TOUCH SCREEN USER INTERFACE FOR THE CORNELL AUTOMOTIVE X-PRIZE COMPETITION CAR

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: Tim K. Sams

Project Advisor: Bruce Land

Degree Date: May 2010

Abstract

Master of Electrical Engineering Program
Cornell University
Design Project Report

Project Title: Touch Screen User Interface for the Cornell Automotive X-Prize Competition Car

Author: Tim K. Sams

Abstract:

The project is to develop, in LabVIEW, the user interface and communication system for the touch screen dashboard to be used in the Cornell Automotive X-Prize competition car. In order to complete this project, I am developing a detailed and exhaustive list of parameters and operating modes to be implemented, installing and ensuring my system compatibility to develop the software, and checking the interface compatibility with the current hardware in place in the vehicle. In the end, the touch screen will be able to report any and all needed data on a real time basis to the driver and passenger of the Cornell AXP car. They will be able to change the operating mode of the vehicle and monitor the current status of all the available metrics to ensure efficient operation of the vehicle. This is a critical component to the success of the Cornell Automotive X-Prize team when it comes to race time, because fuel efficiency is paramount, and not being able to monitor the real time status of the vehicle could have a devastating impact on the vehicles operation in the competition.

Report Approved by
Project Advisor: ______________________________________Date: _______________

Contents

Executive Summary

As a member of the Cornell Automotive X-Prize competition team, I was tasked with the creation of a user-friendly touch panel interface for use as the cars dashboard. For this competition it is imperative that the car operate as efficiently as possible, and the touch panel feedback serves as the eco-feedback to the driver and passenger of the car. Simple parameters that any normal car might have must also be portrayed on this system as well as the data that makes operation as a hybrid possible. It is also necessary to provide quick access to many of the details to facilitate quick debugging if something in the car goes wrong.

As a means of testing, there was a preliminary test-bed car built that used the exact same hardware used here. While some of the programming methods were retained, much of the software was written from scratch to facilitate faster data update rates. The hardware is all National Instruments LabVIEW based programming for the touch panel and FPGA. The FPGA runs the car's systems whereas the touch panel serves more as a monitor. This eliminates the need for the driver and passenger to keep a laptop in the car to check the status.

The software, completely written using LabVIEW, revolves around two main components. The first is the loop on the FPGA that serves as the cars onboard computer which updates all of the necessary data variables and calculates needed values. The second is the loop and display setup on the touch panel which decodes the sent variables and transfers them into a viewable format on the front panel of the screen.

In the final tests of the system, everything performed as desired. We were able to achieve a screen refresh rate of 20 Hz or every 50 ms. This is crucial to the operation of the car because it is the replacement to the cars original speedometer. When used at the shakedown stage of the competition there were no hardware or software issues other than the fact that glare seemed to be somewhat of an issue on the screen itself. One of the largest benefits of this system is the fact that it is very simple to modify on the fly and during the competition, team members were able to quickly update the way the speed was displayed in order to avoid the glare issue.

Design Requirements

The objective of this project was to create a user-friendly and simple to operate user interface for use with the Cornell Automotive X-Prize (AXP) team competition car. The hardware interface to be used was a touch screen interface, implemented on a National Instruments embedded touch panel device. Initially, the program was meant to run on a small 5.6" touch panel device, but through considerable testing and implementations, we found that a much larger device with more processing power was necessary and the requirements changed in the second semester. This made it possible to utilize a 12" diagonal version of the touch panel embedded computer which enabled several new features in the design space as the new touch panel ran the embedded version of Windows XP. The following is a summary of the preliminary goals for this design project:

- Ability to provide real-time vehicle data such as: speed, battery level, miles per gallon, distance traveled, fuel level, and other driving habits as requested.
- Ability to switch between different operating modes, as specified by the AXP Team.
- Smooth, consistent, and simple operation. Should not require extensive practice or experience to operate.
- Extremely reliable, should operate under the normal circumstances experienced within a commercial vehicle.
- Must interface will all existing hardware within the vehicle, if the hardware is not already provided, will research and implement new hardware for our needs.
- The physical unit will conform to the interior design of the vehicle.

It is important to note that throughout the year several of the requirements and environment were changed, allowing for more efficient use of resources, such as:

- Transition from a 6'' form factor screen to a 12'' form factor.
- Removal of Driving mode feature – vehicle did not have different operating modes at the time of implementation

Previous Work

One of the unique aspects of this project is that it was not the first time the AXP team had built a working version of the hybrid-diesel car. In the previous two years, the team developed a working test bed version in a modified Geo Metro sedan. In that car, the first touch panel device was utilized, the simple 5.6” model. Several of the concepts and operations of the original screen were taken from a tech report produced for the AXP team by John Penning [1]. This report provided many of the methods necessary to program and implement custom interfaces on the touch panel device and was extremely useful throughout the duration of this project.

Design Alternatives

When beginning this project, one other possible hardware solution was considered. Knowing the interface was meant to be a simple touch screen, there was always the option of using a microcontroller device (such as an Atmega 644 or a similar device) with a compatible touch panel. This, however, would have required completely custom software and would have been difficult to interface with the Ethernet interface in the car. The brains of the car were already running on a LabVIEW based FPGA device that connected to external hardware through an Ethernet local area network connection. So for overall simplicity and compatibility, the National Instruments LabVIEW based touch panels were the ideal solution. Another factor was that the team was already sponsored by National Instruments, which made requisition of their hardware easy and painless, as well as the support. We had a direct support line with National Instruments engineers anytime we had an issue and it turned out to be a huge benefit.

When we began the project we thought that the smaller 6’’ touch panel device would adequately fit our needs for this system, as it provided a small form factor and enough processing power to compute and display networked data. As it turned out, this was not the case. We found that through implementation and testing the screen proved to be very small and difficult to read, as well as taking a long time to initialize the data connection with the FPGA. So as an alternative we decided to upgrade our touch panel to the 12’’ version with embedded Windows XP and a faster microprocessor. This upgrade virtually eliminated all of our lag issues and provided us with much larger text and easier to read displays.

Design Details

Hardware

First, a little introduction to the hardware units. The preliminary unit, a National Instruments TPC-2106T was a 6'' diagonal touch panel computer that ran Windows CE Embedded edition. This model boasts a 416 MHz Intel XScale Processor and 64 MB onboard SDRAM [2]. For the 12'' TPC-2512, it runs Windows XP Embedded edition, has a 500 MHz AMD LX800 processor, and 512 MB DDR SDRAM onboard memory [3]. So there was a little bit of improvement in speed on the larger screen and a significant improvement in memory.

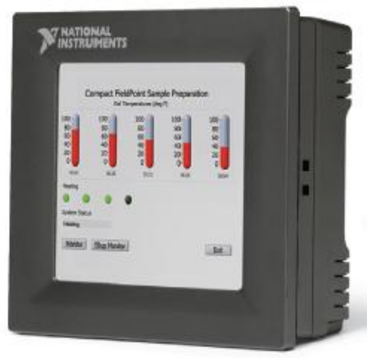
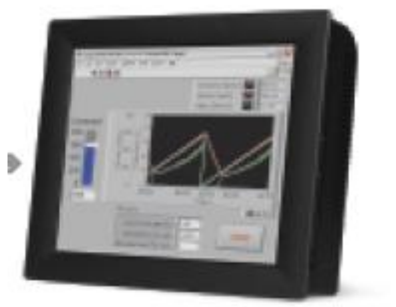



Figure 1. TPC-2106T and TPC-2512 [2,3]

Also a major component in the system was the NI cRIO-9074. This was the controller unit which monitored and controlled all of the electrical systems of the car. It also was the unit used to calculate and provide the data and configuration data over the network to the touch panel. It contains a 400 MHz processor with 2 million available logic gates within the FPGA [4]. The cRIO is also configured using LabVIEW but there is not a viewable front panel for it unless you connect a touch panel or monitor device. You can also monitor the compiled code via the computer interface for debugging, but for independent operation a monitor or touch panel is desired.

For communication between the two devices a hardware Ethernet connection was used. Since there was no need for any other devices on the Ethernet network within the car, the cRIO and touch panel are directly connected via crossover cable. In order to ensure connection between the two, they were placed within the same network via IP address. The IP addresses were pre-assigned in the hardware and were never changed.

Both the cRIO and touch panels were programmable by a very simple network interface. Once the host computer was connected to the same network as the device we just needed to run the program on the host and it compiled it and sent it to the device. This made debugging very simple and made some other programming interfaces look much more complicated. Changes could be implemented almost instantly and the results viewed on the touch panel just as fast.

One of the major hurdles as far as the hardware was concerned was the communication rate between the two devices. At a given time we were trying to send and receive about 20 different variables of different lengths across the network connection. On the original screen we found that there was somewhat of a lag period when we first started to run the programs. It took about 10-15 seconds to initialize all of the communications and start updating the data on the screen. When we switched to the faster and larger touch panel, however, this problem was eliminated and we were able to boost the update rate on the panel to 20 Hz or 20 updates per second. Another issue with the smaller screen was that the text on the panels appeared very small and very difficult to read. Of course this was easily fixed when we moved to the 12'' screen because almost everything we displayed basically doubled in size. Another issue that came up when actually using the screen was glare. When the sun was shining in the car windows there was a decent amount of glare that actually affected performance during competition because the values on the screen could not be read. One of the solutions here was to recess the screen farther into the dash to cover the screen so that the sunlight couldn't reach it.

Software

The software design is where the majority of the work in this project was done. There had been previous implementations in the test-bed car but none of them were a final product that they wanted to use for the final iteration of the car. All of the programming for the interface was done using National Instruments LabVIEW platform. LabVIEW is a data flow based programming language which is very easy to pick up and learn, and is especially useful when implementing rapid prototype systems such as this project.

First it is important to understand the overall architecture used in the software. LabVIEW provides a very robust system for Networking Data Variables. It is basically a simple drag and drop interface once you create the data type and size that you would like in a library. In order to publish the data from the cRIO controller, there must be a loop that writes to the specified network variable. This is what we will call the publisher. On the other end, at the touch panel side, we have a loop that reads the same network variable and formats it for the correct display on the screen. This loop is called the subscriber loop. A very simple example of this architecture is shown below in figure 2.

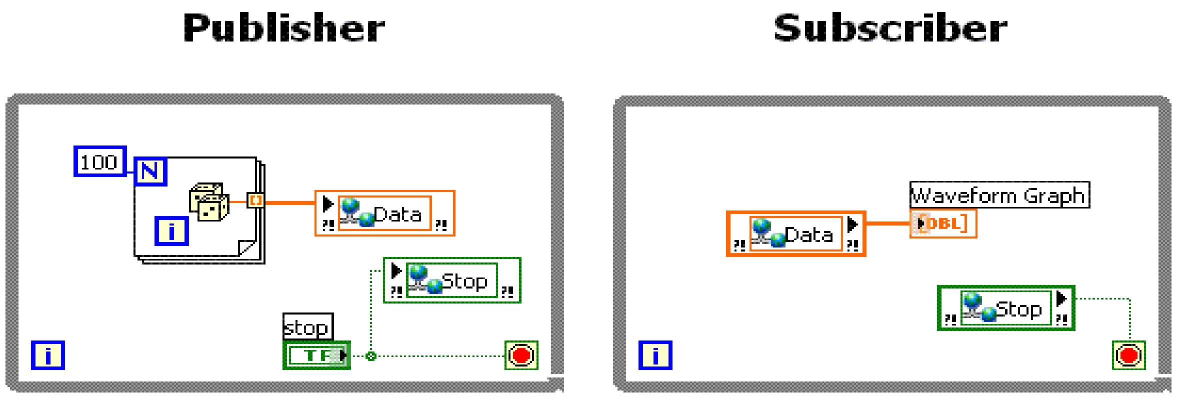


Figure 2. Publisher-Subscriber network variable example [5].

All of the calculation and computation on the necessary data for the displayed front panel was done on the cRIO. After the computation the data was passed into a communication loop that is throttled to run at the same speed as the receiving loop on the touch screen end. In the appendix project figure 1, you can see the setup as the system project with the library showing the necessary system variables.

Since LabVIEW does not provide an efficient way of commenting, I'd like to step through the pieces of the code as seen in the attached code appendix. First, in the touch panel figure 1, you can see the main data update loop for the front panel. Starting from the top left we have the network variable that updates the kW being produced from the generator on board the car. This is being passed as an array value with two entries and is converted into a 2 element structure for the dial on the front panel. Moving downward, the next element is the electrical Miles-Per-Gallon. This is similar to the kW in that it enters as an array and is converted to a 2 element structure that contains the average MPG and the instantaneous MPG. Next is the Miles-per-hour update which just reads the MPH from the network variable, and displays it as a dial on the front panel. Then we have voltage drop, trip miles, and the battery error array. The voltage drop and trip miles are just direct reads of the network variable whereas the battery error array is a collection of Boolean values indicating battery errors. Moving back towards the top we have the odometer reading which is just a straight read from the network and then the UQM direction. The UQM direction indicates whether the car is in Forward, Neutral, or Reverse. This is a 3-element Boolean array and requires a quick conversion in order to indicate the correct direction on the front panel. Next we have the max temperature and the state-of-charge. The max temperature is a direct network read but state of charge required a little bit of manipulation. We take the state of charge and check to see if it is about 40%, if it is not we turn on a small warning light on the front panel to indicate to the driver that the battery charge is low and needs to be attended to. One last element to note about this figure is the tab control. On the front panel below you can switch between different tabs that display lots of different data. The tab control is what is used to select the shown tab, and it changes according to what screen is shown. When that is changed, the selector on the case structure within the loop is changed and different operations are done. This other tab, battery errors, is detailed in touch panel figure 3 and 4 of the appendix.

The next figure in the appendix is the touch panel figure 2. In this figure the only detail is the shutdown operation outside of the loop. This takes a certain Boolean entry of the error array and uses it to trigger a touch panel shutdown by stopping the loop and moving through the sequence structure to run the shutdown program. This was necessary because the panel actually takes a very long time to shutdown, and it is essential that it does so before power is cut off. So the cRIO takes this into account when running its shutdown sequence and is able to remotely control the shutdown of the touch screen.

Finally, we have touch panel figures 3 and 4. These figures show the detailed code necessary to parse the large arrays of Boolean values that indicate the battery errors. What we have done is to divide these errors into their different packs and show the individual error as a light for each pack. The incoming network variable is formatted as a very long array of Boolean entries, but they are in order. So what is done here is the Boolean entries are addressed by the array selector methods and then individually sent to the corresponding row and columns on the front panel to display the errors. This loop only runs when the tab control has selected the battery error update screen.

The next figure is the cRIO figure 1. This figure shows the source of all of the data we use on the touch panel. In the bottom right quadrant is the loop that gathers all of the different data values from several different parts of this program using local variables. Next, the data is taken to the case statement in the middle of the screen where the data is logged into an excel file for retrieval and analysis after the car is run. All of the data is then taken into the next part of the sequence and written to all of the network variables on the far right side. This is important because it was the source of the data that is being displayed on the screen.

The final figure in the appendix is the project figure 1. This figure details the layout and structure of the project. Most notable here is the setup of the libraries for the network variables. As you can see we created a library called Dash variables and placed all of our necessary data into that library. This enable the quick drag and drop to the reads and writes for these variables. Also, at the bottom of the project window you can see how we have added the dashboard computer unit to the system. The VI entitled “12inch.vi” is the main program that runs on the touch panel and what is shown in the previous screen shots. The device is set up within the project using the IP’s that match the cRIO as to enable quick and easy communication between the two.

Once the design of the front panel interface was laid out, all that was necessary was connecting all of the correct data variables. In figure 3 you can see the final layout of the panel. At the bottom of the panel, you can see that there are several green boxes which signify Boolean values which represent the battery pack errors. These are just a quick summary of the more detailed battery system errors that can be seen by clicking the battery errors tab at the bottom control panel. These errors are then separated by the pack number and the specific errors for that pack.

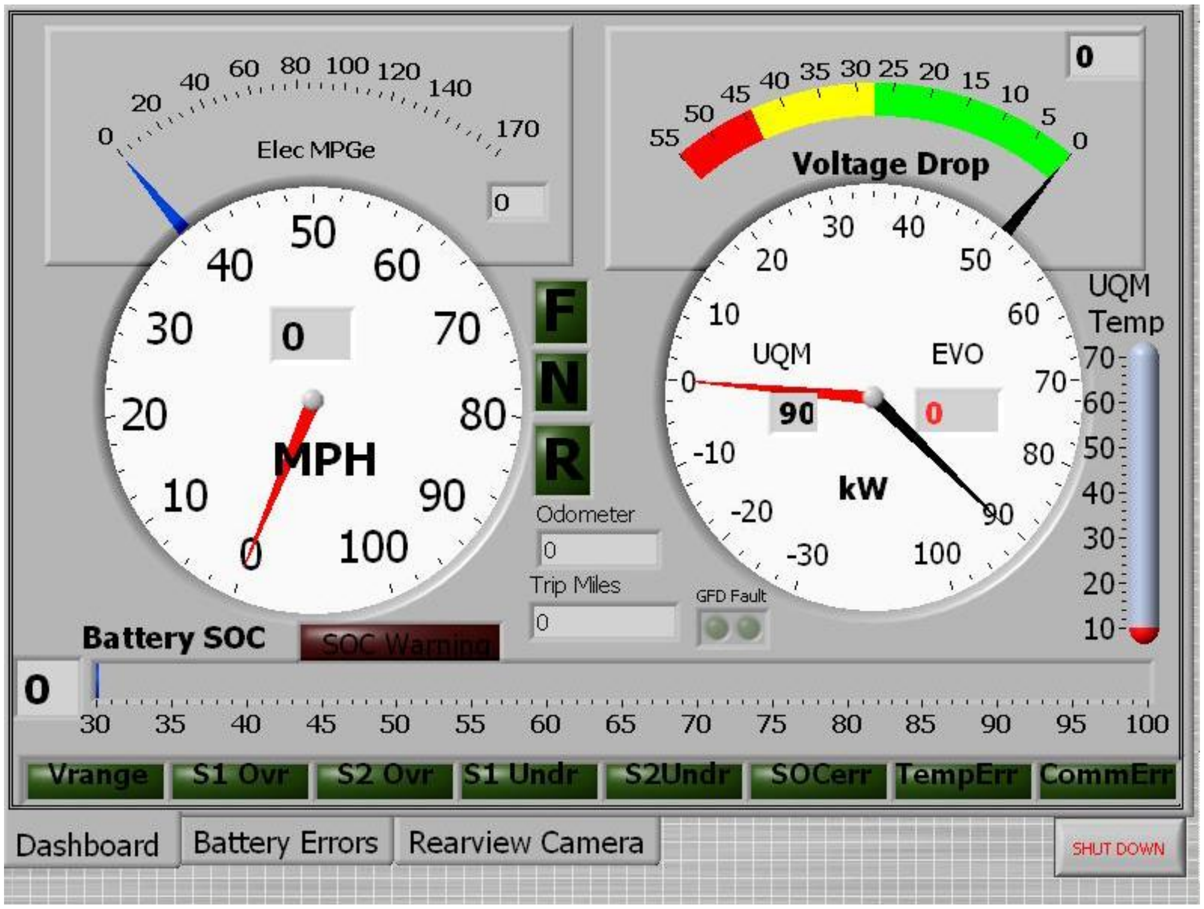


Figure 3. Final Front panel layout

As you can see, the presentation is very simple and provides quick access to many of the variables necessary for debugging problems with the car and to maintain efficient driving conditions.

Results and Conclusions

For this project, the results were determined by the correct operation of the panel. In order to test the panels operation we did a series of very simple tests to verify the data communication was as desired and up to speed. It was difficult to test in the environment of the entire car due to the fact that the car was often in pieces all over the lab and hardly all put together until a day or so before competition. So in order to test the screen we would set up the cRIO and touch panel outside of the car and assign an arbitrary value to all of the different parameters on the screen and make sure that the update efficiently. When we ran these tests there were absolutely no problems. The real test of the screen was its final implementation before the X-Prize shakedown stage at the beginning of May. This was one of the first times the car would be driven solely with the screen and the cRIO without the laptop to also monitor the status. Although I was unable to attend this stage, all reports have been good. After speaking with all the drivers and passengers of the car, there have not have been any issues with the data and update rate of the screen. That being said, there was one issue that required and on-the-fly fix at the competition. The speedometer dial was too difficult to read and resulted in inaccurate speed readings to the driver. So a third panel was added that just showed a large digital reading of the speed of the car to alleviate this issue.

Overall I was very satisfied with the operation of the screen. I do not feel that it was lacking in any way in its operation and functionality. A major factor that contributed to this was the ease of implementation due to the LabVIEW system. I believe that if the system had been a custom system the stability and operation would always be in question and would require substantially more testing for reliabilities sake.

This was just one very small part of a very large student team, but the value added from working on such an innovative project was substantial. The impact of a car that can be produced relatively inexpensively and achieve greater than 100 miles per gallon is immense. Throughout this year, I sat in on an energy seminar given here at Cornell and almost every single talk spoke to the needs of more fuel efficient technologies for our transportation in order to continue life the way we like to live it as a planet. This project may be a very small part of a very small step towards those major changes, but I'm proud to have been a part of it.

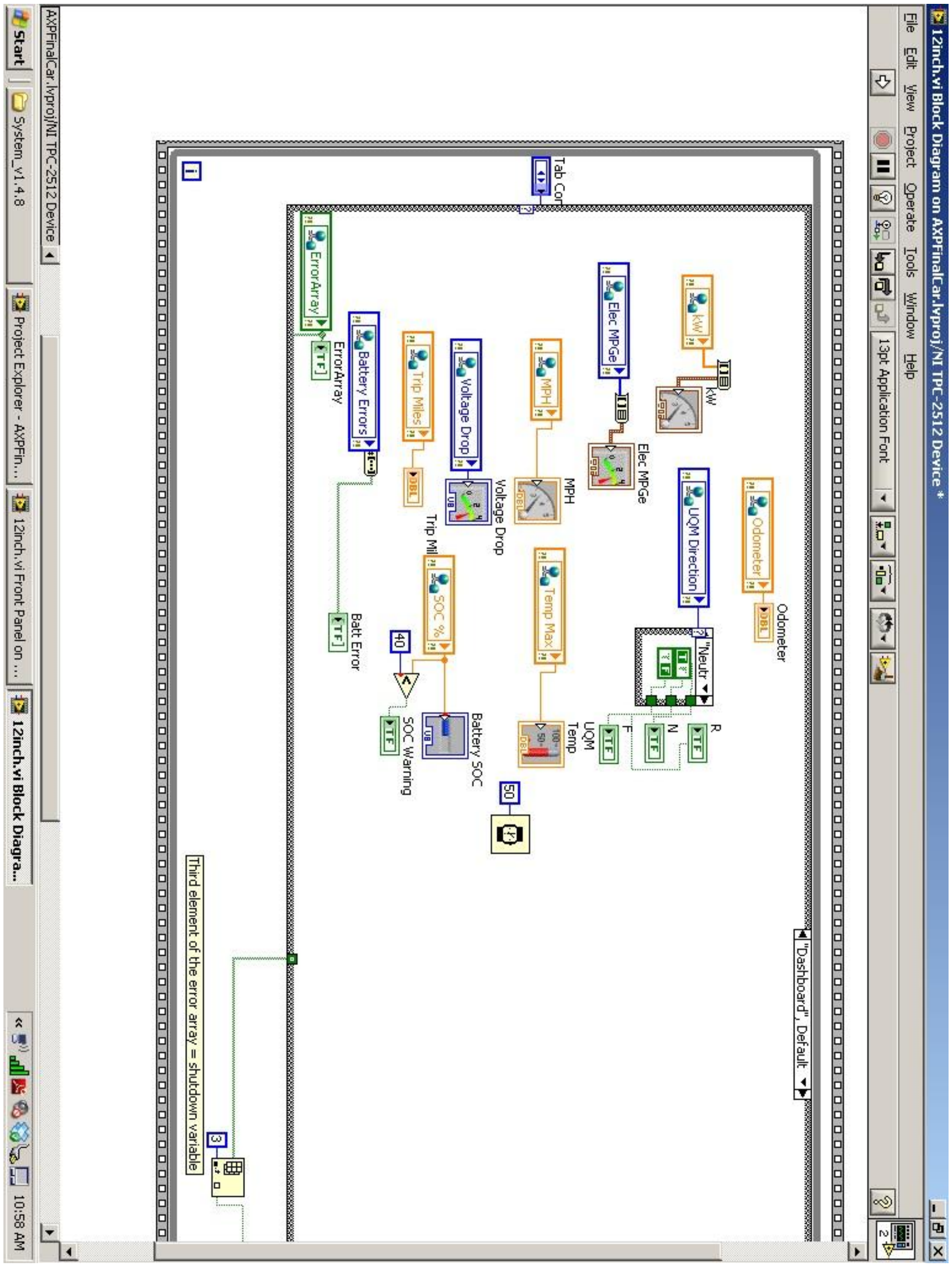
References

1. Penning, John. “Eco-Feedback Hardware Implementation using the NI CompactRIO and TPC-2106T Touch Panel.”, AXP Team, May 2008.
2. National Instruments, “NI TPC-2106 and TPC-2106T Data Sheet”, 2007.
3. National Instruments, “NI TPC-2512 Data Sheet”, 2008.
4. National Instruments, “CompactRIO Integrated Systems with Real-Time Controller and Reconfigurable Chassis NI cRIO-907x”, 2010.
5. "Adding an HMI to your Programmable Automation Controller (PAC) - Developer Zone - National Instruments " 12/7/2009 <http://zone.ni.com/devzone/cda/tut/p/id/2698>.

Appendix

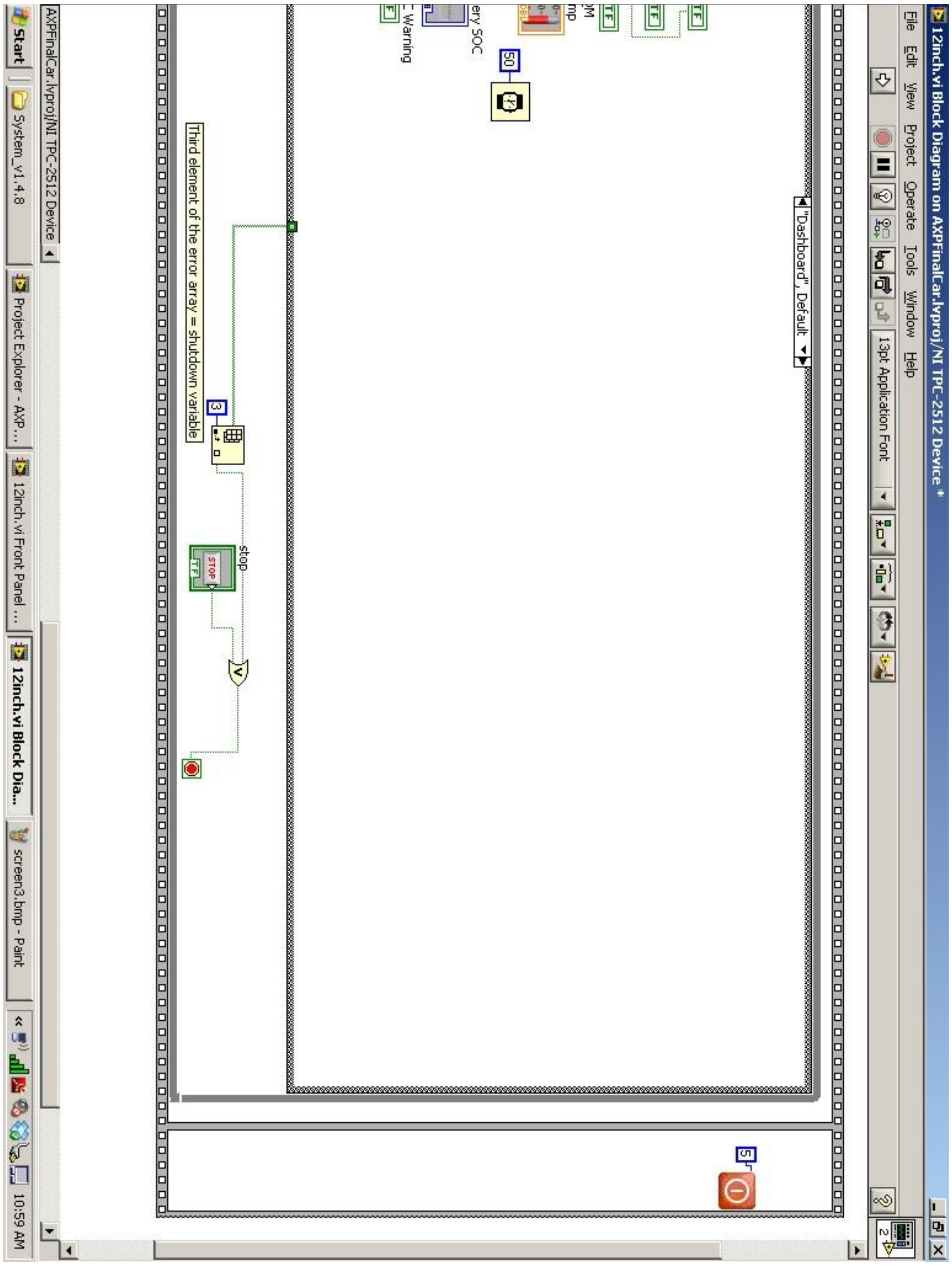
LabVIEW Code Screenshots

Touch Panel Figure 1



12inch.vi Block Diagram on AXPFinalCar.lvproj/NI TPC-2512 Device *
File Edit View Project Operate Tools Window Help
13pt Application Font
Tab Con
"Dashboard", Default
kW
Elec MPGe
MPH
Voltage Drop
Trip Miles
Trip Mil
Battery Errors
ErrorArray
Odometer
UQM Direction
"Neutr
R
N
F
Temp Max
UQM Temp
SOC %
Battery SOC
SOC Warning
40
50
Batt Error
3
Third element of the error array = shutdown variable
AXPFinalCar.lvproj/NI TPC-2512 Device
Start
System_V1.4.8
Project Explorer - AXPFin...
12inch.vi Front Panel on ...
12inch.vi Block Diagra...
10:58 AM

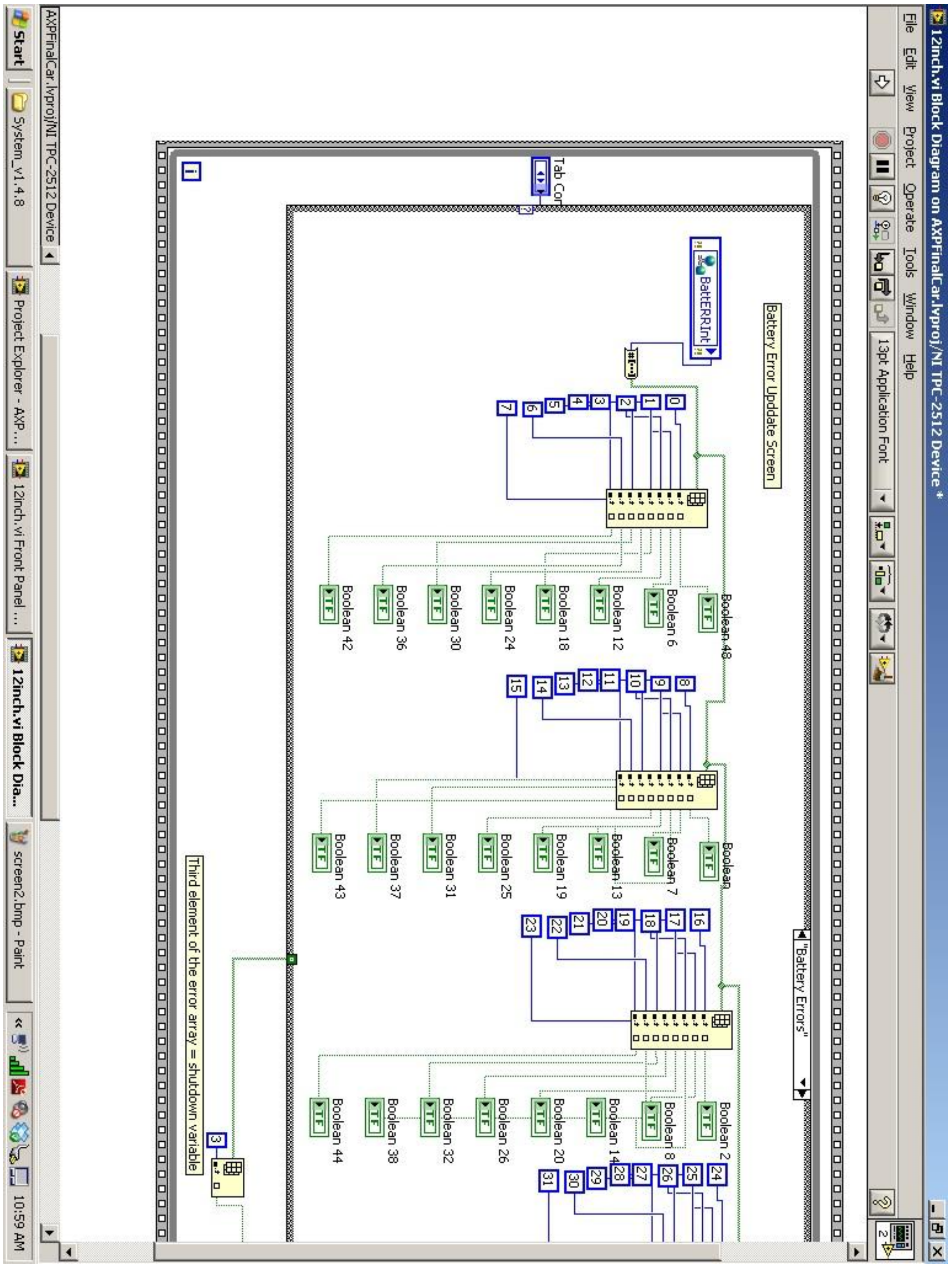
Touch Panel Figure 2



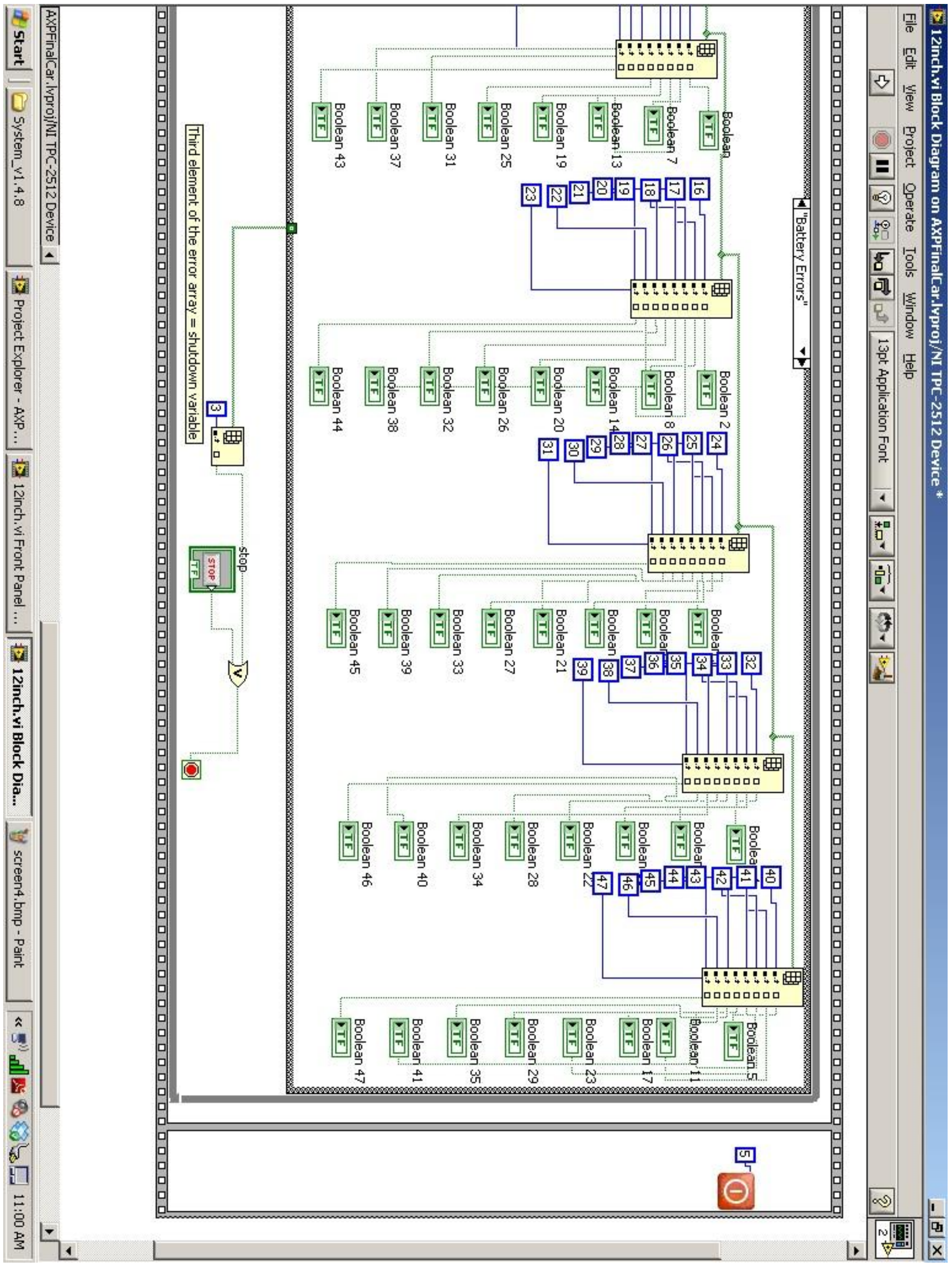
12inch.vi Block Diagram on AXPFinalCar.lvproj/NI TPC-2512 Device *

Third element of the error array = shutdown variable

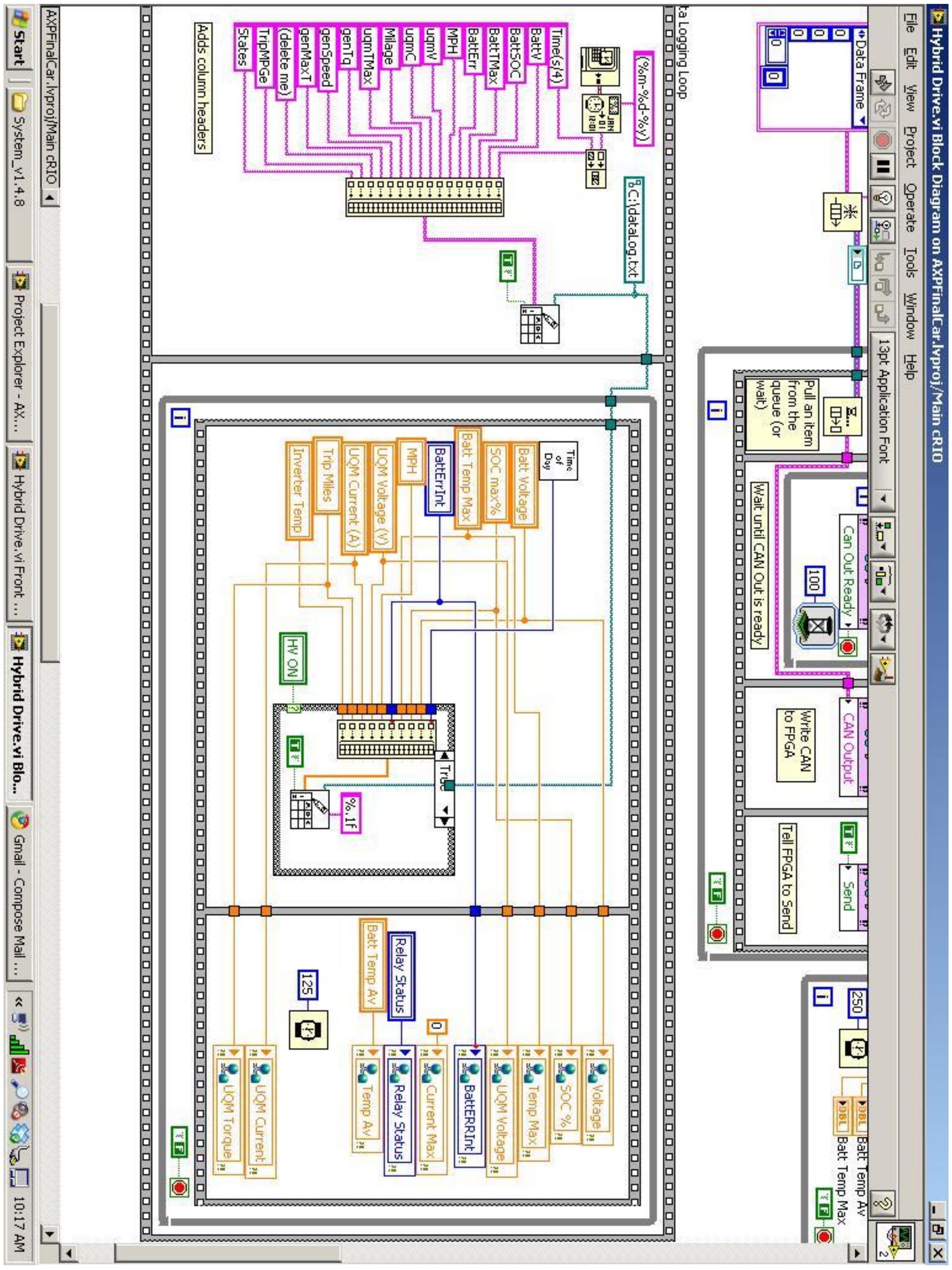
Touch Panel Figure 3



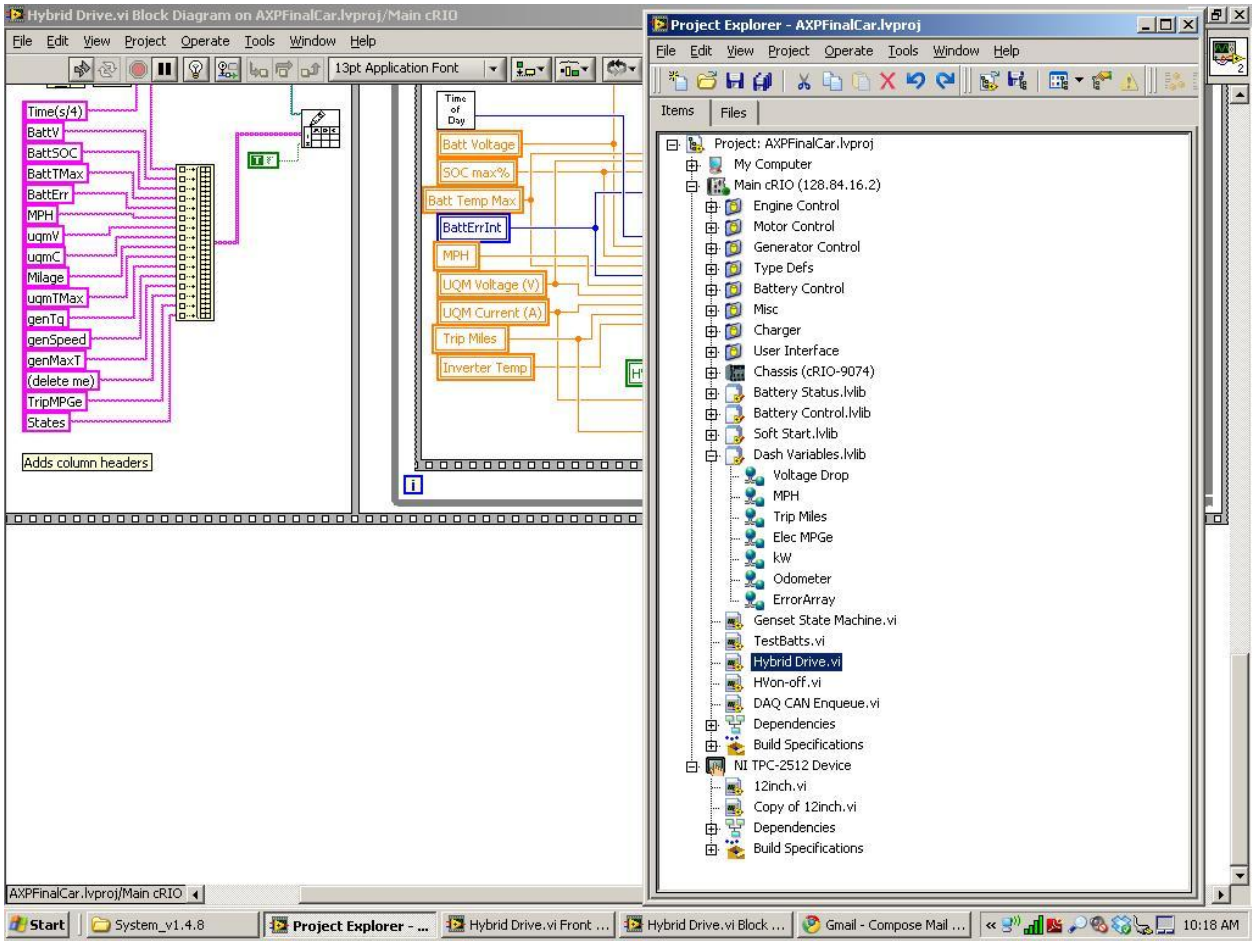
Touch Panel Figure 4



cRIO Figure 1



Project Figure 1



```
Project: AXPFinalCar.lvproj
  My Computer
  Main cRIO (128.84.16.2)
    Engine Control
    Motor Control
    Generator Control
    Type Defs
    Battery Control
    Misc
    Charger
    User Interface
    Chassis (cRIO-9074)
    Battery Status.lvlib
    Battery Control.lvlib
    Soft Start.lvlib
    Dash Variables.lvlib
      Voltage Drop
      MPH
      Trip Miles
      Elec MPGe
      kW
      Odometer
      ErrorArray
    Genset State Machine.vi
    TestBatts.vi
    Hybrid Drive.vi
    HVon-off.vi
    DAQ CAN Enqueue.vi
    Dependencies
    Build Specifications
  NI TPC-2512 Device
    12inch.vi
    Copy of 12inch.vi
    Dependencies
    Build Specifications
```