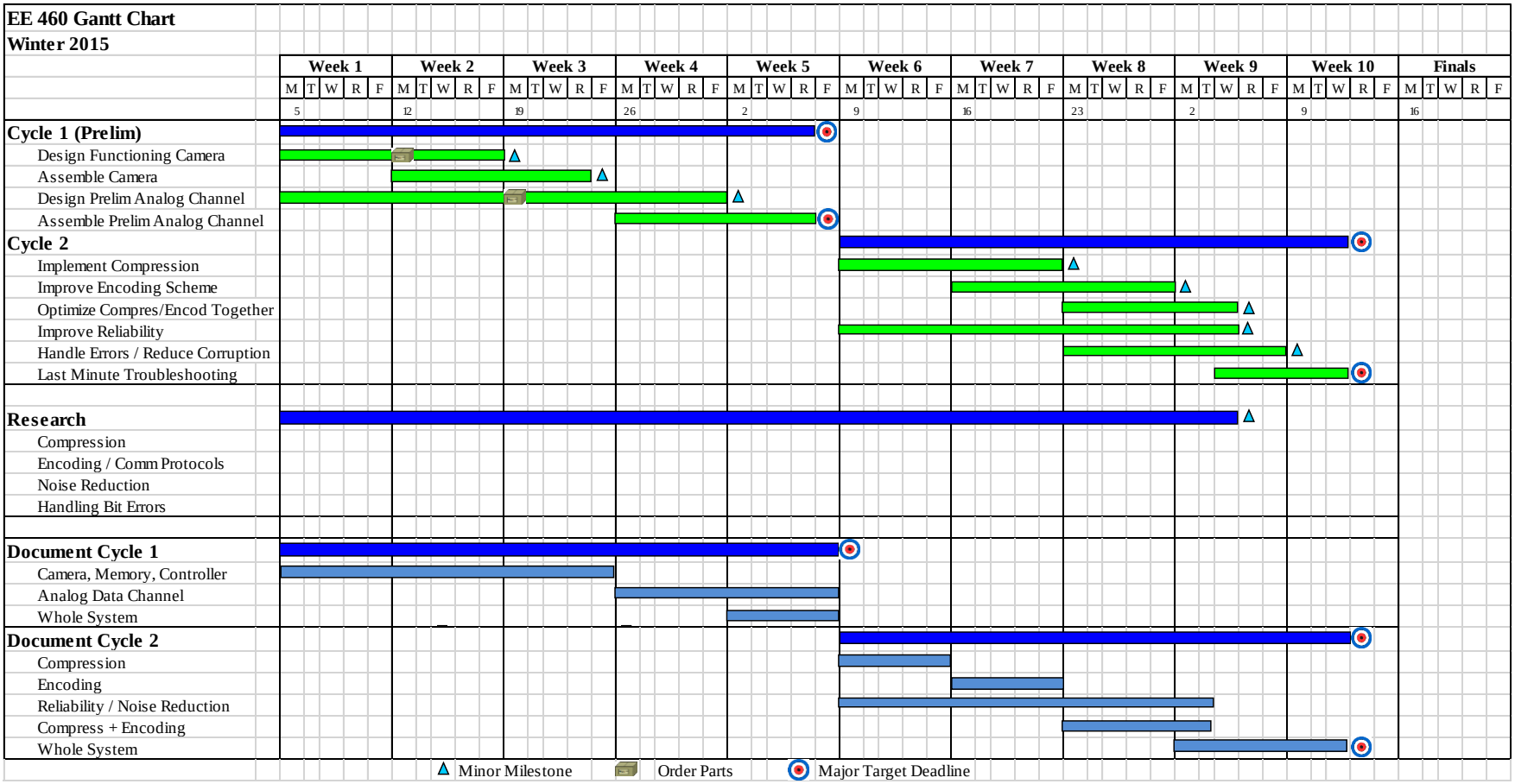
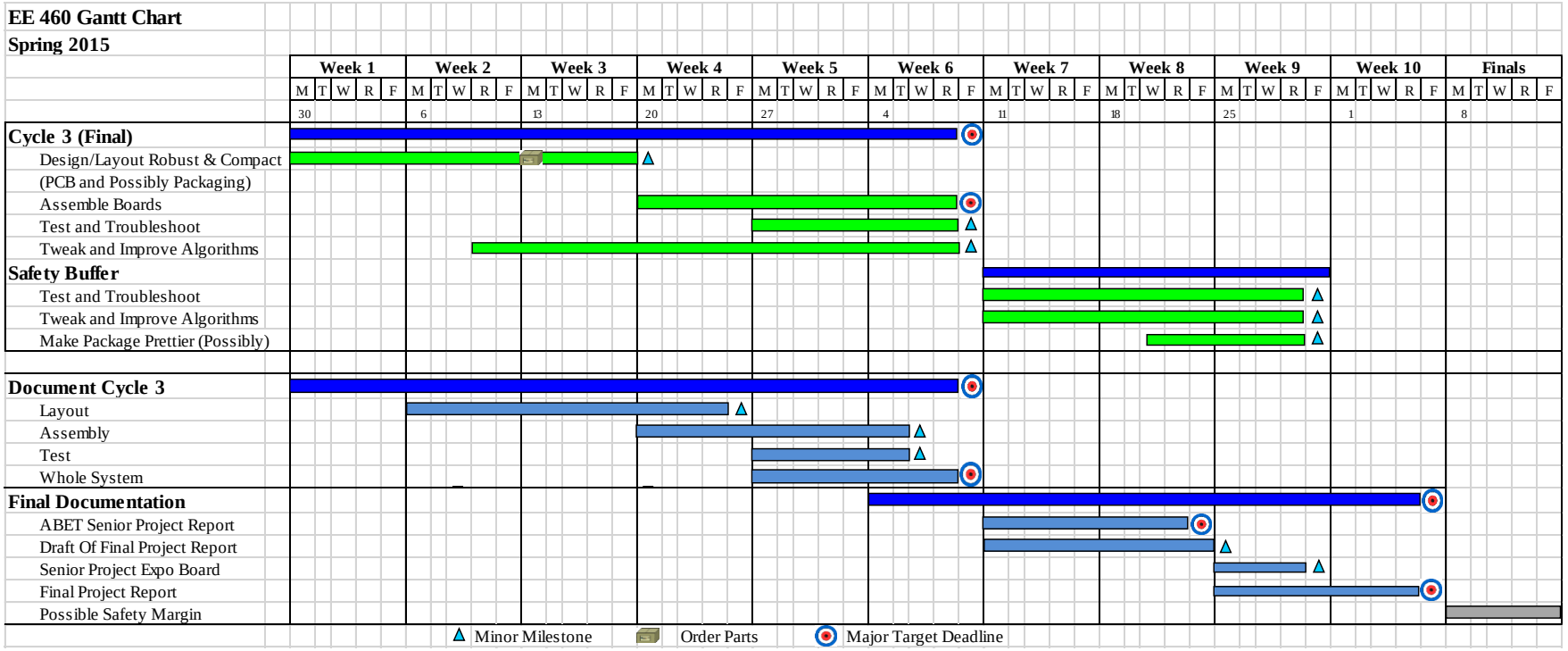


TABLE A5 (c)

PLANNING GANTT CHART – SPRING 2015

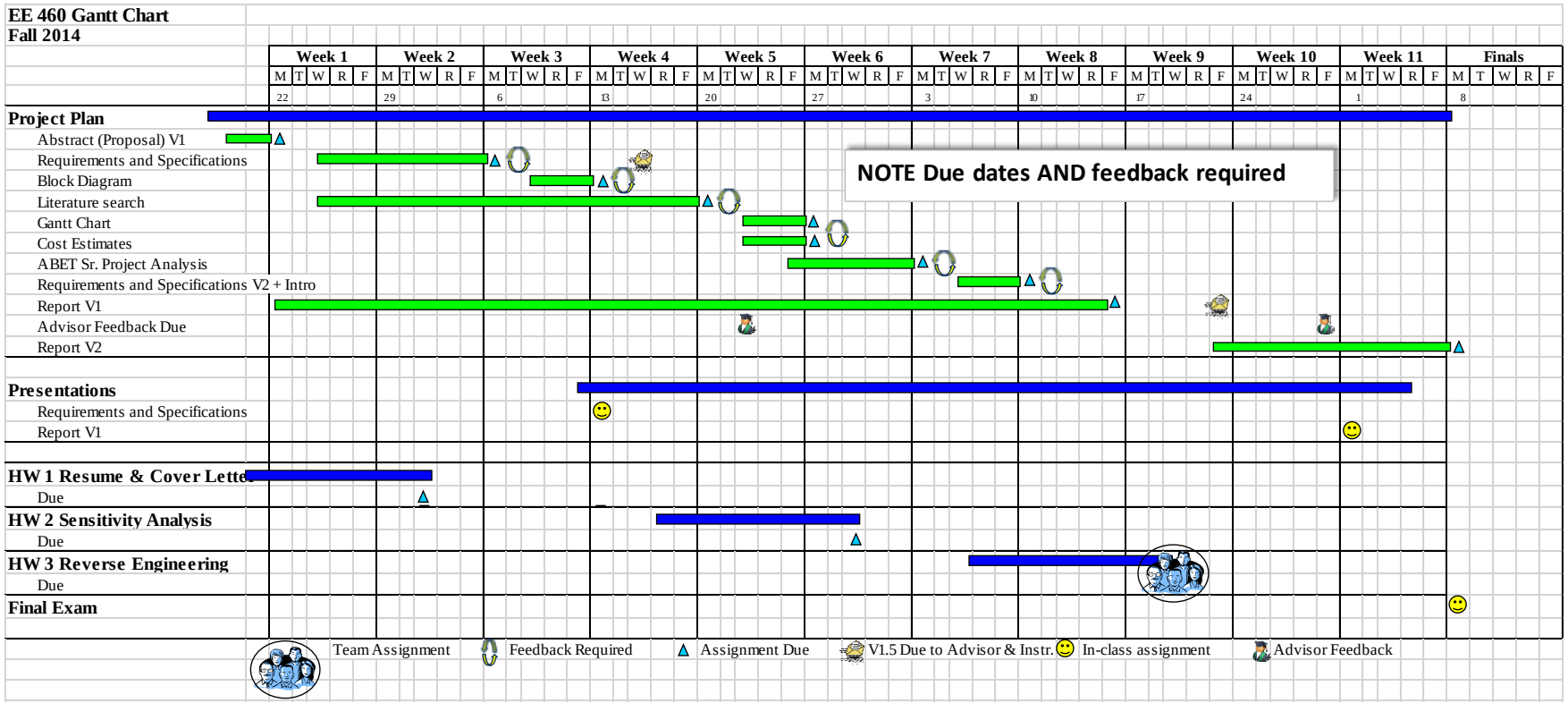


In the initial timeline planning, the first design-build-test cycle corresponds to implementing a basic version of the design on a breadboard. The second design-build-test cycle involves creating an improved version on perf-board. And the last cycle implements a final version of the design on a printed circuit board and with included packaging.

Tables A6 (a), A6 (b), and A6 (c) contain the timeline that the development actually followed.

TABLE A6 (a)

ACTUAL DEVELOPMENT GANTT CHART – FALL QUARTER AND WINTER BREAK 2014 [21]



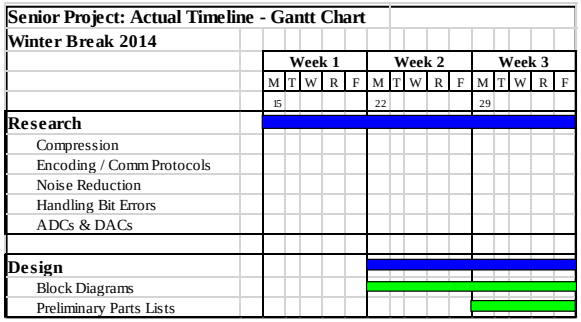


TABLE A6 (b)

ACTUAL DEVELOPMENT GANTT CHART –WINTER 2015

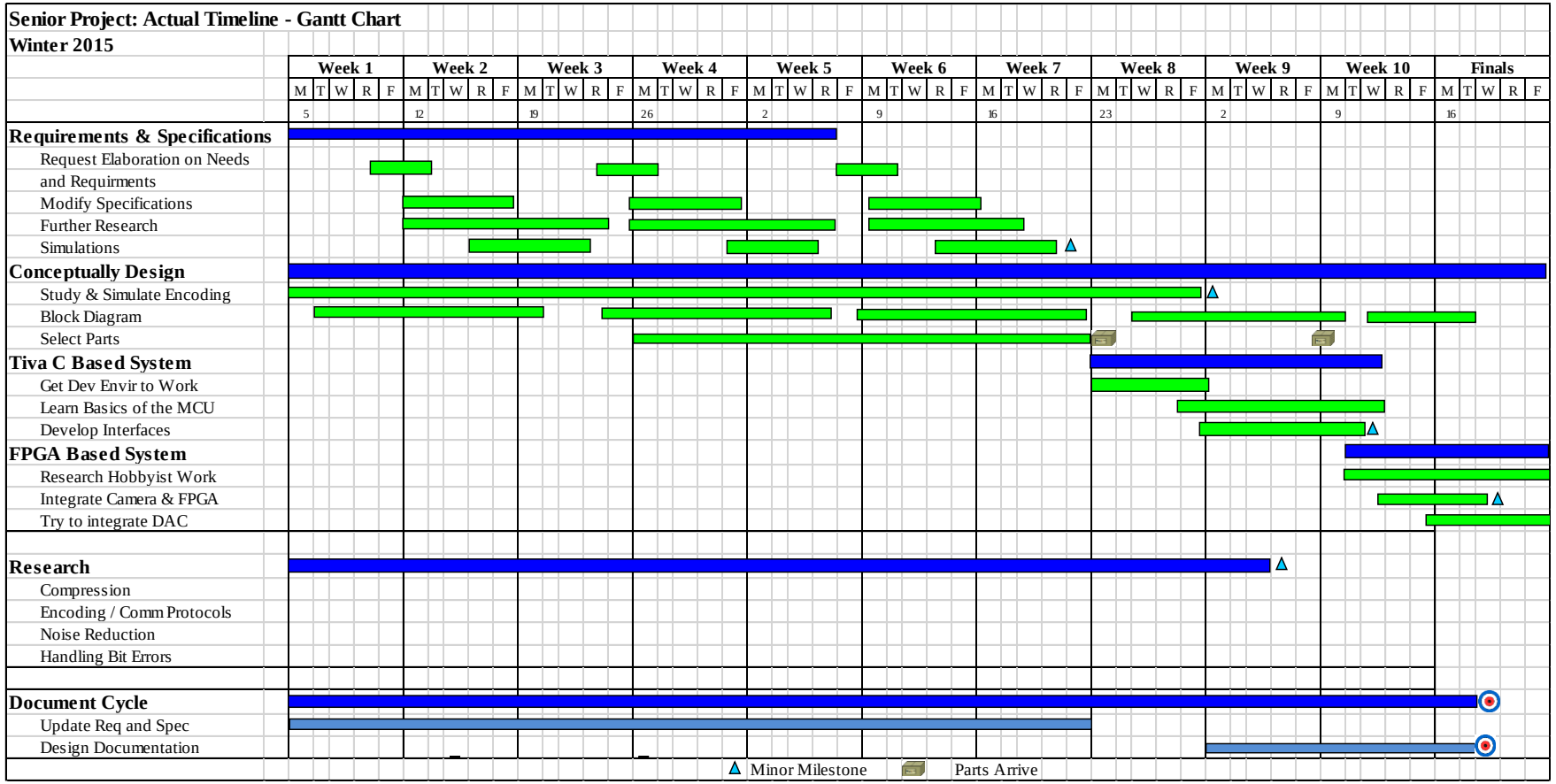
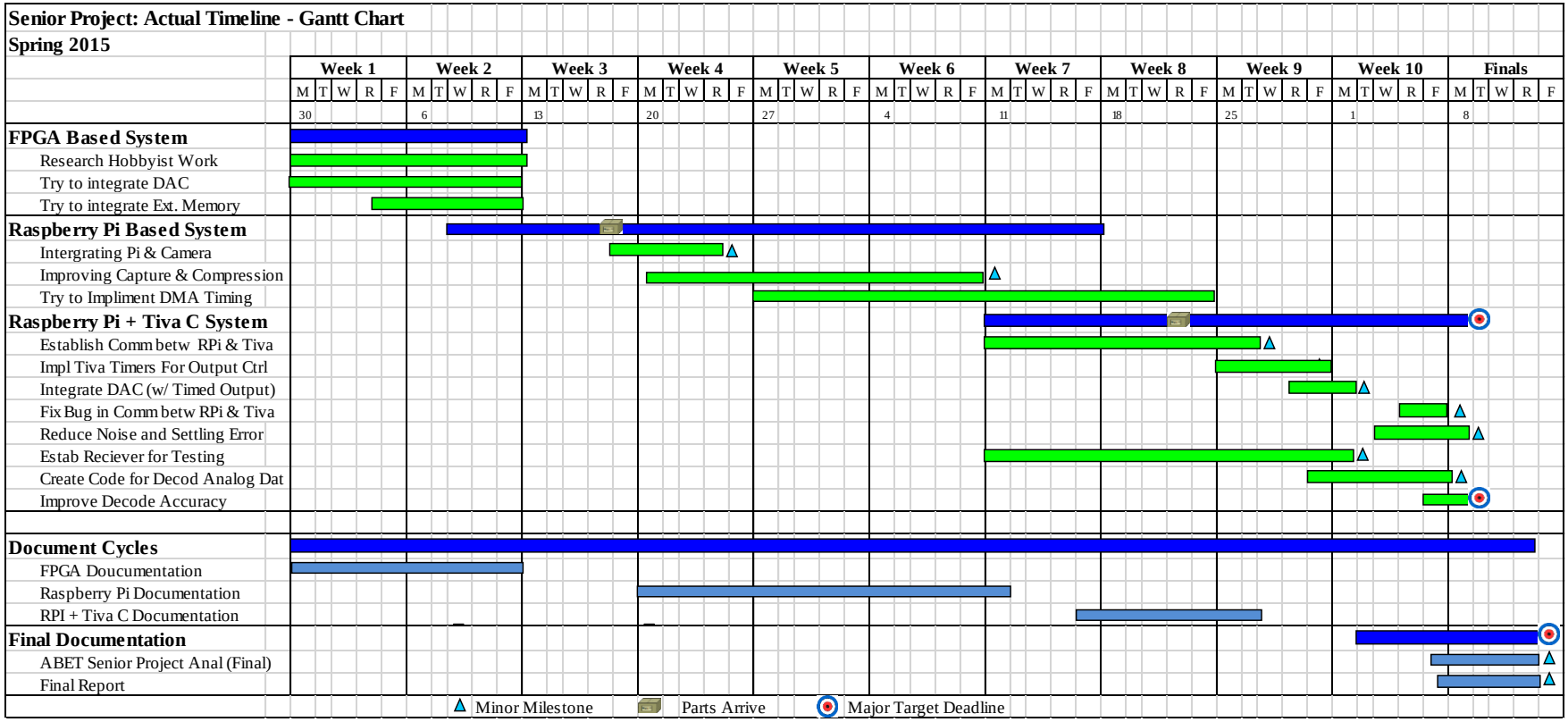


TABLE A6 (c)

ACTUAL DEVELOPMENT GANTT CHART – SPRING 2015



4. If manufactured on a commercial basis

The project has a small direct market, consisting of Orbital ATK itself. The market does have the possibility for expansion to any customers who need a camera module capable of recording and transmitting buffered video data over a low-bandwidth analog data channel. However, considering only Orbital ATK's business, we estimate the possibility of 10-20 rocket launches in the next 10 years, which utilize an average of 3-4 of the modules (corresponding to multiple stages and payload deployments). Assuming the purchase of a few spare units, this brings the total number of modules to the order of 60 units. This averages to 6 units per year. Estimating a manufacturing cost, for a finished unit, of $130-170 ($70-80 in components, $15-30 for OSH Park PCB, and $45-60 assembly). Setting a corresponding $180-220 price point (a $50 profit-margin per unit), this means an estimated $300 dollars of profit per year.

The primary expected cost for the user, after purchase, comes from the installation of the module as well as any required modifications to their existing systems to incorporate the module. Since the module is specifically intended to take minimal effort to integrate into their existing systems, , we expect minimal modification costs. The module itself, however, may require application specific outer packaging ($40-100). The other primary costs come from the labor to install the module, which we estimate as around $60 (2 hours of work, mostly spent verifying and triple checking, at $30 an hour). The module does also draw power, however, a commercial version's power requirement should be low compared to the existing systems on the rocket. In addition, costs associated with the fuel burned to lift the module's weight exist, but compared to the weight of the payload and the rocket itself, the impact is essentially negligible. Unfortunately, all of these costs accrue on a single launch, with little chance of reuse for the module, both because of the harsh operating environment it faces and because Orbital ATK does not directly reuse components after a mission, due to reliability and safety issues.

5. Environmental

Materials that come from limited natural resources compose the product. A few examples of materials used in some fashion include: laminates & resins (PCBs); copper (PCB & wires/cables); silicon (ICs); polymers (IC packages); ceramics (discrete capacitors and resistors); aluminum (metal layers in ICs); tin (solder); and miscellaneous trace heavy metals. The manufacturing processes, both of the module and of its components, also use energy, which may come from any of the typical sources, fossil fuels, nuclear, solar, wind, hydro-electric, etc. In addition, the component manufacture processes also inevitably produce wastes, which, if not properly disposed of or recycled, could potentially get released into the environment.

Unfortunately, if the customer does not take care to properly recycle or dispose of the product, at the end of its lifecycle it could end up as waste or litter. Coupled with its short life cycle (Orbital ATK does not directly reuse components after a mission, due to reliability and safety issues) and manufacturing wastes, this could have a negative impact on other species in the environment. Hopefully, however, the use of 100% ROHS compliant components, the limited quantity produced, and any steps that the customer may take towards properly disposing of the product, should help mitigate the short term and long term environmental impacts.

6. Manufacturability

The design uses all readily available, "off-the-shelf" components. Thus the primary manufacturing challenges consist of the fabrication of the PCB board, and assembling of the components into the completed module (potentially very time consuming), and the programing of the control unit (also costs some time). But perhaps most significantly, the manufacture of the mechanical packaging of the module, and the assembly of the electrical components into the finished package, presents a major production challenge.

7. Sustainability

Unfortunately, the camera operates in a harsh, destructive environment, which makes maintaining the completed device exceedingly difficult. In addition, since reliability constitutes a critical safety issue, Orbital ATK does not directly reuse rocket systems. Thus sustainable use of project resources focuses on proper disposal and recycling of the design at the end of its mission. The limited production quantities help to also improve the sustainability of the overall product.

Improvements or upgrades could potentially include lower power consumption, reduced manufacture and component costs, improved reliability/survivability, additional features that could broaden the potential market (perhaps alternative data channels, and the ability to transmit in real time), and reduction in environmentally harmful components (increase in percentage of ROHS compliant components). The first major upgrade should be implementation on a single, custom PCB board. This will allow for improved reliability, smaller size / lower weight, lower power consumption, and reduced noise. Then improved decoding algorithms can be tuned to utilize the improved noise and delay characteristics of the single board. However, these upgrades involve a significant increase in design time and cost, as it necessitates a lower level focus on the requirements of the processors used, and it entails the higher cost of custom PCBs.

8. Ethical

Considering the project from the framework of utilitarianism, we arrive at the following conclusions. Though the manufacture of the project produces waste and the product itself (if not properly disposed of) may become waste, the target application provides data for modelling and troubleshooting dynamic events on large scale rocket systems. A failure during a dynamic event, could lead to severe repercussions for the rocket system, and if it causes an overall failure of the rocket, this could lead to significant littering and environmental harm, and possibly even risk to human life. Thus if improved models resulting from the use of this module help to prevent failures during dynamic events, then I would argue that the benefit in reducing the risk of large scale damage to the environment outweighs the relatively small harm the production wastes constitute. In addition, reducing the risk of a failure reduces the risk to both human health/safety, and the risk of financial harm, both to Orbital ATK and to their customers (who often pay Orbital ATK to deliver their products to space). From the framework of utilitarianism, creating this product performs greater good than not creating it.

From the framework of ethical egoism, the creation of this senior project helped the student (myself) to gain valuable experience and skills. Both technical skills and project management/develop experience, will help me to perform better engineering, by better meeting customer requirements, and by creating more reliable, sustainable, and beneficial designs in the future.

These arguments are also consistent with the IEEE Code of Ethics. The module provides data for improving the understanding of dynamic events and the technologies used in them, which directly ties into the 5th point of the code. Improvement of these models could help reduce the risk to public health, welfare, and property (points 1. and 8.). Lastly, it gave the designer the opportunity to develop and improve technical competence in several areas (point 6.)

Like any other technology, people could potentially misuse the product for unethical purposes. The module suffers from similar possible misuses to those other camera technologies face (such as “spying”), however, the product does not lend itself well to these misuses. The limited video duration and the narrow bandwidth data channel would hinder, or at least not provide advantage, for most unethical applications I can think of. There already exist a plentiful number of camera modules available that exhibit both lower cost and better suited features for illicit purposes. Thus the chance of unethical misuse of this project remains minimal.

9. Health and Safety

There always exists health and safety concerns associated with any product. During the design and testing process, the designer had to take care to avoid contact with any potentially harmful substances, and ensure that no one else came into contact with them. In addition, the designer had to take care to avoid burns, cuts, and electrocution. Assembly of the completed project features the same safety concerns. The manufacture of the components incorporated also have their associated health concerns, however, the manufacturers from which we purchase the parts deal with these themselves, via established procedures. Use of the design involves the same risks from dealing with the power source connected to the design. The module itself draws low enough power and current so that it's not directly dangerous, however, damage or defect could cause an unintended short and dangerous amounts of current to flow. The design does use 100% ROHS compliant parts, which reduces the risk from toxicity. Installation into a larger system involves the same risks/concerns as any other work on the larger system, and as well as risks stemming directly from the act of installing (for instance, jabbing your hand with a screwdriver, getting your fingers pinched, or cutting your hand).

10. Social and Political

Various individuals have contributed to the project's success, including the project's advisor, Professor Wayne Pilkington, and the company liaison, Johnathan Chan. In addition, the project made use of third party services such as major components vendors, component manufacturers, delivery infrastructure, and caffeine providers. Inside of each of these organizations, a number of individuals worked to complete the contracted tasks. The organizations also inevitably made use of various other suppliers and subcontractors of their own. Lastly, numerous other persons provided advice on developmental challenges of the project, when consulted by the project designer. These persons range from engineering faculty and students, to friends and family of the lead designer, and a generous responder on a help forum. All of the individuals in the supplier, contractor, and subcontractor organizations benefit in a small way from business received from this project. Johnathan Chan and other Orbital ATK employees benefit from a module that they can potentially use or modify to use with their products.

A negative impacted on an indirect stakeholder could include if someone (or creature) is harmed by wastes due to manufacture or improper disposal of the product.

11. Development

During the course of the project, I learned about a number of fields in which I previously had little background. I studied lossy video compression schemes, analog data encoding schemes, and how to reduce noise & bit errors. I learned how to utilize Linux terminal commands and shell scripting, as well as how to write and implement C code in a Linux environment. I experienced using ARM based microcontrollers, making use of (and debugging) manufacturer provided driver libraries, and adjusting optimization and compiler (in particular, stack size) settings. And I gained practice using Matlab for large scale data manipulation, algorithm prototyping, and conceptual simulations.

Please see the Literature Search section for resources I used while developing this knowledge.

Literature Search

[1] Orbital Sciences Corporation, "Rocket Camera Proposal," *Orbital Sciences Corporation*, Unknown Date. [Online]. Available: Cal Poly Polylearn: EE460. [Accessed: Aug. 27, 2014].

[2] Orbital Sciences Corporation, "Rocket Camera Proposal – Revised 10/21/14," *Orbital Sciences Corporation*, Oct. 2014. Unpublished.

[3] M. Noonen, K. Deierling, K. Barraclough, and B. Martin, "Video compression and decompression arrangement having reconfigurable camera and low-bandwidth transmission capability," U.S. Patent 5 926 208, July 20, 1999.

[4] G.J. Sullivan and T. Wiegand , "Video Compression - From Concepts to the H.264/AVC Standard," Proceedings of the IEEE, vol.93, no.1, pp.18,31, Jan. 2005. *A*vailable: IEEE Xplore, http://www.ieee.org [Accessed: 10/20/14]

[5] Kyusam Lim; Jeonghun Kim; Hwangyoung So; Sejin Kang; Suki Kim, "A Low Power QXGA Camera Signal Processor for Mobile Camera Applications," International Conference on Consumer Electronics, 2007. ICCE 2007. Digest of Technical Papers., vol., no., pp.1,2, 10-14 Jan. 2007. Available: IEEE Xplore, http://www.ieee.org [Accessed: 10/20/14]

[6] M.L. Kaddachi, L. Makkaoui, A. Soudani, V. Lecuire, and J. Moureaux, "FPGA-based image compression for low-power Wireless Camera Sensor Networks," 3rd International Conference on Next Generation Networks and Services (NGNS), 2011, vol., no., pp.68,71, 18-20 Dec. 2011. Available: IEEE Xplore, http://www.ieee.org [Accessed: 10/20/14]

[7] Elinux.org, "RPi Low-level peripherals - eLinux.org," 2015. [Online]. Available: http://elinux.org/RPi_Low-level_peripherals#GPIO_Code_examples. [Accessed: 6/12/15].

[8] Analog Devices, "Data Converter Fundamentals Training Module - Analog Devices | DigiKey," 2015. [Online]. Available: http://www.digikey.com/en/ptm/a/analog-devices/data-converters-data-converter-fundamentals?WT.pn_sku=AD5601BKSZ-REEL7TR-ND&WT.z_part_id=760552&WT.z_video_source=Part_Detail. [Accessed: 6/12/15].

[9] Texas Instruments, "Connected LaunchPad Kits," Texas Instruments, 2015. [Online]. Available: http://www.ti.com/ww/en/launchpad/launchpads-connected.html#tabs. [Accessed: 6/12/15].

[10] M. Field, "OV7670 camera - Hamsterworks Wiki!," Hamsterworks.co.nz, 2015. [Online]. Available: http://hamsterworks.co.nz/mediawiki/index.php/OV7670_camera#The_camera.27s_interface. [Accessed: 6/12/15].

[11] Raspberry-projects.com, 'BCM2835 SoC – Raspberry Pi Projects', 2015. [Online]. Available: http://www.raspberry-projects.com/pi/pi-hardware/bcm2835. [Accessed: 6/12/15].

[12] Texas Instruments, "Tiva TM4C123GH6PM Microcontroller," TM4C123GH6PM datasheet, [Online] Available, www.ti.com. [Accessed: 2/27/15]

[13] Analog Devices, "LC2MOS 12-Bit DACPORTs," AD7245A/AD7248A datasheet [Online] Available: Analog, www.analog.com. [Accessed: 6/12/15]

[14] Raspberrypi-spy.co.uk, 'Raspberry Pi B+ GPIO Header Details And Pinout', 2014. [Online]. Available: http://www.raspberrypi-spy.co.uk/2014/07/raspberry-pi-b-gpio-header-details-and-pinout/. [Accessed: 6/12/15].

[15] Wiringpi.com, 'WiringPi', 2015. [Online]. Available: http://wiringpi.com/. [Accessed: 6/12/15].

[16] A. Eames, 'Another way to convert Raspberry Pi Camera .h264 output to .mp4', RasPi.TV, 2013. [Online]. Available: http://raspi.tv/2013/another-way-to-convert-raspberry-pi-camera-h264-output-to-mp4. [Accessed: 6/12/15].

[17] Digilent, "WaveForms SDK Reference Manual," Jan 26, 2015 [Online] Available http://www.digilentinc.com. [Accessed: 5/16/15]

[18] Digilent Forum, “Using ScanShift Acquisition Mode in WaveForms SDK,” 2015. [Online]. Available: https://forum.digilentinc.com/topic/613-using-scanshift-acquisition-mode-in-waveforms-sdk/. [Accessed: 6/12/15].

[19] R. Gonzalez, and R Woods, *Digital Image Processing*, 3rd ed. Upper Saddle River: Prentice Hall, 2007.

[20] Raspberrypi.org, “Raspberry Pi Camera Module -Applications,” Raspberry Pi Documentation 2015. [Online]. Available: https://www.raspberrypi.org/documentation/raspbian/applications/camera.md. [Accessed: 6/12/15].

[21] D. Braun, “EE 460 Gantt Chart – Fall 2014,” Cal Poly Courseware – EE. [Online]. Available: https://courseware.ee.calpoly.edu/~dbraun/courses/ee460/EE460Gantt.html. [Acessed: 10/27/14]

[22] I. Wahidah, A.B. Suksmono, Hendrawan, and T.L.R. Mengko, "Compressive sampling for digital video signal compression involving dynamic sparsity," 7th International Conference on Telecommunication Systems, Services, and Applications (TSSA), 2012, vol., no., pp.46,50, 30-31 Oct. 2012. Available: IEEE Xplore, http://www.ieee.org [Accessed: 10/20/14]

[23] Analog Devices, “LC2MOS 12-Bit Serial DACPORTs,” AD7243 datasheet [Online] Available: Analog, www.analog.com. [Accessed: 6/12/15]

[24] Analog Devices, “12-Bit 100 kSPS A/D Converter,” AD7243 datasheet [Online] Available: Analog, www.analog.com. [Accessed: 6/12/15]

[25] Linear Technology, “10kHz to 150kHz Digitally Controlled Antialiasing Filter and 4-Bit P.G.A,” LTC1564 datasheet [Online] Available: Linear, www.linear.com. [Accessed: 6/12/15]

[26] Digilent, “PmodSF2 Reference Manual,” [Online] Available http://www.digilentinc.com. [Accessed: 6/12/15]

[27] Numonyx, “Numonyx® OmneoTM P5Q PCM,” P5Q PCM datasheet, [Online] Available, Digilent Inc, http://www.digilentinc.com. [Accessed: 6/12/15]

[28] E2e.ti.com, “SPI Speed on ARM microprocessors (Tiva for example) - TM4C Microcontrollers Forum - TM4C Microcontrollers - TI E2E Community,” 2015. [Online]. Available: http://e2e.ti.com/support/microcontrollers/tiva_arm/f/908/p/274452/958695. [Accessed: 6/12/15].

[29] Wiringpi.com, 'WiringPi', 2015. [Online]. Available: http://wiringpi.com/. [Accessed: 6/12/15].

[30] Raspberrypi.org, “Raspberry Pi Forums - DMA and SPI and Interrupts - cant get it to work,” 2015. [Online]. Available: https://www.raspberrypi.org/forums/viewtopic.php?f=71&t=19797. [Accessed: 6/12/15].

[31] Raspberrypi.org, “Raspberry Pi Forums - DMA on GPIO,” 2015. [Online]. Available: https://www.raspberrypi.org/forums/viewtopic.php?f=44&t=8376. [Accessed: 6/12/15].

[32] Wallacoloo, “Wallacoloo/Raspberry-Pi-DMA-Example,” GitHub, 2014. [Online]. Available: https://github.com/Wallacoloo/Raspberry-Pi-DMA-Example/blob/master/dma-gpio.c. [Accessed: 6/12/15].

[33] J. Pihlajamaa, 'Benchmarking Raspberry Pi GPIO Speed | Code and Life', Codeandlife.com, 2012. [Online]. Available: http://codeandlife.com/2012/07/03/benchmarking-raspberry-pi-gpio-speed/. [Accessed: 6/12/15].

[34] Raspberrypi.stackexchange.com, 'How to compile C files in terminal', 2015. [Online]. Available: http://raspberrypi.stackexchange.com/questions/5599/how-to-compile-c-files-in-terminal. [Accessed: 6/12/15].

[35] Raspberrypi.org, “Camera - Hardware,” Raspberry Pi Documentation 2015. [Online]. Available: https://www.raspberrypi.org/documentation/hardware/camera.md. [Accessed: 6/12/15].

[36] A. Allain, "”C File I/O Tutorial - Cprogramming.com.” Cprogramming.com, 2015. [Online]. Available: http://www.cprogramming.com/. [Accessed: 6/12/15].

[37] Codingunit.com, “C Tutorial – Binary File I/O | CodingUnit Programming Tutorials,” 2015. [Online]. Available: http://www.codingunit.com/c-tutorial-binary-file-io. [Accessed: 6/12/15].

[38] M. Samek, “Are We Shooting Ourselves in the Foot with Stack Overflow?”, Embeddedgurus.com, 2015. [Online]. Available: http://embeddedgurus.com/state-space/2014/02/are-we-shooting-ourselves-in-the-foot-with-stack-overflow. [Accessed: 6/12/15].

[39] N. Jones, “Efficient C Tip #13 – use the modulus (%) operator with caution,” Embeddedgurus.com, 2015. [Online]. Available: http://embeddedgurus.com/stack-overflow/2011/02/efficient-c-tip-13-use-the-modulus-operator-with-caution/. [Accessed: 6/12/15].

[40] Oshpark, “OSH Park ~ Pricing and Specifications,” 2015. [Online]. Available: https://oshpark.com/pricing. [Accessed: 6/12/15].

Appendix B – Parts List and Costs

Actual Prototyping Parts Costs:

Table B1 shows the actual cost of prototyping materials from the project.

TABLE B1

ACTUAL PARTS AND EQUIPMENT COSTS

Actual Development Costs (Equipment)					
Final Prototype					
System	*Part Name/Number*	*Vender*	*Manufacturer*	*Base Price*	*Cost (w/ Tax)*
Camera	Raspeberry Pi 5MP Camera	Amazon	Raspberry Pi Foun.	$ 26.95	$ 29.38
Video Data Processor	Raspberry Pi - Model B+	MCM Elec.	Raspberry Pi Foun.	$ 25.00	N/A
Memory	16 GB Micro SD Card	Amazon	Kingston Digital	$ 6.95	N/A
DAC	AD7245AANZ	DigiKey	Analog Devices	$ 19.37	$ 20.92
D/A Control Unit	Tiva C TM4C123GXL Eval	DigiKey	Texas Instruments	$ 16.68	$ 18.01
Misc. Passive	Caps/Resistors/Leads	N/A	N/A	$ 2.00	N/A
			Total -	$ 96.95	$68
Early Prototype					
System	*Part Name*	*Vender*	*Manufacturer*	*Base Price*	*Cost (w/ Tax)*
Camera	OV7670 300KP VGA Module	Amazon	Atomic Market	$ 10.99	$ 10.99
Memory	PmodSF2 - Serial PCM	Amazon	Digilent	$ 14.95	$ 14.95
DAC	AD7243ANZ	DigiKey	Analog Devices	$ 16.68	$ 16.68
			Total:	$ 42.62	$ 42.62
Development and Test Equipment					
Function	*Part Name*	*Vender*	*Manufacturer*	*Base Price*	*Cost (w/ Tax)*
ADC	AD1674JNZ	DigiKey	Analog Devices	$ 29.17	$ 31.50
USB-UART Cable	768-1204	DigiKey	FTDI	$ 15.00	$ 16.20
Antialiasing Filter	LTC1564IG#PBF	DigiKey	Linear Technologies	$ 17.59	$ 19.00
Breakout BNC	FEMALE-20AWG LEADS	DigiKey	Pomona Electronics	$ 17.10	$ 18.47
			Total:	$ 78.86	$ 85.17
	Shipping and Handing:				
Total	$16.50		**Equip. Total**	*Base Price*	*Actual Cost*
			S&H: $16.50	$ 218.43	$ 196.10

Estimates of Production Model Parts Costs:

Table B2 shows an estimate of how much a production model of the design would cost, assuming a production quantity of 1000 per year. It shows both per unit and grand total costs.

TABLE B2

ESTIMATED PRODUCTION MODEL PARTS AND EQUIPMENT COSTS

Parts Costs - Commercial Basis					
Production Model					
System	*Part Name/Number*	*Vender*	*Manufacturer*	*Unit Price*	*1000 Units*
Camera	Raspeberry Pi 5MP Camera	Amazon	Raspberry Pi Foun.	$ 26.95	$ 26,950.00
Video Data Processor	Raspeberry Compute Module	Newark	Raspberry Pi Foun.	$ 30.00	$ 30,000.00
Memory	16 GB Micro SD Card		Kingston Digital	$ 4.00	$ 4,000.00
DAC	AD7245AARZ	DigiKey	Analog Devices	$ 11.37	$ 11,371.50
D/A Control Unit	TM4C123GXL	DigiKey	Texas Instruments	$ 3.57	$ 3,570.00
Misc. Passive	Caps/Resistors/Leads	N/A	N/A	$ 0.10	$ 100.00
PCB	N/A	OSH Park	OSH Park	$ 12.00	$ 12,000.00
Shipping and Handing:	N/A	USPS	N/A	$ 0.50	$ 500.00
			Equip. Total	*Unit Price*	*1000 Units*
				$ 88.49	$ 88,491.50

Appendix C – Project Schedule

Planning and Actual Time Estimates:

Table C1 shows the initially planned estimate of hours to work on the project.

TABLE C1
ESTIMATED PLANNED DEVELOPMENT TIME

Estimated Planned Development Time				
Labor (Weekly)				
Person	*Min*	*Typ*	*Max*	*PERT*
Myself (hrs)	8	18	30	18.3
Labor (Total)				
Person	*Min*	*Typ*	*Max*	*PERT*
Myself (hrs)	160	360	600	366.7
Others (hrs)	10	15	40	18.3
Total	*Min*	*Typ*	*Max*	*PERT*
	170	375	640	385

Table C2 shows the actual estimated of hours of work on the project.

TABLE C2
APPROXIMANT ACTUAL DEVELOPMENT TIME

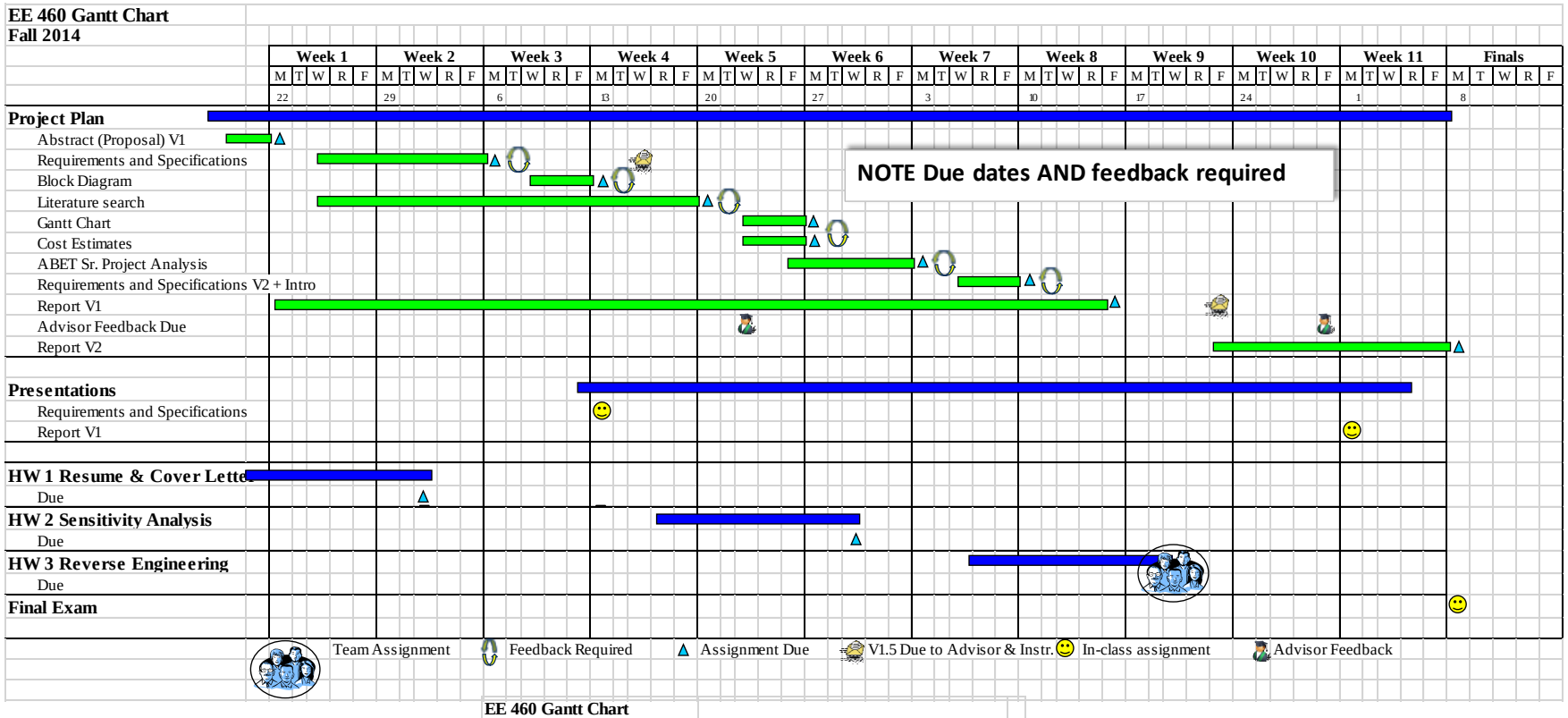
Approx. Actual Development Time				
Labor (Weekly Averaged)				
Person	*Min*	*Typ*	*Max*	*PERT*
Myself (hrs)	8	10	15	10.5
Labor (Total)				
Person	*Min*	*Typ*	*Max*	*PERT*
Myself (hrs)	160	200	300	210.0
Others (hrs)	12	18	25	18.2
Total	*Min*	*Typ*	*Max*	*PERT*
	172	218	325	228

Planning and Actual Timelines:

The initial planning timeline for project development is shown in Tables C3 (a), C3 (b), & C3 (c) below

TABLE C3 (a)

PLANNING GANTT CHART – FALL QUARTER AND WINTER BREAK 2014 [21]



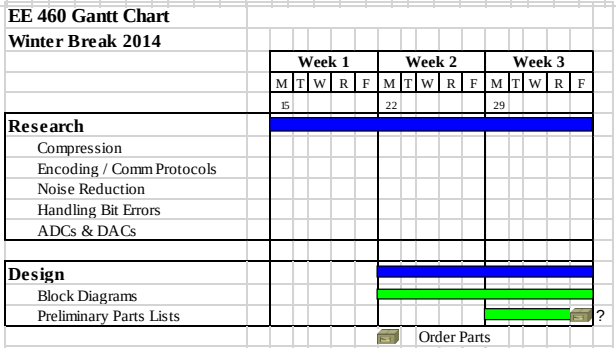


TABLE C3 (b)

PLANNING GANTT CHART –WINTER 2015

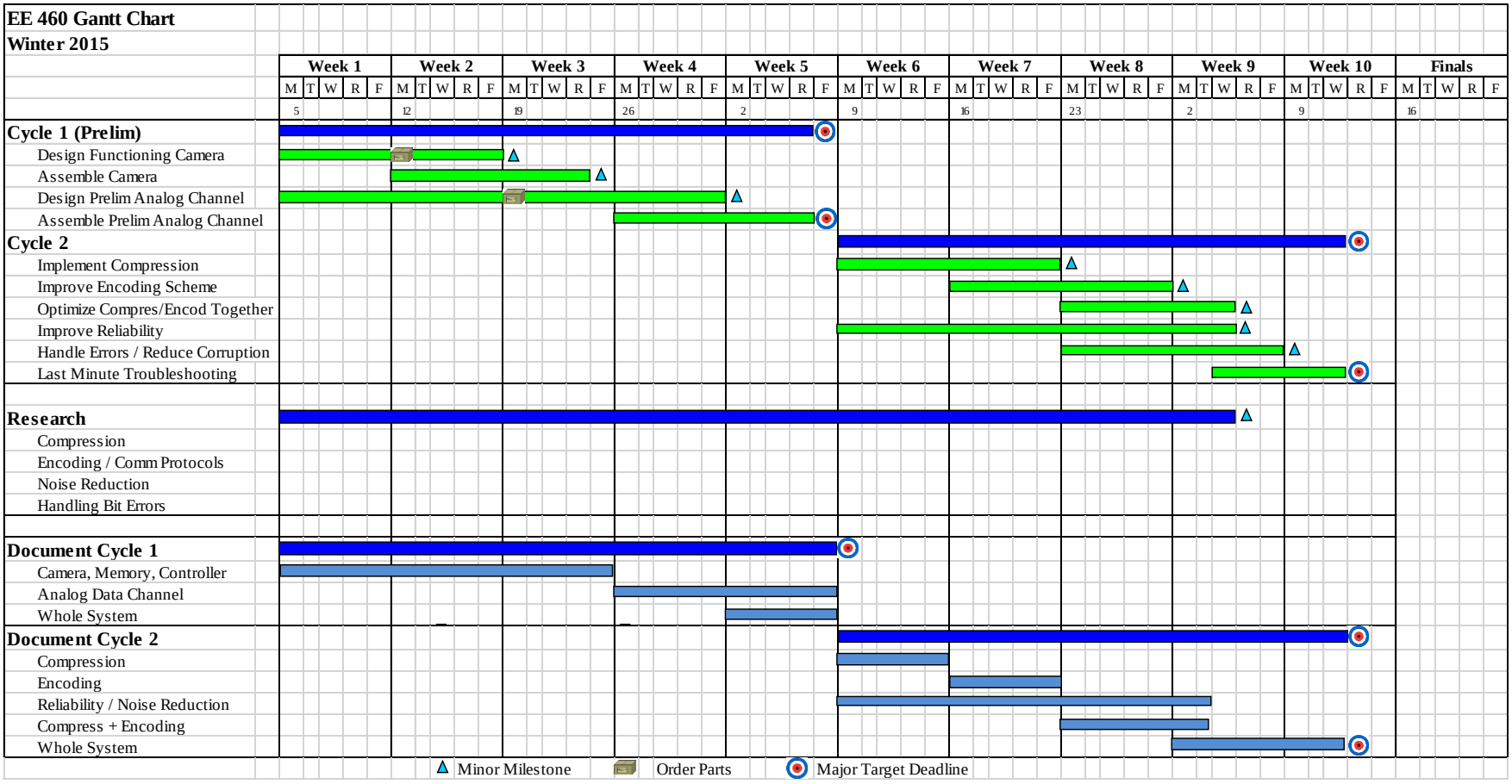
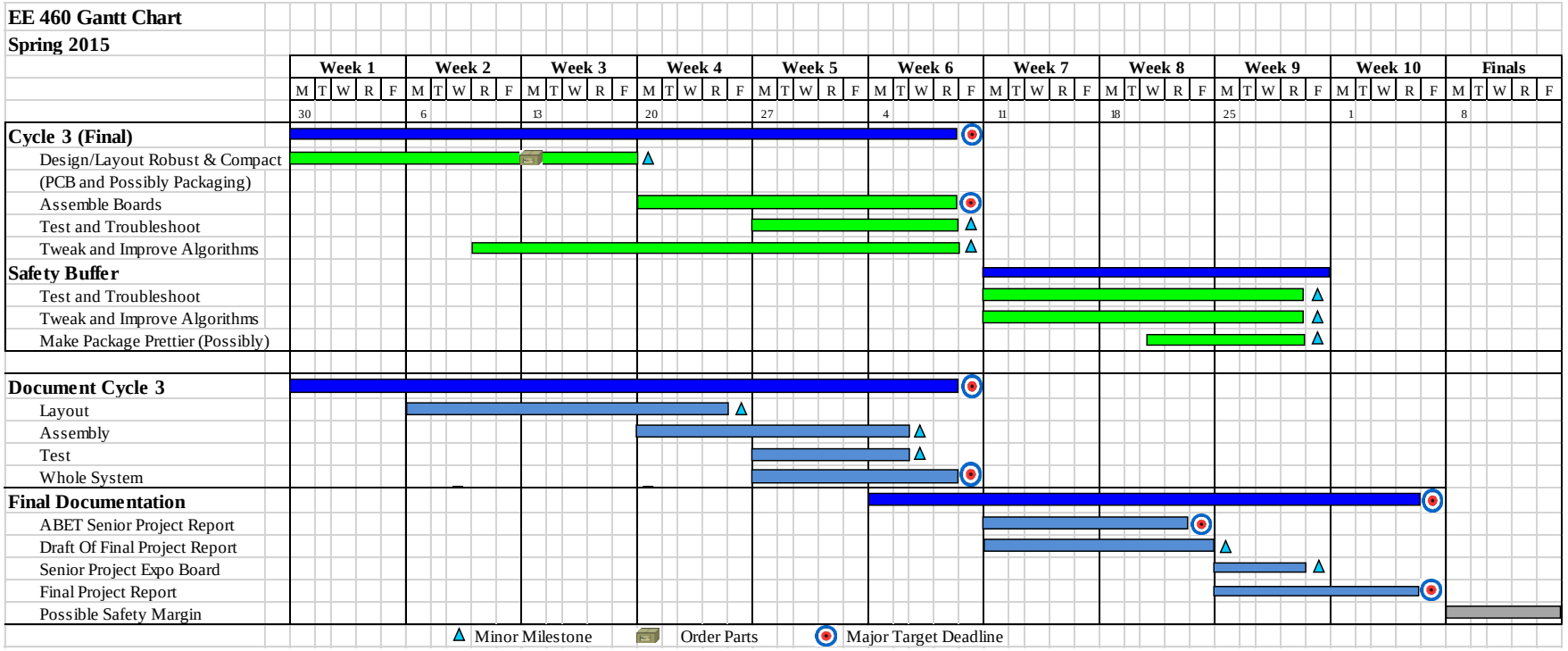


TABLE C3 (c)

PLANNING GANTT CHART – SPRING 2015

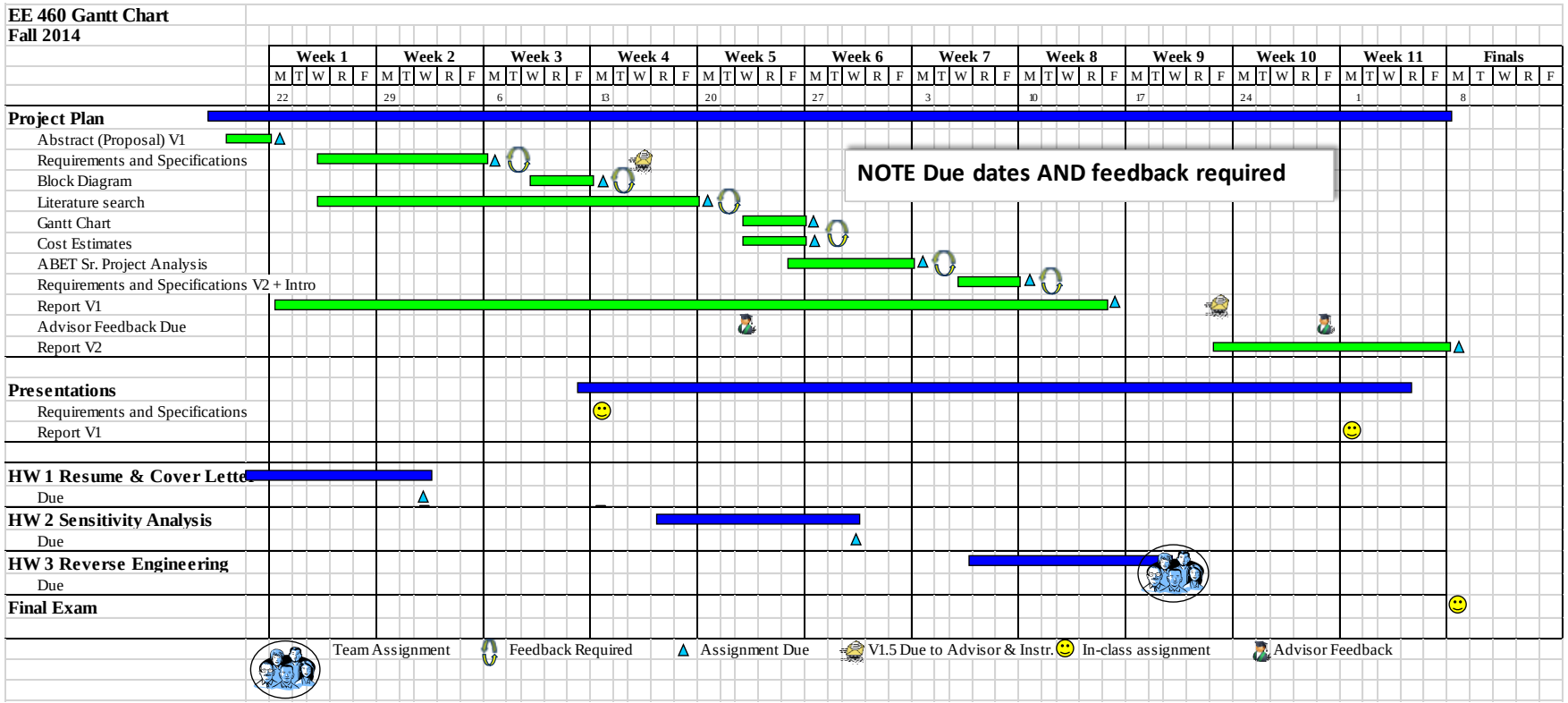


In the initial timeline planning, the first design-build-test cycle corresponds to implementing a basic version of the design on a breadboard. The second design-build-test cycle involves creating an improved version on perf-board. And the last cycle implements a final version of the design on a printed circuit board and with included packaging.

Tables C4 (a), C4 (b), and C4 (c) contain the timeline that the development actually followed.

TABLE C4 (a)

ACTUAL DEVELOPMENT GANTT CHART – FALL QUARTER AND WINTER BREAK 2014 [21]



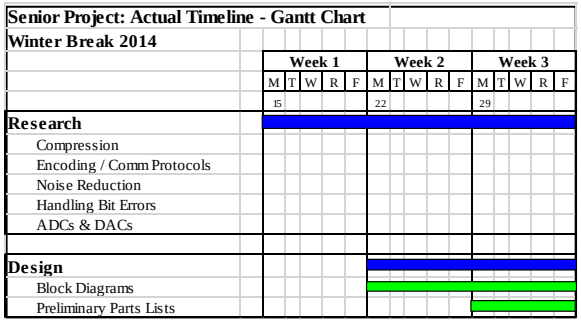


TABLE C4 (b)

ACTUAL DEVELOPMENT GANTT CHART –WINTER 2015

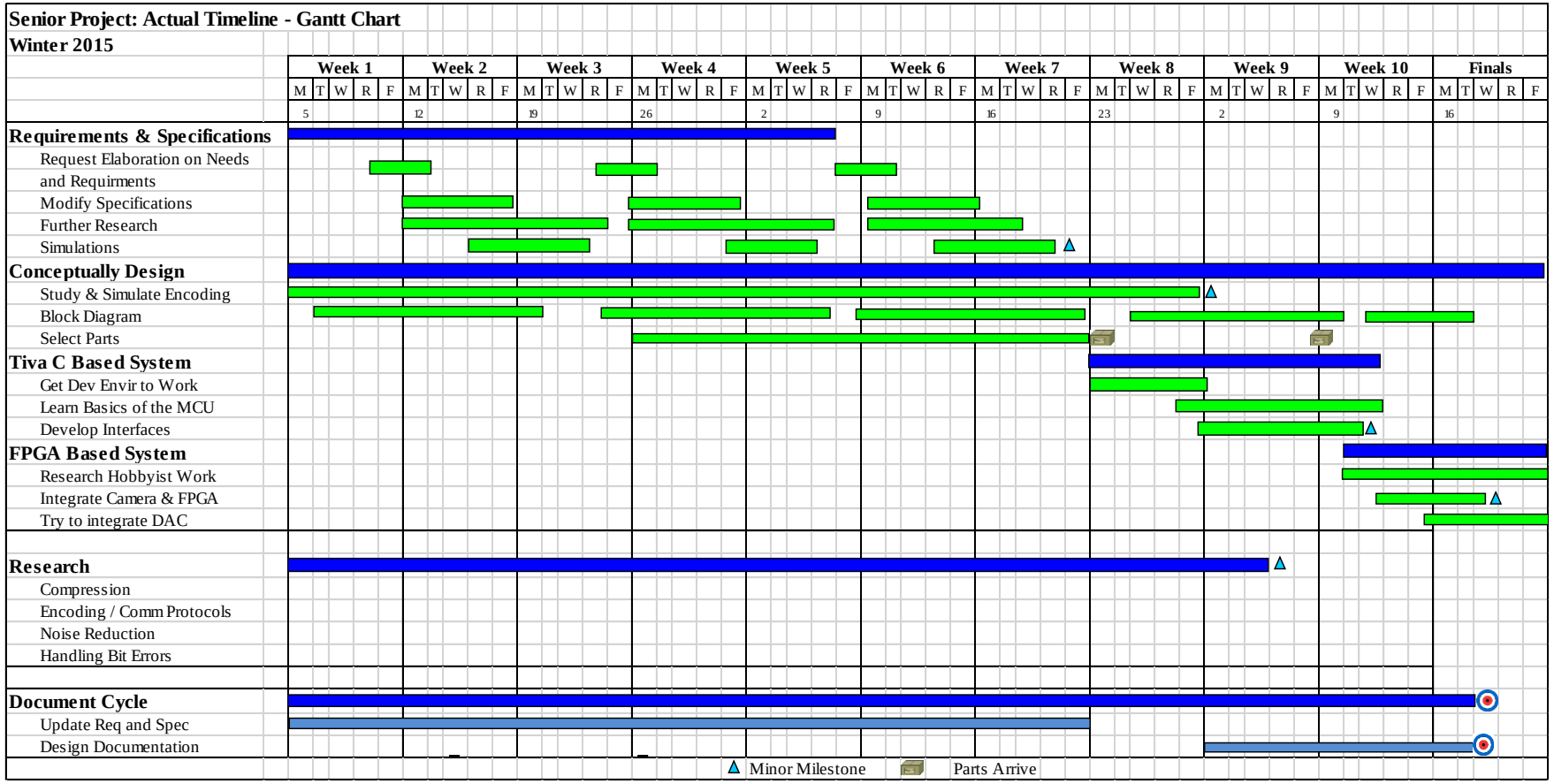
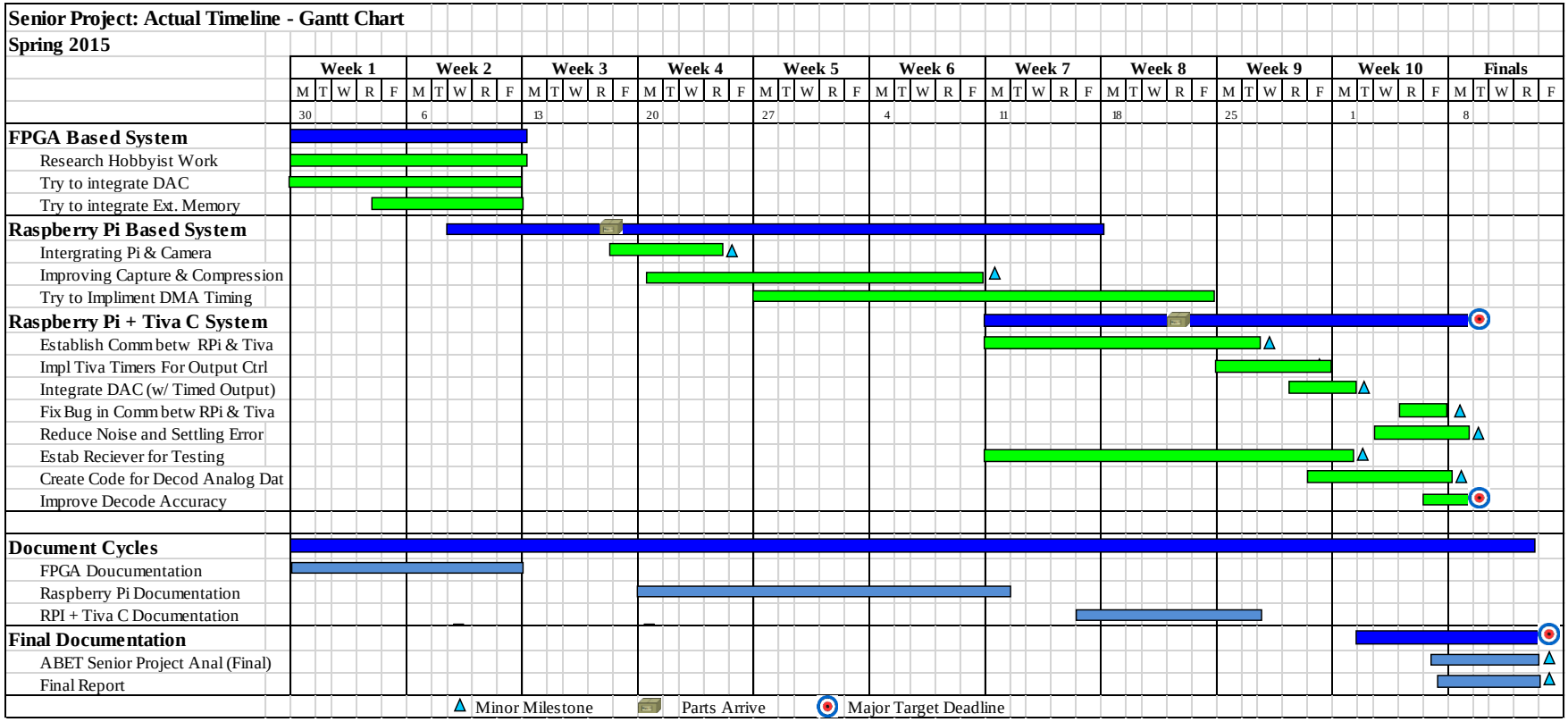


TABLE C4 (c)

ACTUAL DEVELOPMENT GANTT CHART – SPRING 2015



Appendix D – Program Listings

Raspberry Pi Code:

Shell Script for Video Capture, Compression, and Wrapping in .mp4 file – Vid_script_Pi:

```
#!/bin/bash
# Video record,store,play,and convert to .mp4

raspivid -w 640 -h 480 -t 3000 -o scriptest.h264 -fps 30 -e -cfx 128:128
omxplayer scriptest.h264
if [ -e scriptest.mp4 ]; then
    rm scriptest.mp4     # Delete previous otherwise unexpected results
    echo "deleted scriptest.mp4"
fi
MP4Box -add scriptest.h264 scriptest.mp4
```

C code for sending video data to the Tiva C – ComProtVidPlus:

```
/*
-- Author:  Thomas Higdon
-- Last Modification:  6/8/15
*/

//-----------Libraries------------
#include <wiringPi.h>

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

//-----------Definitions------------
#define BCM2 2
#define BCM3 3
#define BCM4 4
#define BCM5 5
#define BCM6 6
#define BCM7 7
#define BCM8 8
#define BCM9 9

#define SendSig 21
#define ReadyPin 20
#define MoreDataPinO 16

//-----------Pre-declare Functions------------
inline void BCM_writeByte(char Byte);
void BCM_ByteOut_Ini();

int main()
{
    //--------Variable Declaration--------
    //int I;
    //uint8_t I8;
    uint8_t NextByte=0;
```

```
FILE *infile;
uint8_t successful_read;

//--------Initialization--------
if (wiringPiSetupGpio() == -1)
    exit(1);
BCM_ByteOut_Ini();
if(piHiPri(99))
{
    printf("Error setting high priority\n");
    return 1;
};

pinMode(ReadyPin,OUTPUT);
pinMode(MoreDataPinO,OUTPUT);

pinMode(SendSig,INPUT);
    pullUpDnControl(SendSig,PUD_DOWN);
//-------------------------------------

//--------Data to Send--------
printf("Opening File to be sent...");
infile = fopen("scriptest.mp4","rb");
if (!infile)
{
    printf("Unable to open file(scriptest.mp4)");
    return 1;
}
printf("opened.\n");
//----------------------------

//-----------------Data Transfer-----------------
//  I8=0;//!!!!!!Just for testing send
//wiringPiISR(HaltSig, INT_EDGE_RISING, &HaltFunc);

digitalWrite(MoreDataPinO,1);
while(1)
{
    successful_read = fread(&NextByte, 1, 1, infile);
    if (successful_read != 1) { break; }
    //---More File Data Ready---
    digitalWrite(ReadyPin,0);
    //printf("Ready Low\n");
    //or//Same
        //I8++;//!!!!!!Just for testing send
        //printf("New Index: %d",I8);

//---Wait to send---
    while(!digitalRead(SendSig)); //trap until not 0
    //printf("\nDetected a Send Request\n");
//---Send Byte---
    BCM_writeByte(NextByte);
    //printf("Sent: %d (from I=%d)\n",NextByte,I8);
//---Signal Ready---
    digitalWrite(ReadyPin,1);
    //printf("Ready High\n");
//---Wait for Send==0---
```

```
        while(digitalRead(SendSig)); //trap until not 1
        //printf("Send Low Detected\n");

    /*
    //---Signal Ready---
        digitalWrite(ReadyPin,0);
        printf("Ready Low\n");
        //or//Same
            I8++;//!!!!!!Just for testing send
            //printf("New Index: %d",I8);
    */
    }
    digitalWrite(MoreDataPinO,0);
    digitalWrite(ReadyPin,0);
    printf("\nDone Reading File and Sending Data.\n");
    NextByte = 0x00;
    fclose(infile);
    return 0;
}


inline void BCM_writeByte(char Byte)
{
    int val;
    val = 0x01 & Byte;       //Bit-masking
    digitalWrite(BCM2, val); //treats any nonzero as high

    val = 0x02 & Byte;
    digitalWrite(BCM3, val); //treats any nonzero as high

    val = 0x04 & Byte;
    digitalWrite(BCM4, val); //treats any nonzero as high

    val = 0x08 & Byte;
    digitalWrite(BCM5, val); //treats any nonzero as high

    val = 0x10 & Byte;
    digitalWrite(BCM6, val); //treats any nonzero as high

    val = 0x20 & Byte;
    digitalWrite(BCM7, val); //treats any nonzero as high

    val = 0x40 & Byte;
    digitalWrite(BCM8, val); //treats any nonzero as high

    val = 0x80 & Byte;
    digitalWrite(BCM9, val); //treats any nonzero as high
}

void BCM_ByteOut_Ini()
{
    int I;
    for (I = 2; I < 10; I++)
    {
        pinMode(I, OUTPUT);
    }
}
```

Tiva C Code:

The C code for Buffering Data from the RPi to the DAC – Com_with_Pi_Timer:

```
/*
--Author:    Thomas Higdon
--Last Modification:  6/8/15
--
--Based in part on Texas Instruments Code Samples for using the Tiva C
--included with the Tiva C development package
--The sample projects: project0.c and timers.c
--TI's copyright statement and disclaimer are below:
      //*****************************************************************************
*
      //
      // project0.c - Example to demonstrate minimal TivaWare setup
      // timers.c - Timers example.
      //
      // Copyright (c) 2012-2013 Texas Instruments Incorporated.  All rights
reserved.
      // Software License Agreement
      //
      // Texas Instruments (TI) is supplying this software for use solely and
      // exclusively on TI's microcontroller products. The software is owned by
      // TI and/or its suppliers, and is protected under applicable copyright
      // laws. You may not combine this software with "viral" open-source
      // software in order to form a larger program.
      //
      // THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
      // NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
      // NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
      // A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
      // CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
      // DAMAGES, FOR ANY REASON WHATSOEVER.
      //
      // This is part of revision 1.1 of the EK-TM4C123GXL Firmware Package.
      //
      //*****************************************************************************
*
*/

//For talking to Pi
#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"

//For Timers
#include "inc/hw_ints.h"
//#include "driverlib/fpu.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
```

```
#include "driverlib/timer.h"

//*****************************************************************************
//
// Define pin mapping.
//
//*****************************************************************************
#define RED_LED   GPIO_PIN_1
#define BLUE_LED  GPIO_PIN_2
#define GREEN_LED GPIO_PIN_3
#define SendPinO GPIO_PIN_0
#define ReadyPinI 0x02
#define MoreDataPinI GPIO_PIN_5
#define DAC_nWR_O GPIO_PIN_3
#define Buff_Full_O GPIO_PIN_7
#define Buff_Low_O GPIO_PIN_4


//*****************************************************************************
//
// The error routine that is called if the driver library encounters an error.
//
//*****************************************************************************
#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif

//*****************************************************************************
//
// Global Variables
//
//*****************************************************************************
static uint_fast8_t gbl_storedData[512]={0};

volatile int gbl_ReceiveIndex;
volatile int gbl_SendIndex;
volatile int32_t gbl_BuffLeadSize;

volatile uint_fast8_t gbl_SendMode=0;
volatile uint_fast32_t gbl_SendCount=0;

const uint_fast32_t MinBuff = 500;
//*****************************************************************************
//Function Declarations
//*****************************************************************************
void init_gpio();
uint_fast8_t read_Pi_Byte();
void write_DAC_Byte(uint_fast8_t Byte);
void Timer0IntHandler(void);
void init_timers();
//*****************************************************************************
//
```

```
// Main 'C'
//
//*****************************************************************************
int
main(void)
{
      //---------------Initiallization--------------
      gbl_ReceiveIndex=-1;
      gbl_SendIndex=-1;
      //int I=0;
      //uint_fast8_t testbyte = 0xAA;
      uint_fast8_t readbyte;
      //uint_fast8_t storedData[512];
   /* //
    // Setup the system clock to run at 50 Mhz from PLL with crystal reference
    //
    SysCtlClockSet(SYSCTL_SYSDIV_4|SYSCTL_USE_PLL|SYSCTL_XTAL_16MHZ|
                   SYSCTL_OSC_MAIN);*/
      //
      // Set the clocking to run directly from the crystal.
      //
      //ROM_SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                                          //SYSCTL_XTAL_16MHZ); //Timer Interrupt
Original
      ROM_SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                                    SYSCTL_XTAL_16MHZ); //TH: Modded to 50 Mhz PLL


    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //For debug
    GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED); //For debug
    //GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, RED_LED); //For debug
    init_gpio();

    write_DAC_Byte(0x00);
    init_timers();

    GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, BLUE_LED);

    //-----------Start of Communication with RPi Loop--------------

    while(GPIOPinRead(GPIO_PORTB_BASE,MoreDataPinI)>0) //For Testing
    {
       while(gbl_BuffLeadSize >= MinBuff); //Wait until buffLead < 500

                    //GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
GREEN_LED|BLUE_LED); //For debug
       GPIOPinWrite(GPIO_PORTB_BASE, SendPinO, 0xFF); //Request Next Byte from Pi
                    //GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
GREEN_LED|RED_LED); //For debug
       while((GPIOPinRead(GPIO_PORTB_BASE,ReadyPinI)&ReadyPinI)==0); //Wait until Pi
says the data is Ready
                    //GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
RED_LED|BLUE_LED); //For debug
       readbyte = read_Pi_Byte(); //Read the actual data in from the Pi
```

```
                    //GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
GREEN_LED);
      GPIOPinWrite(GPIO_PORTB_BASE, SendPinO, 0x00); //Tell Pi Data has been
received
      gbl_ReceiveIndex++;
            //gbl_ReceiveIndex=gbl_ReceiveIndex%512; //Too slow, if statement faster
            if(gbl_ReceiveIndex>511)
            {
                  gbl_ReceiveIndex=0;
            }
      gbl_storedData[gbl_ReceiveIndex] = readbyte; //For Testing

            //write_DAC_Byte(readbyte); //For Testing

      while((GPIOPinRead(GPIO_PORTB_BASE,ReadyPinI)&ReadyPinI)!=0); //Wait for
acknowledge/reset from Pi


      //-----Update BuffLeadSize-----
            if((gbl_ReceiveIndex-gbl_SendIndex) >= 0)
            {
                  gbl_BuffLeadSize=gbl_ReceiveIndex-gbl_SendIndex;
            }
            else
            {
                  gbl_BuffLeadSize=gbl_ReceiveIndex-gbl_SendIndex+512;
            }

      //Begin again
   } //Receive Data Loop

   while(gbl_SendMode!=5); //Wait to finish sending what is stored in the buffer
   ROM_IntMasterDisable(); //Disable timers
   write_DAC_Byte(0x00);   //Turn everything off . . .
   GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, 0x00);


   return 0;
} //-------------End of Main-------------

//****************************************GPIO*********************************
***
//============================GPIO Initialization=============================
void init_gpio()
{
      //-------------------------Receiving End----------------------------
      GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, RED_LED); //For
debug
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB); //Enable Port B
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD); //Enable Port D
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE); //Enable Port E

      //GPIOPinTypeGPIOInput(GPIO_PORTE_BASE,
GPIO_PIN_4|GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1); //PE:
GPIO_PIN_4|GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1 0x1E
```

```
            // Check the arguments.
            ASSERT(_GPIOBaseValid(GPIO_PORTE_BASE));
          // Make the pin(s) be inputs.
          GPIODirModeSet(GPIO_PORTE_BASE,
GPIO_PIN_4|GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1, GPIO_DIR_MODE_IN);
          // Set the pad(s) for standard push-pull operation.
          GPIOPadConfigSet(GPIO_PORTE_BASE,
GPIO_PIN_4|GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPD);
          ///////GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
GREEN_LED|RED_LED); //For debug

      //GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, 0x0F ); //PD:
GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0 <----- This was causing stack overflow,
the following was not
            // Check the arguments.
            ASSERT(_GPIOBaseValid(GPIO_PORTD_BASE));
          // Make the pin(s) be inputs.
          GPIODirModeSet(GPIO_PORTD_BASE, 0x0F, GPIO_DIR_MODE_IN);
          // Set the pad(s) for standard push-pull operation.
          GPIOPadConfigSet(GPIO_PORTD_BASE, 0x0F, GPIO_STRENGTH_2MA,
GPIO_PIN_TYPE_STD_WPD);
          ////////GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
BLUE_LED|RED_LED); //For debug

      //GPIOPinTypeGPIOInput(GPIO_PORTB_BASE, ReadyPinI ); //PB GPIO_PIN_1  <-----
This was
            // Check the arguments.
            ASSERT(_GPIOBaseValid(GPIO_PORTB_BASE));
            // Make the pin(s) be inputs.
            GPIODirModeSet(GPIO_PORTB_BASE, ReadyPinI|MoreDataPinI,
GPIO_DIR_MODE_IN);
            // Set the pad(s) for standard push-pull operation.

      GPIOPadConfigSet(GPIO_PORTB_BASE,ReadyPinI|MoreDataPinI,GPIO_STRENGTH_2MA,GPIO
_PIN_TYPE_STD_WPD);
            /////////GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
BLUE_LED|GREEN_LED|RED_LED); //For debug

      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, SendPinO ); //PB GPIO_PIN_1|pin5

      //-------------------------Sending End---------------------------
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC); //Enable Port C
      SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA); //Enable Port A
      //FOR DEBUG, REMOVED: SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF); //Enable
Port F

      GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, 0xF0 ); //PC:
GPIO_PIN_7|GPIO_PIN_6|GPIO_PIN_5|GPIO_PIN_4

      GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, 0x1C ); //PA:
GPIO_PIN_4|GPIO_PIN_3|GPIO_PIN_2

      GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, GPIO_PIN_2); //PB: GPIO_PIN_2
```

```
	GPIOPinTypeGPIOOutput(GPIO_PORTB_BASE, DAC_nWR_O); //PB: GPIO_PIN_3 controls
DAC's nWR
	GPIOPinWrite(GPIO_PORTB_BASE, DAC_nWR_O, DAC_nWR_O); //Initialize High,
because active low

	GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, BLUE_LED); //For
debug
	return;
}
//============================================================================

uint_fast8_t read_Pi_Byte()
{
	uint_fast32_t Byte = 0x00;
	//GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED|GREEN_LED, BLUE_LED);
	Byte = (GPIOPinRead(GPIO_PORTE_BASE, 0x1E )<<3); //PE:
GPIO_PIN_4|GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1
	Byte |= GPIOPinRead(GPIO_PORTD_BASE, 0x0F ); //PD:
GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1|GPIO_PIN_0
	//GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED|GREEN_LED, 0);
	return Byte;
}

void write_DAC_Byte(uint_fast8_t Byte)
{
	//uint_fast32_t BigInt;
	//BigInt = Byte;
	//GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED|GREEN_LED, BLUE_LED);
	GPIOPinWrite(GPIO_PORTB_BASE, DAC_nWR_O, 0x00); //nWR low to read data to DAC
	SysCtlDelay(ROM_SysCtlClockGet()/11000000); //delay 91 ns
	GPIOPinWrite(GPIO_PORTC_BASE, 0xF0, Byte);//PC:
GPIO_PIN_7|GPIO_PIN_6|GPIO_PIN_5|GPIO_PIN_4
	////////GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED, RED_LED);
	GPIOPinWrite(GPIO_PORTA_BASE, 0x1C, (Byte<<1)); //PA:
GPIO_PIN_4|GPIO_PIN_3|GPIO_PIN_2
	/////////GPIOPinWrite(GPIO_PORTF_BASE, RED_LED|BLUE_LED|GREEN_LED,
RED_LED|GREEN_LED);
	GPIOPinWrite(GPIO_PORTB_BASE, GPIO_PIN_2, (Byte<<2)); //PB: GPIO_PIN_2
	SysCtlDelay(1); //delay 20 ns
	GPIOPinWrite(GPIO_PORTB_BASE, DAC_nWR_O, DAC_nWR_O); //nWR high to latch data
to DAC
	//GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED|GREEN_LED, GREEN_LED);
	return;
}

//***********************************Timers*********************************
void
Timer0IntHandler(void)
{
    //char cOne, cTwo;

    //
    // Clear the timer interrupt.
    //
    ROM_TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT);
```

```
    //
    // -----Send To DAC------
    //
    if(gbl_SendMode == 3) //Mode-3=Sending Data from Buffer; Most timing critical
mode is first tested for
      {
            gbl_SendIndex++;
            //gbl_SendIndex=gbl_SendIndex%512;
            if(gbl_SendIndex>511)
            {
                  gbl_SendIndex=0;
            }
            write_DAC_Byte(gbl_storedData[gbl_SendIndex]);

            //Update BuffLeadSize
            int temp_sub_int = gbl_ReceiveIndex-gbl_SendIndex;
            if((temp_sub_int) > 0)
            {
                  gbl_BuffLeadSize=gbl_ReceiveIndex-gbl_SendIndex;
            }
            else if(temp_sub_int<0)
            {
                  gbl_BuffLeadSize=gbl_ReceiveIndex-gbl_SendIndex+512;
            }
            else
            {
                  gbl_SendMode=5;
            }

      }else
      if (gbl_SendMode == 0) //Initially send 20 zeros
      {
            write_DAC_Byte(0x00);
            gbl_SendCount++;
            if (gbl_SendCount>19)
            {
                  gbl_SendCount = 0;
                  gbl_SendMode = 1;
            }
      }
      else
      if(gbl_SendMode == 1) //Send callibration ramp
      {
            uint_fast32_t calli_data;
            calli_data = gbl_SendCount>>1; //Sends same byte twice before moving on
            write_DAC_Byte(calli_data);
            gbl_SendCount++;
            if (gbl_SendCount>511)
            {
                  gbl_SendCount = 0;
                  gbl_SendMode = 2;
            }
      }
      else
```

```
        if(gbl_SendMode == 2) //Send 20 zeros after callibration
        {
                write_DAC_Byte(0x00);
                gbl_SendCount++;
                if (gbl_SendCount>19)
                {
                        gbl_SendCount = 0;
                        gbl_SendMode = 3;
                }
        }
        else // Just send zero, final state
        {
                write_DAC_Byte(0x00);
                GPIOPinWrite(GPIO_PORTF_BASE, BLUE_LED|GREEN_LED|RED_LED, RED_LED);
        }

    //
    // Update the interrupt status on the display.
    //
    ROM_IntMasterDisable();
    ROM_IntMasterEnable();
}

void init_timers()
{
        //
        // Enable the peripheral used by this example.
        //
        ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

        //
        // Enable processor interrupts.
        //
        ROM_IntMasterEnable();

        //
        // Configure the two{one} 32-bit periodic timers.
        //
    ROM_TimerConfigure(TIMER0_BASE, TIMER_CFG_PERIODIC);
    ROM_TimerLoadSet(TIMER0_BASE, TIMER_A, ROM_SysCtlClockGet()/3330);

    //
        // Setup the interrupts for the timer timeouts.
        //
        ROM_IntEnable(INT_TIMER0A);
        ROM_TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

        //
        // Enable the timers{timer}.
        //
        ROM_TimerEnable(TIMER0_BASE, TIMER_A);
}
```

Analog Data Decoding and Digital Data Reconstruction (Matlab):

The primary piece of reconstruction code is an m-script: ana2file_script_B.m, however it calls 3 functions also created by the project designer, whose code will follow ana2file_script_B.m's code.
The functions are: clust_callibration.m, findbin.m, and writebytes2file.m
An alternatice to clust_callibrated.m is dev_callibration.m

Main Script for translated analog readings back into a digital file – ana2file_script_B.m:

```
% ana2file_script_B.m
%--Author: Tom Higdon
%--Date Modified: 6/13/15
an_data=importdata('ana_vid_data2.csv');

max_an=max(an_data);
min_an=min(an_data);
range=max_an-min_an;

%---------------Callibrate----------------
% -- Find Callibration Signal --
start_sig=find( an_data<(1.1*abs(min_an)), 1, 'first');
trig_start=find( an_data(start_sig:end)>(max_an/5),1,'first' );
trig_start=start_sig+trig_start;
trig_stop=find( an_data(start_sig:end)>(max_an*4/5),1,'first' );
trig_stop=start_sig+trig_stop;
% ------------------------------
%Old: % calli_data=an_data((trig_start-326):(trig_stop+326));
%(255/5)*6+20=326

% Using data from beginning middle and end to try to reduce any component
% drift issues. Adaptive Callibration might further improve.
stop_data=find(an_data>0.100,1,'last');
mid_point=round(mean(trig_start,stop_data));
calli_part1=an_data((trig_start-326):trig_start+15000); % Beginning
calli_part2=an_data((mid_point-7500):(mid_point+7500)); % Middle
calli_part3=an_data((stop_data-15000):stop_data);       % End
calli_data=vertcat(calli_part1,calli_part2,calli_part3);
zer_noise=(calli_data<0);
calli_data(zer_noise)=[];

% To generate callibrated byte code levels
[cent, numpoint2]=clust_callibrate(calli_data); % User created function
%[level, dev_groups]=dev_callibration(calli_data); % Alt created function

mid_fin=cent; %!!!!!!!!!!!!!!!!!!Change if you change which callib type!
%mid_fin=level;

brder=zeros(255,1); % Create borders that can be used to put readings into
for I=2:256         % Bins
    brder(I-1)=(mid_fin(I)+mid_fin(I-1))/2;
end

'Callibrated.'

% Initiallize for Interpretation loop
continue_reading=1;
```

```
Bfnd_flg=0;
nBfnd_cnt=0;
skip_cnt=0;
old_samp=NaN;
samp=0;
nan_count=0;
nan_loc=[];
nan_loc_an=[];
smp_num=(trig_stop+326); %
I=1;
I_B=1;
binned_data=zeros((length(an_data)-smp_num),1);
Byte_data=zeros(ceil(length(binned_data)/3),1);
while(continue_reading==1)
    if(Bfnd_flg==0)  % If you haven't just found a Byte, look for one
        [ samp ]=findbin(an_data(smp_num),brder); %put anlg data in dig bin
        % findbin() is User Created Function

        if(abs(an_data(smp_num)-an_data(smp_num-1))<0.045) % compare with
previous sample

Byte_data(I_B)=findbin(0.8*an_data(smp_num)+0.2*an_data(smp_num),brder); % if
equal, you found a Byte
            Bfnd_flg=1; % set flag that a Byte has been found
            nBfnd_cnt=0; % reset counter since last Byte found
            I_B=I_B+1; % increment Byte array's index
        else % if not equal
            old_samp=samp; % get ready to check the next pair
            nBfnd_cnt=nBfnd_cnt+1; % increment the byte not found counter
        end
         if((nBfnd_cnt>=2)) % Let me know if something is wrong
             Byte_data(I_B)=NaN;
             nan_count=nan_count+1;
             I_B=I_B+1;
             %nan_loc=[nan_loc,I];
             %nan_loc_an=[nan_loc_an,smp_num];
             nBfnd_cnt=0;
         endxlabel('Index')


    else % If you have found a byte recently
        skip_cnt=skip_cnt+1; % increment skip counter
        if(skip_cnt>=2) % after 2 loops (so completely skiping 1 an_samples)
            skip_cnt=0; % reset skip counter
            Bfnd_flg=0; % reset Byte found flag
            nBfnd_cnt=0; % reset Byte not found counter
            % load 1st analog sample after the skipped sample into old_samp
            old_samp=findbin(an_data(smp_num),brder);
        end
    end

    binned_data(I)=samp; % save the digital equivalents of all of the samps

    smp_num=smp_num+1; % increment the index in the original data array
    I=I+1; % increment the index for the shorted binned array
    if(smp_num>(length(an_data))) % stop when at the end of data
```

```
        continue_reading=0;
    end
%     if(I_B>100)
%         if(sum(Byte_data((I_B-10):(I_B-1)))==0)
%             continue_reading=0;
%             'Stopped Binning Early Because zeros'
%         end
%     end
end

Start_data=find(Byte_data>0,1,'first');
End_data=find(Byte_data>10,1,'last');

% .mp4 has 3 leading zeros and one trailing zero, I think
Vid_array = vertcat([0;0;0],Byte_data(Start_data:End_data),[0]);
'Vid_array ready!' %#ok<NOPTS>

success=writebytes2file(uint8(Vid_array),'DecodedVid.mp4');
if(success == 1)
    'Wrote to the File!' %#ok<NOPTS>
end
```

Function that finds calibrated voltage levels by clusting readings into 256 clusters – clust_callibrate.m:

```
function [ cent, pointsperclust ] = clust_callibrate( calli_data )
%--Author: Tom Higdon
%--Date Modified: 6/9/15
len_cal=length(calli_data);
max_an=max(calli_data);
min_an=min(calli_data);
range=max_an-min_an;

%-----cent initiallization-----
cent=zeros(256,1);
cent(1)=0;
for I=1:256
    %cent(I)=(range*(I-1)/(256-1)) + min_an;
    cent(I)=(2*5)*(16*(I-1)/4096); %From DAC (AD7345A) Datasheet pg 10
end

dtocent=zeros(len_cal,256);
pointsperclust=zeros(8,1);
catag=zeros(len_cal,1); %#ok<NASGU>
catag_old=catag;
clust_long=zeros(len_cal,256);
iterate = 1;
iter_count=0;
while iterate==1
    %-------Distance from centroid Calulation-------
    for I=1:256
        dtocent(:,I)=abs(calli_data-cent(I));
    end
```

```
    %Catagorizing
    [~,catag]= min( dtocent, [], 2 ); %Dimension-2

    %---------- Recalculating centroid locations -----------
    for I=1:256
        clust_temp=zeros(len_cal,1);
        cl_temp_ind=find(catag==I);
        size_cl=length(cl_temp_ind);
        clust_temp(1:size_cl)=calli_data(cl_temp_ind);

        %if (size_cl ~= 0)
            cent(I)=sum(clust_temp)./size_cl;
        %end
        pointsperclust(I)=size_cl;
        clust_long(1:size_cl,I)=clust_temp(1:size_cl);
    end

    iter_count = iter_count+1;
    if (iter_count >= 3)
        compar_catag = (catag~=catag_old);
        percent_diff = sum(compar_catag)/len_cal;
        if (percent_diff <= 0.05)
                iterate = 0;
           elseif (iter_count > 35)
                'Could not converge!' %#ok<NOPTS>
                iterate = 0;
        end
    end
    catag_old=catag;
end %while

end
```

Alternative Function that finds calibrated voltage levels based on ramp structure of the calibration signal – dev_callibration.m:

```
function [ level, clust ] = dev_callibration( calli_data )
%--Author: Tom Higdon
%--Date Modified: 6/9/15
len_cal=length(calli_data);
sort_calli=sort(calli_data, 'descend');

clust(256,:)=sort_calli(1:7);
clust(256,:);
cl_mean=mean(clust(256,:));
cl_med_dev=median_deviation(clust(256,:));
for CI=1:7
    if (abs(clust(256,CI)-cl_mean)>(1.5*cl_med_dev))
        clust(256,CI)=NaN;
    else
        last_in_clust=CI;
    end %if
```

```
end %Inter-cluster for
temp_clust=clust(256,(isfinite(clust(256,1:7)))); % Temp array with all non
                                                  %NaN numbers
level(256)=mean(temp_clust);
Start = 1 + last_in_clust;

I=255;
while (I>1)
    last_in_clust=0;
    clust(I,:)=sort_calli((Start):(Start+6));
    cl_mean=mean(clust(I,:));
    cl_med_dev=median_deviation(clust(I,:));
    for CI=1:7
        if (abs(clust(I,CI)-cl_mean)>(1.5*cl_med_dev))
            clust(I,CI)=NaN;
        else
            last_in_clust=CI;
        end %if
    end %Inter-cluster for
    Start=Start+last_in_clust;
    temp_clust=clust(I,(isfinite(clust(I,1:7))));
    level(I)=mean(temp_clust);
    I=I-1;
end
```

Function that returns the byte "bin" of the analog data, based on boundries – findbin.m:

```
function [ bin_value ] = findbin( analog_sample, boundry )
%--Author: Tom Higdon
%--Date Modified: 6/9/15
below=find(boundry<analog_sample,1,'last');
above=find(boundry>analog_sample,1,'first');

if(~isempty(below))
    bin_value=below; %not below+1 because Matlab 1-indexed and value 0-ind
elseif(~isempty(above))
    bin_value=above-1; %should only occur for the case of above==1, val==0
else
    'error, neither high nor low!' %#ok<NOPRT>
    bin_value=NaN;
    %return
end
```

Function that writes bytes to a binary file – writebytes2file.m:

```
function [ success ] = writebytes2file( Byte_vector,filename )
% Essentially taken from Mathwork Documention
fid = fopen(filename,'w');
fwrite(fid,Byte_vector); % Default is one byte at a time
fclose(fid);
success=1;
end
```