Adaptive Battery Management System Testbed

Fast-Prototyping PHEV Application Demonstration

Group 1: Haoyuan Qu, Nan Wang

Group 2: Shaobo Zhang, Zijian He

Mentor: Dr. Mo-Yuen Chow

August, 2012

Abstract

Nowadays, plug-in hybrid electric vehicles (PHEVs) are becoming more and more popular for the rising demand to use renewable energy instead of fossil fuels. However, accurate estimation of battery parameters and states, e.g. state of charge (SOC) and state of health (SOH) is still a major challenge in the purpose of optimizing the functionality of electric vehicles and avoiding drivers' anxiety of relying on the battery power for long distance drives.

To demonstrate, test and validate battery monitoring algorithms, an adaptive battery management system testbed is set up. Group 1 mainly focuses on the hardware part, i.e. to build a PHEV emulator. Group 2, on the other hand, deals with software and has developed a LabVIEW-based battery and electric vehicle data acquisition system. In this report, **Chapter 1** briefly introduces the background and objectives of our project. **Chapter 2** and **Chapter 3** illustrate our system in both hardware and software perspectives. **Chapter 4** summarizes our achievements and possible future work.

The system we have built is a fast-prototyping experimental platform for future battery tests. By implementing the battery monitoring algorithms on this PHEV testbed, the algorithms can be evaluated and optimized and thus make PHEV batteries more reliable.

Index terms: Battery, PHEV Emulator, LabVIEW, GUI

Contents

Chapter 1: Introduction

Plug-in hybrid electric vehicles (PHEVs) mark the trend of future transportation to reduce greenhouse gas emission, local air pollution and dependency on petroleum. One major challenge of the large-scale penetration of PHEVs in the transportation market, however, is the reliability and efficiency of batteries. Lack of effective battery monitoring system prevents consumers from enjoying a smooth trip by arousing anxiety and causing delays due to the recharging time. Currently several battery monitoring algorithms have been developed to estimate battery states.

Since the estimation of SOC in a PHEV battery is very important, we should evaluate how well our estimation algorithms work. Our project provides a battery management system testbed where different battery monitoring algorithms could be tested. First, an electric vehicle emulator has been built to simulate a real PHEV's power system. Also, we have already had a software simulator (called SimBattery) in MATLAB to simulate the battery performance, which runs in Simulink. After we have acquired data from hardware emulator, we use the algorithm in Simulink to estimate SOC of hardware emulator and in the meanwhile we present the input data and result in LabVIEW. To evaluate the feasibility of running the estimate algorithm in a real-time device such as MCU or DSP, we have a graduate student implement the algorithm on a microcontroller.

The expected goals are:

- All the needed data will be shown in a single GUI in LabVIEW.
- Results of estimation will be shown in the same GUI.
- Users can select different EV operating conditions for the emulator.

Chapter 2: PHEV Emulator

2.1 Description

The PHEV emulator refers to the hardware part of the battery management system testbed. It is mainly composed of battery, DC-AC converter, dynamometer, dynamometer controller and AC motor. The chief objective is to simulate the performance of PHEV using both battery and AC outlet as power supply. Since the system is an experimental platform for battery management systems on PHEV/PEVs, it is necessary for the system to be fast-prototyping and organized so that further experiments could be easily implemented.

2.2 Dynamometer

2.2.1 Getting Started

To familiarize ourselves with the dynamometer and its controller, we read the user manual and succeeded in operating the AC motor with a certain torque load. We also managed to set torque unit and maximum speed on the dynamometer controller.

2.2.2 Remote Control

One special feature of the dynamometer controller is remote control. By using a RS-232 connector, the computer can easily send commands to the dynamometer controller and request for torque and speed data. At first, we tested the connection between PC and the dynamometer controller using a sample LabVIEW program, which can manually send a command and acquire torque and speed data. To successfully run the program, we need to install the Virtual Instrument Software Architecture (VISA) driver. The communication between dynamometer controller and PC is based on a command string that is terminated with Carriage Return (CR)-Line Feed (LF). The commands we used are as followings:

Table 1 Commands for Dynamometer

Command Code	Function	Explanation
OD	Prompts to return speed-torque-direction data string	"Output Data" prompt to return data string with this format: SxxxxxTxxxxxRcrlf or SxxxxxTxxxxxLcrlf R or L is the shaft direction indicator, as viewed looking at the dynamometer shaft, where: R = right; clockwise (CW) L = left; counterclockwise (CCW) The speed will equal the displayed value and the torque will be in the same units as displayed on the front panel.

Q#	• Sets torque point to # • Turns brake ON	This is a closed loop command with its own set of PID parameters. The units defined will be the same as those displayed by the Controller.

Later this part of sample program was integrated with Group 2's program. While running the program, we found that I/O error would occur when the Baud rate was 9600bps. To solve the problem, the Baud rate has to be increased to 19200bps.

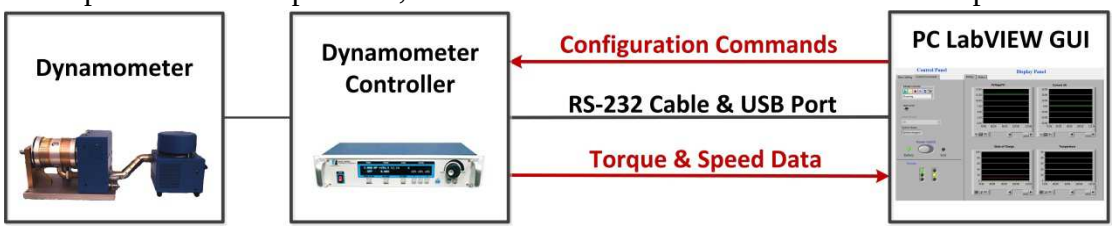

Figure 1 Remote Control of Dynamometer

2.2.3 Alignment

While running the whole system, we found that motor speed measured by dynamometer was zero. After checking the connection between each part, we discovered that the motor and dynamometer were not connected with each other. However, to simply connect the dynamometer to the motor shaft is far from enough. To get the system run smoothly without too much noise, proper alignment is required. Currently we have only aligned the system manually without any professional tool. Therefore, it is highly recommended that the system be aligned professionally by some mechanic students.

2.2.4 Calibration

According to the label on the dynamometer controller, the last time when it was calibrated was in 1997. To ensure its accuracy, it should be calibrated at least once a year. So we decided to calibrate the dynamometer controller. According to the user manual of DSP6000, the basic calibration process includes the following four steps:

- Initial Calibration Procedure
- Torque Offset and Gain
- Accessory Torque Output Offset and Gain
- Auxiliary Input Offset and Gain

The dynamometer, however, was not calibrated in our project. To have the dynamometer work accurately, proper calibration is still needed before implementing battery model tests.

2.2.5 Designing Torque and Speed Profiles

We set up our model basing on the parameter of Audi A4 1.8T's engine. The maximum power is 120 kW/5700 rpm and the maximum torque is 225N·m/1950-4700rpm. Assumed that the car is at a constant speed, thus the angular velocity is also constant.

For driving on the flat road, suppose that the angular velocity is 4500rpm and the torque is 150N·m. When the car is ramping up, the torque is designed to be 225 N·m.

In this situation, the power is 105kW.

When the car is ramping down, the torque get smaller, we make it to be 75N·m. When the car is turning, we assume that the power remain constant, if the angular velocity slow down to 3000 rpm, the torque will be 225N·m.

Under above assumption, we designed a profile to simulate ramp up, ramp down and turning. Then, we built a program to send the user-defined profiles to dynamometer controller in every second. Although the user profile is relatively rough, it is easy to modify it and load it into our LabVIEW program.

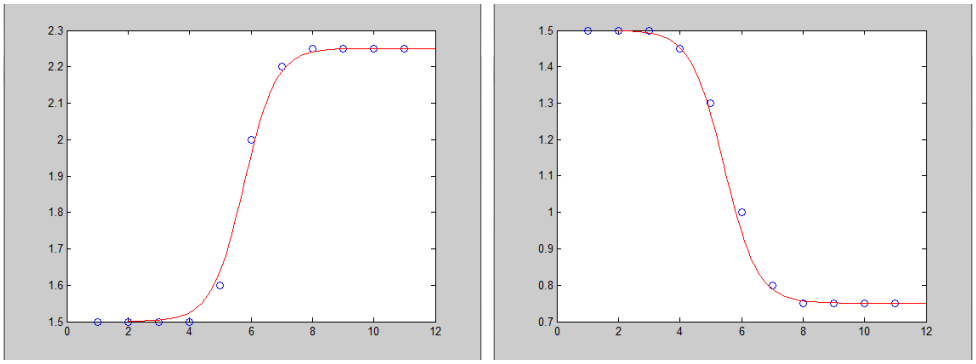

Figure 2 Ramping up and Ramping Down

2.3 DC-AC Converter

2.3.1 Operating Modes

After the DC-AC converter has arrived, we went through the user manual first to get an overall impression of how it works. It has three operating modes, namely "AUTO/REMOTE", "DC OFF" and "CHARGE ONLY". A brief introduction to these three modes is listed in **Table 2**.

Table 2 Operating Modes of DC-AC Converter

Operation Modes	Basic Function
AUTO/REMOTE	The DC-AC converter supplies continuous AC power for the load and works in the inverter mode when disconnected from AC outlet. The battery will be charged if AC is plugged in. This mode also enables remote control.
DC OFF	In this mode, DC power supply from the battery is cut off. AC outlet becomes the only power source.
CHARGE ONLY	The converter will not supply any AC power to the load. Instead, it only acts as a charger of the battery.

2.3.2 Indicator Lights

The DC-AC converter has six indicator lights in total. The left column shows the capacity of battery and the right column indicates the operating mode of the converter.

Figure 3 Front Panel of DC-AC Converter

Table 3 summarizes different meanings of indicator lights in different operating modes according to the user manual. **Table 4** provides an approximate battery charge level according to the LEDs illuminated.

Table 3 Indicator Lights of Operation

Indicator Lights	Explanation
LINE Green LED	If the operating mode switch is set to "AUTO/REMOTE", this light will illuminate continuously when your connected equipment is receiving continuous AC power supplied from a utility/generator source. If the operating mode switch is set to "CHARGE ONLY", this light will blink to alert you that the unit's inverter is off and will not supply AC power in the absence of a utility/generator source or in over/under voltage situations.
INV Yellow LED	This light will illuminate continuously whenever connected equipment is receiving battery-supplied, inverted AC power (in the absence of a utility/generator source or in over/under voltage situations). This light will be off when AC power is supplying the load.
LOAD Red LED	This red light will illuminate continuously whenever the inverter is functioning and the power demanded by the connected appliances and equipment exceeds 100% of load capacity. The light will blink to alert you when the inverter shuts down due to a severe overload mode switch to "DC OFF"; remove the overload and let the unit cool. You may then turn the operating mode switch to either "AUTO/REMOTE" or "CHARGE ONLY" after it has adequately cooled. This light will be off when AC power is supplying the load.

Table 4 Battery Voltage LEDs

LEDs Illuminated	Battery Capacity (Charging/Discharging)
Green	91%-Full
Green & Yellow	81%-90%

Yellow	61%-80%
Yellow & Red	41%-60%
Red	21%-40%
All three lights off	1%-20%
Flashing red	0% (Inverter shutdown)

2.3.3 Remote Control

At first, we planned to use a UTP cable to achieve both remote control and LED display on PC. After carefully studying the user manual, we came to realize that as long as AC is plugged in, the DC-AC converter working in the “AUTO/REMOTE” mode will continuously supply AC power from AC outlet instead of the battery. This deviates from our original goal, which is to remotely control the power source of AC motor.

In our second plan, we have designed a relay circuit to control the AC power supply. The relay could be triggered on by a digital output signal from the DAQ board, which is controlled on the LabVIEW graphical user interface (GUI).

In the relay control circuit, a transistor is used to switch on the relay and provide the current for the coil. The LED was designed to indicate the status of the relay, but was unfortunately hidden from sight after putting the relay circuit into a box. The circuit diagram is shown in **Figure 4**. **Figure 5Figure 5** shows the controllable power outlet we have built using relay both from inside and outside.

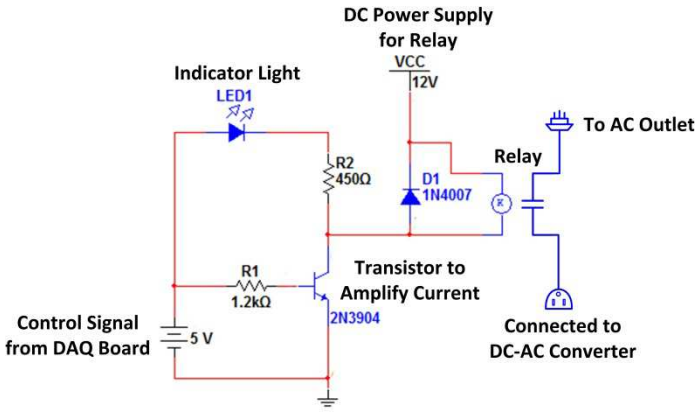

Figure 4 Relay Control Circuit Diagram

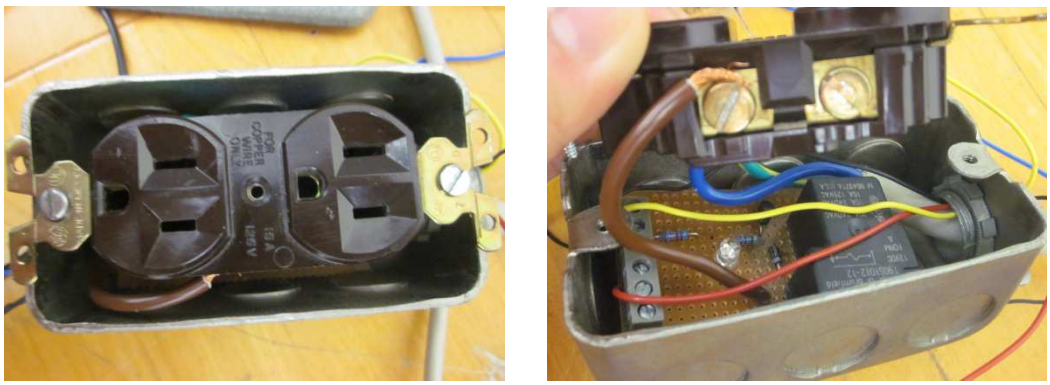

Figure 5 Controllable Power Outlet

For the connection between the AC cord of DC-AC converter and AC outlet, we used an extra AC cord. The hot wire of two AC cords are connected to two metallic contacts respectively, while their neutral and earth wires are directly connected with each other. Because the relay requires a DC voltage of 12V, an additional DC power supply is needed. The trigger signal of the relay control circuit is provided by a digital I/O on the DAQ board, which is controlled by LabVIEW GUI.

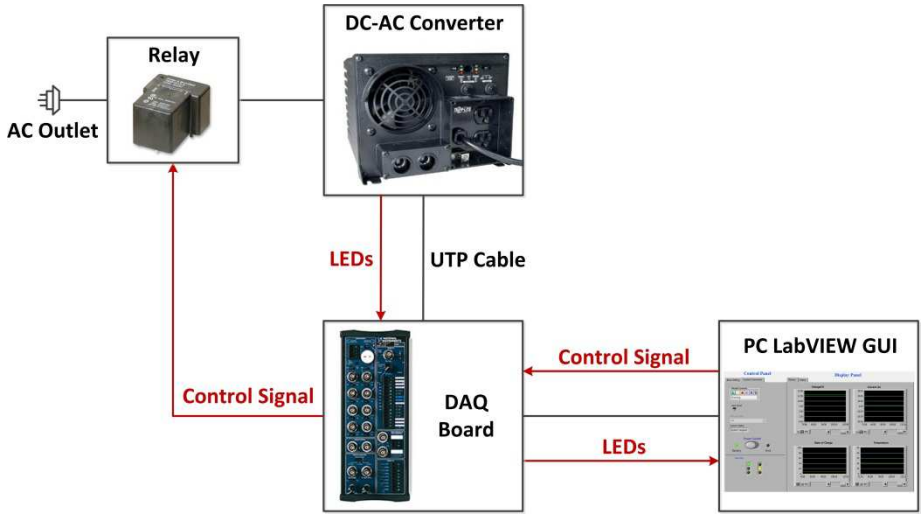

Figure 6 Remote Control of DC-AC Converter

Besides, by using a UTP cable we can display LEDs of DC-AC converter on the LabVIEW GUI. The relationship between UTP cable pins and LEDs are listed in **Table 5**. **Figure 7** shows pin locations on RJ45 plug.[1]

Table 5 UTP Cable Pins

Pins	Indicator Lights
3	Battery Green LED
5	Battery Yellow LED
4	Battery Red LED
6	LINE Green LED
2	INV Yellow LED
8	LOAD Red LED

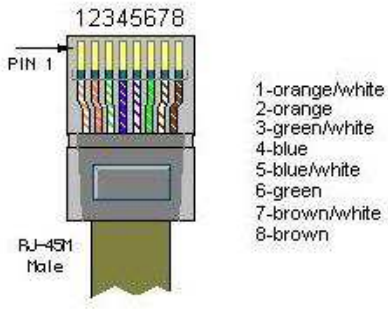

Figure 7 Pin Location on RJ45

In this way, LEDs of DC-AC converter will be shown on both DAQ board and LabVIEW GUI. All of the indicator lights on the DC-AC converter will be illuminated as long as the UTP cable is plugged in.

2.3.4 Grounding

As is suggested in user manual, the DC-AC converter should be grounded. We used 8AWG wire to connect DC-AC converter to the metal table leg in order to ensure safe grounding.

[1] Source: http://classbforum.com/phpBB2/viewtopic.php?t=20&start=0&postdays=0&postorder=asc&highlight=

2.4 Battery

2.4.1 Wire Selection

To successfully connect the battery to DC-AC converter, proper wires should be selected so that it will not be burnt by the current. According to the ratings of DC-AC converter, total combined charger and bypass input DC current should not exceed 12A. To ensure safety, we should consider twice the maximum current, i.e. 24A. According to NEC (National Electrical Code), the copper wire ampacity of 12 AWG wire is 25/25/30A with 60/75/90°C insulation respectively. Therefore, 12 AWG wires are required.

2.4.2 Fuse Selection

It is required by NEC that the positive DC terminal should be directly connected to a UL-listed fuse and fuse block within 450mm of the battery. The fuse size is selected with respect to the wire gauges. Thus, 25A fuses are selected for the system and have also been proved adequate during tests. We have also prepared 30A fuses in case 25A fuses are burnt.

2.4.3 Battery Switching

To compare the performance of two different batteries, we have designed a basic plan to switch from one battery to another.

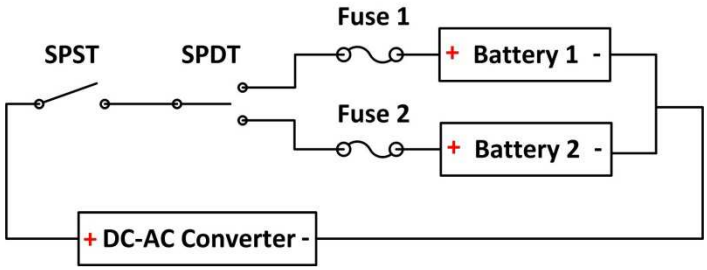

Figure 8 Battery Switching Plan

A single pole single toggle switch is used to switch on battery power supply. For battery switching, a single pole double toggle switch is adopted.

To make the system more beautiful and convenient, we have built a switch panel using a box to wrap up the wires and switches. After the whole system runs successfully, we cleaned up space for it and shortened the wires to make it neat and tidy.

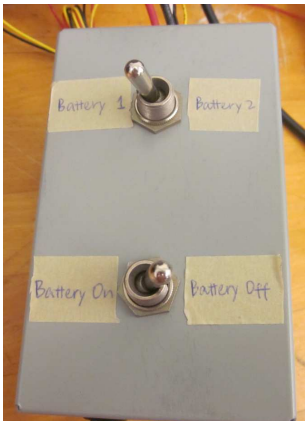

Figure 9 Switch Panel

Chapter 3: Data Acquisition System

3.1 Final Design

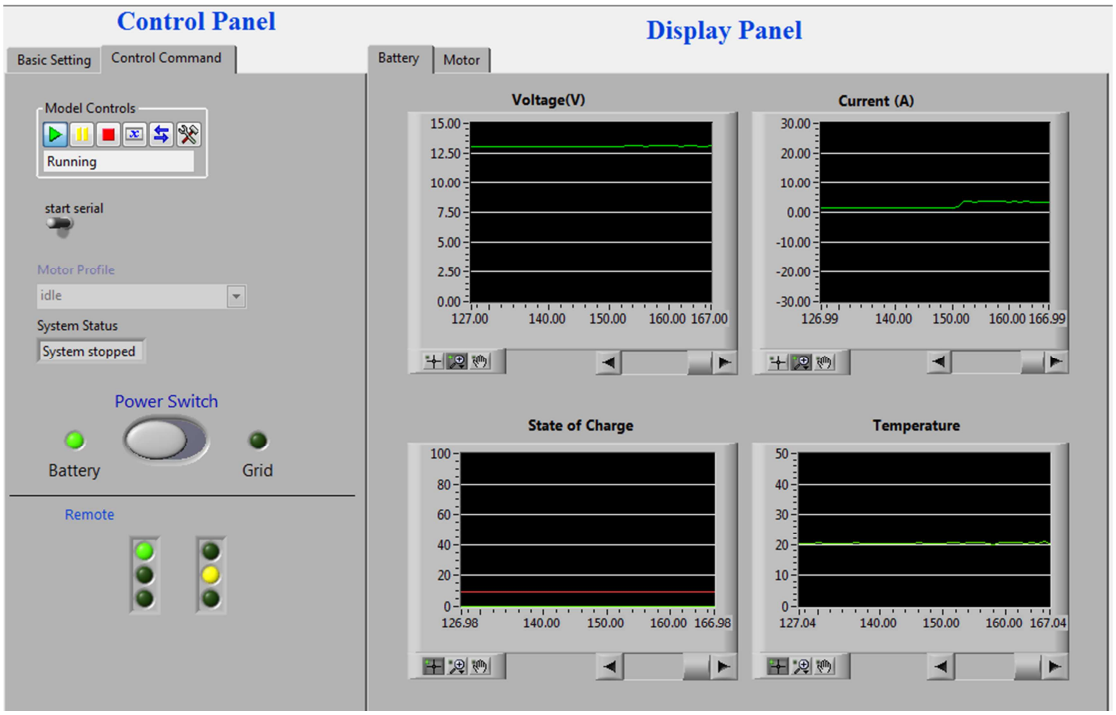

Figure 10 Overview of Front Panel

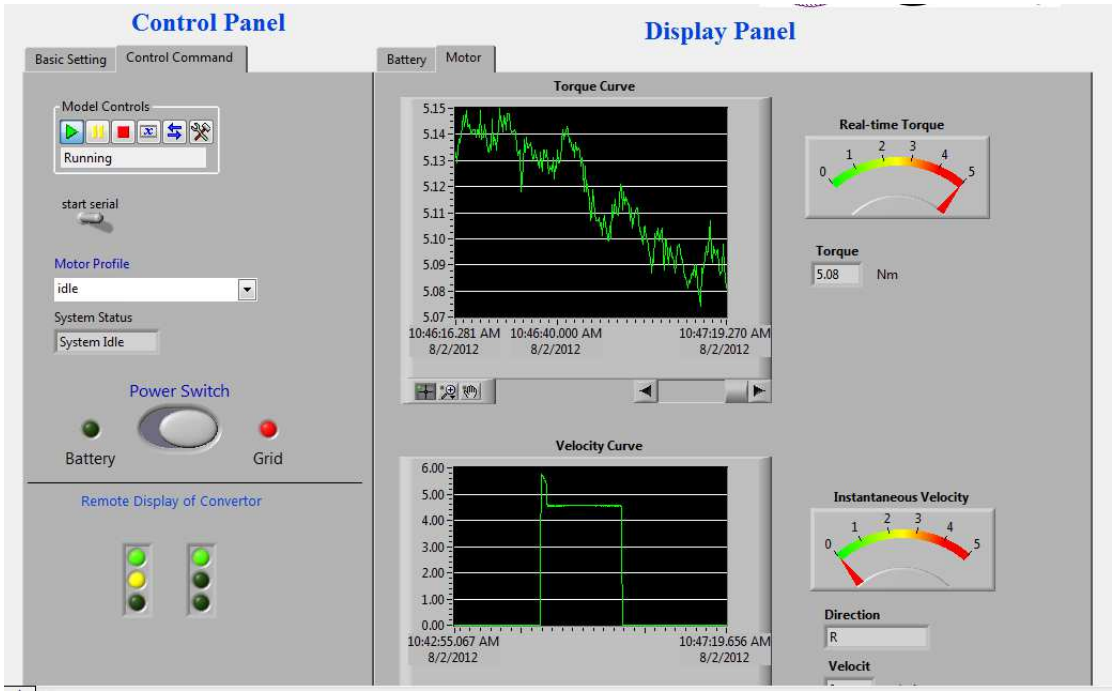

Figure 11 Overview of Front Panel (Continued)

Software design can be classified into these parts:

- Read torque and speed from the dynamometer;

- Read battery voltage, current and temperature;
- Send torque and speed set-point to the dynamometer;
- Remotely display status of the converter;
- Control a relay to change power source between grid and battery;
- Build a data transmission interface between LabVIEW and MATLAB/Simulink.

Because LabVIEW can run each part of program parallelly, we can easily put them together to get our final design. After we classified the design, we developed each part independently and tested them till they can meet our needs. Finally we integrated those parts together and made some necessary adjustments.

3.1.1 Reading Torque and Speed from the Dynamometer

We used serial port to communicate with the dynamometer. By sending "OD", the dynamometer will return instant mechanical torque and rotate speed of the motor in a string with the format of "SdddddTdddd.R(cr)(lf)". Then the program will parse the string to get the data and repeat the process with a period of 100ms. Finally, the data will be plotted onto a chart.

3.1.2 Reading Battery Voltage, Current and Temperature

3.1.2.1 Voltage

A DAQ board was used to measure the voltage of these analogy signals and after acquisition we would convert them to their real values according to their physical characteristics.

To make high-quality measurements, we should follow these rules:

- Maximize the precision and accuracy
- Minimize the noise
- Match the sensor range to the A/D range

Because the maximum input voltage range of the DAQ board that we have is from -10V to 10V and the voltage of the battery can up to 14V or more, we designed a signal conditioning circuit to step-down the input signal. We calibrated the signal conditioning circuit and after get the step-down signal we multiply it with a constant to restore the voltage signal.

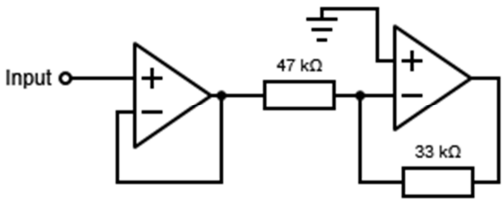

Figure 12 Step-down circuit

3.1.2.2 Current

We use a Hall current sensor to measure the current of battery because it can not

only measure AC current but also DC current and current with arbitrary wave-forms. There is also a linear relationship between the current being measured and the voltage acquired from the output of sensor, so we used a method similar with the one in voltage measurement.

3.1.2.3 Temperature

Although LabVIEW have built-in feature that can measure temperature directly (the conversion from sensor output to real data), we have to measure sensor output and use a look-up table to convert it to temperature because that we decide to drive DAQ board inside MATLAB instead of taking the measurement in LabVIEW and transmit data to MATLAB. If one tries to use analog input from LabVIEW and MATLAB at the same time, there will be some conflict.

3.1.3 Setting Torque and Speed

Similar to read data from dynamometer, we can easily set torque and speed by sending a string with the format of "Q#", the character # represents a floating-point numerical value following the command. Due to the reason that both reading and setting need usage of serial port, if not arranged appropriately there will be access violations. So we put reading and setting in a fixed time sequence to make sure our program will run smoothly. Every 100ms PC will send command to dynamometer for read data and every 10 data acquisition commands we will try to send command for setting torque or speed. In this way, the sampling speed is maintained constantly and there won't be any violation.

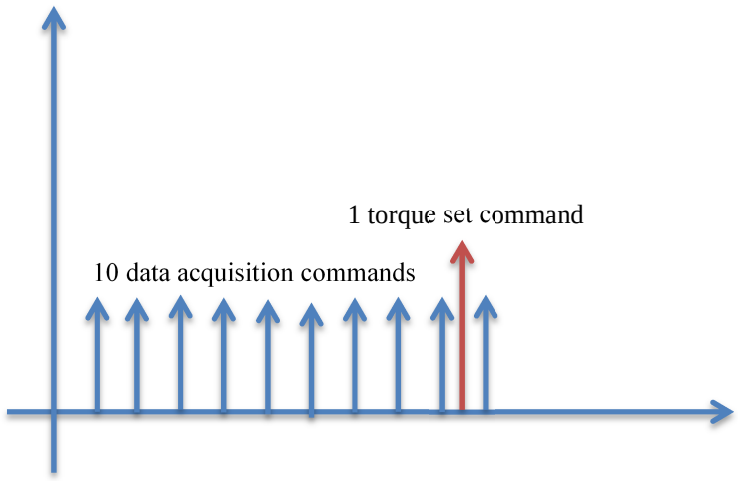

Figure 13 Time Sequence of Sending Command

Because we want to load different profiles into the dynamometer, we have to send different command to it in response of user’s inputs. We use a state-machine model to achieve these state changes.

3.1.4 Data Transmission Interface between LabVIEW and Simulink

At the very beginning of our project, we began to think how our program should work, especially by what mean can we join the control and display GUI in LabVIEW and the algorithm developed in Simulink.

Because both data acquisition and model simulation are at fixed period, but the time used to simulate 1 sec may be short to 1ms or be long to 10s, we cannot simply fix the simulation period and data sampling period to the same value. Our goal is to provide the needed value at each time spot when Simulink began to solve the state of the model at next time spot. A simple example is that, when Simulink has simulated past 1 second, and now it will simulate the next time spot according to its step length, but the simulation maybe very slow and it takes 10s for the model to run over the simulation of the past 1 second. Of course, the model now needs the input data acquired at t=1s, but now t=10s. Same thing will happen when simulation is faster than real world, except that it needs the data acquired in the future. So only by some mechanism that will store the acquired data when the simulation is too slow or halt the simulation when the simulation is too fast can we get the right result and evaluate our model correctly.

We can solve the above problem by introduce a data structure called queue. "In computer science, a queue is a particular kind of abstract data type or collection in which the entities in the collection are kept in order and the principal (or only) operations on the collection are the addition of entities to the rear terminal position and removal of entities from the front terminal position. This makes the queue a First-In-First-Out (FIFO) data structure. In a FIFO data structure, the first element added to the queue will be the first one to be removed. This is equivalent to the requirement that once an element is added, all elements that were added before have to be removed before the new element can be invoked. " from Wikipedia. So if we can realize this data structure, we are able to correctly run model with physical inputs.

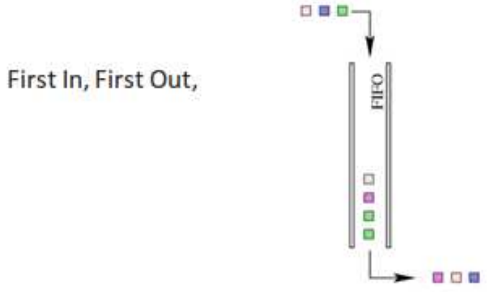

Figure 14 Queue

At first, we came up with an idea that we can measure the input within LabVIEW and write the data into a file while the measurement goes on, and, at the same time, read the file and get the measured data. In MATLAB we can store the data read in a FIFO structure and use the stored data to simulate. But this is too complicated to build and the system will not be stable because the file operation between LabVIEW and MATLAB may happen at the same time and when this happens an error will be reported and the simulation will stop. Part of this idea is realized in last year interns' work, but we decide to look for other better solutions.

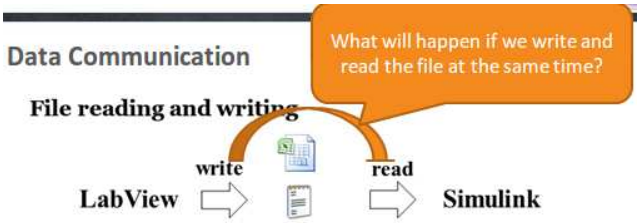

Figure 15 Why we don't use file reading and writing

We heard from our group meeting that MATLAB can take measurement directly using a toolbox called data acquisition toolbox. With it, we are able to use a block called "analog input" which can be chosen to run in asynchronous mode.

> *Asynchronous*
> *Initiates the acquisition when the simulation starts. The simulation runs while data is acquired into a FIFO (First in, First out) buffer. The acquisition is continuous; the block buffers data while outputting a scan/frame of data at each time step.*
> *Synchronous*
> *Initiates the acquisition at each time step. The simulation will not continue until the requested block of data is acquired. This is unbuffered input; the block will synchronously output the latest scan/frame of data at each time step.*

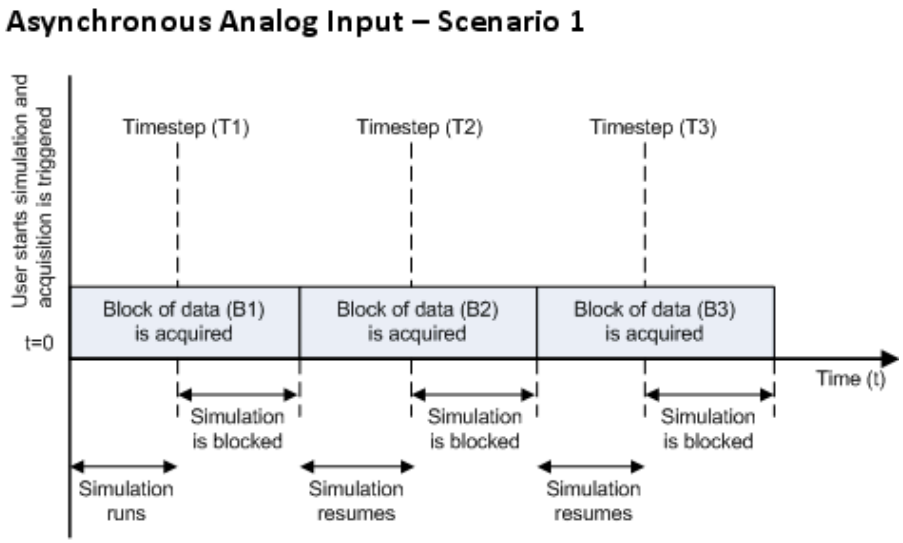

Figure 16 Asynchronous Analog Input--Scenario1

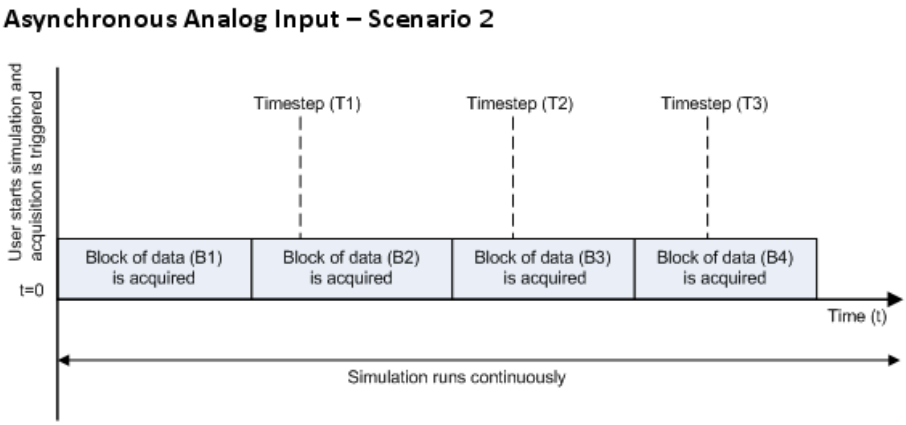

Figure 17 Asynchronous Analog Input--Scenario 2

There is also an important attribute “Block size” that matters how our data will be collected.

> *Block size*
> *The desired number of data samples to output at each time step for each channel. Block size corresponds to the SamplesPerTrigger property for an analog input device. The default value for block size depends on the hardware selected. It must be a positive integer, and be within the range allowed for the selected hardware.*

After solving the data input of Simulink problem, we use an National Instruments product called Simulink Interface Toolkit to show the model output inside LabVIEW VI. It can control the simulation to start, halt or stop and can control or show some Simulink components mapped to controllers or indicators inside LabVIEW VI.

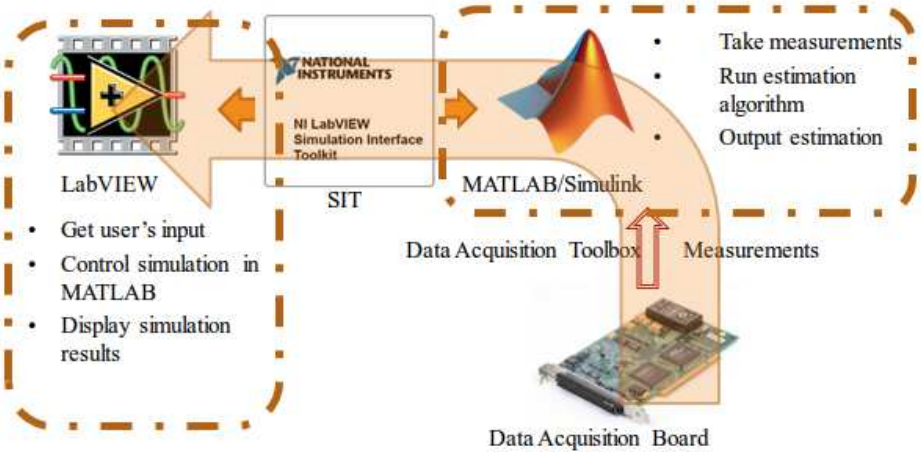

Figure 18 Block Diagram of Data Communication

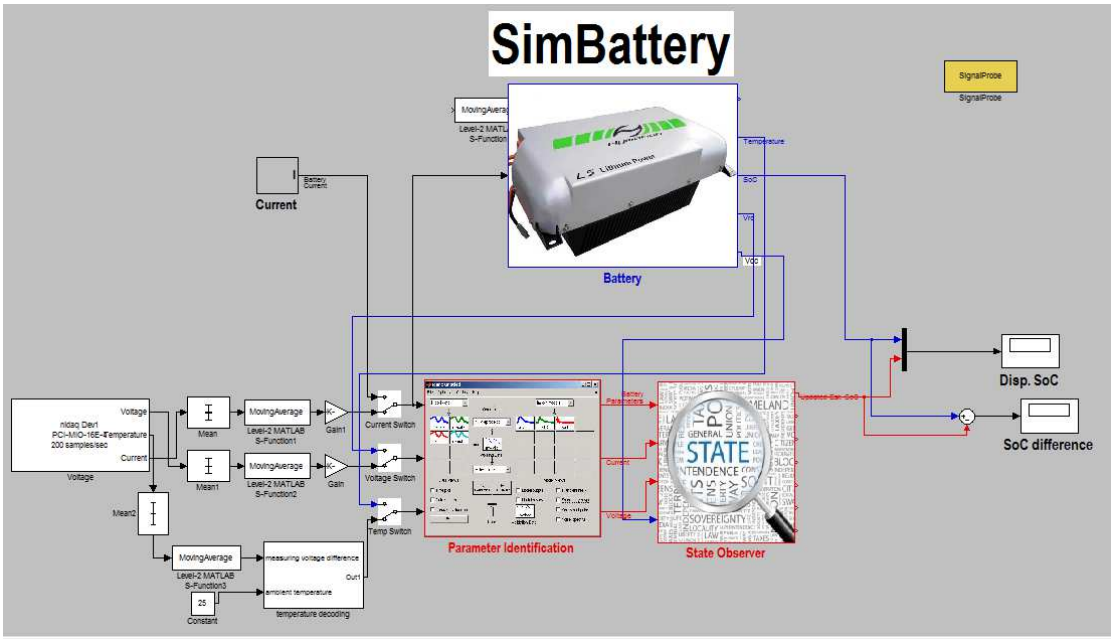

Figure 19 Simulink overview

3.2 Development Steps

We roughly followed the steps assigned in the task instruction document, but we made some changes. For example, we took some work from hardware group that relates with LabVIEW, because we are more familiar with LabVIEW and the whole software we've built. However, the data used to simulate different battery operations is still generated by hardware group because they are more familiar with the power chain of our system.

3.2.1 Get Familiar with Last Year Summer Interns' Work

1) Know what will our system work for
   a) Estimating SOC, EOL and SOH of a PHEV's battery is very important
   b) We should evaluate how our estimating algorithm works
2) Read the final report of the last year interns' work.
3) Get familiar with the existing GUI in LabVIEW and how to run it in accordance with *SimBattery*.
   a) Two days learning how to use LabVIEW (e.g. installment, reading courses of study, running examples)
   b) Running last year summer interns' work, sadly to find that the model we get in Simulink has some problems
   c) Reconstruction of the model using an runnable model in Simulink

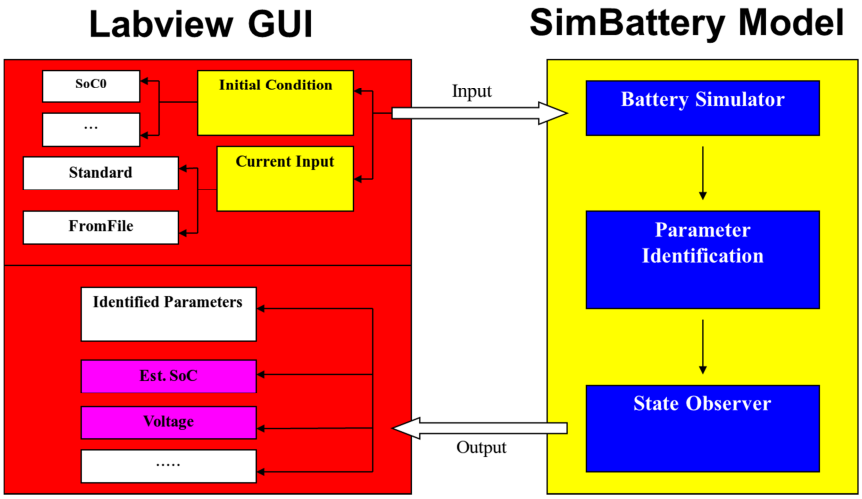

Figure 20 How LabVIEW and Matlab interact in last year's work

3.2.2 Design Data Transfer Method between LabVIEW and Matlab

1) Found that last year's work may not meet our demand, for the hardness of understanding diagram and though the functions are very similar but have very different design style.
2) Learned that we can use Matlab to read DAQ board directly.
3) Designed our final program structure.
4) Based on the range of the DAQ card input signals and range of the output signal of the battery voltage, we find that we need a step down circuit. So we designed and implemented it.

3.2.3 Design Signal Conditioning Circuits

1) Designed an interface circuit to connect the voltage of the battery to the input port of the DAQ card.
2) Based on the range of the DAQ card input signals and range of the output signal of the current sensor and the temperature sensor, we find we don't need design signal conditioning circuits for current sensor.
3) Parsed raw voltage data of temperature difference electromotive force into temperature difference
4) Tested the digital input of battery remote monitoring based on Week2's work.
5) Improved signal conditioning circuit by adding an buffer to the input port of the divider.
6) Calibrate the current sensor and replace battery in the sensor.
7) Build a simple LabVIEW program to monitor the discharging and charging current of battery in the need of diagnosing why the fuse burn during operation and get the waveform of battery current. Choose fuse according to the maximum root-mean-square value (RMS) of current. And the waveform of battery current is kind of like pulse width modulation.

8) Write a program that can send different serial setting points to dynamometer according to the profile that user has chosen. I’ve tested it at Friday and it can read and write data at the same time. (This originally is not our part of work, but we decided to change our work for a higher efficiency).
9) Write an abstract of our work.

3.2.4 Wrap up

1) Revised our abstract.
2) Began to write our poster.
3) Integrated our work into final GUI, do some debug and find that although our program can collect data correctly, the output estimation SOC of both *Simbattery* and real battery doesn't look right. So we decided to design a moving average filter for the input of data.
4) Write final report and user manual.

3.3 Difficulties

3.3.1 Getting Familiar with Last Year Interns' Work

Because we don't familiar with LabVIEW and Simulink Interface Toolkit at first, we have some difficulties in running last year work. When we run the VI, it will ask us to indicate where the fold that Simulink model file exists because the path stored inside the VI wasn't the path we are currently using. We assigned it the right path and model file, and hit start simulation, it will produce a simulation error inside Simulink says "Frame size.....". We followed the steps on users guide and run the preparation model in Simulink before we got this error. We know the model can run independently without LabVIEW, so we try to run it outside LabVIEW. However, the error message appears too. Because this model is designed by ADAC lab and modified by last year interns, so we couldn't find out which part of the model is wrong. After discussion, we decided to modify a runnable version of MATLAB instead of trying to debug where is wrong. This is because debugging an unfamiliar program will take a lot of time and the modification of last year interns is not too much and we can rebuild the program in a quite short time. We asked for a runnable version of model and modified it with parts of model from last year intern's work and finally, last year intern's work was able to run.

3.3.2 Reading Data from Dynamometer

We compose the command according to the user's guide of dynamometer and send command periodically to the dynamometer. But at first no matter how we change the period, the dynamometer will produce I/O Errors all the time. We double checked the configuration of serial port both on the PC and dynamometer, but the error still existed. Finally we got to the conclusion that the serial line was too long and because of that the error rate maybe too high. So we increased the baud rate from 9600 to 19200 and the

dynamometer worked with much fewer errors.

3.3.3 Using Data Acquisition Toolbox in Simulink

We wanted to use DAT to acquire data from DAQ board in Simulink. Although we can find the board in command line in Matlab, and also successfully take samples from the input but we failed to find the board in Simulink (we can only find windows sound wave card). After hard search on the Internet, we find a post on a Chinese forum that we have to register our device in Matlab. Without the step, we cannot use the board in Simulink.

The command is, in our case, **daqregister('nidaq')**.

3.3.4 Install Simulation Interface Toolkit

SIT only supports MATLAB version earlier than 2010b, but in our environment, the version is 2011b, so the install program of SIT couldn't recognize MATLAB and are not able to install the file needed. We tried to install the files needed by hand. First, we added the root path of SIT to path variable in MATLAB. Then we put some files found in SIT into the directory of MATLAB. After these steps, we need to add two commands to the script that MATLAB will run every time it starts. The commands are "NISIT_AddPaths;NISITServer;".

If MATLAB was not installed correctly, the path variable may be damaged and needs to be repaired, just follow the instruction show in MATLAB to repair path variable.

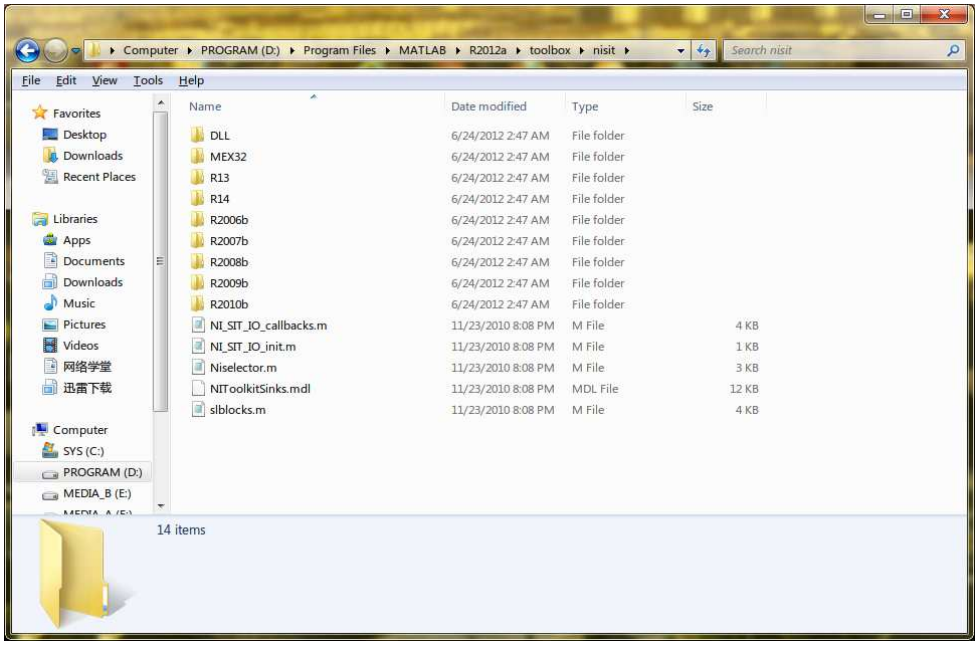

Figure 21 Copy files to MATLAB

Chapter 4: Achievements and Future work

4.1 Achievements

- ✓ Remotely monitor and control converter.
- ✓ Remotely monitor output torque and speed of AC motor.
- ✓ Simulate AC motor operations such as “ramp up” and “ramp down”
- ✓ Run simulation with different battery estimation algorithm for immediate analysis.

4.2 Conclusion & Future Work

- ✓ A basic fast prototyping platform of battery estimation algorithm has been realized with our platform
- ✓ We are currently improving the platform to be more user-friendly in order to meet different application perspective, and preliminary results are encouraging.
- ✓ Driving profiles could be more delicately designed by adding more operating conditions. It would be better to have someone professionally design the profiles.
- ✓ Another charger could be added to the system to charge the battery while the other is supplying power.

Acknowledgments

We sincerely acknowledge Dr. Mo-Yuen Chow for the guidance during these five weeks. We would also like to thank everyone in ADAC lab for hosting us and kindly giving us suggestions on our posters, especially Habiballah Rahimi Eichi, Udai Muhammed, Yuan Zhang and Rudy Salas for help with the project.