ZIGBEE DATA ACQUISITION (DAQ) MODULE FOR THE CORNELL UNIVERSITY BAJA RACING TEAM

A Design Project Report

Presented to the Engineering Division of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering (Electrical)

by

Joseph S. Neiss

Project Advisor: Professor Bruce Land

Degree Date: January 2009

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Zigbee Data Acquisition (DAQ) Module for the Cornell University Baja Racing Team

Author: Joseph S. Neiss

Abstract: The project is to develop a sensor data acquisition (DAQ) module that can communicate with a PC using the IEEE 802.15.4 Zigbee protocol. The data collected can be used by the Cornell University Baja Racing Team to assist in optimizing their mechanical design for SAE Mini-Baja competitions.

The technical approach will require the design, fabrication, integration and testing of a DAQ sensor circuit board, the development of a Zigbee data packetization scheme, and a user-friendly interface to analyze the captured data on a Windows XP personal computer.

Integration testing consists of DAQ Module POST tests and Base Station Console simulation testing with Baja vehicle system testing occurring in Spring 2009.

Deliverables will include a DAQ schematic, bill of materials, board layout, PC interface code, and a prototype of the DAQ Module. The Master of Engineering Design Report will serve as the technical reference and user's manual.

Report Approved by

Project Advisor: ______________________________ Date: ______________

Table of Contents

Executive Summary

Each year, the Cornell Baja Racing Team competes against colleges from around the world in the Society of Automotive Engineer's (SAE's) Mini-Baja Competition. The competition requires college teams to design and construct an off-road racing vehicle. The vehicles are judged in several categories including a Design Element and Dynamic Element. The team is constantly seeking ways to optimize the mechanical performance of their vehicle to gain a greater advantage in the Dynamic Element events as well as points for ingenuity in optimization methods and unique systems.

The design concept addresses the team's need for a wireless data acquisition system while maintaining low weight, low power, small size, and the versatility for future design expansion. The team uses several metrics to determine performance including the continuously-variable transmission (CVT) speed, engine RPM, and the RPM-to-speed ratio. In order to assist in the data acquisition and analysis of these metrics, I have developed an in-vehicle Data Acquisition (DAQ) Module, a complementary base-station PC interface to analyze the data, and a wireless communication protocol between the DAQ and base-station using the IEEE 802.15.4 Zigbee standard.

Integration testing of the entire system in-vehicle will occur when the Baja vehicle has been completed in the Spring of 2009. DAQ Module integration consists of a series of POST tests that exercise the on-board circuitry. The main microprocessor issues several initialization and configuration commands to the General Display, SD Card and Zigbee Transceiver. DAQ Module testing has been executed and recorded in a video online at: http://www.youtube.com/watch?v=fkLEGHPIz50. Base Station console testing has been performed with simulated serial inputs from an RF modem. The Base Station code has been proven to correctly parse, identify, display, and store properly formatted data packets per the intended design.

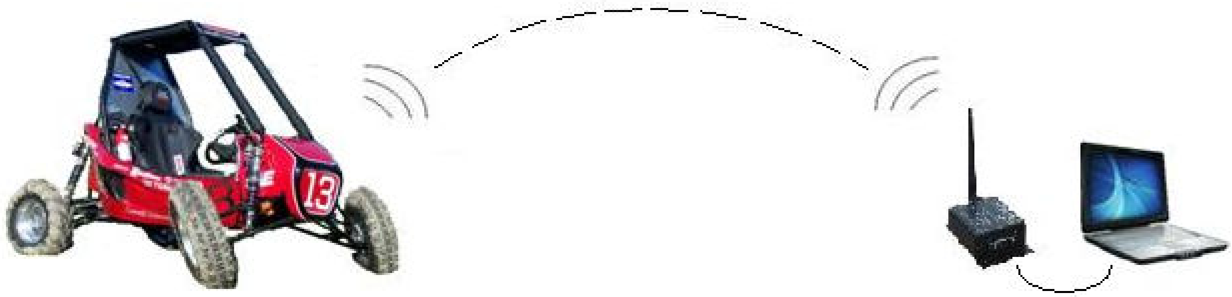
An electrical schematic, bill of material, board layout and prototype module were developed for the DAQ module. A base-station console and RF modem interface were developed using Labview. And code flow charts were developed as an operational aid for the mechanical team and future developers.

System Purpose & Overview

Each year, the Cornell Baja Racing Team competes against colleges from around the world in the Society of Automotive Engineer's (SAE's) Mini-Baja Competition. The competition requires college teams to design and construct an off-road racing vehicle. The vehicles are judged in several categories including a Design Element and Dynamic Element. The team is constantly seeking ways to optimize the mechanical performance of their vehicle to gain a greater advantage in the Dynamic Element events as well as points for ingenuity in optimization methods and unique systems.

The team uses several metrics to determine performance including the continuously-variable transmission (CVT) speed, engine RPM, and the RPM-to-speed ratio. In order to assist in the data acquisition and analysis of these metrics, the system design concept addresses the team's need for a wireless data acquisition system while maintaining low weight, low power, small size, and the versatility for future design expansion. A high-level depiction of the system is shown in Figure 1 - System Illustration. The data used by the DAQ system would allow the Mechanical Team to optimize their tire pressure, CVT weights, belts, and make other adjustments as necessary.

Figure 1 - System Illustration



The DAQ system consists of two complementary units: the in-dash DAQ module and the PC "base station". The DAQ module is a microcontroller-based circuit board with interfaces to sensors on the CVT and engine, a wireless transceiver, an SD memory card, LED circuitry, and an LCD display circuit. The LCD display circuit design and function is currently in development and is completely separate from the data acquisition, storage and wireless transmission functions presented as part of the DAQ system. Because the DAQ will be in mobile system, it is powered by common "AA" batteries and controlled by an "ignition" toggle switch on the dashboard console. The DAQ microcontroller performs the necessary calculations on the sensor inputs and packetizes the data appropriately for storage on an SD card, display on the LEDs, and transmission to the base station system via the wireless transceiver. The DAQ microcontroller also has provisions for communication of the same packetized data to the LCD circuitry.

The Base Station system is designed around a user console interface with a serial connection to the wireless transceiver. Through the console, the user is be able to initialize the wireless connection, choose a text file in which to store the data, and view near real-time data with graphical indicators. The console is designed with considerations for technical expansion to allow for future additional sensors depending on the anticipated needs of the team. Configuration parameters are modifiable on the console to assist in troubleshooting and the evaluation of additional sensors in the future. A screen capture of the console interface is attached in Appendix E - Labview Base Station Console.

Design Requirements

The Cornell University Baja Racing Team did not have any specific quantitative requirements for the design therefore there are no “shall” requirements. After several discussions with the Mechanical Engineering team as to the intended usage and scope, I compiled at the following derived requirements:

1. General Operation
    - 1.1. The DAQ Module should have a method of capturing Engine RPM and Speed data.
    - 1.2. The DAQ Module should be able to capture Engine RPM and Speed data with a minimum rate of 2 datapoints per second.
    - 1.3. The Engine RPM-to-Speed ratio should be able to be calculated from the data captured from paragraph 1.2
    - 1.4. The DAQ Module should be capable of operating while the Baja vehicle is actively driving (mobile).
    - 1.5. The DAQ Module should have a method of communicating wirelessly with a laptop running a Windows XP Operating System.
    - 1.6. The DAQ Module design should allow provisions for additional sensors.

2. Wireless Communication
    - 2.1. Wireless communication should be designed to allow for maximum transmission range.
    - 2.2. Wireless communication should be designed for ease of setup and use.

3. Power Consumption
    - 3.1. The DAQ Module should be designed to minimize power consumption.
    - 3.2. The DAQ Module should have a replaceable or rechargeable battery.
    - 3.3. The DAQ Module should have a commonly available type of battery.

4. Cost
    - 4.1. The DAQ Module should be designed with cost-efficiency in mind.

5. Mechanical Design
    - 5.1. The DAQ Module design should consider mechanical mounting considerations.
    - 5.2. The DAQ Module should be capable of being mounted to the front dashboard just above the steering column.
    - 5.3. The DAQ Module size should be of sufficiently small size to allow installation in the front dashboard.
    - 5.4. The DAQ Module should be designed with expected vehicle vibration and operator/handler “indelicate” usage in mind.

Since the intention of the DAQ is also to be used in SAE competition for Design Element competition, the DAQ Module must be compliant to the SAE Baja Competition Rules. The relevant rules from the 2009 SAE Baja Competition are listed with corresponding paragraph numbers below.

Relevant 2009 SAE Competition Rules

21.4.15 Battery Requirements

The batteries must be sealed and not leak in the event of a roll over. The batteries can only provide power to accessories on the vehicle (brake light, reverse light & beeper, data acquisitions, and other instrumentation). Final approval on any batteries used will come from the National Technical Inspectors. The battery must be able to provide power to safety items for the duration of the entire event. Cars will be black flagged if safety equipment is not functioning. Batteries must be mounted with sound engineering practice. The mounting must prevent the battery from coming loose during a roll over.

21.4.16 Onboard Instrumentation/Data Acquisition

Onboard instrumentation/data acquisition is allowed; the power for this instrumentation must be from approved batteries per 21.4.15.

32.7 Kill Switches

Each vehicle must be equipped with two (2) easily accessible kill switches turning off the ignition and the entire electrical system of the car. Brake light, reverse light and reverse alarm are required not to be turned off.

Design Considerations

When developing the solution for the DAQ system, there were several technical solutions that were considered. To help determine the optimal solution, trade studies were conducted to determine the weighted benefit of potential solutions. The critical areas for design consideration were:

- Processing element
- Wireless protocol
- Power source
- PC and user interface

Processing Element

The selection of the processing element is between using various microcontrollers or FPGAs. The derived requirements of the project are to have low power draw, the necessary support for I/O, low cost, be easily programmable for future expansion, and replaceable in the event of damage. While FPGA's are favored by supporting more than enough I/O, there are more advantages by using microcontrollers for this application. The standard FPGA "QFP" or "BGA" package types do not allow for easy replacement without moderate to advanced soldering skills and expensive solder rework equipment. On the other hand, microcontrollers (and most common package ICs) can be purchased with sockets for very little additional cost. When working on an undergraduate team designing a vehicle for racing, it is extremely beneficial to have the ability to replace components that may have been damaged during installation, racing, or handling.

A PIC18LF4585-I/P Microchip Technology microcontroller is the solution that was selected for the DAQ system. It has unique SPI and UART interfaces, 24Kx16 program memory size, 3.25Kx8 RAM size, several integrated Analog-to-Digital Converters (ADC's), an industrial temperature range, and a 40 pin DIP package. When used in conjunction with an Aries "Eject-a-DIP" socket, the microcontroller is easily replaceable. The cost for both the microcontroller and socket, as of the printing of this document, is $20 which is still about ½ the price of a comparable FPGA.

Wireless Protocol

Currently available commercial technology allows for a wide range of wireless solutions including, but not limited to, IEEE 802.11 (Wi-Fi), IEEE 802.15.1 (Bluetooth) and IEEE 802.15.4 (Zigbee). The key metrics to consider between these protocols are data rate, power, range of transmission, cost to implement, and complexity in implementation. For the purposes of the DAQ system, security features are not considered, but are recommended as an interesting expansion for any future developers. The general trade-offs between the technologies are indicated in Table 1 - Wireless Technology Average Metric Comparison.

The advantages of Zigbee over Wi-Fi and Bluetooth are the lower average power consumption for comparable ranges. Zigbee transceivers, at higher power, can broadcast up to 1 mile line-of-sight. The data rate is substantially lower, but will suffice for transmitting at the rate required by paragraph 1.2. Another advantage of Zigbee is the ability to transmit with "transparent" networking. The DAQ Module and PC RF Modem can act as serial buffers, transmitting between the two nodes with very little network overhead as comparable to Wi-Fi. For initialization purposes, this is an ideal solution for a team looking to "plug-n-play" a wireless DAQ system.

Table 1 - Wireless Technology Average Metric Comparison

	Wi-Fi	Bluetooth	Zigbee
Data Rate	19 Mbps[1]	3 Mbps[2]	250 kbps[3]
Output Power	32 mW	2.5 mW	1-2 mW
Range	95 m[4]	10 m	100-120 m
Relative Cost	High	Low	Low

Power Source

The derived power requirements under paragraph 3 and the mobility requirement of paragraph 1.4 focused the choice of power source to be a common type of battery. A comparison of commonly used batteries is shown in Table 2 - Common Battery Comparison.

Table 2 - Common Battery Comparison

Battery Type	Voltage (V)	Weight (g)	Size (cm^3)	Power (mAh)			
				Alkaline	NiMH	Zinc-Carbon	Lithium
Coin	1.5 – 3	3 – 7	~1	115	NC[5]	NC[6]	250
AAA	1.5	10 – 13	4	1,200	900	540	500
AA	1.5	21 – 27	8	2,700	2,300	1,100	2,900
C	1.5	65 – 80	27	8,000	6,000	3,800	7,200[7]
D	1.5	105 – 160	57	12,000	10,000	8,000	19,000[8]
9V	9	38	22	565	175	400	540

The majority of integrated circuit components require input voltages of 5V or lower, such as 3.3V. An input voltage of 5V can be broken down with other smaller voltage regulators for 3.3V IC's and signaling, so the battery should ideally source 5V. The expected power consumption was estimated from the datasheets of the major components (see Appendix B - Bill of Materials) and tabulated in Table 3 - Power Consumption.

[1] Typical bandwidth using IEEE 802.11g

[2] Data rate using Bluetooth 2.0

[3] Raw over-the-air data rate per channel in the 2.4GHz band

[4] Using typical stock antenna, line of sight, outdoors

[5] Not commonly available.

[6] Not commonly available.

[7] Not commonly available. Voltage is 3V.

[8] Not commonly available. Voltage is 3.6V

Table 3 - Power Consumption

Component	Input Current	Voltage Source	Power
PIC18LF4585	200 mA (max)	5 V	1.00 W
MAX7221	330 mA (typ)	5 V	1.65 W
Zigbee 50mW Transceiver	340 mA (max)[9]	3.3 V	1.12 W
Total	-	-	**3.77 W**

Based on an approximate 4.02 W estimate for the power requirement, the calculated current on a 5V battery source is 754mA. With a maximum expected continuous runtime of 2 hours without battery replacement, the expected power requirement is 1,508 mAh. Given this calculation and the power-to-weight ratio, four Alkaline, Lithium-ion, or NiMH AA batteries in series are considered the ideal choice and should provide 2,300 to 2,900 mAh. Temperature-dependent battery performance was acknowledged but considered outside the scope of the design for further investigation.

For compliance to the SAE competition rule 32.7, regarding the scope of Kill Switches, the battery is connected in serial with the kill switch to chassis ground. This will effectively turn off the DAQ Module in the event the Kill Switch is used.

User Interface

The goal of the user interface is to provide an intuitive and user-friendly interface for initiating data capture and analysis. MATLAB, Labview and Visual Basic modules were considered as potential solutions for a user interface to the DAQ system. MATLAB libraries provide a wealth of computational and analytic tools, but they are highly specified and would require a specialized serial driver to interact with the RF modem as a PC I/O device. A Visual Basic module, as well, would require a specialized serial driver to interact with the modem. It also lacks the available processing tools available in MATLAB. Labview is GUI based with easily modifiable flow diagrams and graphical indicators. Labview also includes several types of device drivers and file I/O "blocks" that would facilitate storing information retrieved from the RF modem to a file.

Labview is the solution selected for the DAQ System's PC interface. It was chosen primarily for several reasons. The Labview libraries contain Serial communication, File I/O, and graphical indicator blocks available for easy integration in a GUI interface. The interaction and flow of data between these blocks is left to the designer. With forward planning, the designer's architecture can easily allow for additional sensor indicators and file formatting.

[9] Based on 295mA transmit current and 45 mA receive current (peak, not continous).

Technical Approach

The DAQ Module and PC interfaces required distinctly separate, yet complementary design. A block diagram of the DAQ Module is included in Appendix A - DAQ Module Block Diagram. The purpose of the DAQ Module is to acquire the data from the sensors, process it, display any debug indications via the LED display, store the data to the SD card, and transmit the data through a Zigbee transceiver.

A screen capture of the base station console is included in Appendix E - Labview Base Station Console. A high-level flow diagram of the console functionality is illustrated in Appendix F - Labview Console Flow Diagram. The purpose of the base station console is to provide a serial interface to a Zigbee RF Modem, capture all data to a .txt file, and to provide some graphical indications in "real-time" of the data. The data cannot be displayed in real-time because the DAQ Module is programmed to transmit in packet bursts. The received data therefore cannot be considered true real-time data.

DAQ Module Physical Design and Construction

The basis for the DAQ Module is based around the microcontroller selection and architecture. Refer to Appendix D - DAQ Module Physical Layout for an illustration of the actual board layout. A Microchip Technology PIC18LF4585-I/P is the centerpiece of the board, indicated by marker (1). The microcontroller is connected to three input signals to capture the CVT-speed, sparkplug ignition frequency, and driver dash button activity.

A Cherry Hill Magnetic-Proximity Sensor is mounted securely near the drive shaft of the engine, aimed at the drive shaft. With each rotation of the drive shaft, the CVT rotates. With a fixed gear ratio, the driven axle speed can be calculated. The driven axle speed, along with the tire diameter can be used to calculate the number of rotations the tires experience. This correlates directly to the speed of the vehicle assuming there is no slip between the tires and the ground. This implies that the speed experiences a margin of error when the car is in the air or on slippery terrain.

Figure 2 - Briggs & Stratton 10 HP OHV Intek Engine

The engine used in the Cornell University Baja vehicle is a Briggs & Stratton 10 hp OHV Intek, as regulated by SAE Competition rules. A picture of the engine is shown in Figure 2 - Briggs & Stratton 10 HP OHV Intek Engine. To capture the tachometer data, a single wire is wrapped around the base of the sparkplug in the engine. As the engine fires, the wire will acquire a voltage based on Lenz's Law. As the sparkplug fires, the current through the engine causes a potential voltage in the wire by the magnetic field it creates. The sparkplug fires at fairly regular intervals, increasing as more energy is demanded of the engine. The timing of the intervals can be captured to determine the Engine RPM. It is important to note that the engine can create very high potentials on the wire this way. While the exact amplitude is irrelevant for calculating the Engine RPM, it may cause damage to the DAQ Module if not accounted for. For this reason, the DAQ Module includes an opto-isolator on the Engine RPM signal, effectively “isolating” the microcontroller input from any spikes in voltage from the engine.

The driver input is a simple contact-switch pushbutton which can be mounted within reach of the driver. The driver pushes the button after the completion of a lap to capture the lap time in the microcontroller. With some code modification, the button can also be used for future development to crudely communicate with the pit crew through the wireless transceiver.

When these inputs are processed, the data is packetized and sent to the onboard SD card and Zigbee transceiver. For development and debug purposes, a Maxim MAX7221 LED Driver and a five 7-segment LED array are wired to the microcontroller as well. Communication with the SD card and LED Driver is through an SPI bus arranged in a parallel architecture. The Zigbee transceiver is connected directly to the UART channel on the microcontroller.

DAQ Module Mechanical Interface

There are nine connectors to the DAQ module, each with reference designators starting with “J”. The function of each connector is shown in Table 4 - Connector Functions. Refer to the schematics in Appendix C - Electrical Schematic for signal details. The battery harness assembly is connected the DAQ via the J1 connector bringing 5V and Ground onto the board. The 5V power bus goes off-board to the ignition switch via the J2 connector. The ignition switch is a simple contact switch activated when the user flips the toggle of the switch connecting the 5V power bus to the majority of the DAQ Module circuitry. Without the appropriate harnesses in J1 and J2, the DAQ will not operate at all, nor will the microcontroller be capable of being programmed.

Table 4 - Connector Functions

Connector Reference Designator	Function	Connector Type
J1	Battery Input	Molex shielded
J2	Ignition Switch Input	Molex shielded
J3	Microcontroller Programmer	Breakaway Header
J4	SD Card (off-board header)	Molex shielded
J5	Tire Diameter Tuner	Breakaway Header
J6	Misc Sensor I/O	Molex shielded
J7	LCD Microcontroller	TBD
J8	LCD I/O	TBD
J9	SD Card (on-board socket)	SD Card Socket

The J3 connector serves as an onboard programmer header for the DAQ microcontroller. The target device voltage is provided at the board connector for recognition by the programmer. The J4 connector and J9 socket are redundant connectors. The J9 socket is intended to allow an SD card to be installed into a common SD socket connector. The SD socket datasheet was rather ambiguous regarding its physical layout and the orientation of some SD Card-specific signals such as the Card Lock signal and Card Present signal. As a risk reduction measure to interpreting the SD socket datasheet incorrectly, the

J4 connector was added to allow for a modifiable harness between the DAQ Module and an off-board SD Card breakout board shown in Figure 3 - SD Card Breakout Board. Because SPI is not intended for long distance transmission, the breakout cable for the J4 connector must be as short as possible, specifically no more than a few inches. Through integration testing, it was discovered that communication would fail intermittently at a cable length of approximately 4 inches. The basic operation of the SD Card socket connector, J9, has not been verified in integration testing as of the creation of this report. For future development, the J4 connector may be removed as a risk reduction if J9 is validated operationally. The card lock signal and other supplementary signals on J9 currently have no way of being tested without modification of the DAQ Module hardware and software.

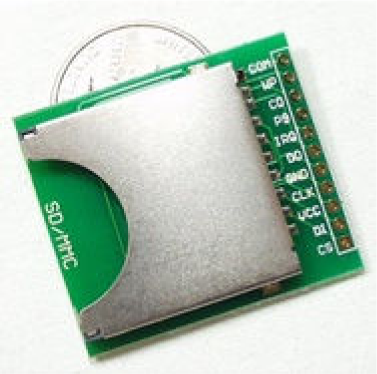


Figure 3 - SD Card Breakout Board

The J5 connector allows the user to insert and remove a potentiometer to configure the wheel diameter, in inches, and store the value in the DAQ microcontroller's EEPROM. The output of the potentiometer is read by a DAQ microcontroller ADC and converted into a wheel diameter value. For future development, it is recommended to replace the J5 connector with a surface mount potentiometer to reduce the cost of the system. J6 is the Misc I/O Connector of the DAQ Module. The design intent is to channel all I/O external to the dashboard enclosure through the J6 connector. The CVT sensor, Engine RPM signal, and driver pushbutton signals are routed through this connector and out of the dashboard. An 18-pin connector was chosen to allow ample room for additional sensors in future designs. The LCD circuitry of the DAQ utilizes the J7 and J8 connectors for the LCD microcontroller and LCD I/O respectively. Currently, this is a work in progress and considered outside the scope of the DAQ system design.

The board measures 6.5" x 4.275", which is approximately the size of the LCD display to be used with the LCD circuitry on the DAQ Module. The LCD mounting plate has four standoff holes at its four corners. The DAQ Module is designed to accommodate standoffs in the same locations for simple mechanical assembly between the DAQ Module and the LCD display. The Baja vehicle's dashboard enclosure will provide a bezel faceplate that will mate with the LCD display mounting plate and allow the entire DAQ Module assembly to be mounted securely to the dashboard of the vehicle.

<u>DAQ Module Microcontroller Operation</u>

The DAQ microcontroller provides the interface between the sensor I/O, SD Card, Zigbee transceiver, and future LCD circuitry. The microcontroller code is written in the "C" programming language and was compiled using Microchip Technology's MPLAB IDE and the C18 compiler. The code in *main.c* is designed to run continuously with no exit conditions until power is removed from the DAQ Module.

Upon application of power, the DAQ Module will perform a series of "one-time" initialization commands before entering an infinite "while" loop in *main.c*. The initialization sequence performed is:

1. All *main.c* variables are instantiated and assigned initial values as necessary.

2. Setup the SPI bus. This opens the microcontroller's SPI bus and configures the SPI device chip selects as outputs on the microcontroller. The LED Display Driver is configured over the SPI bus and the General Display LEDs are cleared.
3. Setup the UART bus. This opens the microcontroller's UART channel to the Zigbee transceiver.
4. The timers and capture registers are opened and configured for capturing time, CVT and Engine RPM signals.
5. The last stored wheel diameter value is retrieved from the microcontroller EEPROM to be used in speed calculations.
6. General Display "POST" and SD Card Initialization. A simple counter is used to exercise all segments on the General Display LEDs to identify bad LED segments. The counter will count from 0 to 9 with each 7-segment LED incrementing simultaneously. The counter will then decrement from 9 to 0 in a similar fashion. As the counter is incrementing to exercise the LED segments, the microcontroller will attempt to initialize the SD card by writing a series of configuration commands.

   Note: The effective time spent initializing the SD card, if unsuccessful, will increase the total time the LED's will increment from 0 to 9. If the SD card is able to be initialized, the incrementing will proceed "quickly". If the SD card is unresponsive or not present, the incrementing will proceed "slowly".

After these commands have been executed, the microcontroller will enter its "forever" loop. In general, the "forever loop" will perform the following high-level tasks:

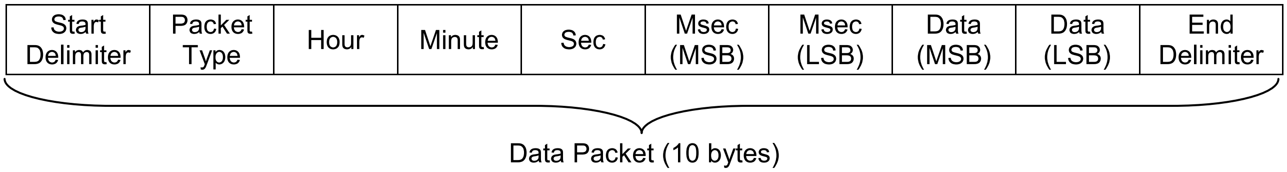
- If 51 new datapoints have been collected, write all 51 datapoints to the SD card in a block write.
- If any new datapoints have been collected, send them to the Zigbee transceiver.
- If the driver is depressing the pushbutton for an extended time, check the Tire Diameter Tuner signal for a change in the tire diameter as indicated by the user. Another depression of the pushbutton will save the new tire diameter value and return to the main function.
- If the driver is depressing the pushbutton for a short time, interpret it as a lap time "stopwatch" indication.
- Otherwise, capture the CVT and sparkplug signals, calculate the speed and tachometer reading from them, and store them as datapoints.

Datapoints are structs defined with variables indicating the type of data point, the course-time hours, minutes, seconds, milliseconds, and the data point data "value". The type of data point can currently be a Speed, Tachometer or Fuel type. As datapoints are collected, they are sent in packetized form to the Zigbee transceiver individually and to the SD card in a block write when enough datapoints have been collected to fill an SD card sector.

Data Packetization Scheme

The data capture packetization scheme is used for storing data to the SD card over the SPI bus and to transmit through the Zigbee transceiver over the UART channel. The scheme uses a fixed length packet size of 10-bytes. Each packet is structured as shown in Figure 4 - Data Packet Structure. Each field is 1 byte in length.

Figure 4 - Data Packet Structure



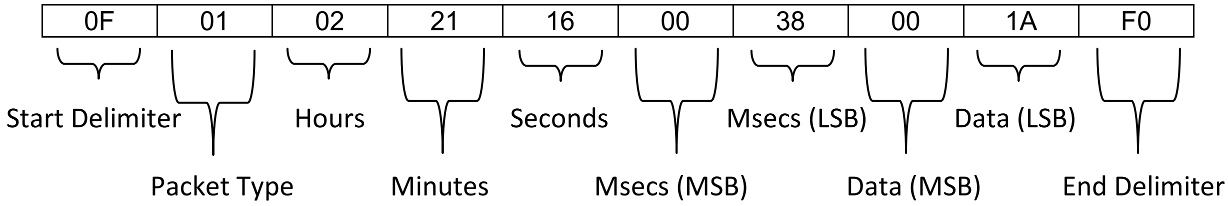
Start Delimiter	Packet Type	Hour	Minute	Sec	Msec (MSB)	Msec (LSB)	Data (MSB)	Data (LSB)	End Delimiter

The structure was designed to allow for all reasonable quantities and values of time, data, and packet types. The design currently uses three unique packet types to capture the data. They are defined as listed in Table 5 - Data Packet Type Values. The field values are easily changeable in both the DAQ module code and Labview Console code. The field values are defined in *define.h* for the C code and on the Labview Console as “flags”. They must be identical, respectively, between the Labview and C code in order to function properly. For future development, one may easily include additional data packet types for additional sensors such as accelerometers, GPS coordinates, and engine temperature with the appropriate code additions. Relevant hardware additions would also need to be included. A specific example of a data packet is illustrated in Figure 5 - Data Packet Example. The interpretation of the packet would be a “Speed” packet captured at 2h 33m 22.56s with a value of 26 (mph).

Table 5 - Data Packet Type Values

Packet Type	Field Value
Speed	0x01
Tachometer	0x02
Fuel	0x03

Figure 5 - Data Packet Example

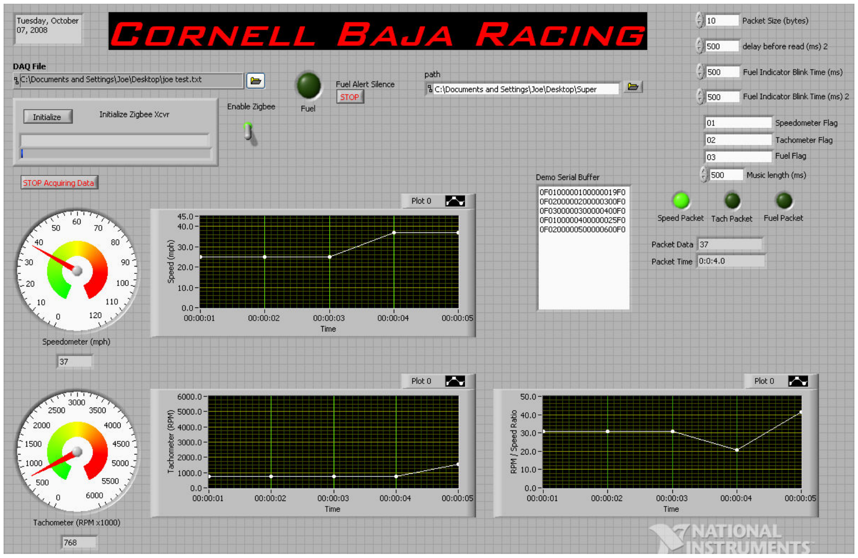


0F	01	02	21	16	00	38	00	1A	F0

Base Station Console Operation

The Labview Base Station Console controls the serial channel interface between the PC and the Zigbee transceiver at the base station. It also graphically indicates the values of the data packets received from the DAQ Module and stores those values in a formatted text file on the PC. The high-level flow of the Labview Console operation is illustrated in Appendix F - Labview Console Flow Diagram. A screenshot of the console is shown in Figure 6 - Screenshot of Base Station Console. Refer to Appendix E - Labview Base Station Console for a larger picture of the console.

Figure 6 - Screenshot of Base Station Console



Tuesday, October 07, 2008

DAQ File: C:\Documents and Settings\Joe\Desktop\joe test.txt

path: C:\Documents and Settings\Joe\Desktop\Super

10 Packet Size (bytes)
500 delay before read (ms) 2
500 Fuel Indicator Blink Time (ms)
500 Fuel Indicator Blink Time (ms) 2
01 Speedometer Flag
02 Tachometer Flag
03 Fuel Flag
500 Music length (ms)

Demo Serial Buffer

```
0F0100000100000019F0
0F0200000200000300F0
0F0300000300000400F0
0F0100000400000025F0
0F0200000500000600F0
```

Packet Data: 37
Packet Time: 0:0:4.0

Speedometer (mph): 37
Tachometer (RPM x1000): 768

When the Labview console is first instantiated, the user is able to configure certain operating parameters by modifying the parameter values on the right side of the console, shown in Figure 7 – Console Parameter Section. The user then “executes” the Labview module. The module will open a .txt file, selected by the user, to store all received data for analysis at a later time if desired, shown in Figure 8 – Console Parameter Section.

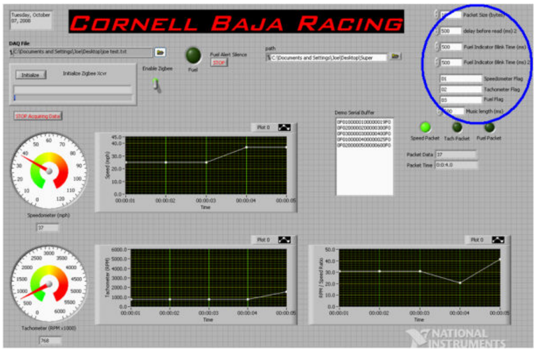


Figure 7 - Console Parameter Section

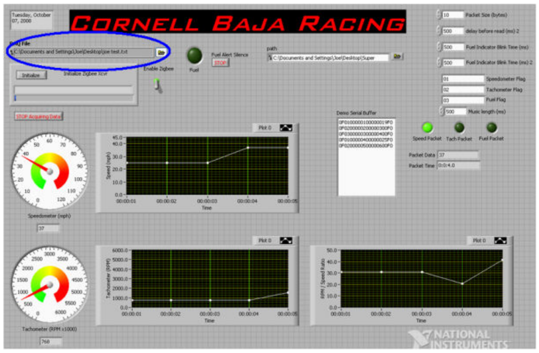


Figure 8 - Console Data File Entry Field

The user would then choose when to “Initialize” the Zigbee RF Modem. Labview will send a few basic commands to ensure the module is present and then continuously pull any data from the serial buffer for processing per the DAQ Module packetization scheme format. The initialization area of the console is indicated in Figure 9 - Zigbee Initialization Window   Figure 10 - Console Graphical Indicators. The graphical indicators on the console display the recent data for speed, engine RPM, and the calculated "RPM / speed" ratio shown in Figure 10 – Console Graphical Indicators in blue, green, and red respectively.

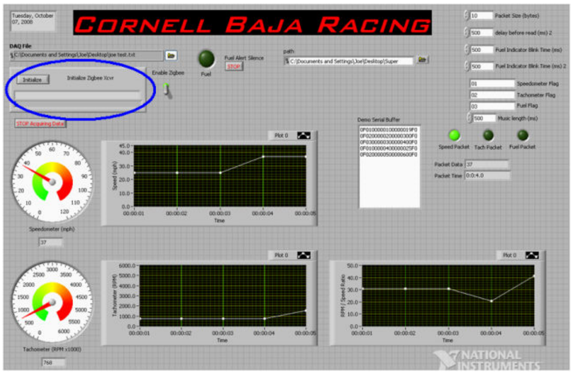


Figure 9 - Zigbee Initialization Window

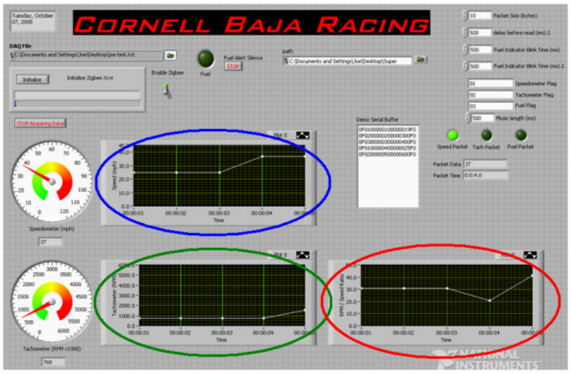
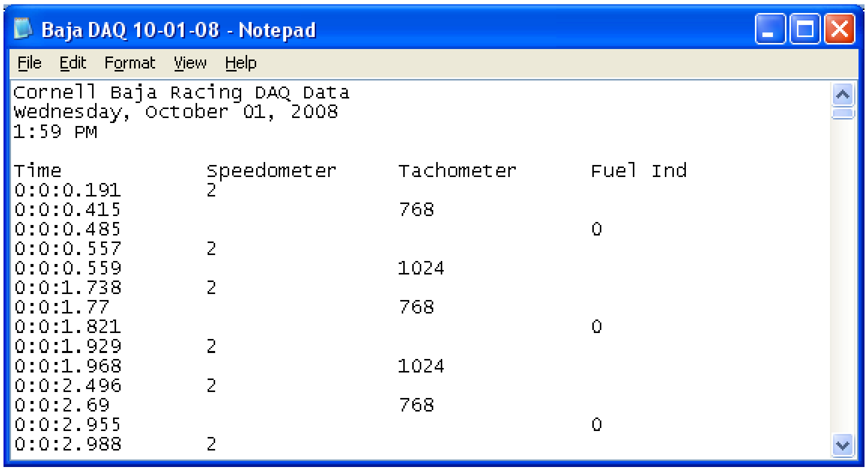


Figure 10 - Console Graphical Indicators

The module will read the serial buffer stream, parse the packet fields, convert to decimal and concatenate values as necessary, update the appropriate indicators, and store the values to the .txt file with a pre-defined (tab-delimited) format. With tab-delimited formatting, the data headers and the data can be copied and pasted directly into an Excel spreadsheet and plotted using Excel graphing tools. It can also be used as an input to a MATLAB program as well for more advanced analysis using MATLAB toolboxes. A screenshot of an example DAQ data file produced by the Base Station Console module is shown in Figure 11 - Example DAQ File.

Figure 11 - Example DAQ File



```
Cornell Baja Racing DAQ Data
Wednesday, October 01, 2008
1:59 PM

Time            Speedometer     Tachometer      Fuel Ind
0:0:0.191       2
0:0:0.415                       768
0:0:0.485                                       0
0:0:0.557       2
0:0:0.559                       1024
0:0:1.738       2
0:0:1.77                        768
0:0:1.821                                       0
0:0:1.929       2
0:0:1.968                       1024
0:0:2.496       2
0:0:2.69                        768
0:0:2.955                                       0
0:0:2.988       2
```

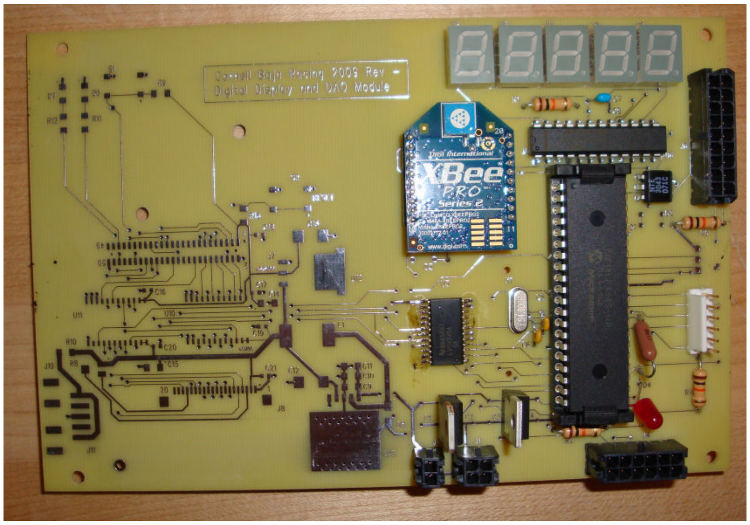
The module will continuously check the serial buffer for data until the user selects the "Stop Acquiring Data" button. This will force the module to complete processing of the remaining serial buffer packets, then close the serial port, .txt file and report any errors to the user.   The module will then return to an

“Idle” state where the user may restart the process by modifying any configuration parameters as necessary.

Assembly, Simulation and Testing

The DAQ Module printed circuit board (PCB) schematic and board layout was captured using PCB123 software. The PCB was purchased through a Baja team sponsorship with Sunstone Circuits®. After the board and components were received, they were assembled and soldered to the board. A picture of the assembled board is shown in Figure 12 - Assembled DAQ Module. The LCD circuitry components on the board (left-half) were not assembled during integration. Also note that Figure 12 shows a DAQ module without a soldermask. The first DAQ was manufactured without a soldermask in anticipation of a board re-spin due to design errors or improvements. Ordering without a soldermask saved 50% of the cost of the board through Sunstone Circuits®.

Figure 12 - Assembled DAQ Module


Cornell Baja Racing 2009 Rev -
Digital Display and DAQ Module
XBee
PRO
Series 2

The board was compiled using MPLAB IDE software and programmed using a USB-based MPLAB ICD 2 programmer to the DAQ Module J3 connector.

DAQ Module Testing

Complete DAQ Module testing cannot be performed until the Baja vehicle has been built, which is scheduled for Spring 2009. Currently, the DAQ Module runs a Power-On Self-Test which initializes and tests the General Display circuitry, SD card and Zigbee transceiver. Refer to Appendix H - DAQ Module Code Excerpts for sections of POST and initialization code.

POST, as executed in the DAQ Module, will initialize and exercise the General Display LED's, SD Card circuit and Zigbee Transceiver. The user will notice the following activity as POST is being executed:

1. Power is applied by the user.
2. The Debug LED will blink once to indicate that the microcontroller is "alive".

3. The General Display LED's will count from 0 to 9 and then from 9 to 0. This confirms the SPI channel is opened, the LED Driver is functional, and will allow the user to check for damaged 7-segment LEDs.
4. The General Display will indicate "00001" as the PIC performs the SD Card initialization and POST test. Initialization sends basic configuration commands to the SD card to configure it for block writing. If any errors* are found during the test, the Debug LED will blink 5 times before continuing on.
5. The General Display will indicate "00002" as the PIC performs the Zigbee POST test. If any errors* are found during the test, the Debug LED will blink 5 times before continuing on.

*Note: The user is responsible for troubleshooting the DAQ Module if any errors are indicated.

Testing of the Initialization and POST code was successful in identifying when the SD Card and/or Zigbee were attached or were not attached or initialized properly. The SD card code was also successfully tested by writing some "dummy" packets to the card and verifying the card contents on a PC using an SD/MMC card reader. A video demonstration of the DAQ Module testing is online at: http://www.youtube.com/watch?v=fkLEGHPIz50.

Base Station Console

Testing of the Base Station Console was done by simulating the serial input from the Zigbee RF Modem and verifying the correct packet parsing, interpretation, display, and data file storage. A "Demo Serial Buffer" was created inside the console to allow the user to simulate a number of packets in the serial buffer. An "Enable Zigbee" toggle switch was also added in the console to bypass the initialization of the Zigbee transceiver and pull the serial data from the Demo Serial Buffer instead.

In simulation, the console performed as expected with all properly formatted packets. A serial buffer size of 5 packets (50 bytes) was input into the Console code and it was able to parse and interpret the packets, graphically display the data and store it to a data file. In a separate test, the console was also able to successfully initialize the RF Modem over the serial channel in the Zigbee Initialization code. This was verified by configuring some network parameters over the serial channel during the initialization sequence. RF Modem activity was noticed during this test, proving that communication was recognized.

Complete DAQ System Test

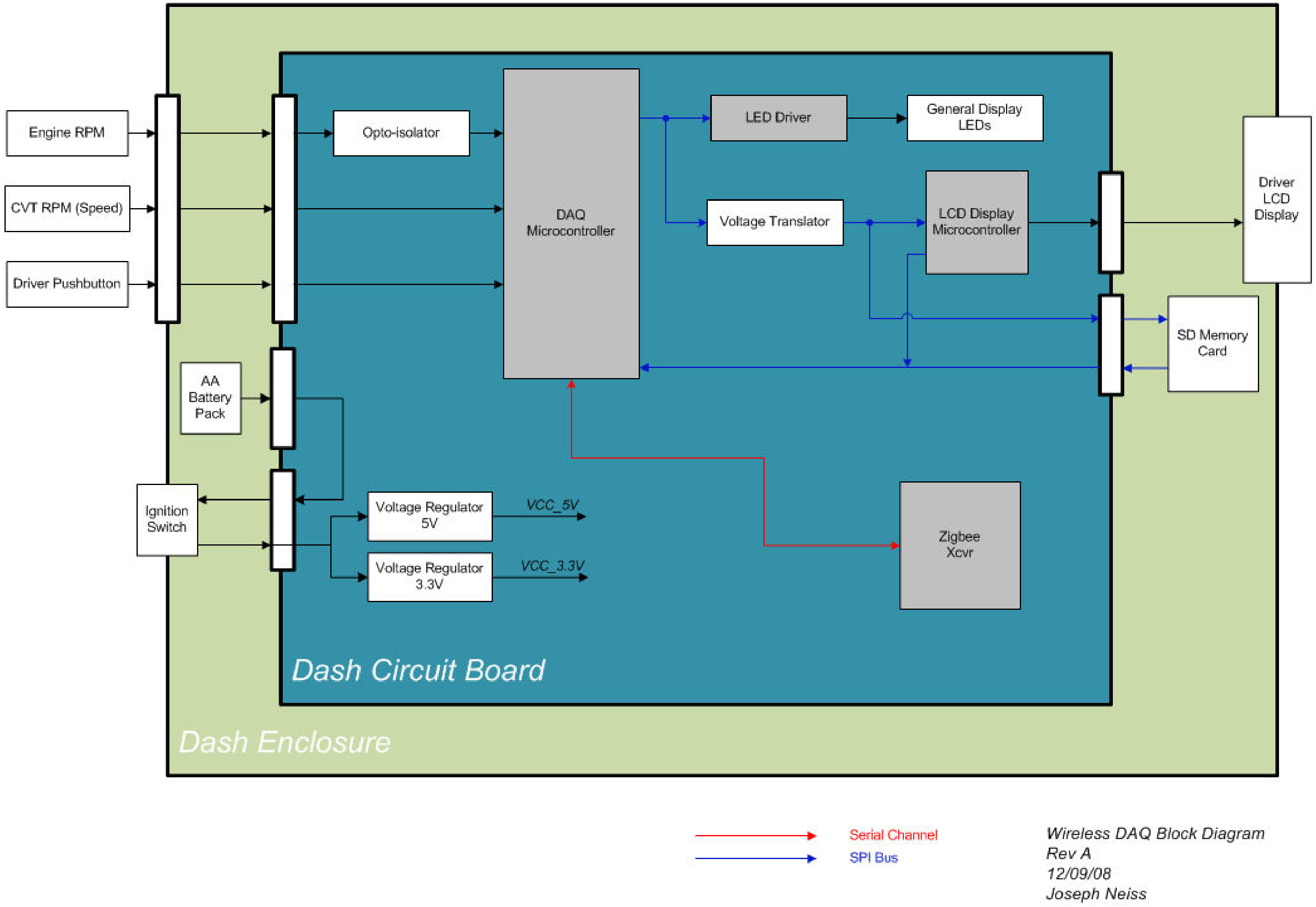
As mentioned previously, the complete DAQ System Test cannot be performed until the Baja vehicle has been built and the DAQ installed in the system. Once the sensors are installed in the car, the following tests will be performed to ensure proper operation:

1) Sensor inputs will be verified using the Debug LED to indicate sampling activity. The general display will be used to show calculated values. This will be used to verify the calculations are correct and/or that the data is in an expected range.
2) Verify successful SD card writes of collected sensor data in packetized form. This will verify that the data is being captured with a timestamp and packet type and that block writes to the SD card are occuring.

3) Transmit packets to the Base Station. Packetization integrity can be verified using an indicator display on the console. This will verify that the data stream and packetization scheme is synced between the DAQ Module and Base Station.
4) Verify that the received packets at the Base Station are stored to the data file appropriately. This functionality has already been tested with high confidence in the integration phase of the design.

Based on the testing already performed, the remainder of the system should function per the intended design. Errata and design notes are documented in Appendix G - Design Errata and Future Revision Notes.

Appendix A - DAQ Module Block Diagram

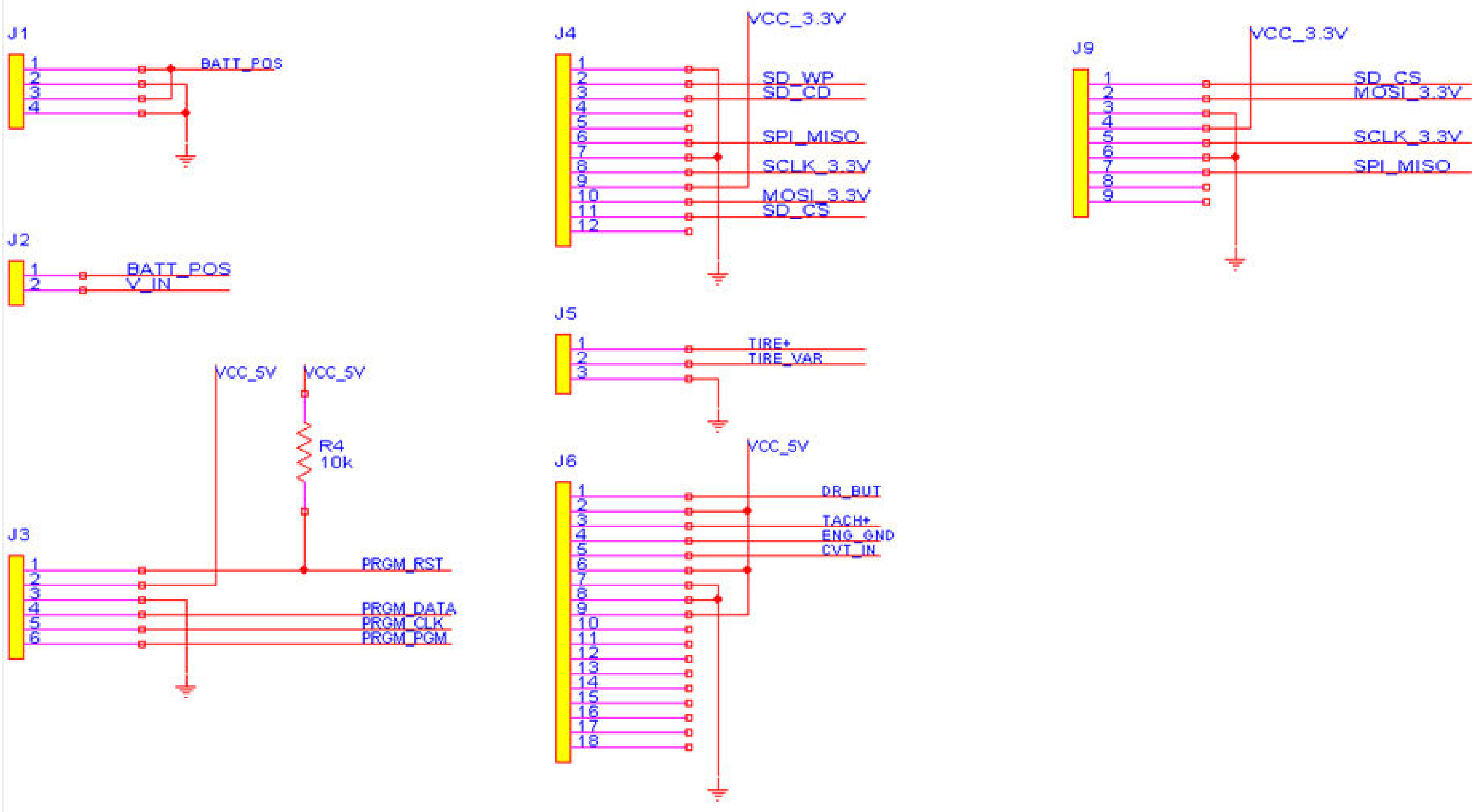


Appendix B - Bill of Materials

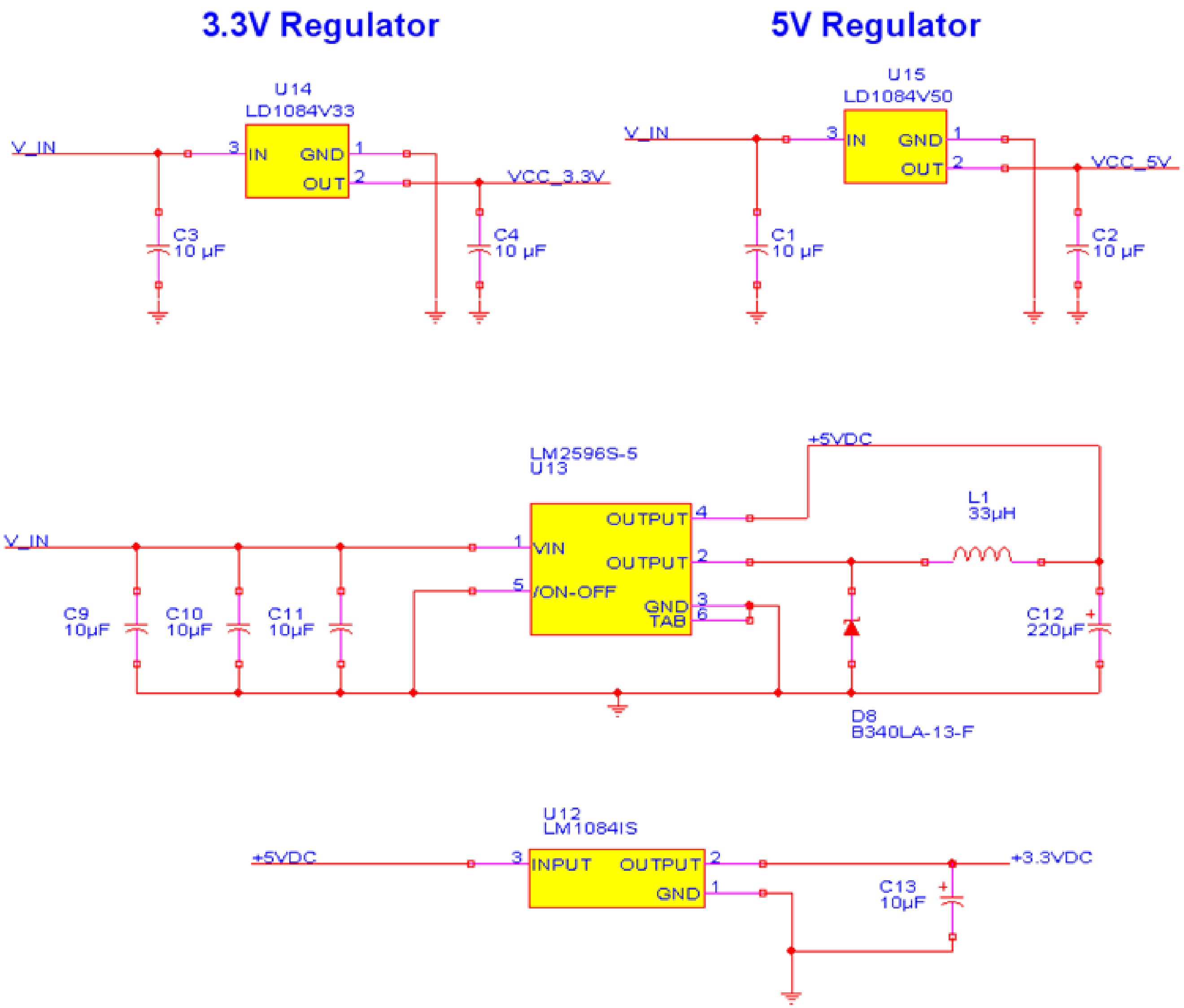
FN	Ref Des	Part Number	Noun	Manufacturer	Vendor PN	Vendor	Qty	Ext Cost
1	[redacted]	PCB_SCHEM_2009-001	Schematic	Baja ECE 2009	[redacted]	[redacted]	[redacted]	$0.00
2			PCB	Baja ECE 2009		Sunstone Electronics	1	$128.23
3	U1	PIC18LF4585-I/P	Microcontroller	Microchip Technology	PIC18LF4585-I/P-ND	Digikey	1	$12.13
4	U2	MAX7221CNG+	LED Driver	Maxim Integrated Products	MAX7221CNG+-ND	Digikey	1	$10.81
5	U3	SN74LVC4245ADW	Octal Bus Translator	Texas Instruments	296-8515-5-ND	Digikey	1	$0.88
6	U4	XBP24-Z7CIT-004	MODULE ZIGBEE-PRO W/CHIP ANT	Digi International	602-1100-ND	Digikey	1	$36.65
7	U5-9	LC3951-11EWAK	LED, 7 Segment, 1 Digit	LEDTECH ELECTRONICS	308179	Jameco Electronics	5	$1.00
8	U10							
9	U11							
10	U12	LM1084IS-3.3/NOPB	3.3V Regulator	National Semiconductor	LM1084IS-3.3-ND	Digikey	1	$3.15
11	U13	LM2596S-5.0/NOPB	5V Regulator, TO-263	National Semiconductor	LM2596S-5.0-ND	Digikey	1	$4.63
12	U14	LD1084V33	IC REG LDO POS 3.3V 5.0A TO-220	STMircoelectronics	425-2347-5-ND	Digikey	1	$2.64
13	U14	AP1084T33L-U	IC REG LDO 5A 3.3V TO-220	Diodes, Inc.	AP1084T33L-UDI-ND	Digikey	1	$1.61
14	U15	LD1084V50	IC REG LDO POS 5.0V 5.0A TO-220	STMircoelectronics	497-3417-5-ND	Digikey	1	$3.70
15	U16	NTE3043	Optoisolator NPN Transistor Output	NTE	526-NTE3043	Mouser Electronics	1	$3.10
16	J1	43045-0413	Connector, 4 pin, Batteries	Molex/Waldom Electronics Corp	WM2649-ND	Digikey	1	$1.61
17	J2	43045-0213	Connector, 2-pin, Dash Switch	Molex/Waldom Electronics Corp	WM2648-ND	Digikey	1	$1.16
18	J3	3-641213-6	Connector, 6-pin, Programmer	Tyco Electronics / Amp	A30774-ND	Digikey	1	$1.80
19	J4	43045-1213	Connector, 12-pin, SD Card SPI	Molex/Waldom Electronics Corp	WM2653-ND	Digikey	1	$3.34
20	J5	3-641213-3	Connector, 3-pin, Tire Tuner	Tyco Electronics / Amp	A30771-ND	Digikey	1	$1.05
21	J6	43045-1813	Connector, 18-pin, Misc I/O	Molex/Waldom Electronics Corp	WM2656-ND	Digikey	1	$4.73
22	J7	FLE-125-01-G-DV	Header, 50-pin	Samtec	11P4427	Newark	1	$4.06
23	J8						1	$0.00
24	J9		Socket, SD Card	4UCON	PRT-00136	Sparkfun Electronics	1	$3.95
25	J10-11	S3B-XH-SM4-TB (LF)(SN)	LCD Power Connector	JST	455-2263-1-ND	Digikey	1	$0.59
26			2mm 10pin XBee Socket	4UCON	PRT-08272	Sparkfun Electronics	2	$2.00

27		CFR-50JB-10K	Resistor, 10 kΩ, ±5%, 1/2W	Yaego	10KH-ND	Digikey	5	$0.10
28		FK26X5R1A106K	Capacitor, 10µF, 10V, ±10%, X5R	TDK Corporation	445-2874-ND	Digikey	5	$4.34
29		RPER71H104K2K1A03B	Capacitor, 0.1µF, 50V, ±10%, X7R	Murata Electronics North America	490-3811-ND	Digikey	1	$0.64
30		FK26Y5V0J476Z	Capacitor, 47µF, 6.3V, +22%/-33%, Y5V	TDK Corporation	445-2883-ND	Digikey	1	$0.64
31		K330J15C0GF5TH5	Capacitor, 33pF, 50V, ±5%, C0G	Vishay/BC Components	BC1036CT-ND	Digikey	2	$0.16
32	X1	FOXSLF/040	Oscillator, 4.0 MHZ	Fox Electronics	631-1097-ND	Digikey	1	$0.53
33		40-C182-10	40 PIN LOCK/EJECT IC SOCKET	Aries Electronics	A540-ND	Digikey	1	$8.83
34		FJ-20-D-03.00-4	FFC cable connector	Samtec		Samtec samples	1	$0.00
35		ZF1-20-01-T-WT	Header	Samtec		Samtec samples	1	$0.00
36		ECJ-HVB1C106M	Input Capacitor, 10uF 16V X5R	Panasonic ECG	PCC2417CT-ND	Digikey	3	$2.19
37		T495D227K010ATE125	Output Capacitor, 220uF 10V	Kemet	399-3881-1-ND	Digikey	1	$1.90
38	L1	7447709330	Inductor, Power 33uH 4.2A	Wurth Electronics Inc	732-1244-1-ND	Digikey	1	$4.03
39		B340LA-13-F	Diode, Schottky 40V, 3A SMA	Diodes, Inc.	B340LA-FDICT-ND	Digikey	1	$1.12
40		573300D00010G	Heat Sink, TO-263	Aavid Thermalloy	HS338CT-ND	Digikey	1	$0.91
41		5499910-8	Header, 34-pin .100 Latched	Tyco Electronics / Amp	AHE34H-ND	Digikey	1	$4.94
42	R8	3314J-1-203E	Trimpot 10-20k-Ohm	Bourns, Inc.	3314J-203ECT-ND	Digikey	1	$1.56
43		3478	Spacer, threaded 1/4in Length	Keystone Electronics	3478K-ND	Digikey	3	$0.89
44		90272A105	Screws 4-40, 3/16"	McMaster		McMaster	3	$0.05
45		98032A421	Washer .125ID, .25OD	McMaster		McMaster	3	$0.05
46	S1-2	B3S-1000P	Button, Momentary NO	Omron Electronics	SW836CT-ND	Digikey	2	$1.32
47		MCR18EZPF4700	Resistor, 470-Ohm 1206 1%	Rohm	RHM470FRCT-ND	Digikey	10	$0.50
48		MCR18EZHF1001	Resistor 1.0-KOhm 1206 1%	Rohm	RHM1.00KFCT-ND	Digikey	10	$0.50
49		C0603C104K8RACTU	.1uF bypass Capacitor, 0603	Kemet	399-1095-1-ND	Digikey	4	$0.12
50		F931A106MAA	10uF Tantalum Capacitor, 1206	Nichicon	493-2351-1-ND	Digikey	5	$0.70
						Total Cost		**$265.65**

Appendix C - Electrical Schematic

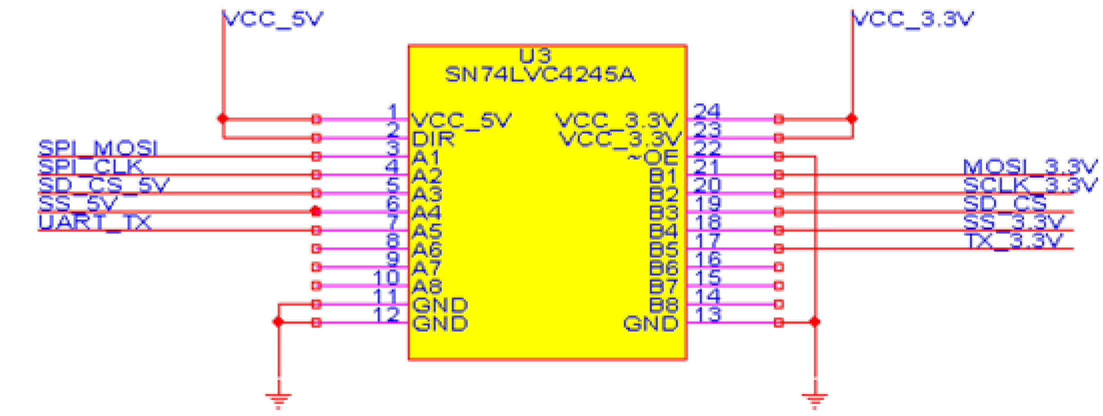


Connectors

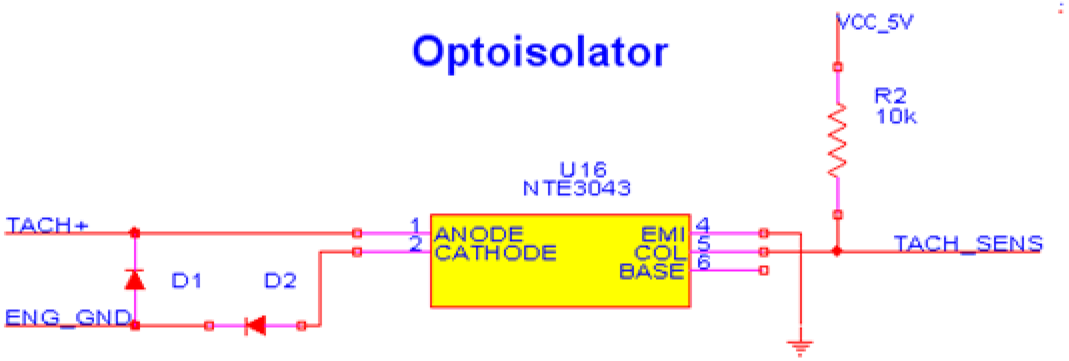


Voltage Regulators

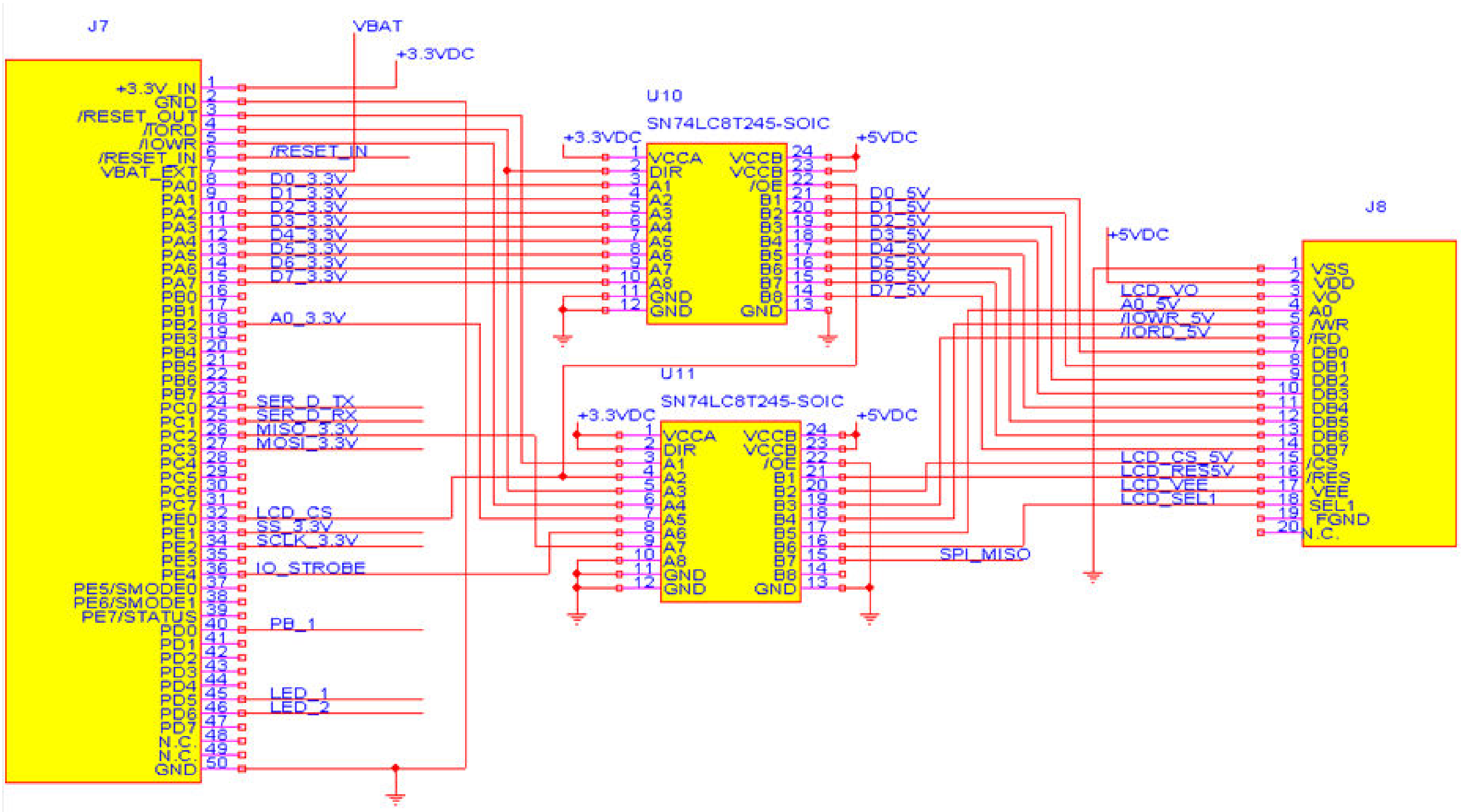
Voltage Translator for SD Card Circuit



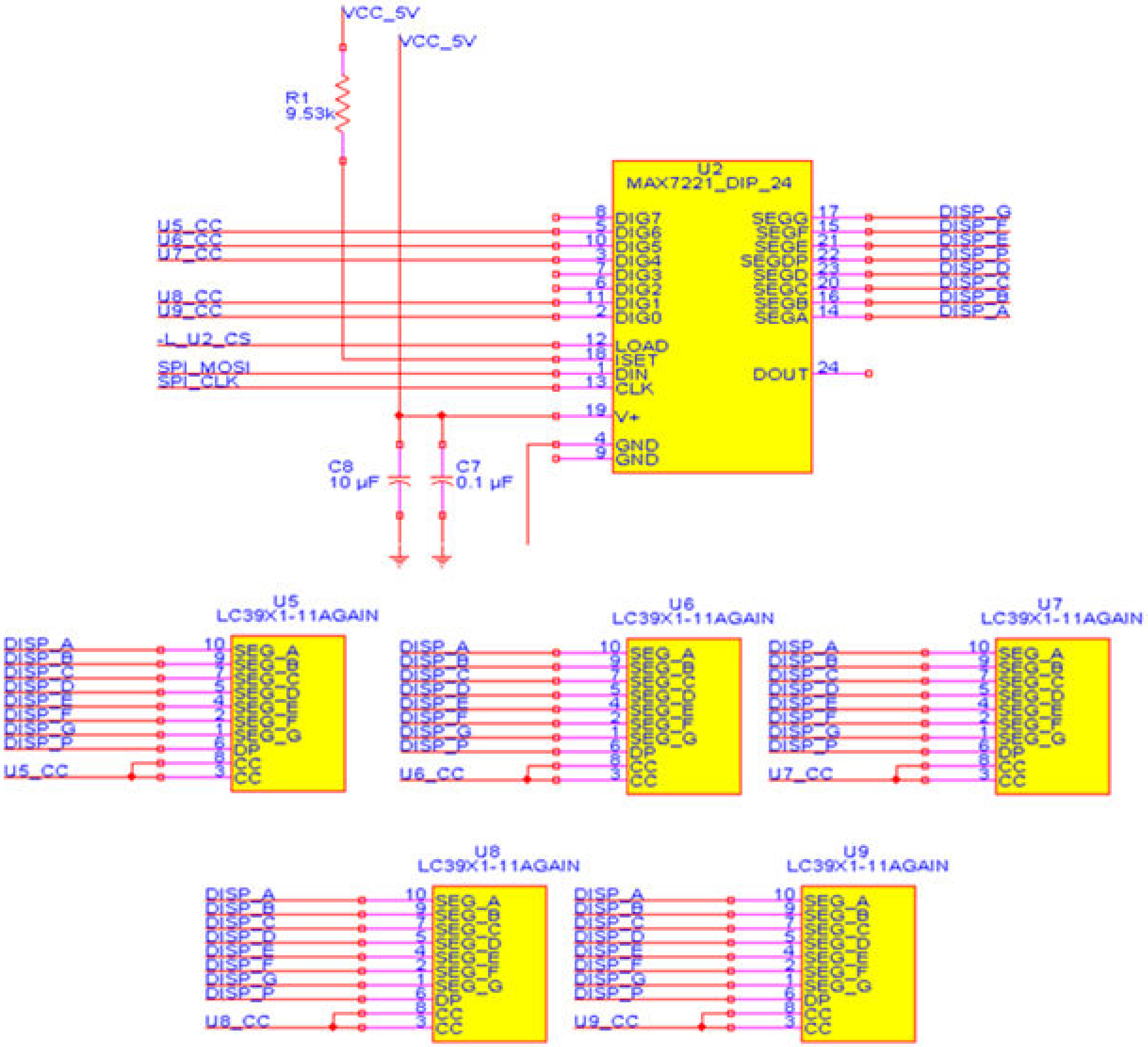
Optoisolator



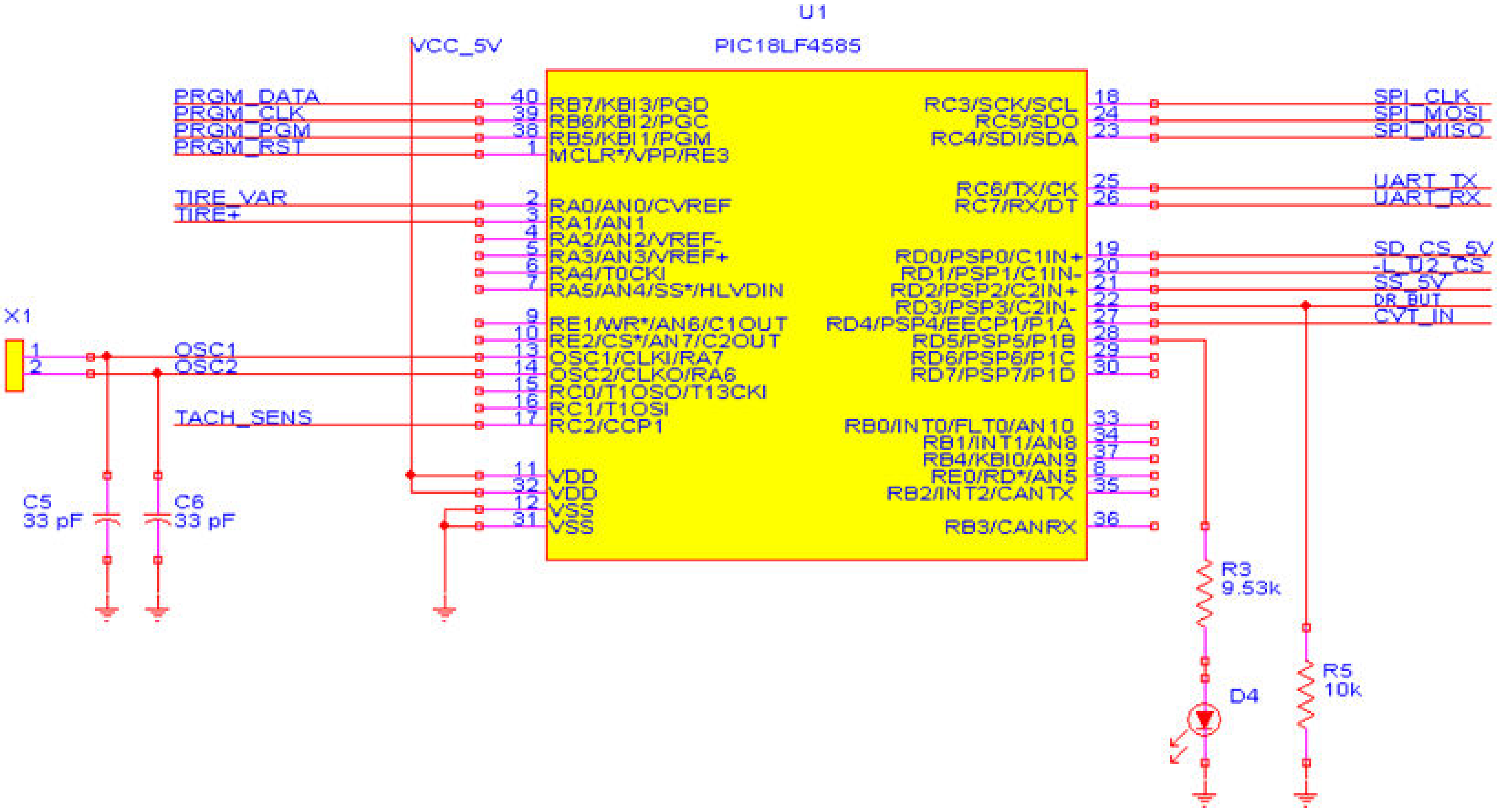
Voltage Translator & Opto-isolator



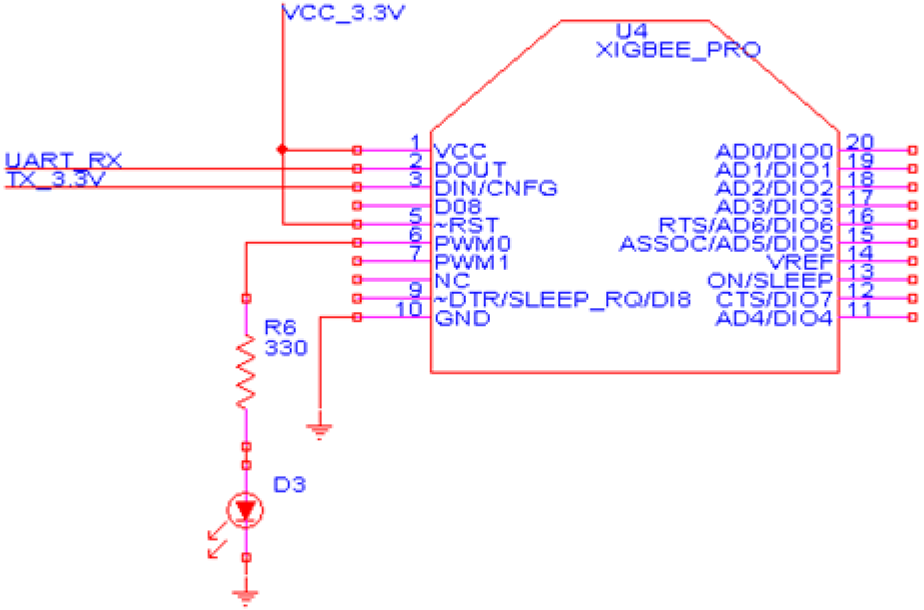
Display Controller Circuitry



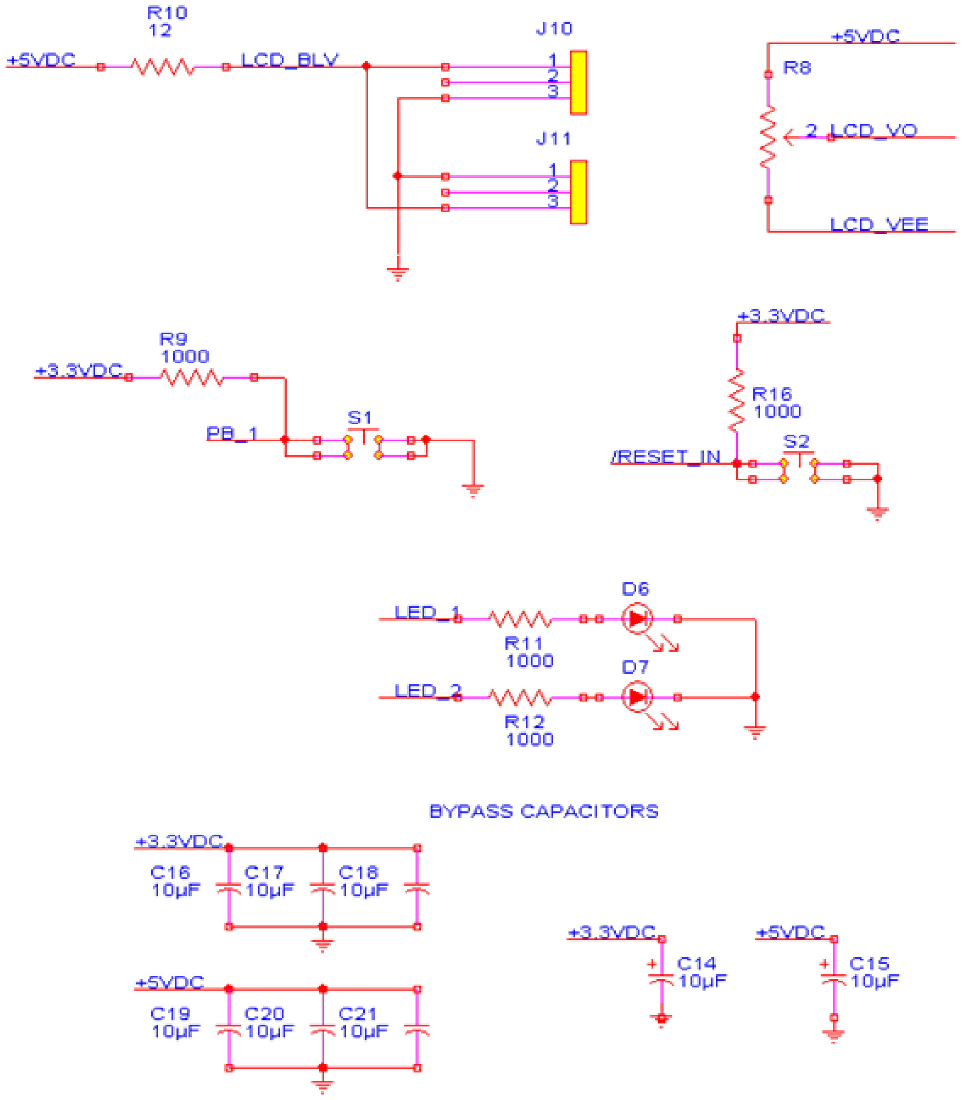
General Display IC’s



DAQ Microcontroller

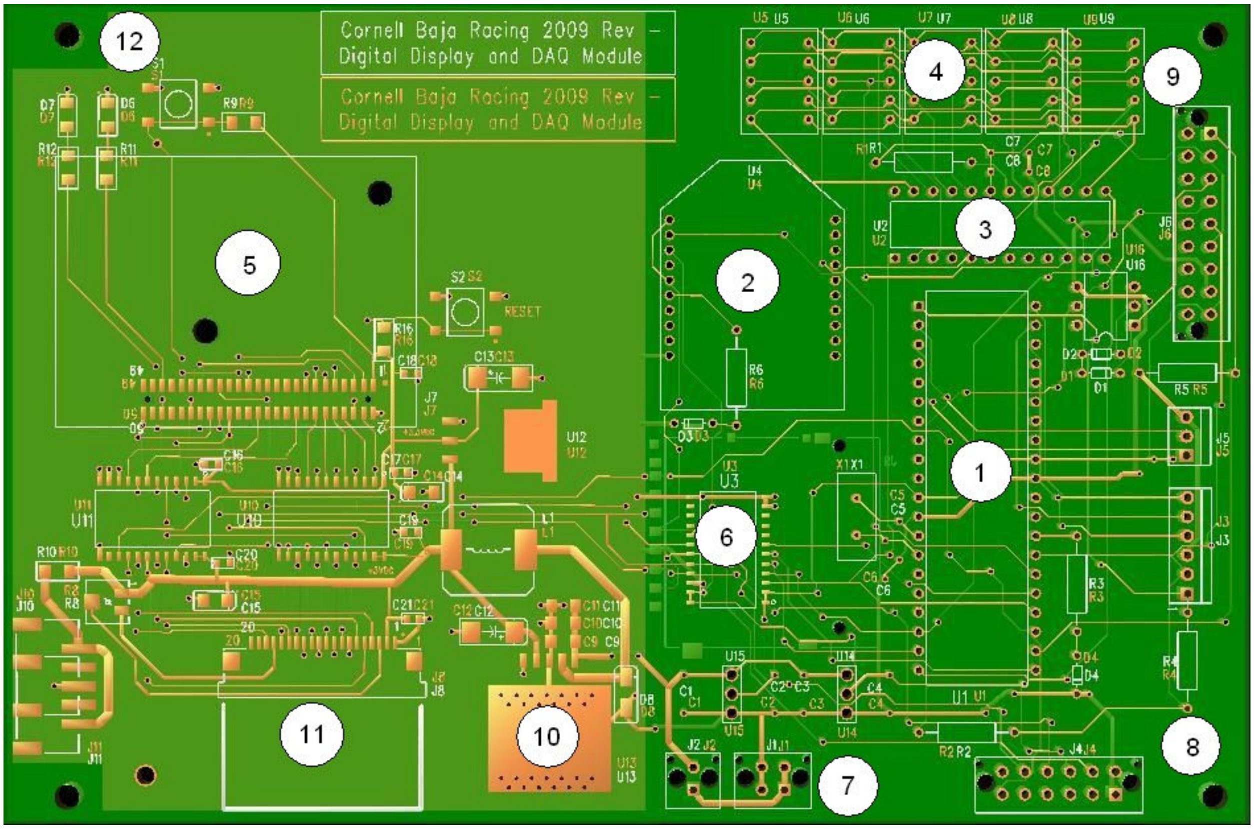


Zigbee Transceiver



Misc Components

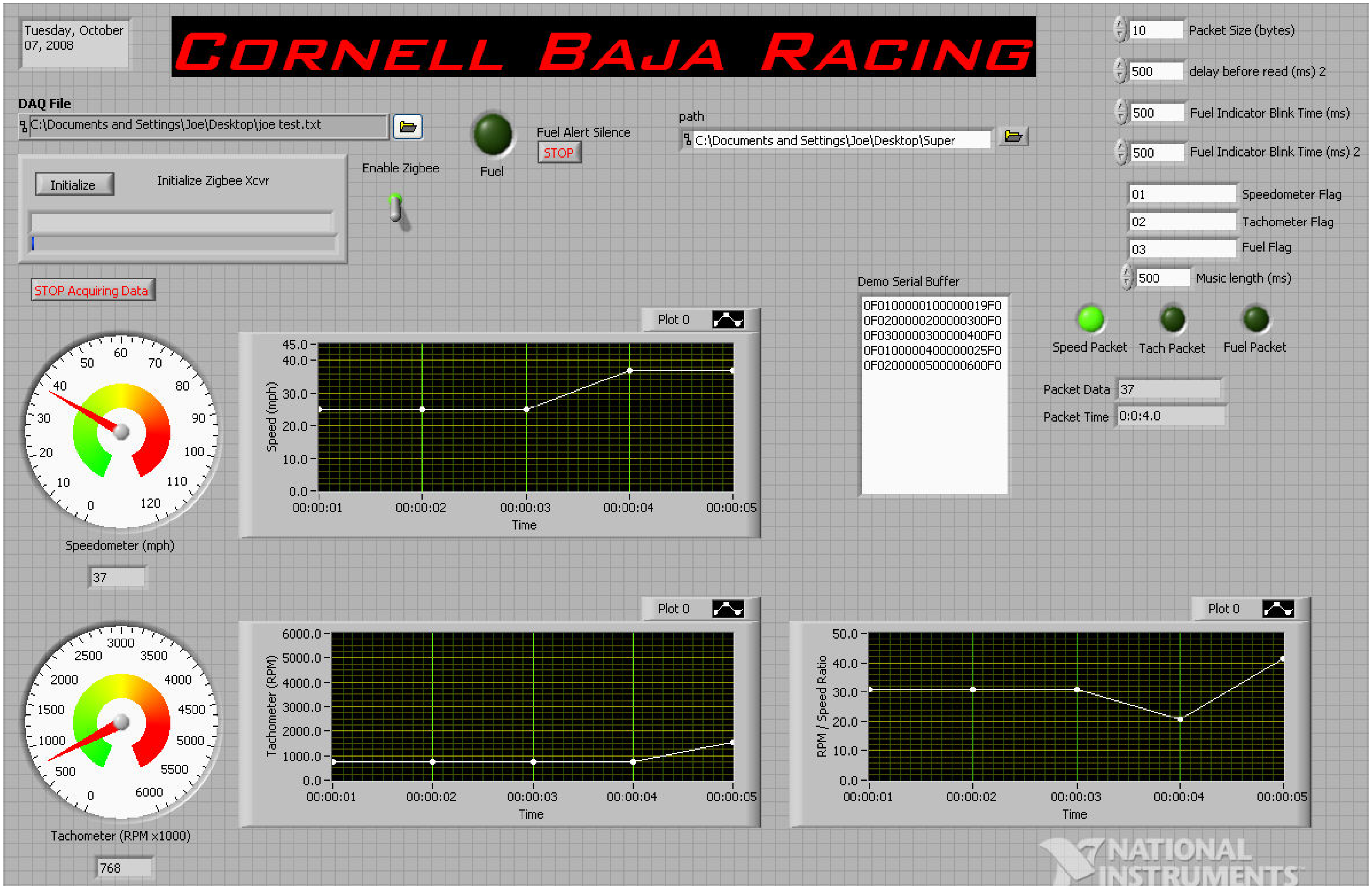
Appendix D - DAQ Module Physical Layout



References

1. DAQ Microprocessor
2. Zigbee Transceiver
3. LED Driver
4. General LED Display
5. LCD Display Microprocessor
6. Voltage Translator
7. Battery and Ignition Switch Connectors
8. SD Memory Card Connector
9. Sensor I/O Connector
10. Voltage Regulator
11. LCD Cable Connector
12. Mechanical Standoff Locations (x4)

Appendix E - Labview Base Station Console



Tuesday, October 07, 2008

CORNELL BAJA RACING

DAQ File: C:\Documents and Settings\Joe\Desktop\joe test.txt

path: C:\Documents and Settings\Joe\Desktop\Super

Packet Size (bytes): 10

delay before read (ms) 2: 500

Fuel Indicator Blink Time (ms): 500

Fuel Indicator Blink Time (ms) 2: 500

Speedometer Flag: 01

Tachometer Flag: 02

Fuel Flag: 03

Music length (ms): 500

Demo Serial Buffer:

```
0F0100000100000019F0
0F0200000200000300F0
0F0300000300000400F0
0F0100000400000025F0
0F0200000500000600F0
```

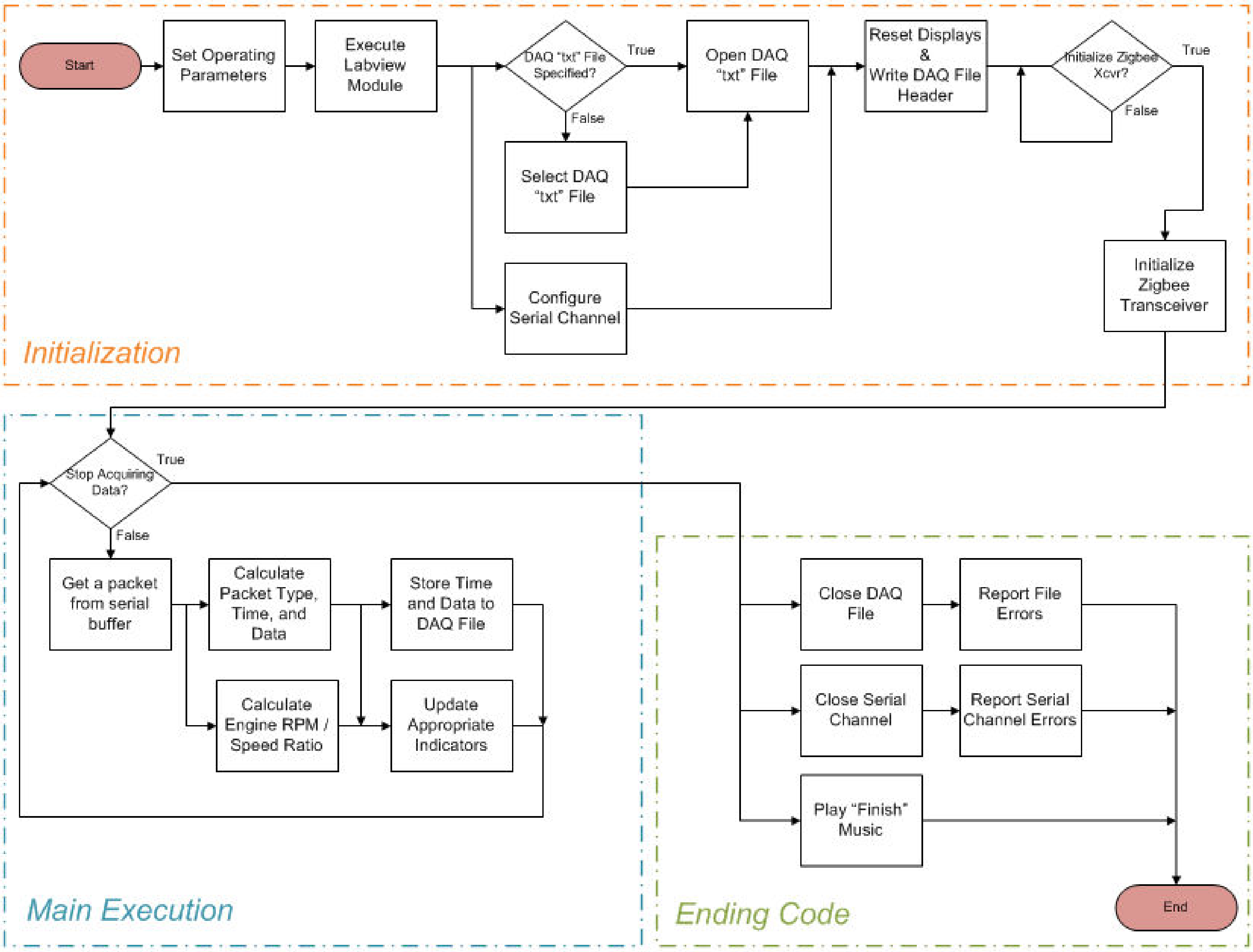
Packet Data: 37

Packet Time: 0:0:4.0

Speedometer (mph): 37

Tachometer (RPM x1000): 768

Appendix F - Labview Console Flow Diagram



Appendix G - Design Errata and Future Revision Notes

Errata

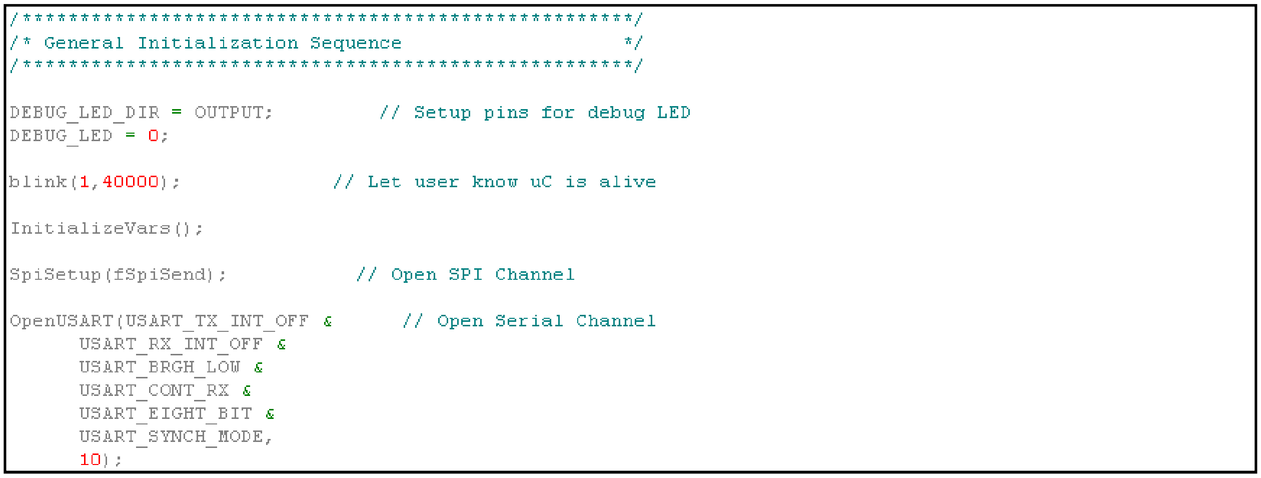
1. Analysis - The LED Driver, U2, on the schematic requires pins 4 and 9 to be tied directly to Ground. The LED Driver will not function without this change.
   Temporary Solution – Add wire (24 – 28 AWG) from U2.4 to U2.9 on B-side of the DAQ Module. Add wire (24 - 28 AWG) from U2.9 to U16.4 on B-side of the DAQ Module.
   Reglass Solution – Update schematic to tie U2.4 and U2.9 directly to Ground. Route as necessary on the board.

Future Revision Notes

1. Connector J4 and the all trace routings to the connector can be removed based on the validation of the J9 socket connector.
2. Recommend moving J9 socket connector to board edge to facilitate card installation and removal.
3. Connector J5 should be replaced with a surface mount potentiometer and the traces can be re-arranged as necessary.

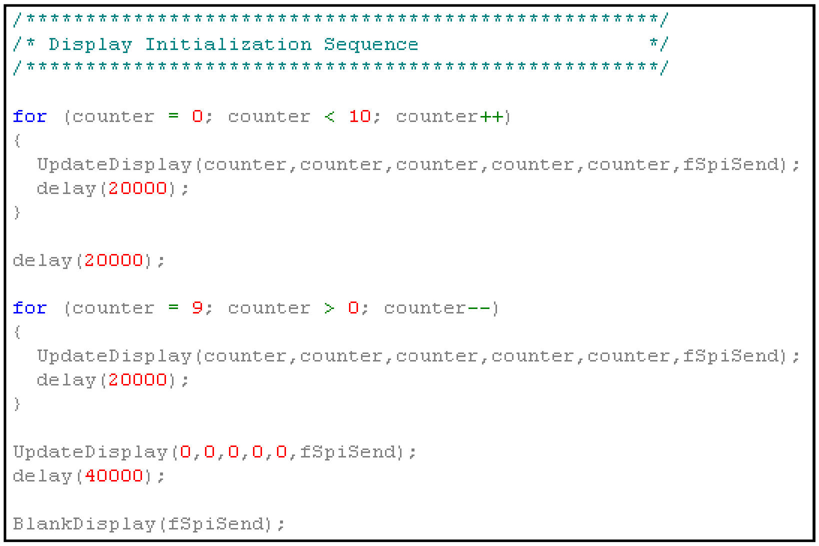
Appendix H - DAQ Module Code Excerpts

Initialization of Ports and I/O



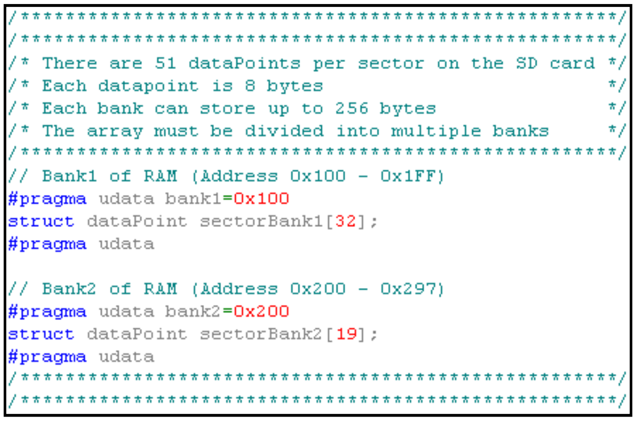
```
/*******************************************************/
/* General Initialization Sequence                     */
/*******************************************************/

DEBUG_LED_DIR = OUTPUT;         // Setup pins for debug LED
DEBUG_LED = 0;

blink(1,40000);              // Let user know uC is alive

InitializeVars();

SpiSetup(fSpiSend);            // Open SPI Channel

OpenUSART(USART_TX_INT_OFF &       // Open Serial Channel
      USART_RX_INT_OFF &
      USART_BRGH_LOW &
      USART_CONT_RX &
      USART_EIGHT_BIT &
      USART_SYNCH_MODE,
      10);
```

General Display POST



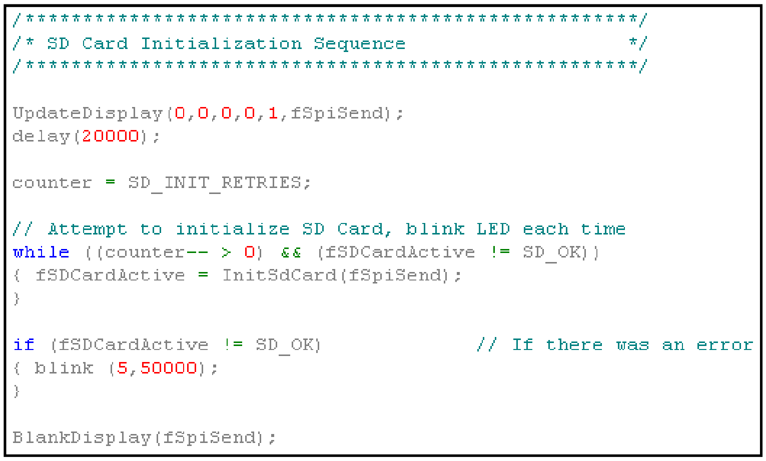
```
/*******************************************************/
/* Display Initialization Sequence                     */
/*******************************************************/

for (counter = 0; counter < 10; counter++)
{
  UpdateDisplay(counter,counter,counter,counter,counter,fSpiSend);
  delay(20000);
}

delay(20000);

for (counter = 9; counter > 0; counter--)
{
  UpdateDisplay(counter,counter,counter,counter,counter,fSpiSend);
  delay(20000);
}

UpdateDisplay(0,0,0,0,0,fSpiSend);
delay(40000);

BlankDisplay(fSpiSend);
```

PIC Memory Declaration



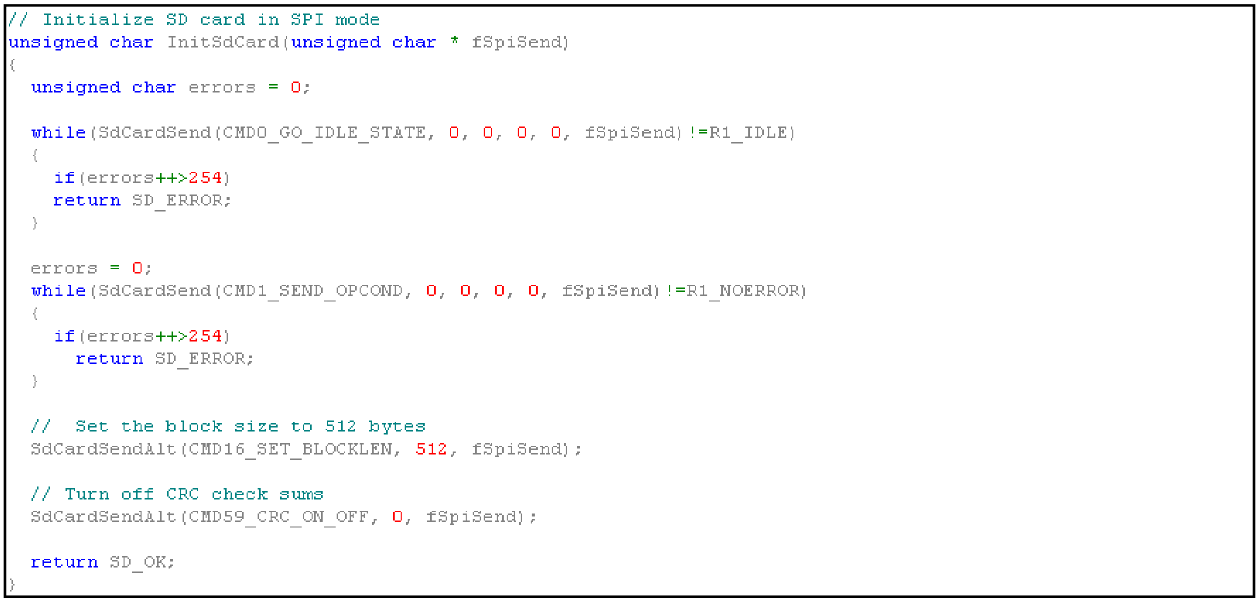
```
/*****************************************************/
/*****************************************************/
/* There are 51 dataPoints per sector on the SD card */
/* Each datapoint is 8 bytes                         */
/* Each bank can store up to 256 bytes               */
/* The array must be divided into multiple banks     */
/*****************************************************/
// Bank1 of RAM (Address 0x100 - 0x1FF)
#pragma udata bank1=0x100
struct dataPoint sectorBank1[32];
#pragma udata

// Bank2 of RAM (Address 0x200 - 0x297)
#pragma udata bank2=0x200
struct dataPoint sectorBank2[19];
#pragma udata
/*****************************************************/
/*****************************************************/
```

SD Card POST



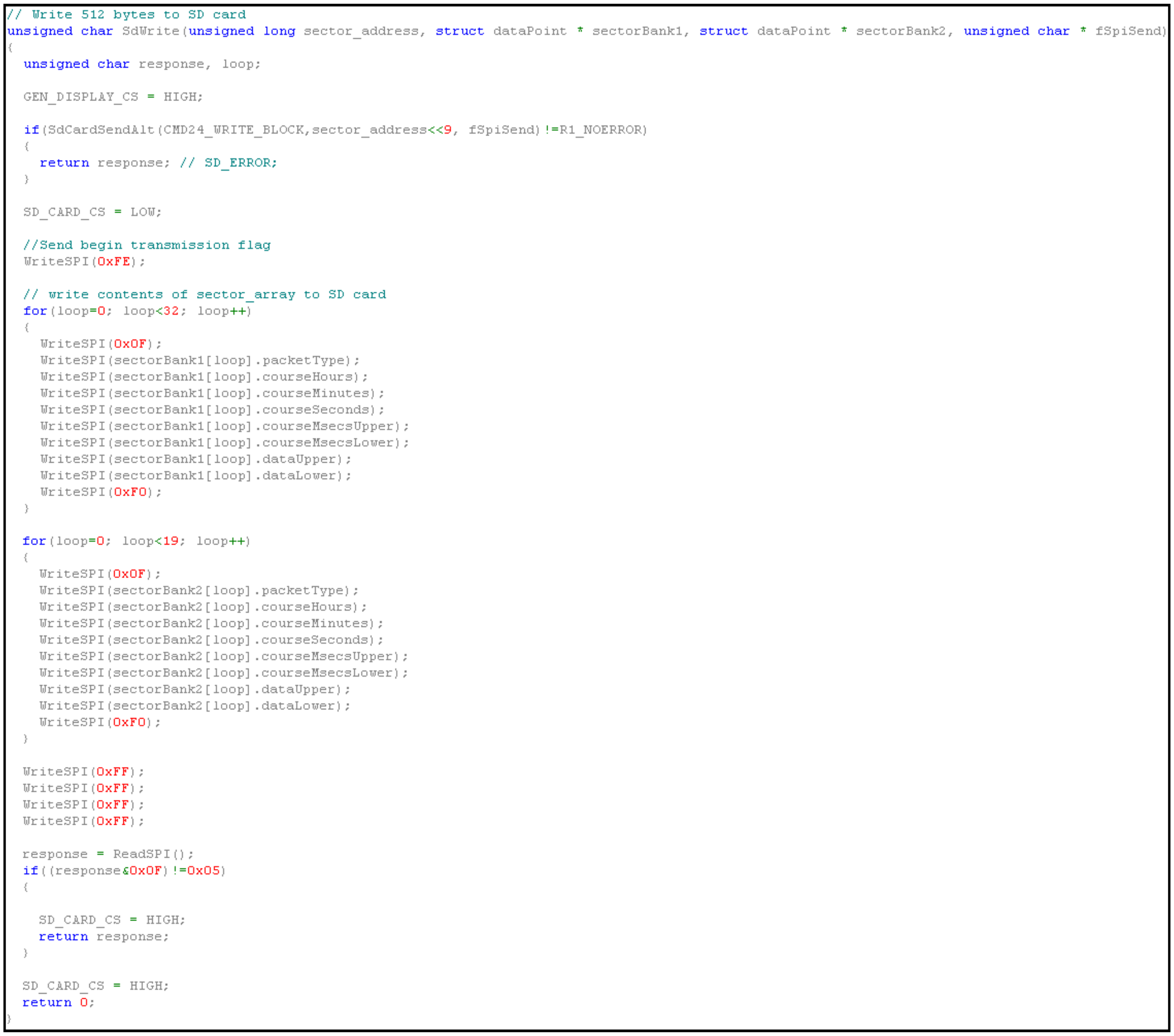
```
/*****************************************************/
/* SD Card Initialization Sequence                   */
/*****************************************************/

UpdateDisplay(0,0,0,0,1,fSpiSend);
delay(20000);

counter = SD_INIT_RETRIES;

// Attempt to initialize SD Card, blink LED each time
while ((counter-- > 0) && (fSDCardActive != SD_OK))
{ fSDCardActive = InitSdCard(fSpiSend);
}

if (fSDCardActive != SD_OK)             // If there was an error
{ blink (5,50000);
}

BlankDisplay(fSpiSend);
```

SD Card Initialization



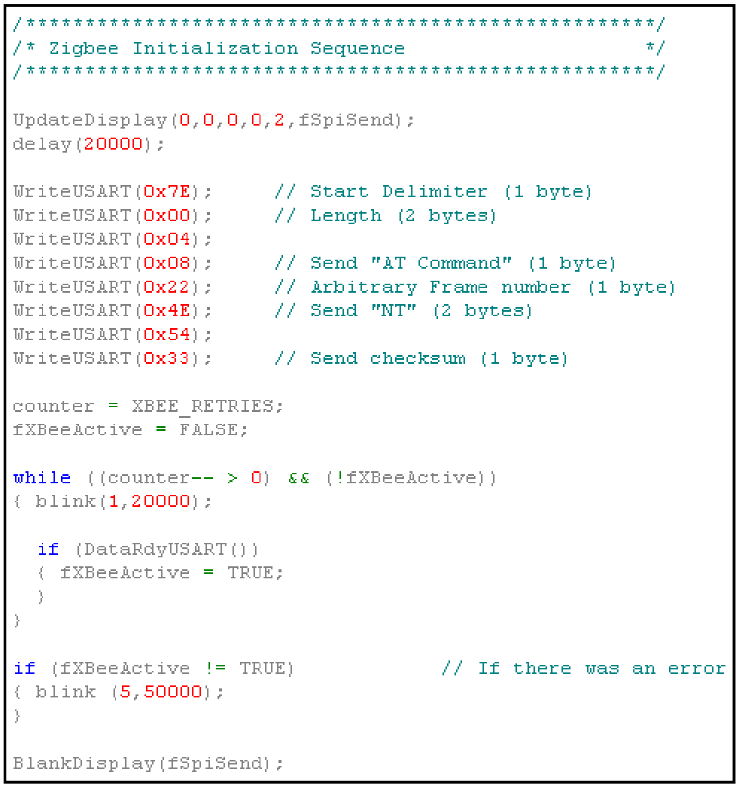
```
// Initialize SD card in SPI mode
unsigned char InitSdCard(unsigned char * fSpiSend)
{
  unsigned char errors = 0;

  while(SdCardSend(CMD0_GO_IDLE_STATE, 0, 0, 0, 0, fSpiSend)!=R1_IDLE)
  {
    if(errors++>254)
    return SD_ERROR;
  }

  errors = 0;
  while(SdCardSend(CMD1_SEND_OPCOND, 0, 0, 0, 0, fSpiSend)!=R1_NOERROR)
  {
    if(errors++>254)
      return SD_ERROR;
  }

  //  Set the block size to 512 bytes
  SdCardSendAlt(CMD16_SET_BLOCKLEN, 512, fSpiSend);

  // Turn off CRC check sums
  SdCardSendAlt(CMD59_CRC_ON_OFF, 0, fSpiSend);

  return SD_OK;
}
```

SD Card Writing



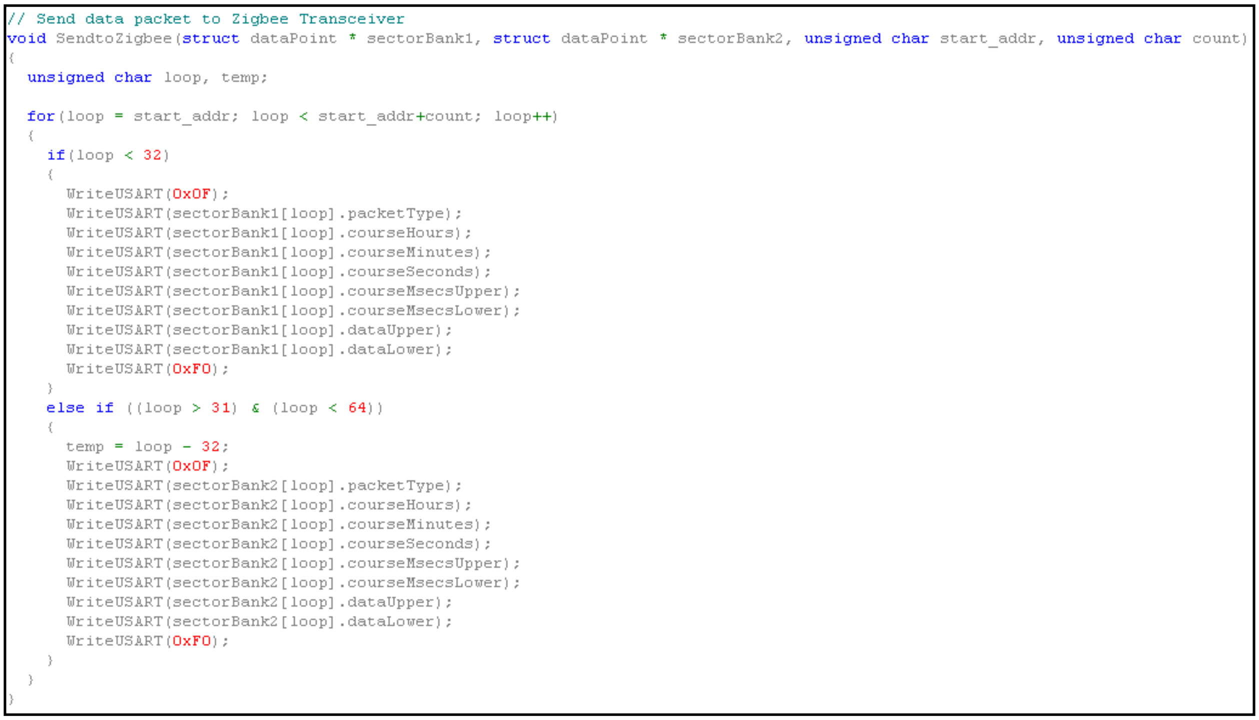
```
// Write 512 bytes to SD card
unsigned char SdWrite(unsigned long sector_address, struct dataPoint * sectorBank1, struct dataPoint * sectorBank2, unsigned char * fSpiSend)
{
  unsigned char response, loop;

  GEN_DISPLAY_CS = HIGH;

  if(SdCardSendAlt(CMD24_WRITE_BLOCK,sector_address<<9, fSpiSend)!=R1_NOERROR)
  {
    return response; // SD_ERROR;
  }

  SD_CARD_CS = LOW;

  //Send begin transmission flag
  WriteSPI(0xFE);

  // write contents of sector_array to SD card
  for(loop=0; loop<32; loop++)
  {
    WriteSPI(0x0F);
    WriteSPI(sectorBank1[loop].packetType);
    WriteSPI(sectorBank1[loop].courseHours);
    WriteSPI(sectorBank1[loop].courseMinutes);
    WriteSPI(sectorBank1[loop].courseSeconds);
    WriteSPI(sectorBank1[loop].courseMsecsUpper);
    WriteSPI(sectorBank1[loop].courseMsecsLower);
    WriteSPI(sectorBank1[loop].dataUpper);
    WriteSPI(sectorBank1[loop].dataLower);
    WriteSPI(0xF0);
  }

  for(loop=0; loop<19; loop++)
  {
    WriteSPI(0x0F);
    WriteSPI(sectorBank2[loop].packetType);
    WriteSPI(sectorBank2[loop].courseHours);
    WriteSPI(sectorBank2[loop].courseMinutes);
    WriteSPI(sectorBank2[loop].courseSeconds);
    WriteSPI(sectorBank2[loop].courseMsecsUpper);
    WriteSPI(sectorBank2[loop].courseMsecsLower);
    WriteSPI(sectorBank2[loop].dataUpper);
    WriteSPI(sectorBank2[loop].dataLower);
    WriteSPI(0xF0);
  }

  WriteSPI(0xFF);
  WriteSPI(0xFF);
  WriteSPI(0xFF);
  WriteSPI(0xFF);

  response = ReadSPI();
  if((response&0x0F)!=0x05)
  {

    SD_CARD_CS = HIGH;
    return response;
  }

  SD_CARD_CS = HIGH;
  return 0;
}
```

Zigbee POST



```
/**********************************************************/
/* Zigbee Initialization Sequence                        */
/**********************************************************/

UpdateDisplay(0,0,0,0,2,fSpiSend);
delay(20000);

WriteUSART(0x7E);      // Start Delimiter (1 byte)
WriteUSART(0x00);      // Length (2 bytes)
WriteUSART(0x04);
WriteUSART(0x08);      // Send "AT Command" (1 byte)
WriteUSART(0x22);      // Arbitrary Frame number (1 byte)
WriteUSART(0x4E);      // Send "NT" (2 bytes)
WriteUSART(0x54);
WriteUSART(0x33);      // Send checksum (1 byte)

counter = XBEE_RETRIES;
fXBeeActive = FALSE;

while ((counter-- > 0) && (!fXBeeActive))
{ blink(1,20000);

  if (DataRdyUSART())
  { fXBeeActive = TRUE;
  }
}

if (fXBeeActive != TRUE)              // If there was an error
{ blink (5,50000);
}

BlankDisplay(fSpiSend);
```

Zigbee Packet Writing



```
// Send data packet to Zigbee Transceiver
void SendtoZigbee(struct dataPoint * sectorBank1, struct dataPoint * sectorBank2, unsigned char start_addr, unsigned char count)
{
  unsigned char loop, temp;

  for(loop = start_addr; loop < start_addr+count; loop++)
  {
    if(loop < 32)
    {
      WriteUSART(0x0F);
      WriteUSART(sectorBank1[loop].packetType);
      WriteUSART(sectorBank1[loop].courseHours);
      WriteUSART(sectorBank1[loop].courseMinutes);
      WriteUSART(sectorBank1[loop].courseSeconds);
      WriteUSART(sectorBank1[loop].courseMsecsUpper);
      WriteUSART(sectorBank1[loop].courseMsecsLower);
      WriteUSART(sectorBank1[loop].dataUpper);
      WriteUSART(sectorBank1[loop].dataLower);
      WriteUSART(0xF0);
    }
    else if ((loop > 31) & (loop < 64))
    {
      temp = loop - 32;
      WriteUSART(0x0F);
      WriteUSART(sectorBank2[loop].packetType);
      WriteUSART(sectorBank2[loop].courseHours);
      WriteUSART(sectorBank2[loop].courseMinutes);
      WriteUSART(sectorBank2[loop].courseSeconds);
      WriteUSART(sectorBank2[loop].courseMsecsUpper);
      WriteUSART(sectorBank2[loop].courseMsecsLower);
      WriteUSART(sectorBank2[loop].dataUpper);
      WriteUSART(sectorBank2[loop].dataLower);
      WriteUSART(0xF0);
    }
  }
}
```

Appendix I - Labview Code Excerpts

Figure 13 - Serial Channel Configuration, Display Initialization, and DAQ Data File Header Writing

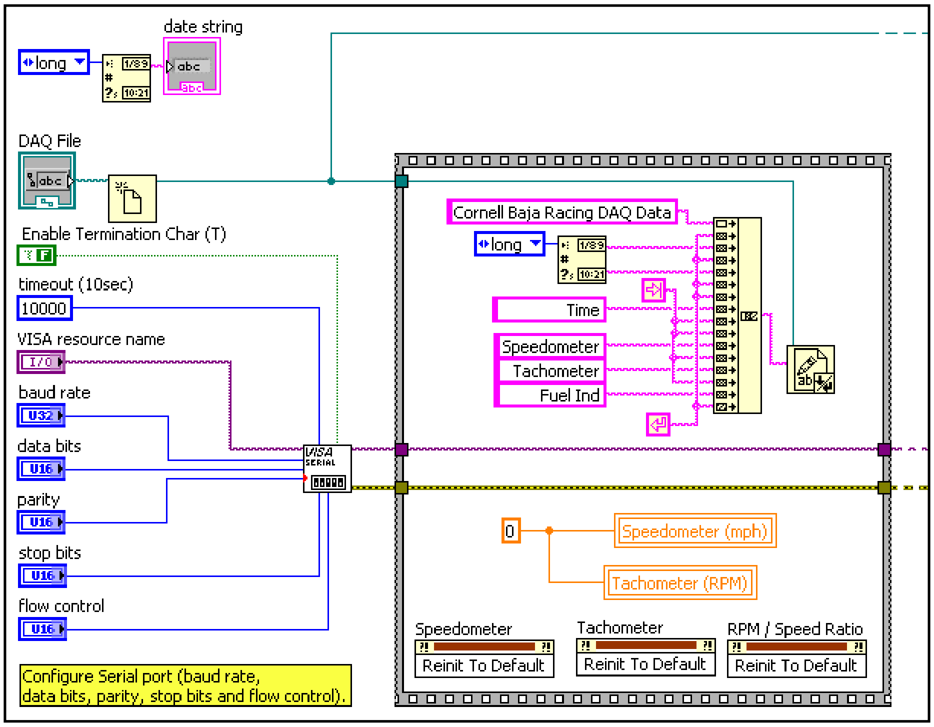


Figure 14 - Zigbee Initialization Code

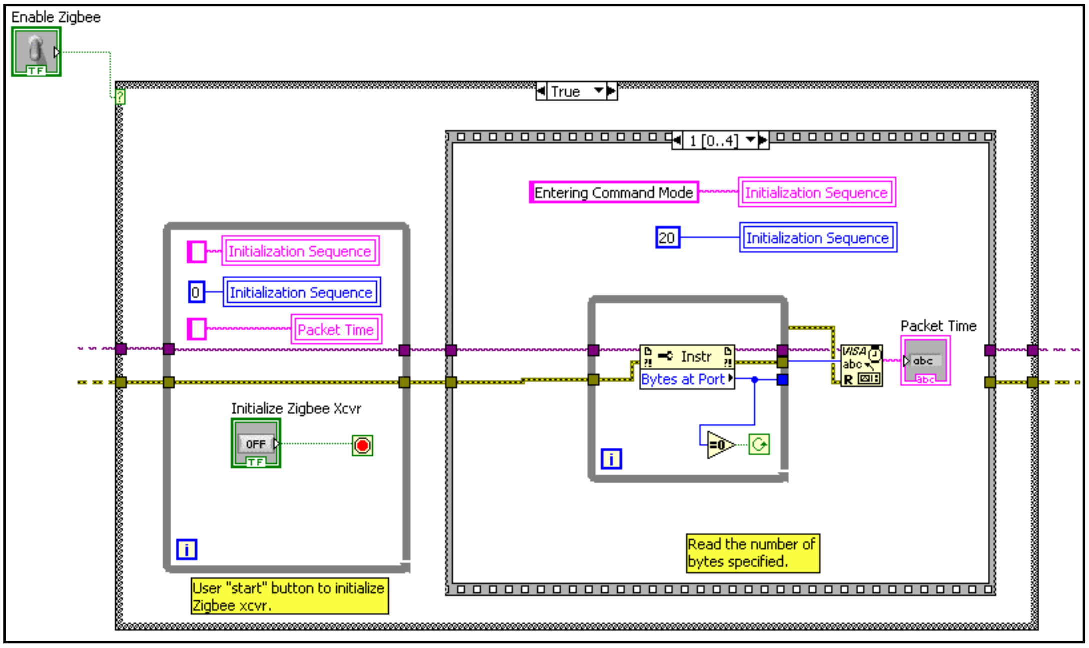


Figure 15 - Main Code

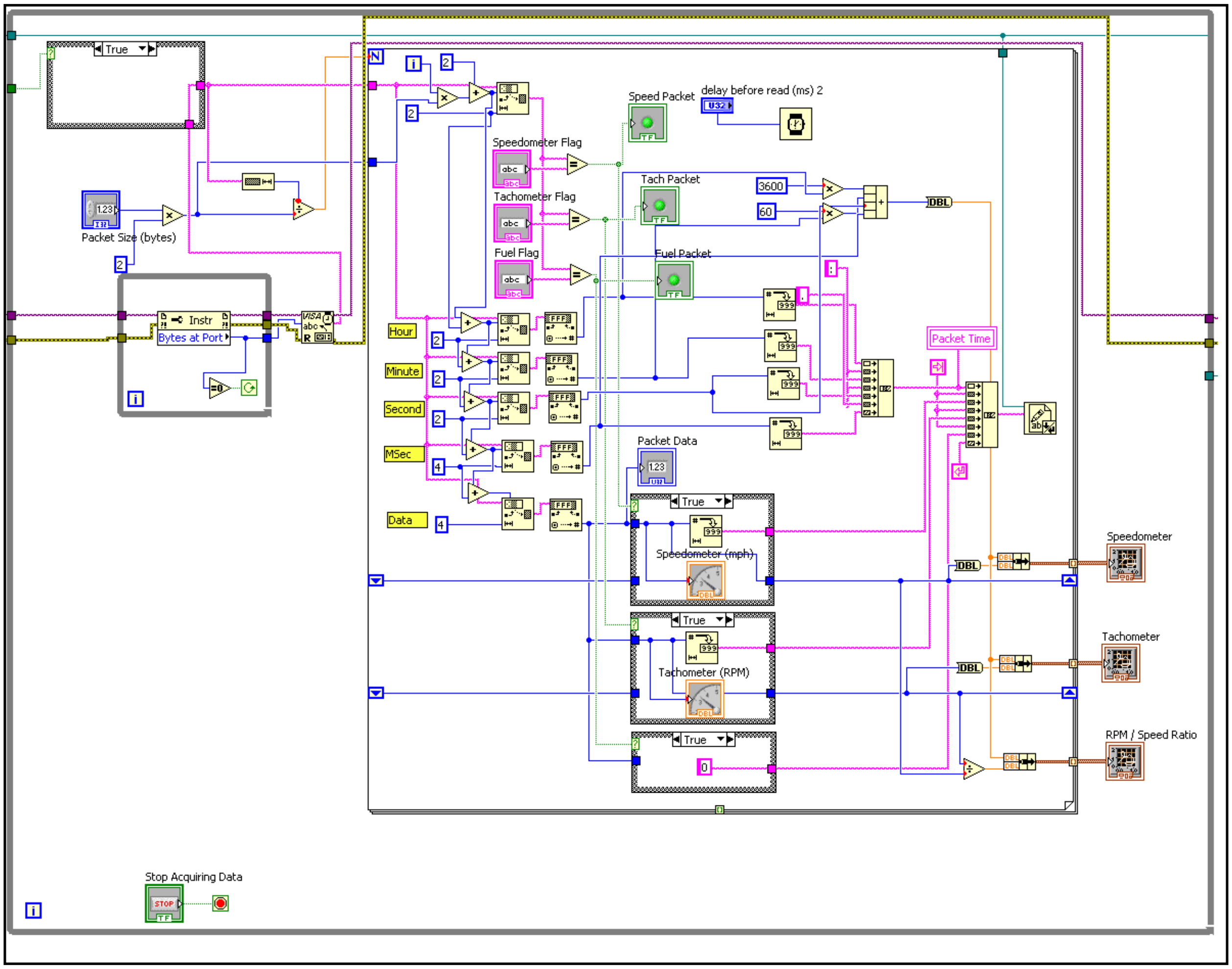
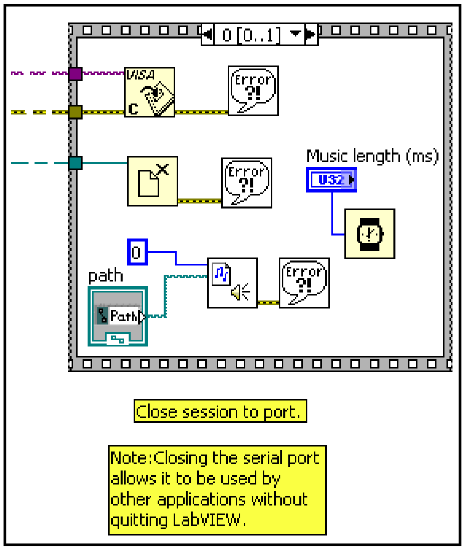


Figure 16 - Ending Code



Close session to port.

Note:Closing the serial port allows it to be used by other applications without quitting LabVIEW.