SOLAR HOT WATER HEATER NET ENERGY MONITOR

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

Xun Yang

Project Advisor: Dr. Bruce Land

Degree Date: May 2011

Abstract

Master of Electrical Engineering Program

Cornell University

Design Project Report

Project Title: Solar Hot Water Heater Net Energy Monitor

Author: Xun Yang

Abstract: An embedded device is created to monitor the net energy of the solar water heater system in order to understand the heater's performance and daily household hot water energy usage. The device is created based on an AVR mega644 microcontroller. It collects the data from the solar water heater including the water temperature and output water flow speed in order to make the energy calculation. All collect data is periodically logged on a SD card and is further imported to PC and analyzed in a MATLAB program with extreme ease. The project eventually offers an intuitive digital interface for the solar heater and makes the interaction easier which enables the users to better understand the energy harvest and consumption and make contributions to the green planet.

Table of Contents

Table of Figures

Abstract

This design report is written to provide the detailed information on all aspects of the MEng project – Solar hot water heater net energy monitor. The design report starts with the introduction of the project including some background information. The report also has the software and hardware sections which cover how the project is constructed and the result section tells about its outcome. The conclusion section summarized the achievement and gives glimpse of the possible future implementations. The appendix section includes all the embedded code and any other related information. By reading the design report, one should be very clear about how the design works and be able to duplicate the project.

Introduction

Overview

The project is to create an embedded device to measure the net energy delivery of the hot water in a home-use solar hot water system. The goal of the project is to enable the users to have a clear sense of how well the solar water heater performs and both the amounts of the daily household energy usage on the hot water and the water usage. The introduction of the net energy concept offers an intuitive measure of how much energy is taken advantage of from the solar heater.

The project provides a complete solution to monitor the net energy. The solution walks through three major stages. First of all, it collects data from the water heater. Then data is logged on a SD card on a periodic basis. The log file stored on the SD card can be further imported in customized program or MATLAB for analysis. The following figure shows the overview of the design.

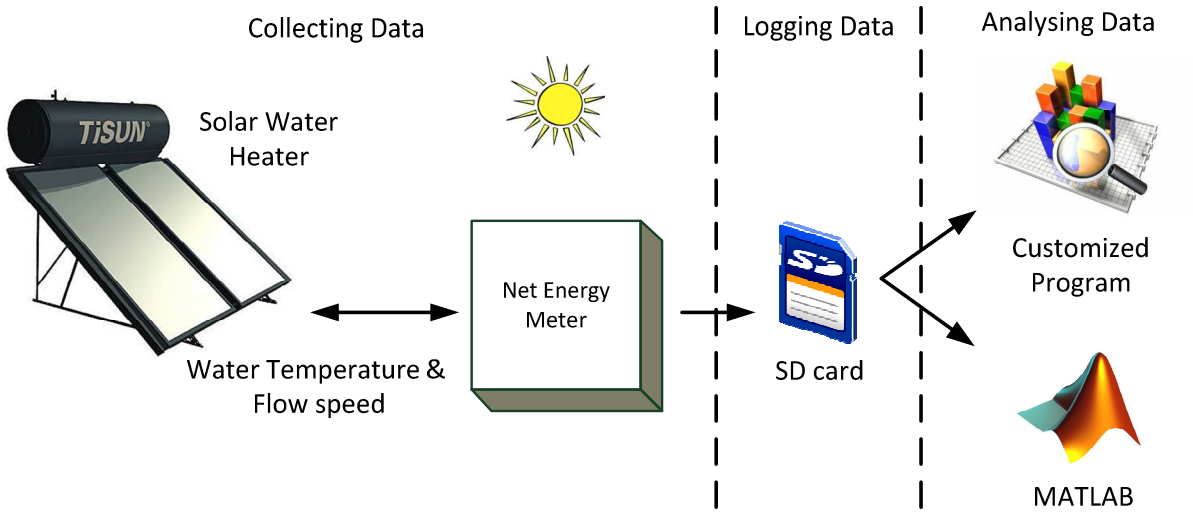

Figure 1 Conceptual Overview

The following picture shows the actual well-finished end product.

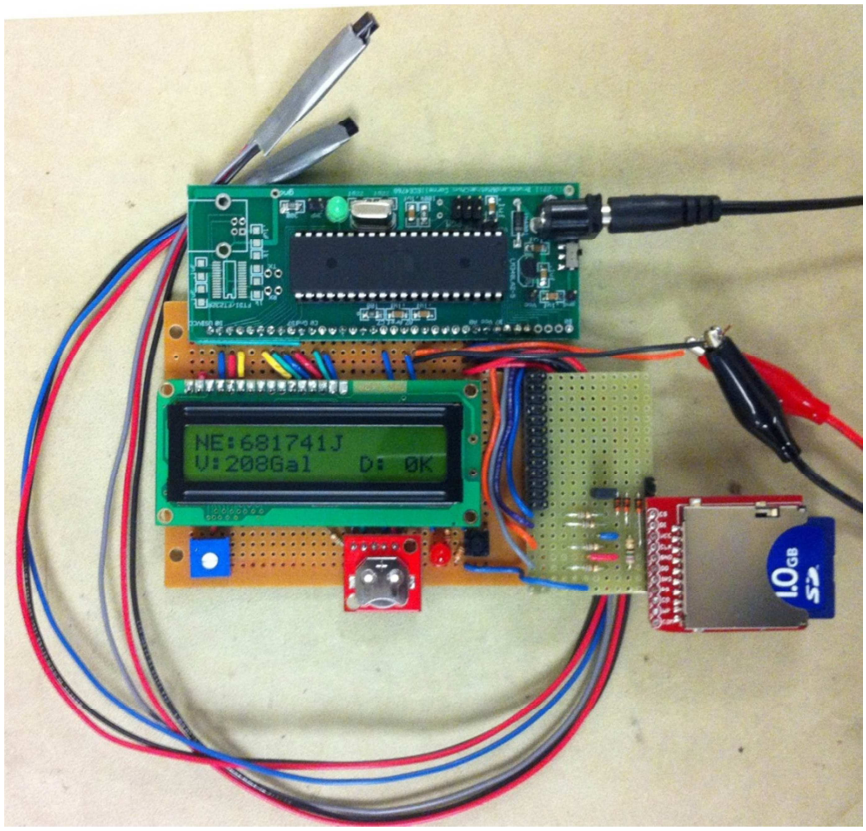

Figure 2 Project Overview

Background

The solar water heater system is for home use and the tank can store up to 80 gallons of water which equals to about 300 liters in metric unit. During the daytime, the heating component heats the cold water in the tank and the heated water is stored in the tank located in the basement before it is used. The temperature of the water cycling in the water heater has a range of 10 ˚C to 90 ˚C (41 ˚F to 194 ˚F). A digital water flow meter consistently monitors the output water flow speed from the tank and toggles the output digital signal 75 times per gallon of water going through.

Design

Rationale

Energy becomes an increasingly popular topic in the current world as petroleum resource on earth inevitably runs out eventually someday in the future. People care about the future generations and keep discovering new alternative energies to replace the dominant role of the precious natural resources. This is definitely the radical method of solving the energy crisis; however, on the other hand, improving the energy usage efficiency can also be a crucial way to solve the energy crisis. Among the new alternative energies, the solar energy is the easiest one to access for common people. To take better advantage of the solar energy,

it helps significantly to improve the efficiency if people know how the energy is utilized and how much they benefit from this alternative energy source.

However, the problems come when people want to figure out the actual amount of the energy they acquire from the sun. It is inaccurate and overestimated to measure the net energy of the solar hot water by checking the output power of the solar heater. There are two major reasons attributing to the problem. First, the inconsistent sunshine intensity provides inconsistent heat delivery. Second, the heat emission of the hot water when storing in the water tank is unmeasurable.

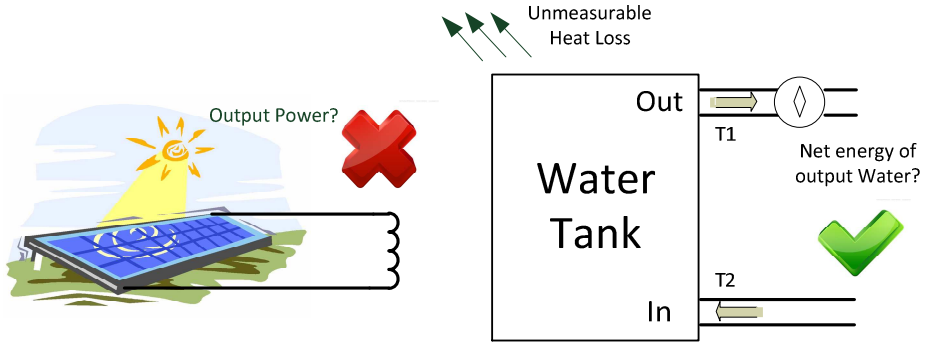

Figure 3 Ways to Measure the Net Energy

Since it is difficult to measure the energy input, it becomes obvious to measure the net energy by just measuring the energy difference between the water going in and out.

Theory

To obtain the information on the heat delivery of the hot water, several critical measurements to the temperatures and water flow speed are made. One critical concept when calculating the energy delivery is the specific heat of the water. The heat change in the water can be calculated as following.

$$H = C_p * (T1 - T2) * \Delta m$$

Shown in the formula above, the calculation of energy different H involves several parameters. T1 and T2 represent the temperature of the water going out from and into the tank respectively. Δm is mass of the mass of the water. C_p is the specific heat capacity of water.

However, the formula is modified a little for customization in order to simplify the calculation. To be more specific, the electrical pulses generated by the water flow meter provide information on the output water flow speed. The measurements and the calculations can be taken along with the unit volume of 1/75 gallon (50.472 ml) of water since one level transition of the output signal of the flow meter indicate a volume of 1/75 gallon of water. Since the water has a trivial specific heat capacity change under the desired temperature range, C_p is the constant value equals to 4.1813 J ·g−1 ·K−1. Another assumption to simplify the calculation is the consistent volume of the water under different temperatures which means 1 ml water weight 1 gram. The temperature T1 and T2 must be presented in unit of Kelvin in the calculation and the conversion is

$$K = (^{\circ} F + 459.67) \times 5 / 9$$

The final customized formula to calculate the net energy of a unit volume of water is shown as following. T1 and T2 is the converted temperature in unit of Kelvin.

$$\Delta H = 211 * (T1 - T2)$$

The total net energy is the accumulation of the net energy in a unit volume presenting below. The unit is in Joule.

$$H = \int_0^v 211 * (T1 - T2) * dv$$

Functional Structure

To implement the desired functions, four major components must be constructed and cooperated. They include the data acquisition, time keeping, data logging and the user interface. Specifically, taking energy conservation into consideration, the LCD is utilized to provide a visual interface to display information. Time is also necessary when writing a new entry so a real time clock is added to keep track of time. SD card is employed to provide storage medium for the log file. The following figure shows a clear view of the design structure.

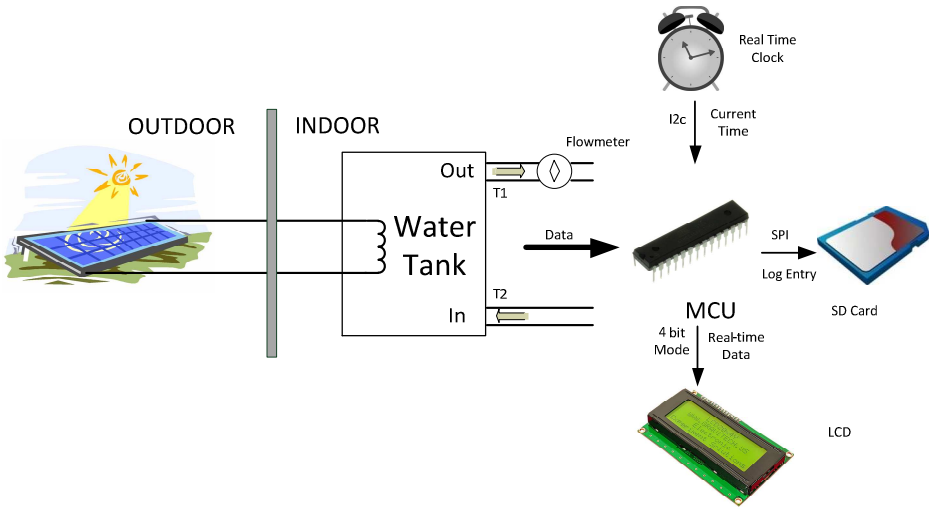

Figure 4 Structural Overview

Hardware/ Software Tradeoffs

One significant tradeoff of this project is the accuracy of tracking the energy flow. The absolute perfect way to measure the net energy of the solar water heater is to measure both the total energy of the water entering and exiting the tank and subtract them for the difference. However, the limitation of having only one flow meter measuring the output water must be considered. Thus, the most optimal solution is to measure the temperature of the water in and out simultaneously and calculates the net energy using the difference of the in-out temperatures.

Software

Software structure

To provide the desired functionalities, the project is comprised of 5 primary functional components.

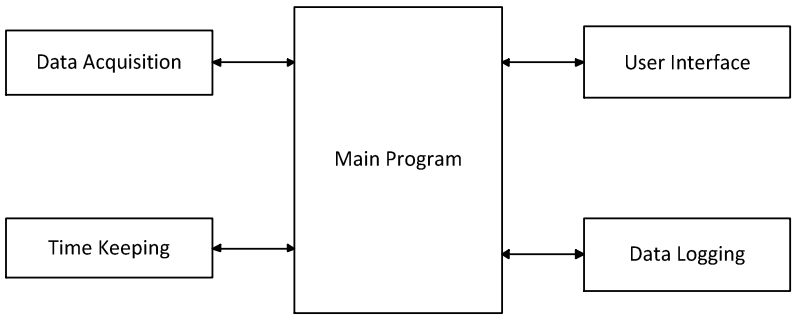

Figure 5 Software Structure

As shown in the figure above, the functional components include the main program, the data acquisition component, the time keeping component, the data logging component and the user interface. The integrity of these parts ensures the functionality of the project.

In the following paragraphs of the section, each component is introduced with detailed implementation.

Main Program

The main program written in the MCU is the brain of the project. It coordinates all the operations and manages the timing schedule. It is in charge of interpreting the ADC conversion result and response to the flow meter triggering signal. It also listens to user input and executes command accordingly. Another very important role of the brain is that it compiles all the information into one log entry and writes it to the SD card. Since the MCU is the brain of the entire system, it always takes the role of master in the communications with real time clock and SD card.

Data Acquisition

Theory

The original signals generated from temperature sensors are analog and must be digitized before any operation in the MCU. One function the energy monitor has is the real-time temperature reading. Thus, the temperature sensors are sampled every 0.5 second. The conversion result stored in the ADC register is in 8-bit mode and the conversion to meaningful reading is shown as following.

$$Temperature = ADC$$

One nice tricky design of to acquire the temperature reading is to take advantage of the MCU internal reference voltage of 2.56V. Since the analog output of the temperature sensor is 10mv/ ˚F and one unit in the ADC result is 2.56/256 = 10mv, the ADC result reading is exactly the actual temperature reading in Fahrenheit.

Actual Implementation

Although the theoretical design is with an easy-to-use reference voltage of 2.56V which can directly present the ADC conversion result to a meaningful reading, the actual electrical world is not as beautiful as what people expected. According to the datasheet, the internal reference voltage in the microcontroller has a 10% error which significantly results in accuracy of the temperature reading. However, in regard to one specific device, the internal reference voltage is consistent over time. Thus, by measuring the analog reference voltage pin on the microcontroller, the actual reference voltage can be found out along with the actual error. The following table shows the measurement result and reference voltage error.

Table 1 ADC Reference Voltage Error

Theoretical Value	Actual Measurement	Error
2.56 V	2.52 V	1.5625%

Fortunately, due to the consistency of the error, the ADC conversion can be calibrated in order to get rid of the reference voltage error. The calibration operation is done by multiplying the calibration coefficient of 1.0159.

$$Temperature_{actual} = ADC * 1.0159$$

Since the energy calculation is based on the unit of Kelvin, all the temperature measurements must be converted in unit of Kelvin. The conversion is done right after a sampling cycle of both temperatures. The following figure shows the processes.

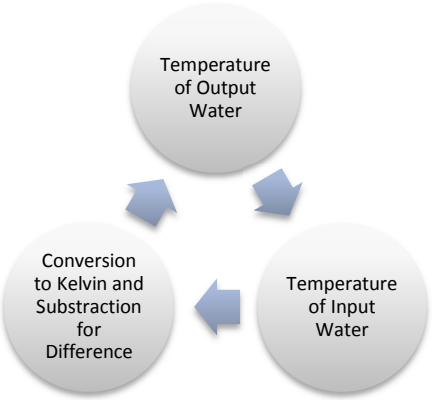

Figure 6 Data Acquisition Processes

User Interface

LCD

The LCD displays all the related information for the system and interacts with human users. There are three display modes which display different content respectively. To switch among screen displays, the users need to press the pushbutton. The three modes are the Clock mode, the Thermometer mode and the Energy meter mode. The software controls the LCD to refresh every 0.25 second with updated information.

In Clock mode, the whole screen is a pure clock which tells the current date and time with accuracy to one second.

In Thermometer mode, the screen displays the real-time temperature of the input and output water in Fahrenheit.

In Energy meter mode, the screen displays the total amount of accumulated net energy, the total output water volume in Gallon and the temperature of the input and output water in Kelvin. Since the user cares about the energy reading most, the energy meter mode is the default mode for the LCD.

Pushbutton

A pushbutton is utilized to switch among different display modes. The pushbutton is debounced using a state machine with a checking time interval of 30ms. The following state diagram shows how the state machine works.

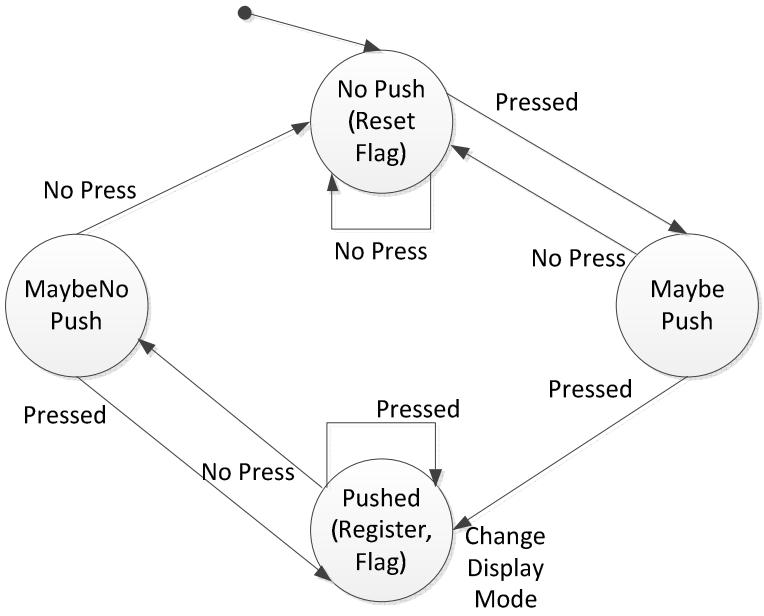

Figure 7 Debounce State Diagram

Data Logging

Logging Entry

One dominant function of an energy monitor is to have a logging function which logs all historical data for future analysis. Since the resolution of the historical data is not very high and taking storage capacity into account, recording one entry every one minute is a good choice. A entry is compiled along with information on time, temperature and energy and written to the SD card. The format will be arranged in the way that a single row represents one sample with the following format.

```
"Date, Time, Temperatures, Water Consumed, Net energy"
```

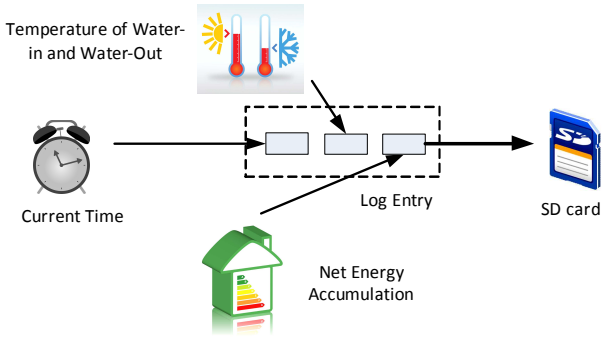

Figure 8 Log Entry

The data logged to the SD card is well-formatted so that the data can be easily imported to MATLAB for data analysis purpose. The data stored on the SD card will never be erased until the space runs out.

Fat File System

Fat file system must be implemented if the log files need to be recognized by a PC system. It actually becomes the most critical aspect in the SD card interface. Fortunately, the Fat file system library for embedded system is already existed for use and can be modified for customization to be fit in. The following diagram shows the structure of the Fat file system.

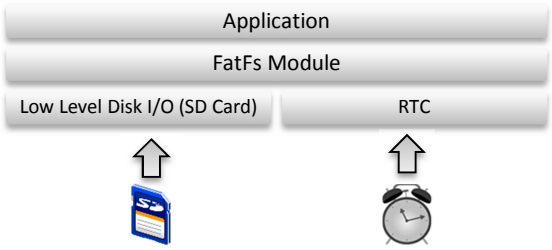

Figure 9 Fat File System Structure

SD Card Operation

There are 5 steps to do when adding a new entry to the SD card. Each step is must be completed prior to the next step. Otherwise, the writing operation terminated and the logging action fails. The steps are shown in the following figure.

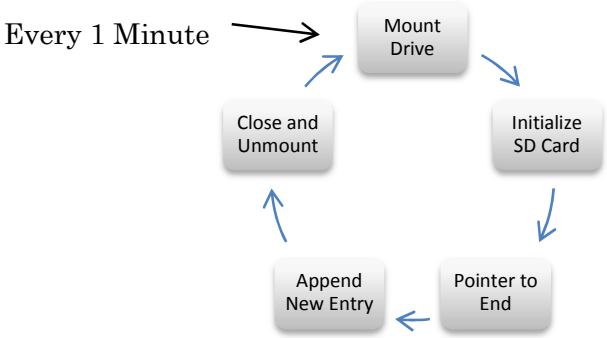

Figure 10 SD Logging Processes

Time Keeping

Time keeping is very critical to this project, because the project is supposed to be running on a nonstop basis. The clock must obtain the correct information on the current time for the data logging purpose. The need of an independent external clock which keeps the time even when the power is off emerges. The solution is to use a real time clock (RTC) which is supplied by a back-up battery. The RTC provides two significant features for the project. First of all, it maintains the time very accurately and makes the current time easy to obtain by sending certain command to the RTC. Second, the RTC relieve the MCU by taking the responsibly of keeping time so that the microcontroller can have more resources to process the data. The communication interface between the MCU and the RTC is the two wire interface (also known as I^2C which is already embedded on the MCU. The software sends specific command to the RTC to acquire the time information. To be more specific, the registers stores the time information and accessible for the MCU to read value from. The RTC is accessed every 0.6 second to get the current time. The following processes show a complete cycle of how the MCU obtains a complete set of time information from the RTC.

Figure 11 Time Reading Processes

Water Meter

Since the output of the flow meter toggles every 1/75 gallon of water, it can be easy to measure the accumulated volume of the water and functions as a digital water meter. Along with the logging function, the water usage at any specific time over the day can be easily found out. Every time the system enters the pin change interrupt, the water usage increases and eventually present the accumulated water usage reading in unit of gallon.

RTC Initial Configuration

The RTC module has an on-board battery which enables the module to memorize the calendar until year 2100. However, the current time must be accurately programmed in the registers. To program the RTC module, multiple register writing operations must be done to write the current time in. First, send the writing command to the RTC to be in programming mode. Second, move the pointer to the desired register. Finally, write the current time value to the register. One efficient way to program the current time accurately is to run the configuration program at the exact time which written in the configuration program.

Status Indicator

A red LED light is installed to provide two major functions. First of all, as part of the water meter, the LED flashes every 1/15 gallon of water which enables the users to have a visual

feeling of water speed. Second, the LED also indicates the logging status. If anything goes wrong during the logging process, the LED is one and manual reset must be conducted.

High Level Time Scheduling

People care about their energy usage on daily basis. Thus, the accumulated net energy resets when a new day starts. The SD card log entry is also added on a periodic basis of 1 minute. Taking advantage of the time registers in the program, the following actions are executed under different circumstances.

Table 2 High Level Timing Schedule

Register	Condition	Action
Second	Equal 30	Write A New Entry
Hour, Minute, Second	All Equal 0	Reset Net Energy to 0

Log Data Analysis

An MATLAB program is specifically written to analyze the log data. It provides a very users friendly visual interface to present data in a meaningful and understandable way. The program does operation in primarily two steps. The user selects the driver first and the program loads the log file and parses it into memory. Then the data is analyzed and plotted in the figures. Further customized statistical analysis can be easily done with the easy-to-use MATLAB function with little effort.

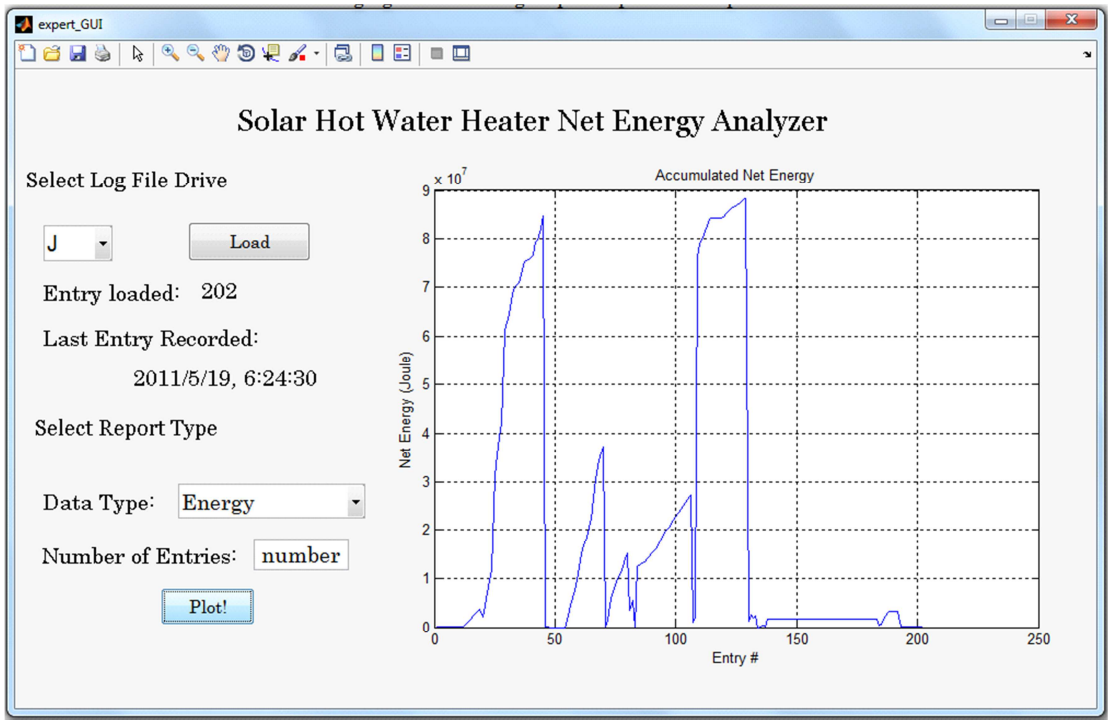

Figure 12 MATLAB Data Analyzer

Hardware

Hardware structure

The following figure shows an overview of the hardware structure.

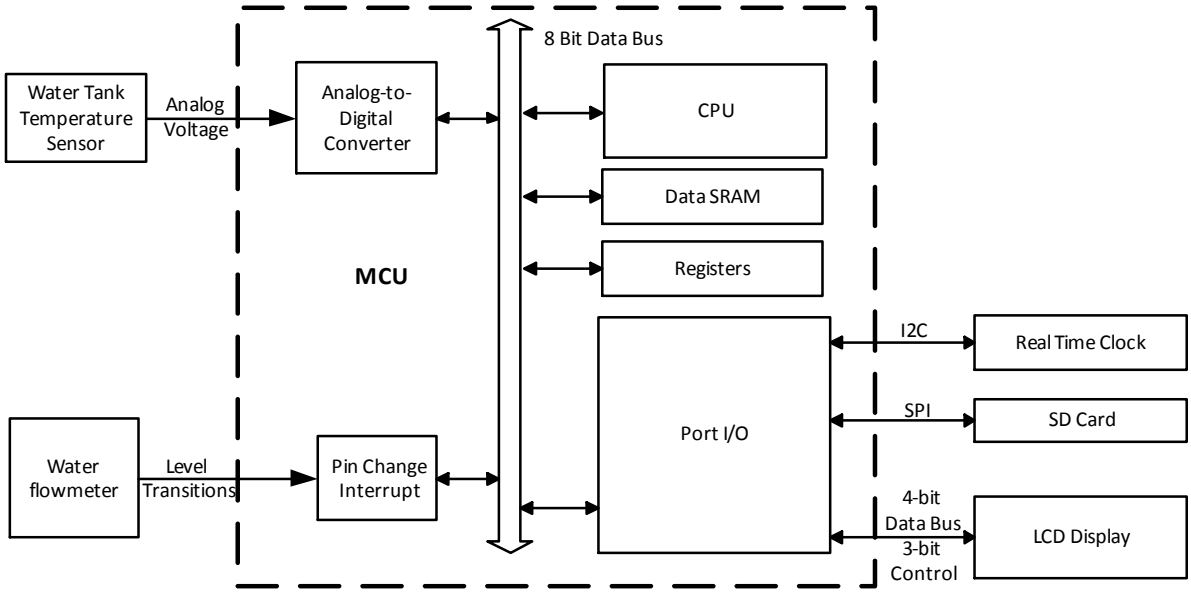

Figure 13 Project Hardware Structure

The following picture clearly indicates the hardware components giving a better understanding of the hardware.

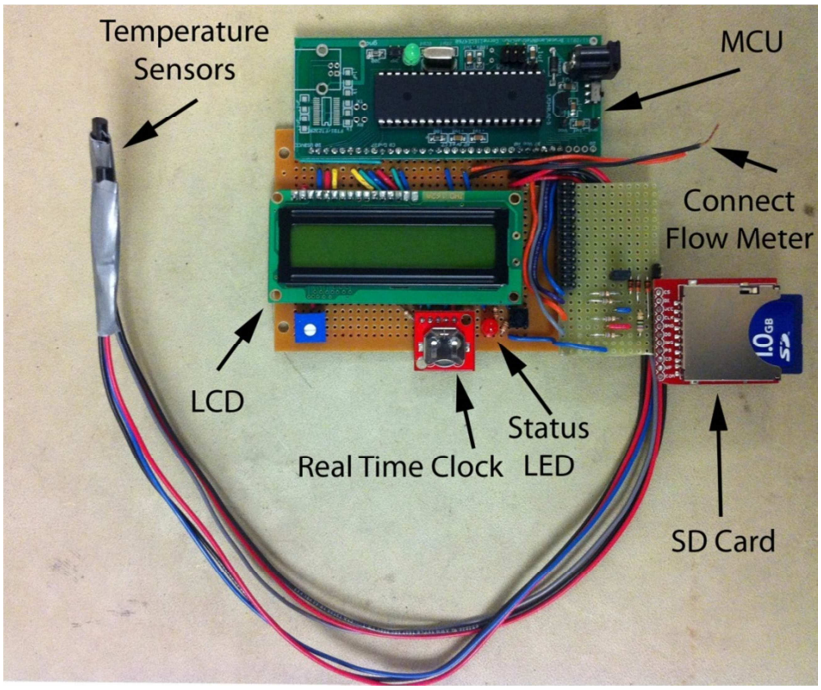

Figure 14 Hardware Overview

Microcontroller

The microcontroller used in the project is the Mega 644 with 64K Byte in-System Programmable Flash from ATMEL. It takes the role of the main brain in the project. This microcontroller has four ports which accommodate sufficient physical connection for peripherals. In addition, it has an 8 channel 10-bit A/D converter. The developing platform is based on the STK500 prototyping board, but the final end product utilizes the target board designed for ECE 4760 class. The following pictures show the actual board and the PCB board drawing.

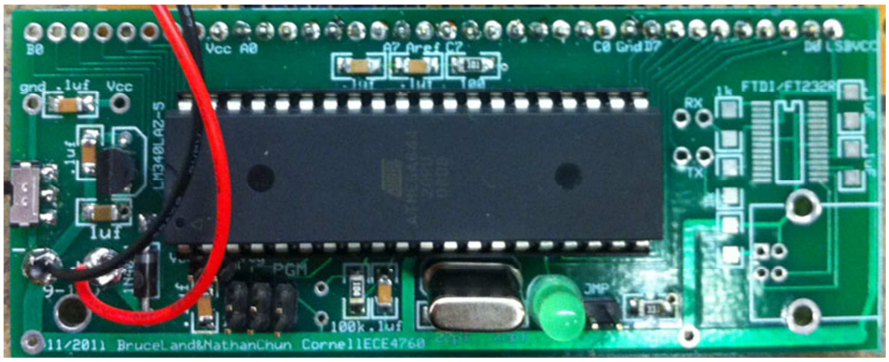

Figure 15 Target Board

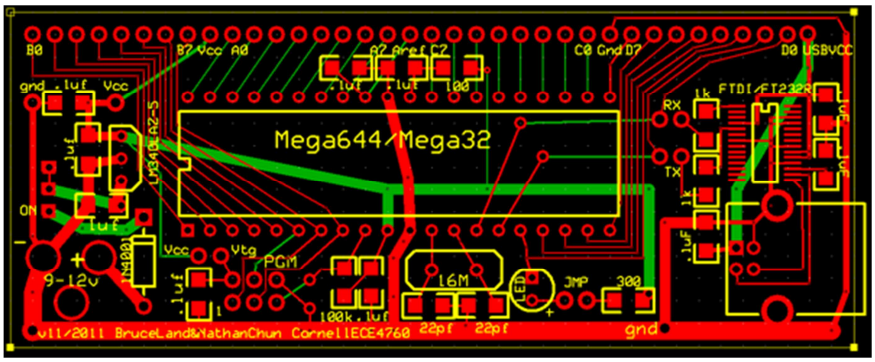

Figure 16 Target Board PCB Drawing

Temperature sensor

The project uses two LM34 temperature sensors to detect the water temperatures. The reason to choose this sensor is because it provides a nice accuracy and the output voltage is formatted along with the temperature change in unit of Fahrenheit. The output voltage from the sensor is fed into Channel 1 and Channel 2 of the ADC on the MCU. The reference voltage is chosen to be the internal one of 2.56V.

According to the datasheet, the output voltage of the temperature sensor is

$$Vout = 10.0mv/^{\circ} F$$

As shown in the picture.

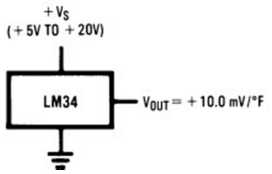

Figure 17 Temperature Sensor Output

LCD display

A 2*16 LCD is employed to display the data reading for the users. The data communication between the MCU and the LCD display is in 4-bit. Three additional control lines are added for command sending.

Figure 18 2*16 LCD

SD card

A 1G SD card is used to store the log data. The card slot provide interface between the SD card and the MCU. The communication between SD card and the MCU is the SPI.

Figure 19 SD Card & Slot

Real Time Clock (RTC)

The real time clock comes along with a back-up battery which can run the module for a minimum of 9 years (17 years typical) without external 5V power. However, during most of the time, it powered up along with the MCU. In other words, the lifetime can be even further extended. This module has calendar stored in the memory to year 2100 which allows the user do nothing for the time setting when year transition happens. The communication interface it uses to talk to the MCU is the two wire interface (also known as I^2C).

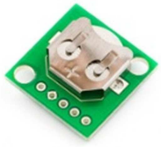

Figure 20 Real Time Clock

Circuit schematics

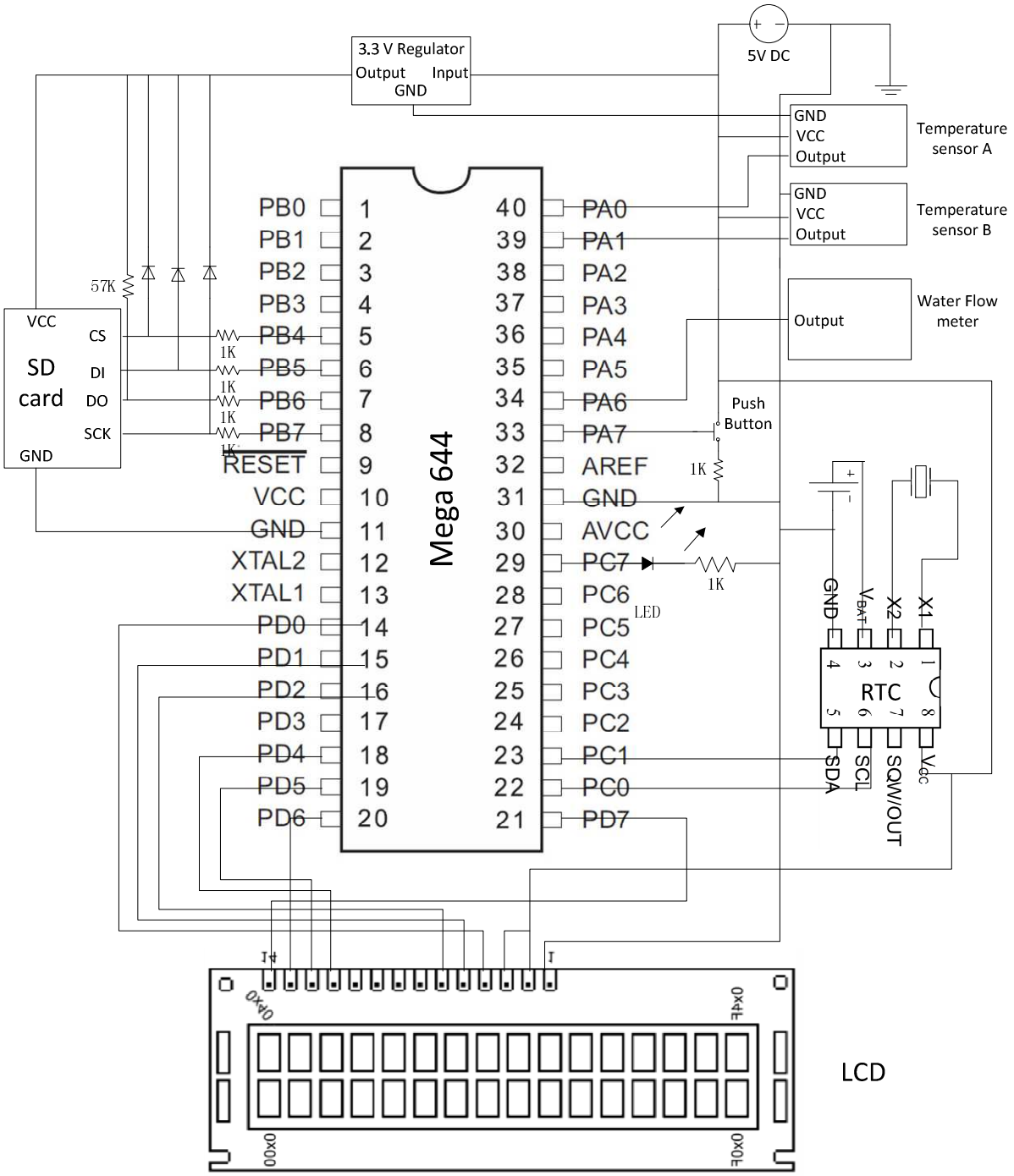

Figure 21 Circuit Schematics

Result

Accuracy

The project turns out to be very effective in terms of the energy calculation and data logging. The accuracy it achieves is far more enough for common users just to under the performance of the solar water heater and the amount of energy consumed every day. Due to the fact that the project is only for home use and budget is limited, extreme accuracy is not essential to pursue. The following study shows how the accuracy is and gives a simple idea of the decent design tradeoffs made.

Time Keeping

The real time clock module has a DS1307 chip on board with a crystal as the clock. According to the datasheet, the accuracy of the clock is dependent upon the accuracy of the crystal and the accuracy of the match between the capacitive load of the oscillator circuit and the capacitive load for which the crystal was trimmed. However, the detailed information on the crystal is unknown. From the observation and tests conducts on the RTC module, the module has an approximate 3 minutes' delay after continuously running for a month. This can lead to a further conclusion that the delay is about half an hour after a continuous run of a year. The following table shows a specific delay time after a period of continuous operation.

Table 3 Real Time Clock Accuracy Analysis

Continuous Operation Time	Error
1 Month*	3 Minutes' Slow
3 Month	9 Minutes' Slow
1 Year	36 Minutes' Slow
4 Year	2.4 Hours' Slow
Actual Test Result*	

Energy Calculation

The tradeoff is made to simplify the energy calculation and relieve the pressure on the MCU resource. In the previous section, the final customized energy calculation formula is shown as below with T1 and T2 in unit of Kelvin.

$$\Delta H = 211 * (T1 - T2)$$

The world is not perfect, but the constant coefficient is equal to an approximated value of 211. The specific heat capacity of water C_p is defined as a constant and equals to 4.1813 J/(g*K). The first accuracy analysis is under the assumption that the 1 ml water weight exact 1g. The following equations show how the constant coefficient 211 comes from.

$$\frac{1}{75} Gallon = 50.5\ ml$$
$$50.4721571\ ml * 1 = 50.5 g$$
$$50.4721571\ g * \frac{4.1813\text{J}}{(\text{g} * \text{K})} = 211.0 \frac{J}{K}$$

From the calculation shown above, trivial calculation error is introduced to find the net energy. However, the previous calculation is done under the assumption that 1 ml water weight exact 1g. Unfortunately, this is not perfectly true in reality, taking the temperature and the volume-weight conversion into consideration, the following table shows the accuracy analysis result.

Table 4 Energy Calculation Accuracy Analysis

Temperature(˚C)	Water Density(g/ml)	True Constant Value	Error
5	0.9970479	210.4162216	-0.277%
20	0.9982071	210.6608582	-0.161%
30	0.9922187	209.3970709	-0.765%
50	0.9880393	208.5150536	-1.192%
80	0.9718007	205.0880719	-2.883%
90	0.965323	203.7210231	-3.573%

As shown in the table above, the error becomes larger than the one under the constant water density assumption. However, even at 90 degree, the error is well under 5% which is completely acceptable for a home-use device.

The accuracy tests show that the project satisfies the accuracy requirement.

Valid Range

Variable Types

The project is designed for non-stop operation. The accumulated net energy is the biggest concern since it may overflow if improper numeric type is assigned to the variable. As stated in the previous design section, the net energy accumulation reset itself every 24 hours. Thus, at least it must ensure the variable would never overflow during a day. The variable which stores the accumulated net energy has a numeric type of unsigned long. The following calculation shows the maximum volumes of water it may tolerate under worst case scenario. Assuming the temperature difference is always as large as 80 K (90 ˚C to 10 ˚C).

$$Unsign\ Long\ Type\ Range: 4{,}294{,}967{,}295$$
$$Tolerated\ Number\ of\ Unit\ Volume: \frac{4294967295}{80 * 211} = 2544411.19\ \left(\frac{1}{75} Gallon\right)$$
$$Total\ Tolerated\ Volume: 2544411.19 * \frac{1}{75} = 3392.55\ Gallon$$

The calculation shows above illustrate a fact that the variable would fail if and only if the solar heat water output more than 3392.55 gallons of 90 ˚C (194 ˚K) hot water. The interesting following calculation gives a simple idea whether it is possible.

$$Required\ Solar\ Heater\ Output\ Power\ (assuming\ work\ full\ 12\ hours\ a\ day): 99420\ W$$
$$Continuous\ Running\ Water\ Speed(assuming\ running\ for\ 12\ hours\ a\ day): 4.7\ gallons/min$$
$$Maximum\ Water\ Heater\ Tank\ Turnovers\ in\ a\ day\ (80\ gallon\ size): 42.4\ Times$$

It can be easily concluded that it is very impossible that the overflow would happen during a single day even under the worst case scenario.

Storage Capacity

The project employs a 1G SD card for logging function. The storage size is chosen in consideration of the storage price and availability. The following calculation proves that the storage size is appropriate and sufficient for the project.

$$Bytes\ in\ 1G: 1024 * 1024 * 1024 = 1073741824\ bytes$$
$$Conservative\ approximation\ of\ an\ average\ entry\ size: 60\ bytes$$
$$Number\ of\ entries\ possiblely\ made: \frac{1073741824}{60} = 17895697$$
$$Sustainable\ Time\ Period\ in\ Hours: \frac{17895697}{60} = 298\,262\ Hours$$
$$Sustainable\ Time\ Period\ in\ Days: \frac{298262}{24} = 12428\ Days$$
$$Sustainable\ Time\ Period\ in\ Years: \frac{12428}{365} = 34\ Years$$

The conclusion can be clearly made that 34 years continuous storage pace is completely sufficient for an ordinary home-use device. A further solid conclusion can be made that the project is safe and stable to run on nonstop basis.

Conclusion

Summary

The project provide an intuitive and convenient solution to create a visual interface for a solar heater to display related energy information for the users and enables them to have a better understanding of the machine's performance and the daily household energy usage on the hot water. Additionally, by implementing the solution, the usage of the water in volume can also be known which can arouse people's attention to save the water resource. The outcome of the project agrees with the design and the requirements. The accuracy and valid rang is sufficient to operate under nonstop use without any extra manual maintenance.

Further Improvements

The project creates an energy monitor solution for a solar water heater. However, based on this foundation, it can be easily enhanced with many other functions with little effort on modification. The following figure shows a glimpse of possible improvements.

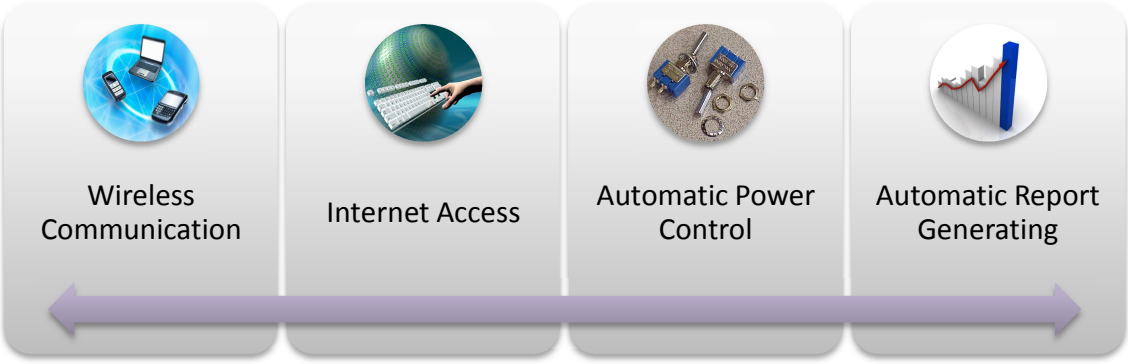

Figure 22 Further Improvements

First of all, the wireless communication can be utilized to replace the SD card to provide a more convenient way to log and store the data. The real-time logging can be easier if wireless communication is established. Second, if the Internet access is enabled, the energy information can be better taken care of by the online services and shared in a more rapid pace and larger scale. In addition, it can have auto-diagnosis function which enables itself to operate under minimum power usage. Finally, it can be further improved to have automatic report generating function which sends out a short text message or email to the users at the end of the day to let them know the relation energy information over the past day. All these ideas can be achieved without significant amount of efforts and fund.

Past Achievements & Milestones

The project progresses along with the ECE 4760 Design with Microcontroller class because lots of the necessary knowledge is learned from the class and achievements are made along the way. However, a significant amount of effort and time is spent on the end of the semester for SD card interfacing and final integrations. Milestones are shown as following.

- Milestone 1: Basic Structure Constructed (Mid-March)
- Milestone 2: RTC Communicating and Data Collection (Mid-April)
- Milestone 3: SD card interfacing and MATLAB data analyzer (Mid-May)

Development Experiences

The project involves significant amount of effort and time over the semester. Although actual project construction is majorly done during the second half of the semester, the first half is extremely important because it provide the essential knowledge base which leads the possibility of the completion of the project.

One major frustration of the entire project comes on the SD card interface. Due to the requirement of being recognizable by a PC for the log file, the Fat file system must be implemented. This requires a lot of internal operation on the MCU before an entry is added to the SD card. By taking advantage of the FatFs library, the writing operation can be successfully done. However, with the completion of the software issue, new hardware problem emerges due to the fact that the SD card is very sensitive to the operating voltage. With days of endless working, the problems are finally solved and the project reaches the final success.

Reference

Datasheet

- ATMEL Mega644 Microcontroller Datasheet
  http://www.atmel.com/dyn/resources/prod_documents/doc2593.pdf
- Real Time Clock Module with DS1307 Datasheet
  http://www.sparkfun.com/datasheets/Components/DS1307.pdf

Project

- Embedded Foot Pronation Detection
  http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2011/ylw3/webpage/index.html
- MCU developer: Fat File System Library
  http://www.basementcode.com/
- GPS Data Logger with Wireless Trigger
  http://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2009/jsm66_mpk28/jsm66_mpk28/index.html

Appendix

Cost and budget

The following table shows an incomplete list of the parts used in the project along with its cost.

Table 5 Project Budget

Device	Quantity	Price	Cost
Breakout Board for SD	1	9.95	9.95
Mega644	1	8	8
Solder Board	1	2.5	2.5
DIP Socket	2	0.5	1
Real Time Clock Module	1	19.95	19.95
Temperature Sensor - LM34	2	From Lab	0
Diodes	3	From Lab	0
SD card	1	Donated	0
		Total	41.4

Codes

Major codes written for the project are listed here. Due to the large size of the supporting library, the complete codes can be found in the zipped code package.

MCU Main Program

```
/*****************************************************************************
       This program is the main program serves the energy monitor M.Eng project
       There are three major components including RTC, SD card, Temperature
       Author: Xun Yang      ECE Cornell
       This version only involves RTC and Temperature. The LCD displays the
       temperature and current local time information with pushbutton
       switch the displays
       Connection:           RTC:    SCL(PC.0)
                                             SDA(PC.1)
                                     LCD:    PORTD(except D.4)
                                     SD:             PORTB(7,6,5,4)
                                     ADC:    ADC0 (PA.0)
                                             ADC1 (PA.1)
                                     SD:             CLK (PB.7)
                                                     DO      (PB.6)
                                                     DI      (PB.5)
                                                     CS      (PB.4)
                                     Pushbutton: PA.7
                                     Flowmeter:      PA.6
       ADC:
       Reference Voltage: Internal 2.56v
       Output format: 8 bit (0~256)
       NOTE -- MUST UNMOUNT the Aref jumper
       RTC:
       The output of the RTC include sec,min,day,date,month,year
       The Time information is capture every 1 second
*****************************************************************************/
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/pgmspace.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stddef.h>
```

```
#include <util/delay.h>


//LCD function enabled
#include "lcd_lib.h"

//TWI interface enable
#include "i2cmaster.h"

//Include SD card interface library
#include "fatfs/ff.h"
#include "fatfs/diskio.h"

//set up ASSERT
#ifndef __ASSERT_USE_STDERR
#define __ASSERT_USE_STDERR
#include <assert.h>
#endif

//Timing control constants
#define RTCinterval 60 //unit in 10 ms
#define ADCinterval   50 //unit in 10 ms
#define LCDinterval   25 //unit in 10 ms
#define PUSHinterval 3 //unit in 10 ms
#define LEDinterval 200 //unit in 10 ms

//State machine state names
#define NoPush 1
#define MaybePush 2
#define Pushed 3
#define MaybeNoPush 4

//******Function declaration**********
void MCUInitialization(void);
//SD card functions
void SDInitialization(void);
void SDAddNewEntry(void);
void SDClose(void);
//RTC functions
void RTCreset(void);
void RTCread(void);
unsigned char bcd2dec(unsigned char data);

//FAT System Structure set-up
FATFS FileSystemObject;
FIL logFile;
unsigned int bytesWritten;    //Number of bytes of the written data
unsigned char SDcounter;      //Write new entry once a minute

//8-bit mode are used in ADC output
//Interpreted value in Fahrenheit
unsigned char tempin;
unsigned char tempout;

//Temperature information display
unsigned char tempindisp[16] ; //"68F, 129K"
unsigned char tempoutdisp[16] ;

//Time information display
unsigned char datedisp[16] ;
unsigned char clockdisp[16];

//Energy Information Display
unsigned char energydisp[16];
unsigned char gallondisp[16];
unsigned char tempdiffdisp[6];

//Screen Mode
// 0--Temperature info; 1--Time info
unsigned char LCDmode;
```

```
//Pushbutton debouncer variable
unsigned char PushState;      //state machine

//RTC status
unsigned char error; //0 indicate no error in the TWI

//RTC reading
unsigned char second;
unsigned char minute;
unsigned char hour;
unsigned char weekday;
unsigned char date;
unsigned char month;
unsigned char year;
unsigned char WeekDayDisp[4];

//Global Timing Control Variables
volatile unsigned char RTCTimer; //timer counter for updating RTC
volatile unsigned char ADCTimer; //timer counter for updating ADC
volatile unsigned char LCDTimer; //timer counter for updating LCD
volatile unsigned char PUSHTimer; //timer counter for updating Pushbutton

//Energy variable
volatile unsigned long NetEnergy;     //Stored in jour
volatile unsigned long GallonUsed;    //True Gallon/75
volatile unsigned char GallonInterval;       //Count Flow Interrupt
volatile unsigned char LEDTimer;             //LED Timer and Turn Off when Stops

//Temperature value in Kevin
volatile unsigned char tempdiffK;

//*********************************************************
//timer 0 compare ISR 10ms tick
//Called to decrement the Timer1 and Timer2 in diskio
ISR (TIMER0_COMPA_vect)
{
        //FATsystem Timing Control
        disk_timerproc();
        //Update the timer for RTC
        if (RTCTimer>0)        --RTCTimer;
        //Update the timer for ADC
        if (ADCTimer>0)        --ADCTimer;
        //Update the timer for LCD
        if (LCDTimer>0)        --LCDTimer;
        //Update the timer for Pushbutton
        if (PUSHTimer>0)       --PUSHTimer;
        //Update the LED timer
        if (LEDTimer >0) --LEDTimer;
}


//External interrupt 1
//Called to intergral energy
ISR (PCINT0_vect)
{
        NetEnergy = NetEnergy+211*tempdiffK;
        GallonUsed++;
        //LED flashed every 1/15 gallon = 252.3 ml
        if (GallonInterval < 5)
        {
                GallonInterval++;
        }
        else
        {
                //Toggle every 1/15 gallon
                PORTC ^= 0b10000000;
                GallonInterval = 0;
                //Initialize LED timer;
                LEDTimer = LEDinterval;
        }
}
```

```
int main(void)
{
	//Initialize System
	MCUInitialization();

	// measure and display loop
	while (1)
	{
		/*****************************************************************************
			Update RTC information every 1s. Store information in variables
		*****************************************************************************/
		if (0==RTCTimer)
		{
			//clear RTC pointer to 0
			RTCreset();

			//Get time information
			RTCread();

			//Data manipulation
			//No conversion needed for weekday
			second = bcd2dec(second);
			minute = bcd2dec(minute);
			hour = bcd2dec(hour);
			date = bcd2dec(date);
			month = bcd2dec(month);
			year = bcd2dec(year);

			//Ready for LCD printing
			sprintf(datedisp,"   20%02d/%02d/%02d",year,month,date);
			sprintf(clockdisp,"%02d:%02d:%02d",hour,minute,second);
			//Find weekday and ready for display
			switch(weekday)
			{
			    case 1:
			        strcpy(WeekDayDisp, "MON");
					break;
			    case 2:
			        strcpy(WeekDayDisp, "TUE");
					break;
			    case 3:
			        strcpy(WeekDayDisp, "WED");
					break;
				case 4:
				    strcpy(WeekDayDisp, "THU");
					break;
				case 5:
				    strcpy(WeekDayDisp, "FRI");
					break;
				case 6:
				    strcpy(WeekDayDisp, "SAT");
					break;
				case 7:
				    strcpy(WeekDayDisp, "SUN");
					break;
				default:
					strcpy(WeekDayDisp, "UNK");
			}

			//reset the task timer
			RTCTimer = RTCinterval;
		}

		/*****************************************************************************
			Update ADC information every 500ms. Information stored in variables
			ADC involves calibration of reference voltage from 2.56V to 2.52V
		*****************************************************************************/
		if (0==ADCTimer)
```

```
{
	//*******get the sample from channel 0************
	tempin= ADCH;
	//Switch source to channel 1
	ADMUX += 1;
	//start another conversion
	ADCSRA |= (1<<ADSC) ;
	//Calibration
	tempin= (int)((float)(tempin)*1.0159); //Reading is already in F
	//Convert result to string and display
	sprintf(tempindisp,"IN:%3dF",tempin);

	//Convert result to string and display
	sprintf(energydisp,"NE:%ldJ",NetEnergy);



	//*****repeat the same process for channel 1*******
	tempout= ADCH;
	//Switch source to channel 0
	ADMUX -= 1;
	//start another conversion
	ADCSRA |= (1<<ADSC) ;
	//Calibration
	tempout= (int)((float)(tempout)*1.0159); //Reading is already in F

	//Calculate the temperature difference
	if (tempout > tempin)
		tempdiffK = (int)((float)(tempout-tempin)*0.555556);
	else
		tempdiffK = 0;

	//Convert result to string and display
	sprintf(tempoutdisp,"OUT:%3dF",tempout);
	//sprintf(gallondisp,"V:%dGal D:%dK",GallonUsed/75,tempdiffK);
	sprintf(gallondisp,"V:%dGal",GallonUsed/75);
	sprintf(tempdiffdisp,"D:%2dK",tempdiffK);

	//Reset Timer control
	ADCTimer = ADCinterval;
}

/*************************************************************************
	Update LCD screen every 250 ms. Mode selected by the pushbutton
*************************************************************************/
if (0==LCDTimer)
{
	//Display Temperature information in mode 0
	if(0 == LCDmode)
	{
		//Update Temp 1
		LCDGotoXY(0, 0);
		LCDstring(tempindisp, strlen(tempindisp));
		//Update Temp 2
		LCDGotoXY(0, 1);
		LCDstring(tempoutdisp, strlen(tempoutdisp));
	}


	//Display time information in mode 1
	else if(1 == LCDmode)
	{
		//Update Calendar
		LCDGotoXY(0, 0);
		LCDstring(datedisp, strlen(datedisp));
		//Update WeekDay
		LCDGotoXY(1, 1);
		LCDstring(WeekDayDisp, strlen(WeekDayDisp));
		//Update Clock
		LCDGotoXY(7, 1);
		LCDstring(clockdisp, strlen(clockdisp));
```

```
        }
        else
        {
                //Update Net Energy Information
                LCDGotoXY(0,0);
                LCDstring(energydisp, strlen(energydisp));
                //Update Water Usage Information
                LCDGotoXY(0,1);
                LCDstring(gallondisp, strlen(gallondisp));
                LCDGotoXY(11,1);
                LCDstring(tempdiffdisp, strlen(tempdiffdisp));
        }
        //Reset Timer control
        LCDTimer = LCDinterval;
}


/************************************************************************
        Debounce Pushbutton by checking state every 30ms.
************************************************************************/
if (0==PUSHTimer)
{
        //State machine debounce the pushbutton
        switch (PushState)
        {
                case NoPush:
                        if (PINA & 0x80) PushState=MaybePush;
                        else PushState=NoPush;
                        break;
                case MaybePush:
                        if (PINA & 0x80)
                        {
                                //Updating LCD screen mode
                                if(LCDmode<2)
                                        LCDmode++;
                                else
                                        LCDmode=0;
                                LCDclr(); //clear previous screen
                                PushState=Pushed;
                        }
                        else PushState=NoPush;
                        break;
                case Pushed:
                        if (PINA & 0x80) PushState=Pushed;
                        else    PushState=MaybeNoPush;
                        break;
                case MaybeNoPush:
                        if (PINA & 0x80) PushState=Pushed;
                        else
                        {
                           PushState=NoPush;
                        }
                        break;
        }

        //Reset Timer control
        PUSHTimer = PUSHinterval;
}

/************************************************************************
        Debounce Pushbutton by checking state every 30ms.
************************************************************************/
if (0 == LEDTimer )
{
        //Turn LED off when Time expires
        PORTC |= 0b10000000;
}

/************************************************************************
        Add New Entry when second register equel to 30
        equivalent to 1 minute per entry
```

```
                ***************************************************************************/
                if ((30 == second) && SDcounter)
                {
                        //SD Card Operations
                        SDInitialization();
                        SDAddNewEntry();
                        SDClose();
                        SDcounter = 0;
                }
                else if (32 == second)
                {
                        SDcounter = 1;
                }

                /***************************************************************************
                        Clear Accumulated Energy Information when a new day starts
                ***************************************************************************/
                if ((0 == second) && (0 == minute) && (0 == hour))
                {
                        NetEnergy = 0;
                        GallonUsed = 0;
                        LCDclr(); //Clear Obselete Screen
                }

        }
}


void MCUInitialization(void)
{
        //Start-up LCD
        LCDinit();      //initialize the display
        LCDcursorOFF();
        LCDclr();       //clear the display

        //initilize LCD display mode
        LCDmode = 2; //Energy Meter Display

        //set up timer 0 for 10 mSec ticks
        TIMSK0 |= 1 << OCIE0A;  /* enable interrupt for timer match a */
        OCR0A = 156;  /* 10 ms interrupt at 16MHz */
        TCCR0B |= (1 << CS02) | (1 << CS00);  //speed = F_CPU/1024

        //Initialize External Interrupt
        PCICR = 1<<PCIE0; // turn on pin change interrupt 0
        PCMSK0 = 1<<PCINT6 ;       // PA.6 configured as pin change interrupt.

        //LED Port Initilization
        //Red LED for Logging Status
        DDRC |= 1<<DDC7;
        PORTC |= 0b10000000;

        //init the controlling timers
        RTCTimer = RTCinterval;
        ADCTimer = ADCinterval;
        LCDTimer = LCDinterval;
        PUSHTimer = PUSHinterval;
        LEDTimer = LEDinterval;

        //init the state machine
        PushState = NoPush;

        // init the TWI communication
        //I2C frequency 80K Hz
        i2c_init();

        //init the A to D converter
        //right adj /Internal Aref 2.56V/Channel 0
        ADMUX = ((1<<ADLAR) | (1<<REFS1) | (1<<REFS0)) ;

    //enable ADC and set prescaler to 1/128*16MHz=125,000
```

```
        //and clear interupt enable
        ADCSRA =((1<<ADEN) | (1<<ADSC)) + 7 ;

        //Initialize Energy Meter
        GallonUsed = 0;
        NetEnergy = 0;
        tempdiffK = 0;

        //SD Card counter
        SDcounter = 1;

        sei();
}


/*--------------Functions Written to Support SD card---------*/
void SDInitialization(void)
{
        if(f_mount(0, &FileSystemObject)!=FR_OK)
        {
          //Status LED ON Indicating Error
          PORTC &= 0b01111111;
        }

        DSTATUS driveStatus = disk_initialize(0);

        if(driveStatus & STA_NOINIT ||
           driveStatus & STA_NODISK ||
           driveStatus & STA_PROTECT
           )
        {
          //Status LED ON Indicating Error
          PORTC &= 0b01111111;
        }

        //Open File
        if(f_open(&logFile, "/ENERGY.log", FA_READ | FA_WRITE | FA_OPEN_ALWAYS)!=FR_OK)
        {

          //Status LED ON Indicating Error
          PORTC &= 0b01111111;
        }
}

void SDAddNewEntry(void)
{
        f_lseek(&logFile, f_size(&logFile));
        f_printf(&logFile,
"20%2d/%2d/%2d,%2d:%2d:%2d,%d,%d,%d,%ld,%ld\n",year,month,date,hour,minute,second,weekday,tempin,
tempout,NetEnergy,GallonUsed);
}

void SDClose(void)
{
        //Close and unmount.
        f_close(&logFile);
        f_mount(0,0);
}
/*-----------------------------------------------------------------*/

/*--------------------Functions Written to Support RTC------------*/
//TWI master transmitter
//Clear the RTC pointer to 0
//Call every time prior to reading
void RTCreset(void)
{
        while(i2c_start(0xD0));
        while(i2c_write(0x00));       //Reset pointer to 0
        i2c_stop();
}
```

```
//TWI master receiver
//Read the RTC registers
//Located with RTC internal pointer
void RTCread(void)
{
	while(i2c_start(0xD1));

	//Read follow-up 7 registers
	second = i2c_readAck() & 0x7f; //get rid of first control bit
	minute = i2c_readAck();
	hour = i2c_readAck() & 0x3f; //get rid of format register bits
	weekday = i2c_readAck();
	date = i2c_readAck();
	month = i2c_readAck();
	year = i2c_readNak();  //Follow up with a stop
	i2c_stop();
}

//Data display conversion function
//Convert data in BCD to Decimal
unsigned char bcd2dec(unsigned char data)
{
	unsigned char tens;
	unsigned char ones;

	tens = data>>4;
	ones = data & 0x0f;
	data = tens*10 + ones;
	return(data);
}
/*-------------------------------------------------------------------*/
```

Real Time Clock Configuration

```
/*************************************************************************
	This code is designed to communicate with the RTC
	and set up the internal registers to current time
	The interface will be I2C
	Setting:		MCU	----	RTC
					PC.0	----	SCL
					PC.1	----	SDA
	UART: PD.0(Rx) & PD.1(Tx)
	The output of the RTC include sec,min,day,date,month,year
	The Time information is capture every 1 second
 *************************************************************************/

#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>

#include "uart.h"
#include "i2cmaster.h"

// UART file descriptor
// putchar and getchar are in uart.c
FILE uart_str = FDEV_SETUP_STREAM(uart_putchar, uart_getchar, _FDEV_SETUP_RW);

//Timing control constants
#define interval 1000  //unit in ms


//TWI status definition
#define START 0x08
#define ADDRESSW 0x18
#define DATASENT 0x28
#define ADDRESSR 0x40

volatile unsigned int time; //timer counter
```

```
//RTC status
unsigned char error; //0 indicate no error in the TWI

//RTC reading
unsigned char second;
unsigned char minute;
unsigned char hour;
unsigned char weekday;
unsigned char date;
unsigned char month;
unsigned char year;


//TWI master transmitter
//Clear the RTC pointer
void RTCreset(void)
{
	//unsigned char status;

	while(i2c_start(0xD0));
	while(i2c_write(0x00));	//Reset pointer to 0
	i2c_stop();
}

void RTCprog(void)
{
	//Set the local current time
	while(i2c_start(0xD0));
	while(i2c_write(0x00));	//Reset pointer to 0

	//Program follow-up 7 registers
	while(i2c_write(0x00));	//Second 00
	while(i2c_write(0x27));	//Minute 27
	while(i2c_write(0x18));	//Hours 18
	while(i2c_write(0x03));	//day 3
	while(i2c_write(0x18));	//Date 18
	while(i2c_write(0x05));	//Month 05
	while(i2c_write(0x11));	//Year 2011

	i2c_stop();
}

//TWI master receiver
//Read the RTC registers
//Located with RTC internal pointer
void RTCread(void)
{
	//unsigned char status;

	while(i2c_start(0xD1));

	//Read follow-up 7 registers
	second = i2c_readAck() & 0x7f; //get rid of first control bit
	minute = i2c_readAck();
	hour = i2c_readAck() & 0x3f; //get rid of format register bits
	weekday = i2c_readAck();
	date = i2c_readAck();
	month = i2c_readAck();
	year = i2c_readNak();  //Follow up with a stop
	i2c_stop();

}

//Data display conversion function
//Convert data in BCD to Decimal
unsigned char bcd2dec(unsigned char data)
{
	unsigned char tens;
	unsigned char ones;

	tens = data>>4;
```

```
        ones = data & 0x0f;
        data = tens*10 + ones;

        return(data);
}


//timer 0 compare ISR
ISR (TIMER0_COMPA_vect)
{
        //Update the task time
        if (time>0)    --time;

}

int main(void)
{
        //set up timer 0 for 1 mSec ticks
        TIMSK0 = 2;            //turn on timer 0 cmp match ISR
        OCR0A = 249;   //set the compare reg to 250 time ticks
        TCCR0A = 0b00000010; // turn on clear-on-match
        TCCR0B = 0b00000011;  // clock prescalar to 64

        //init the task timer
        time = interval;

        // init the UART -- uart_init() is in uart.c
        uart_init();
        stdout = stdin = stderr = &uart_str;
        fprintf(stdout,"Starting RTC testing\n\r");

        // init the TWI communication
        //I2C frequency 80K Hz
        i2c_init();

        //Program the RTC to the current time
        //April 13 2011 15:30:00
        RTCprog();

        sei();


        // measure and display loop
        while (1)
        {

                if (0==time)
                {
                        //Toggle LEDs
                        PORTA ^= 0x01;

                        //clear RTC pointer to 0
                        RTCreset();

                        //Get time information
                        RTCread();

                        //Data manipulation
                        //No conversion needed for weekday
                        second = bcd2dec(second);
                        minute = bcd2dec(minute);
                        hour = bcd2dec(hour);
                        date = bcd2dec(date);
                        month = bcd2dec(month);
                        year = bcd2dec(year);


                        //results to hyperterm
                        printf("Current Time: 20%d/%d/%d %d
%d:%d:%d\n\r",year,month,date,weekday,hour,minute,second);
```

```
                        //reset the task timer
                        time = interval;
                }
        }
}
```

Major Supporting Libraries

I²C

```
/*************************************************************************
* Title:    I2C master library using hardware TWI interface
* Author:   Peter Fleury <pfleury@gmx.ch>  http://jump.to/fleury
* File:     $Id: twimaster.c,v 1.3 2005/07/02 11:14:21 Peter Exp $
* Software: AVR-GCC 3.4.3 / avr-libc 1.2.3
* Target:   any AVR device with hardware TWI
* Usage:    API compatible with I2C Software Library i2cmaster.h
*************************************************************************/
#include <inttypes.h>
#include <compat/twi.h>

#include "i2cmaster.h"

/*************************************************************************
 Initialization of the I2C bus interface. Need to be called only once
*************************************************************************/
void i2c_init(void)
{
  /* initialize TWI clock: 100 kHz clock, TWPS = 0 => prescaler = 1 */

  TWSR = 1;                         /* no prescaler */
  TWBR = 152;  /* SCK = 50K hz */

}/* i2c_init */


/*************************************************************************
  Issues a start condition and sends address and transfer direction.
  return 0 = device accessible, 1= failed to access device
*************************************************************************/
unsigned char i2c_start(unsigned char address)
{
    uint8_t   twst;

        // send START condition
        TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) return 1;

        // send device address
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wail until transmission completed and ACK/NACK has been received
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_MT_SLA_ACK) && (twst != TW_MR_SLA_ACK) ) return 1;

        return 0;

}/* i2c_start */


/*************************************************************************
```

```
 Issues a start condition and sends address and transfer direction.
 If device is busy, use ack polling to wait until device is ready

 Input:   address and transfer direction of I2C device
*************************************************************************/
void i2c_start_wait(unsigned char address)
{
    uint8_t   twst;


    while ( 1 )
    {
            // send START condition
            TWCR = (1<<TWINT) | (1<<TWSTA) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst != TW_START) && (twst != TW_REP_START)) continue;

        // send device address
        TWDR = address;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wail until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits.
        twst = TW_STATUS & 0xF8;
        if ( (twst == TW_MT_SLA_NACK )||(twst ==TW_MR_DATA_NACK) )
        {
            /* device busy, send stop condition to terminate write operation */
                TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);

                // wait until stop condition is executed and bus released
                while(TWCR & (1<<TWSTO));

            continue;
        }
        //if( twst != TW_MT_SLA_ACK) return 1;
        break;
     }

}/* i2c_start_wait */


/*************************************************************************
 Issues a repeated start condition and sends address and transfer direction

 Input:   address and transfer direction of I2C device

 Return:  0 device accessible
          1 failed to access device
*************************************************************************/
unsigned char i2c_rep_start(unsigned char address)
{
    return i2c_start( address );

}/* i2c_rep_start */


/*************************************************************************
 Terminates the data transfer and releases the I2C bus
*************************************************************************/
void i2c_stop(void)
{
    /* send stop condition */
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWSTO);
```

```
        // wait until stop condition is executed and bus released
        while(TWCR & (1<<TWSTO));

}/* i2c_stop */


/*************************************************************************
  Send one byte to I2C device

  Input:    byte to be transfered
  Return:   0 write successful
            1 write failed
*************************************************************************/
unsigned char i2c_write( unsigned char data )
{
    uint8_t   twst;

        // send data to the previously addressed device
        TWDR = data;
        TWCR = (1<<TWINT) | (1<<TWEN);

        // wait until transmission completed
        while(!(TWCR & (1<<TWINT)));

        // check value of TWI Status Register. Mask prescaler bits
        twst = TW_STATUS & 0xF8;
        if( twst != TW_MT_DATA_ACK) return 1;
        return 0;

}/* i2c_write */


/*************************************************************************
 Read one byte from the I2C device, request more data from device

 Return:  byte read from I2C device
*************************************************************************/
unsigned char i2c_readAck(void)
{
        TWCR = (1<<TWINT) | (1<<TWEN) | (1<<TWEA);
        while(!(TWCR & (1<<TWINT)));

    return TWDR;

}/* i2c_readAck */


/*************************************************************************
 Read one byte from the I2C device, read is followed by a stop condition

 Return:  byte read from I2C device
*************************************************************************/
unsigned char i2c_readNak(void)
{
        TWCR = (1<<TWINT) | (1<<TWEN);
        while(!(TWCR & (1<<TWINT)));

    return TWDR;

}/* i2c_readNak */
```

Fat File System

```
/*-----------------------------------------------------------------------*/
/* MMC/SDSC/SDHC (in SPI mode) control module  (C)ChaN, 2007             */
/*-----------------------------------------------------------------------*/
/* Only rcvr_spi(), xmit_spi(), disk_timerproc() and some macros         */
/* are platform dependent.                                               */
/*-----------------------------------------------------------------------*/
```

```
#include <avr/io.h>
#include "diskio.h"


/* Definitions for MMC/SDC command */
#define CMD0   (0x40+0)       /* GO_IDLE_STATE */
#define CMD1   (0x40+1)       /* SEND_OP_COND (MMC) */
#define ACMD41 (0xC0+41)      /* SEND_OP_COND (SDC) */
#define CMD8   (0x40+8)       /* SEND_IF_COND */
#define CMD9   (0x40+9)       /* SEND_CSD */
#define CMD10  (0x40+10)      /* SEND_CID */
#define CMD12  (0x40+12)      /* STOP_TRANSMISSION */
#define ACMD13 (0xC0+13)      /* SD_STATUS (SDC) */
#define CMD16  (0x40+16)      /* SET_BLOCKLEN */
#define CMD17  (0x40+17)      /* READ_SINGLE_BLOCK */
#define CMD18  (0x40+18)      /* READ_MULTIPLE_BLOCK */
#define CMD23  (0x40+23)      /* SET_BLOCK_COUNT (MMC) */
#define ACMD23 (0xC0+23)      /* SET_WR_BLK_ERASE_COUNT (SDC) */
#define CMD24  (0x40+24)      /* WRITE_BLOCK */
#define CMD25  (0x40+25)      /* WRITE_MULTIPLE_BLOCK */
#define CMD55  (0x40+55)      /* APP_CMD */
#define CMD58  (0x40+58)      /* READ_OCR */


/* Port Controls  (Platform dependent) */
#define SELECT()       PORTB &= ~(1 << PORTB4)              /* MMC CS = L */
#define DESELECT()     PORTB |= (1 << PORTB4)        /* MMC CS = H */

//#define SOCKPORT     PINB                  /* Socket contact port */
#define SOCKPORT 0x10   //ECE476 Modified: always assume card inserted.
#define SOCKWP         0x20                  /* Write protect switch (PB5) */
#define SOCKINS               0x10                  /* Card detect switch (PB4) */

#define FCLK_SLOW_BITS (1 << SPR1 | 1 << SPR0)  // fosc/128  = 125kHz
#define FCLK_FAST_BITS (1 << SPR0)  //fosc/16 = 1MHz
#define FCLK_MASK (1 << SPR1 | 1 << SPR0)
#define FCLK_SLOW()    SPCR = (SPCR & ~FCLK_MASK) | FCLK_SLOW_BITS  /* Set slow clock (100k-400k)
*/
#define FCLK_FAST()    SPCR = (SPCR & ~FCLK_MASK) | FCLK_FAST_BITS  /* Set fast clock (depends on
the CSD) */


/*--------------------------------------------------------------------------

   Module Private Functions

---------------------------------------------------------------------------*/

static volatile
DSTATUS Stat = STA_NOINIT;    /* Disk status */

static volatile
BYTE Timer1, Timer2;   /* 100Hz decrement timer */

static
BYTE CardType;                /* Card type flags */


/*-----------------------------------------------------------------------*/
/* Transmit a byte to MMC via SPI  (Platform dependent)                  */
/*-----------------------------------------------------------------------*/

#define xmit_spi(dat)  SPDR=(dat); loop_until_bit_is_set(SPSR,SPIF)



/*-----------------------------------------------------------------------*/
/* Receive a byte from MMC via SPI  (Platform dependent)                 */
/*-----------------------------------------------------------------------*/
```

```
static
BYTE rcvr_spi (void)
{
	SPDR = 0xFF;
	loop_until_bit_is_set(SPSR, SPIF);
	return SPDR;
}

/* Alternative macro to receive data fast */
#define rcvr_spi_m(dst)	SPDR=0xFF; loop_until_bit_is_set(SPSR,SPIF); *(dst)=SPDR



/*-----------------------------------------------------------------------*/
/* Wait for card ready                                                   */
/*-----------------------------------------------------------------------*/

static
BYTE wait_ready (void)
{
	BYTE res;


	Timer2 = 50;	/* Wait for ready in timeout of 500ms */
	rcvr_spi();
	do
		res = rcvr_spi();
	while ((res != 0xFF) && Timer2);
	//PORTA = ~res;
	return res;
}



/*-----------------------------------------------------------------------*/
/* Deselect the card and release SPI bus                                 */
/*-----------------------------------------------------------------------*/

static
void release_spi (void)
{
	DESELECT();
	rcvr_spi();
}



/*-----------------------------------------------------------------------*/
/* Power Control  (Platform dependent)                                   */
/*-----------------------------------------------------------------------*/
/* When the target system does not support socket power control, there   */
/* is nothing to do in these functions and chk_power always returns 1.   */

static
void power_on (void)
{
	for (Timer1 = 3; Timer1; );	/* Wait for 30ms */
	//PORTB = 0b10110101;		/* Enable drivers */
	//DDRB  = 0b11000111;
	//SPCR = 0b01010000;		/* Initialize SPI port (Mode 0) */

	//It didn't work with the above code, this is more clear anyway
	SPCR = (1<< SPE) | (1 << MSTR) | (FCLK_SLOW_BITS);  //SPI Mode 0
	SPSR = 0b00000000;
	//outputs are MOSI, SCLK, and CS
	DDRB = (1 << DDB7 | 1 << DDB5  | 1 << DDB4);
	PORTB = (1 << PORTB5 | 1 << PORTB4);
	DESELECT();
}

static
```

```
void power_off (void)
{
	SELECT();				/* Wait for card ready */
	wait_ready();
	release_spi();

	SPCR = 0;				/* Disable SPI function */
	//DDRB  = 0b11000000;		/* Disable drivers */
	//PORTB = 0b10110000;
	Stat |= STA_NOINIT;		/* Set STA_NOINIT */
}

static
int chk_power(void)		/* Socket power state: 0=off, 1=on */
{
	//return (PORTE & 0x80) ? 0 : 1;
	return 1;
}

/*-----------------------------------------------------------------------*/
/* Receive a data packet from MMC                                        */
/*-----------------------------------------------------------------------*/

static
BOOL rcvr_datablock (
	BYTE *buff,			/* Data buffer to store received data */
	UINT btr			/* Byte count (must be multiple of 4) */
)
{
	BYTE token;


	Timer1 = 10;
	do {							/* Wait for data packet in timeout of
100ms */
		token = rcvr_spi();
	} while ((token == 0xFF) && Timer1);
	if(token != 0xFE) return FALSE;	/* If not valid data token, retutn with error */

	do {							/* Receive the data block into buffer
*/
		rcvr_spi_m(buff++);
		rcvr_spi_m(buff++);
		rcvr_spi_m(buff++);
		rcvr_spi_m(buff++);
	} while (btr -= 4);
	rcvr_spi();						/* Discard CRC */
	rcvr_spi();

	return TRUE;					/* Return with success */
}



/*-----------------------------------------------------------------------*/
/* Send a data packet to MMC                                             */
/*-----------------------------------------------------------------------*/

#if _READONLY == 0
static
BOOL xmit_datablock (
	const BYTE *buff,	/* 512 byte data block to be transmitted */
	BYTE token			/* Data/Stop token */
)
{
	BYTE resp, wc;


	if (wait_ready() != 0xFF) return FALSE;

	xmit_spi(token);					/* Xmit data token */
```

```
	if (token != 0xFD) {	/* Is data token */
		wc = 0;
		do {                                                    /* Xmit the 512 byte data
block to MMC */
			xmit_spi(*buff++);
			xmit_spi(*buff++);
		} while (--wc);
		xmit_spi(0xFF);                                         /* CRC (Dummy) */
		xmit_spi(0xFF);
		resp = rcvr_spi();                              /* Reveive data response */
		if ((resp & 0x1F) != 0x05)           /* If not accepted, return with error */
			return FALSE;
	}

	return TRUE;
}
#endif /* _READONLY */



/*-----------------------------------------------------------------------*/
/* Send a command packet to MMC                                          */
/*-----------------------------------------------------------------------*/

static
BYTE send_cmd (
	BYTE cmd,		/* Command byte */
	DWORD arg		/* Argument */
)
{
	BYTE n, res;


	if (cmd & 0x80) {	/* ACMD<n> is the command sequense of CMD55-CMD<n> */
		cmd &= 0x7F;
		res = send_cmd(CMD55, 0);
		if (res > 1) return res;
	}

	/* Select the card and wait for ready */
	DESELECT();
	SELECT();
	if (wait_ready() != 0xFF) return 0xFF;
	//PORTA = 0xFF;

	/* Send command packet */
	xmit_spi(cmd);                                          /* Start + Command index */
	xmit_spi((BYTE)(arg >> 24));		/* Argument[31..24] */
	xmit_spi((BYTE)(arg >> 16));		/* Argument[23..16] */
	xmit_spi((BYTE)(arg >> 8));			/* Argument[15..8] */
	xmit_spi((BYTE)arg);				/* Argument[7..0] */
	n = 0x01;                                                       /* Dummy CRC + Stop */
	if (cmd == CMD0) n = 0x95;			/* Valid CRC for CMD0(0) */
	if (cmd == CMD8) n = 0x87;			/* Valid CRC for CMD8(0x1AA) */
	xmit_spi(n);

	/* Receive command response */
	if (cmd == CMD12) rcvr_spi();		/* Skip a stuff byte when stop reading */
	n = 10;                                                         /* Wait for a valid response
in timeout of 10 attempts */
	do
		res = rcvr_spi();
	while ((res & 0x80) && --n);

	return res;			/* Return with the response value */
}



/*-----------------------------------------------------------------------
```

```
   Public Functions

-----------------------------------------------------------------------*/


/*-----------------------------------------------------------------------*/
/* Initialize Disk Drive                                                 */
/*-----------------------------------------------------------------------*/

DSTATUS disk_initialize (
        BYTE drv              /* Physical drive nmuber (0) */
)
{
        BYTE n, cmd, ty, ocr[4];

        //PORTA = 0;
        if (drv) return STA_NOINIT;                 /* Supports only single drive */
        if (Stat & STA_NODISK) {
                return Stat;   /* No card in the socket */
        }
        //PORTA |= 1;

        power_on();    /* Force socket power on */
        FCLK_SLOW();
        for (n = 10; n; n--) rcvr_spi();      /* 80 dummy clocks */
        ty = 0;
        BYTE res = send_cmd(CMD0, 0) ;

        //PORTA = ~res;
        if (res == 1) {                       /* Enter Idle state */
                //PORTA |= 2;
                Timer1 = 100;   /* Initialization timeout of 1000 msec */
                if (send_cmd(CMD8, 0x1AA) == 1) {     /* SDHC */
                        //PORTA |= 4;
                        for (n = 0; n < 4; n++) ocr[n] = rcvr_spi();        /* Get trailing return
value of R7 resp */
                        if (ocr[2] == 0x01 && ocr[3] == 0xAA) {                             /* The
card can work at vdd range of 2.7-3.6V */
                                while (Timer1 && send_cmd(ACMD41, 1UL << 30));       /* Wait for
leaving idle state (ACMD41 with HCS bit) */
                                if (Timer1 && send_cmd(CMD58, 0) == 0) {             /* Check CCS
bit in the OCR */
                                        for (n = 0; n < 4; n++) ocr[n] = rcvr_spi();
                                        ty = (ocr[0] & 0x40) ? CT_SD2 | CT_BLOCK : CT_SD2;
                                }
                        }
                } else {                                                            /* SDSC or MMC */
                        //PORTA |= 8;
                        if (send_cmd(ACMD41, 0) <= 1)         {
                                ty = CT_SD1; cmd = ACMD41;    /* SDSC */
                                //PORTA |= 16;
                        } else {
                                ty = CT_MMC; cmd = CMD1;      /* MMC */
                        }
                        while (Timer1 && send_cmd(cmd, 0));                 /* Wait for leaving
idle state */
                        if (!Timer1 || send_cmd(CMD16, 512) != 0)    /* Set R/W block length to
512 */
                                //PORTA |= 32;
                                ty = 0;
                }
        }
        CardType = ty;
        release_spi();

        if (ty) {                     /* Initialization succeded */
                Stat &= ~STA_NOINIT;          /* Clear STA_NOINIT */
                FCLK_FAST();
        } else {                      /* Initialization failed */
                power_off();
        }
```

```
	return Stat;
}



/*-----------------------------------------------------------------------*/
/* Get Disk Status                                                       */
/*-----------------------------------------------------------------------*/

DSTATUS disk_status (
	BYTE drv		/* Physical drive nmuber (0) */
)
{
	if (drv) return STA_NOINIT;		/* Supports only single drive */
	return Stat;
}



/*-----------------------------------------------------------------------*/
/* Get Disk Time                                                         */
/*-----------------------------------------------------------------------*/
DWORD get_fattime () {
  return 0;
}

/*-----------------------------------------------------------------------*/
/* Read Sector(s)                                                        */
/*-----------------------------------------------------------------------*/

DRESULT disk_read (
	BYTE drv,			/* Physical drive nmuber (0) */
	BYTE *buff,			/* Pointer to the data buffer to store read data */
	DWORD sector,		/* Start sector number (LBA) */
	BYTE count			/* Sector count (1..255) */
)
{
	if (drv || !count) return RES_PARERR;
	if (Stat & STA_NOINIT) return RES_NOTRDY;

	if (!(CardType & CT_BLOCK)) sector *= 512;	/* Convert to byte address if needed */

	if (count == 1) {	/* Single block read */
		if ((send_cmd(CMD17, sector) == 0)	/* READ_SINGLE_BLOCK */
			&& rcvr_datablock(buff, 512))
			count = 0;
	}
	else {				/* Multiple block read */
		if (send_cmd(CMD18, sector) == 0) {	/* READ_MULTIPLE_BLOCK */
			do {
				if (!rcvr_datablock(buff, 512)) break;
				buff += 512;
			} while (--count);
			send_cmd(CMD12, 0);				/* STOP_TRANSMISSION */
		}
	}
	release_spi();

	return count ? RES_ERROR : RES_OK;
}



/*-----------------------------------------------------------------------*/
/* Write Sector(s)                                                       */
/*-----------------------------------------------------------------------*/

#if _READONLY == 0
DRESULT disk_write (
	BYTE drv,			/* Physical drive nmuber (0) */
```

```
	const BYTE *buff,	/* Pointer to the data to be written */
	DWORD sector,		/* Start sector number (LBA) */
	BYTE count			/* Sector count (1..255) */
)
{
	if (drv || !count) return RES_PARERR;
	if (Stat & STA_NOINIT) return RES_NOTRDY;
	if (Stat & STA_PROTECT) return RES_WRPRT;

	if (!(CardType & CT_BLOCK)) sector *= 512;	/* Convert to byte address if needed */

	if (count == 1) {	/* Single block write */
		if ((send_cmd(CMD24, sector) == 0)	/* WRITE_BLOCK */
			&& xmit_datablock(buff, 0xFE))
			count = 0;
	}
	else {				/* Multiple block write */
		if (CardType & CT_SDC) send_cmd(ACMD23, count);
		if (send_cmd(CMD25, sector) == 0) {	/* WRITE_MULTIPLE_BLOCK */
			do {
				if (!xmit_datablock(buff, 0xFC)) break;
				buff += 512;
			} while (--count);
			if (!xmit_datablock(0, 0xFD)) /* STOP_TRAN token */
				count = 1;
		}
	}
	release_spi();

	return count ? RES_ERROR : RES_OK;
}
#endif /* _READONLY == 0 */



/*-----------------------------------------------------------------------*/
/* Miscellaneous Functions                                               */
/*-----------------------------------------------------------------------*/

#if _USE_IOCTL != 0
DRESULT disk_ioctl (
	BYTE drv,		/* Physical drive nmuber (0) */
	BYTE ctrl,		/* Control code */
	void *buff		/* Buffer to send/receive control data */
)
{
	DRESULT res;
	BYTE n, csd[16], *ptr = buff;
	WORD csize;


	if (drv) return RES_PARERR;

	res = RES_ERROR;

	if (ctrl == CTRL_POWER) {
		switch (*ptr) {
		case 0:		/* Sub control code == 0 (POWER_OFF) */
			if (chk_power())
				power_off();		/* Power off */
			res = RES_OK;
			break;
		case 1:		/* Sub control code == 1 (POWER_ON) */
			power_on();				/* Power on */
			res = RES_OK;
			break;
		case 2:		/* Sub control code == 2 (POWER_GET) */
			*(ptr+1) = (BYTE)chk_power();
			res = RES_OK;
			break;
		default :
```

```
                        res = RES_PARERR;
                }
        }
        else {
                if (Stat & STA_NOINIT) return RES_NOTRDY;

                switch (ctrl) {
                case CTRL_SYNC :              /* Make sure that no pending write process */
                        SELECT();
                        if (wait_ready() == 0xFF)
                                res = RES_OK;
                        break;

                case GET_SECTOR_COUNT :       /* Get number of sectors on the disk (DWORD) */
                        if ((send_cmd(CMD9, 0) == 0) && rcvr_datablock(csd, 16)) {
                                if ((csd[0] >> 6) == 1) {     /* SDC ver 2.00 */
                                        csize = csd[9] + ((WORD)csd[8] << 8) + 1;
                                        *(DWORD*)buff = (DWORD)csize << 10;
                                } else {                                        /* SDC ver 1.XX or
MMC*/
                                        n = (csd[5] & 15) + ((csd[10] & 128) >> 7) + ((csd[9] & 3)
<< 1) + 2;
                                        csize = (csd[8] >> 6) + ((WORD)csd[7] << 2) +
((WORD)(csd[6] & 3) << 10) + 1;
                                        *(DWORD*)buff = (DWORD)csize << (n - 9);
                                }
                                res = RES_OK;
                        }
                        break;

                case GET_SECTOR_SIZE : /* Get R/W sector size (WORD) */
                        *(WORD*)buff = 512;
                        res = RES_OK;
                        break;

                case GET_BLOCK_SIZE :  /* Get erase block size in unit of sector (DWORD) */
                        if (CardType & CT_SD2) {      /* SDC ver 2.00 */
                                if (send_cmd(ACMD13, 0) == 0) {       /* Read SD status */
                                        rcvr_spi();
                                        if (rcvr_datablock(csd, 16)) {
        /* Read partial block */
                                                for (n = 64 - 16; n; n--) rcvr_spi(); /* Purge
trailing data */
                                                *(DWORD*)buff = 16UL << (csd[10] >> 4);
                                                res = RES_OK;
                                        }
                                }
                        } else {                                        /* SDC ver 1.XX or MMC */
                                if ((send_cmd(CMD9, 0) == 0) && rcvr_datablock(csd, 16)) {  /* Read
CSD */
                                        if (CardType & CT_SD1) {      /* SDC ver 1.XX */
                                                *(DWORD*)buff = (((csd[10] & 63) << 1) +
((WORD)(csd[11] & 128) >> 7) + 1) << ((csd[13] >> 6) - 1);
                                        } else {                                        /* MMC */
                                                *(DWORD*)buff = ((WORD)((csd[10] & 124) >> 2) + 1) *
(((csd[11] & 3) << 3) + ((csd[11] & 224) >> 5) + 1);
                                        }
                                        res = RES_OK;
                                }
                        }
                        break;

                case MMC_GET_TYPE :           /* Get card type flags (1 byte) */
                        *ptr = CardType;
                        res = RES_OK;
                        break;

                case MMC_GET_CSD :            /* Receive CSD as a data block (16 bytes) */
                        if (send_cmd(CMD9, 0) == 0            /* READ_CSD */
                                && rcvr_datablock(ptr, 16))
                                res = RES_OK;
```

```
                        break;

                case MMC_GET_CID :              /* Receive CID as a data block (16 bytes) */
                        if (send_cmd(CMD10, 0) == 0          /* READ_CID */
                                && rcvr_datablock(ptr, 16))
                                res = RES_OK;
                        break;

                case MMC_GET_OCR :              /* Receive OCR as an R3 resp (4 bytes) */
                        if (send_cmd(CMD58, 0) == 0) {       /* READ_OCR */
                                for (n = 4; n; n--) *ptr++ = rcvr_spi();
                                res = RES_OK;
                        }
                        break;

                case MMC_GET_SDSTAT :  /* Receive SD statsu as a data block (64 bytes) */
                        if (send_cmd(ACMD13, 0) == 0) {      /* SD_STATUS */
                                rcvr_spi();
                                if (rcvr_datablock(ptr, 64))
                                        res = RES_OK;
                        }
                        break;

                default:
                        res = RES_PARERR;
                }

                release_spi();
        }

        return res;
}
#endif /* _USE_IOCTL != 0 */


/*-----------------------------------------------------------------------*/
/* Device Timer Interrupt Procedure  (Platform dependent)                */
/*-----------------------------------------------------------------------*/
/* This function must be called in period of 10ms                        */

void disk_timerproc (void)
{
        static BYTE pv;
        BYTE n, s;


        n = Timer1;                                          //* 100Hz decrement timer
        if (n) Timer1 = --n;
        n = Timer2;
        if (n) Timer2 = --n;

        n = pv;
        pv = SOCKPORT & (SOCKWP | SOCKINS);   //* Sample socket switch

        //Removed by jsm66, we don't have lines for WP or INS
    //This code was just making it think there was never a card.
        /*if (n == pv) {                                     //* Have contacts stabled?
                s = Stat;

                if (pv & SOCKWP)                     //* WP is H (write protected)
                        s |= STA_PROTECT;
                else                                         //* WP is L (write enabled)
                        s &= ~STA_PROTECT;

                if (pv & SOCKINS)                    //* INS = H (Socket empty)
                        s |= (STA_NODISK | STA_NOINIT);
                else                                         //* INS = L (Card inserted)
                        s &= ~STA_NODISK;

                Stat = (s & ~STA_NODISK & ~STA_PROTECT);
                //PORTA = ~Stat;
```

```
       } */
}
```

MATLAB Data Analyzer

```
%--------------------------------------------------------------------------
%      This MATLAB script is specifically written to analyze the log data generated
%      on the Energy Monitor Device.
%      It provide extreme easy-to-use user interface and data are catogorized and
%      plotted in sperated figures
%--------------------------------------------------------------------------

function varargout = expert_GUI(varargin)
% EXPERT_GUI M-file for expert_GUI.fig
%      EXPERT_GUI, by itself, creates a new EXPERT_GUI or raises the existing
%      singleton*.
%
%      H = EXPERT_GUI returns the handle to a new EXPERT_GUI or the handle to
%      the existing singleton*.
%
%      EXPERT_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in EXPERT_GUI.M with the given input arguments.
%
%      EXPERT_GUI('Property','Value',...) creates a new EXPERT_GUI or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before expert_GUI_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to expert_GUI_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help expert_GUI

% Last Modified by GUIDE v2.5 19-May-2011 11:09:39

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @expert_GUI_OpeningFcn, ...
                   'gui_OutputFcn',  @expert_GUI_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before expert_GUI is made visible.
function expert_GUI_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to expert_GUI (see VARARGIN)

% Choose default command line output for expert_GUI
handles.output = hObject;
handles.filename='C:\\ENERGY.log';
handles.record=0;
handles.datatype=1;
```

```
handles.datanum=0;
set(hObject,'toolbar','figure');
%clear the hostorical figure
cla(handles.axes1,'reset')
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes expert_GUI wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = expert_GUI_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
switch get(handles.popupmenu1,'Value')
    case 1
        str='C';
    case 2
        str='D';
    case 3
        str='E';
    case 4
        str='F';
    case 5
        str='G';
    case 6
        str='H';
    case 7
        str='I';
    case 8
        str='J';
    case 9
        str='K';
    case 10
        str='L';
    case 11
        str='M';
    case 12
        str='N';
    case 13
        str='Z';
    otherwise
end

handles.filename=[str ':\\ENERGY.log'];
% Update handles structure
guidata(hObject, handles);


% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1


% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function selectnum_Callback(hObject, eventdata, handles)
%store the contents of input1_editText as a string. if the string
%is not a number then input will be empty
handles.datanum = str2num(get(hObject,'String'));

%checks to see if input is empty. if so, default input1_editText to zero
if (isempty(handles.datanum))
     set(hObject,'String','0')
else
    sz=size(handles.record);
    if handles.datanum>sz(1)
        handles.datanum=sz(1);
    end
end
guidata(hObject, handles);

% hObject    handle to selectnum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of selectnum as text
%        str2double(get(hObject,'String')) returns contents of selectnum as a double


% --- Executes during object creation, after setting all properties.
function selectnum_CreateFcn(hObject, eventdata, handles)
% hObject    handle to selectnum (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



% --- Executes on button press in loadbutton.
function loadbutton_Callback(hObject, eventdata, handles)
% hObject    handle to loadbutton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
fid=fopen(handles.filename,'r');

if (fid==-1) %fail to open
    set(handles.datacount,'String','None!');
    set(handles.ltime, 'String', 'No Log File Exists!');
else
    sline=fgetl(fid); %get the first line

    cc=0;
    while ischar(sline)
        cc=cc+1;
        [A, count]=sscanf(sline, '%d/%d/%d,%d:%d:%d,%d,%d,%d,%d,%d');
        if cc==1
            handles.record=A';
        else
            handles.record=[handles.record; A'];
        end
        sline=fgetl(fid);
```

```
    end
    fclose(fid);

    handles.datanum=cc;

    set(handles.datacount,'String',cc);
    stime=sprintf('%d/%d/%d, %d:%d:%d', ...
        handles.record(cc, 1),handles.record(cc, 2), handles.record(cc, 3),...
        handles.record(cc, 4), handles.record(cc, 5), handles.record(cc, 6));
    set(handles.ltime, 'String', stime);

    guidata(hObject, handles); %updates the handles
end


% --- Executes on selection change in typeselect.
function typeselect_Callback(hObject, eventdata, handles)
handles.datatype=get(handles.typeselect,'Value');
guidata(hObject, handles); %updates the handles

% hObject    handle to typeselect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns typeselect contents as cell array
%        contents{get(hObject,'Value')} returns selected item from typeselect


% --- Executes during object creation, after setting all properties.
function typeselect_CreateFcn(hObject, eventdata, handles)
% hObject    handle to typeselect (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in myplot.
function myplot_Callback(hObject, eventdata, handles)
    sz=size(handles.record);
    %clear the figure
    cla(handles.axes1,'reset')
    guidata(hObject, handles); %updates the handles

    axes(handles.axes1)

    x = (sz(1)-handles.datanum+1):sz(1);
    switch handles.datatype
        case 1  %temperature, two y vectors
            y1 = handles.record((sz(1)-handles.datanum+1):sz(1),8);
            y2 = handles.record((sz(1)-handles.datanum+1):sz(1),9);
        case 2  %energy
            y1 = handles.record((sz(1)-handles.datanum+1):sz(1),10);
        case 3  %water volume
            y1 = handles.record((sz(1)-handles.datanum+1):sz(1),11)/75;
        otherwise
    end


    %plots the x and y data
    plot(x,y1);
    grid on
    if handles.datatype==1 %if temperature is selected
        hold on
        plot(x,y2,'r');
    end
    %adds a title, x-axis description, and y-axis description addording to
```

```
%specific data types
xlabel('Entry #');
switch handles.datatype
    case 1   %temperature
        title('Temperature of Water IN & OUT');
        ylabel('Temperature(Fahrenheit)');
        legend('Water Temp in','Water Temp out');
    case 2 %energy
        title('Accumulated Net Energy');
        ylabel('Net Energy (Joule)');
    case 3  %volume
        title('Accumulated Water Usage');
        ylabel ('Volume (Gallon)');
    otherwise
end
guidata(hObject, handles); %updates the handles
```