Low Cost E-Reader

Fall 2014–Spring 2015

Author: Allison King
Advisor: Visiting Assistant Professor Chen-Huan Chiang

Department of Engineering
Swarthmore College

May 14, 2015

Abstract

A basic e-reader was manufactured from fundamental parts using a microcontroller, LCD screen, and SD card. The three components were successfully integrated together to create a functioning device that could "flip" through the pages of a book stored in text file format on the SD card and display this text on the LCD screen. The goal was to minimize cost while maintaing functionality. The cost of each individual component totaled to slightly under $50, where each part was bought individually (not in bulk). The e-reader software was developed on an Atmel Xplained Mini development board and ported to a standalone microcontroller that runs on battery power. An Atmega328p microcontroller, Nokia 6100 LCD Shield, and 16GB SDHC microSD card were used for this project.

Acknowledgments

Immense thanks to Professor Chen-Huan Chiang for advising me throughout this project, for staying up until the late hours answering my questions, and for putting up with my frustrations when nothing seemed to be working. Thanks also to Professor Erik Cheever for showing interest in the project from the very beginning and for the advice. I would also like to thank Ed Jaoudi for teaching me how to use a solder and for never failing to find all of the parts I needed regardless of what or where they were.

I would also like to take the time to thank those incredible people in my life who still hung out with me even when I seemed to end every sentence with "...but it should work *in theory*". In particular– Wendy for believing in me enough to hold something still as I soldered for the first time as well as for being my number one partner in mischief, Michelle for constant genuine encouragement and all of the snacking from CVS chocolate to sour worms, Steven for late night Wawa trips and for making me feel a bit better knowing I was not the most stressed (sorry), Karl for enduring all these classes with me the past four years and for being my number one victim of mischief, Oscar for GONDOR, Iris for all around brightness, Jess for helping me unsolder my SD card that one time, and Jimmy, for all of those adventures, whether they were through Philadelphia, Seattle, Hogwarts, or Republic City, I've loved them all.

Lastly, to my grandmother who first opened my imagination through reading to me each and every night stories about the Monkey King and if dinosaurs were still alive, to my parents who showed no concern and only bought me more books when I started talking with a British accent and waving chopsticks around casting spells, and to my brother, even though he told me very bluntly and dream crushing-ly that Narnia did not exist when I only asked if we had a wardrobe in the house. This e-reader was inspired by all of you!

Contents

List of Figures

1 Introduction

"I was once in New York, and I listened to a talk about the building of private prisons– a huge growth industry in America. The prison industry needs to plan its future growth– how many cells are they going to need? How many prisoners are there going to be, 15 years from now? And they found they could predict it very easily, using a pretty simple algorithm, based on asking what percentage of 10 and 11-year olds couldn't read...

The simplest way to make sure that we raise literate children is to teach them to read, and to show them that reading is a pleasurable activity. And that means, at its simplest, finding books that they enjoy, giving them access to those books, and letting them read them."

-Neil Gaiman

1.1 Objective

The goal of this project is to design a functioning e-reader at minimal cost. A functioning e-reader is defined as one that can store e-book files and display this text to a screen in "pages". These "pages" can then be flipped through to progress through the book.

1.2 Motivation

Currently, the average price of the most competitive e-readers range from seventy to two hundred dollars. However, the cost to output text to a screen is in theory very small. E-readers in the current market have a lot more functions than just outputting text to screen though such as internet capabilities, dictionaries, touch screen, brightness sensitivity and more, making them near luxury items. Much of the cost is hypothesized to also come from buying into the creator's market place (e.g. Amazon or Barnes and Noble). If these extra features were stripped away to just the text– the most essential part of a book– with no attachment to a company, then costs could decrease dramatically and make books more accessible to the general public.

E-readers not only conserve paper, but e-book files are typically cheaper to buy than their paper counterparts. Furthermore, many books ranging from *Pride and Prejudice* to *The Adventures of Huckleberry Finn* are all available in the public domain for free. If teachers were to invest in low cost e-readers, many of the required texts could be supplied to students at a much lower cost to the district, thereby saving resources that could be allocated elsewhere. Making books and literature as accessible as possible encourages learning and provides opportunity to explore many subjects.

1.3 Background

E-readers began to appear in the market as early as 1998, though the idea to create electronic files for books has been around for even longer, possibly since 1971. Due to limited memory capabilities and short battery life at the time, these early e-readers were unsuccessful. In 2006, the Sony Reader became the first to use electronic paper for e-reading purposes, but it was only in 2007 when Amazon released its first Kindle that e-readers became extremely popular and desirable with the Kindle selling out in its first five hours on the market. In 2010, Amazon reported having sold more e-books than hardcover books that year.

However, the rise of the e-reader has been accompanied by many economic issues as well. In 2012, the United States Department of Justice sued Apple and five major publishers for colluding the price of e-books. The reason for the collusion was largely due to Amazon's monopoly on the e-book and the e-reader market. Because the Kindle can only support .mobi or .azw files which can only be purchased from Amazon, Amazon has effectively gained complete control of the market and can therefore afford to release an e-reader as expensive as the Amazon Voyage, priced at two hundred dollars. Figure 1 shows which e-readers can support which files. The Kindle is the only e-reader that does not provide support for the "free and open e-book" file, the ePub file. Third party softwares such as Calibre can convert ePub to Kindle compatible format mobi, but library loans, typically in ePub format, protect from this conversion. Therefore library e-books become inacessible to Kindle owners.

It is also interesting to note that Amazon is known to sell their Kindles at a loss, meaning the e-reader components are more expensive than their sticker price. This tactic helps them gain monopology over the sale of their e-books, with the selling of these electronic files being the source of their profit. This project seeks to investigate just how expensive the components required to make an e-reader would be to see what the cost would be to break free of the Amazon monopoly.

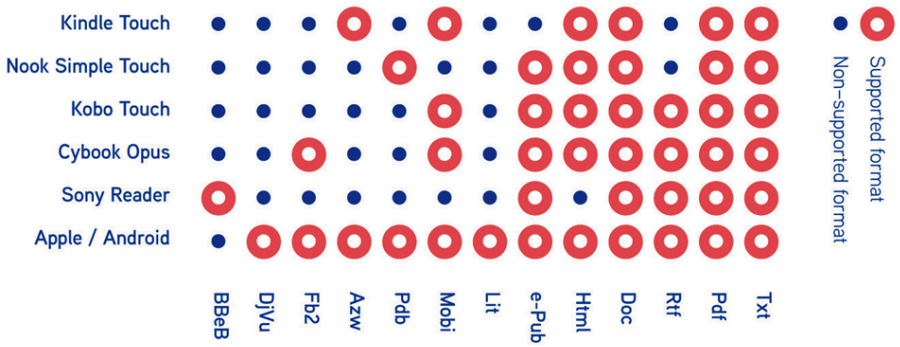

Figure 1: E-reader E-book Compatability

Due to the e-reader's history of monopolization and price collusion, it is therefore interesting

from not just an engineering perspective but also an economical perspective to see at what minimal cost an e-reader can be produced to maximize enjoyment of books.

1.4 Contribution

In this project, I was able to successfully integrate the Atmega328p microcontroller with a Nokia GLCD screen, a task I that to my knowledge has been done before with these specific parts. Furthermore, by creating this e-reader from scratch, I was able to learn first hand the process of developing an embedded system. I learned the benefits of different types of microcontrollers, how to choose one, how to assemble a programming cable, how to write C code that could be programmed to a microcontroller, the process of developing and debugging code on a development board, and how to port the design to a standalone pseudo printed circuit board design. Through interfacing the LCD screen and the SD card, I learned not only how to communicate between the microcontroller and different peripherals, but also about drivers, SD card formats, and file systems. Lastly, I believe this project proves that a very rudimentary e-reader can be made at a relatively cheap cost– a cost that can only go down with the advances of modern technology.

2 Theory

A basic e-reader can be created with only a few major components:

1. Microcontroller: store code and flash memory	\$3-\$4
2. LCD Screen: output text to be read	\$22
3. SD Card: store e-book files to be read by microcontroller	\$2-\$8
4. SD Card breakout board: can eject SD card	\$10.00
5. Battery: power e-reader	\$5-\$10

The projected cost of this e-reader should be between \$42 and \$54, depending on SD card compatibility and estimated battery usage. This cost does not include development costs.

2.1 The AVR Microcontroller

The microcontroller chosen for this project was Atmel's Atmega328p. This microcontroller is an AVR microcontroller, an 8-bit RISC in Harvard architecture with on-chip flash memory. This chip also has GPIO (general purpose input/output) ports, I2C, SPI, and UART, though only I/O and SPI (serial peripheral interface) are needed in the proposed final implementation. For the proposed e-reader implementation, I2C is also an option for accessing external flash memory, though the relatively large amount of flash memory (32 kilobytes) on the Atmega328p should suffice. The I/O ports is used for receiving user input (i.e. a page turn or changing modes), while SPI is used for both communicating to the LCD screen and the SD card.

Though e-book files are small (for reference: 234 KB (*The Art of War*), 1010 KB (*Harry Potter and the Deathly Hallows*)), they are still too large for storing on the on-chip flash memory of the microcontroller. Therefore the SD card will be reserved for storing book files, while the microcontroller's flash will be only used for the e-reader implementation code.

2.1.1 Programming to Microcontroller

Unlike development boards such as Arduino, a single microcontroller cannot be programmed by connecting it to a computer via USB. Instead, a programmer is needed. A programmer is a device that can connect the microcontroller to the computer through a programming calbe (usually a USB cable) and program through the microcontroller's ISP (in-system programming) port. For example, AVR uses SPI interface for in-system programming. Figure 2 shows the pin connections necessary between a programmer and its microprocessor. Therefore, connections between a microcontroller

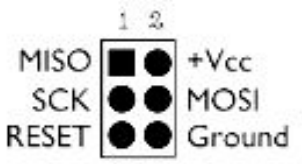

Figure 2: 6 Pin ISP Connector

and a programmer can easily be made through a target board, where the microcontroller can just be attached to a DIP (dual in-line package) socket so that it can be easily programmed and removed. Once the programmer is connected through the 6 pin ISP connector, only a 2x3 DIP header is needed on the target board to then make the connections as shown in Figure 3 for a 28 pin DIP header specifically for the Atmega328p.

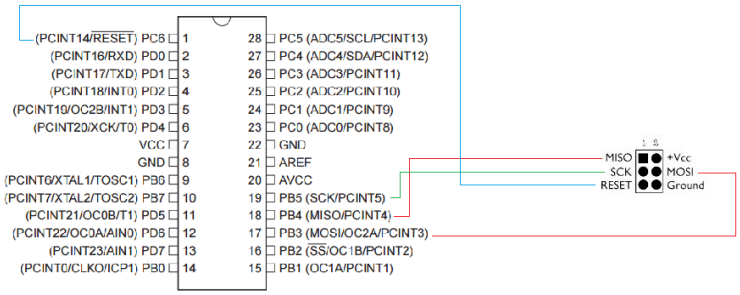

Figure 3: Target Board to Atmega328p Connections

In order to program code onto the microcontroller, the code must be in Intel HEX format. HEX files convey binary information to the microcontroller by burning the machine code onto the ROM of the controller. The process of going from a C or C++ file to a Hex file is typically done by a make file, converting the source file to an object, then elf, then hex file. This process of compiling code for the microcontroller on a PC is known as cross compiling. Programs like Atmel Studio can automatically compile a source file into a hex file without issuing command line statements.

2.2 Connecting to LCD

The microcontroller can output text to an LCD via its SPI. In this case, the Nokia 6110 LCD shield was used due to previous experience with the shield in a former class, its SPI capabilities, and its on shield joystick that can be adapted to a page turner.

2.2.1 Hardware Connections

Hardware connections between the microcontroller and the LCD shield are similar to those between the ISP connector and the target board in subsection 2.1.1 in that ports of the same name are connected– the SCK (serial clock) of the microcontroller is connected to the SCK of the LCD shield, MOSI to MOSI, and so on. A shield was useful in this project since all of the pins are easily accessible and, in this case, fit into the Arduino footprint also used by the development board.

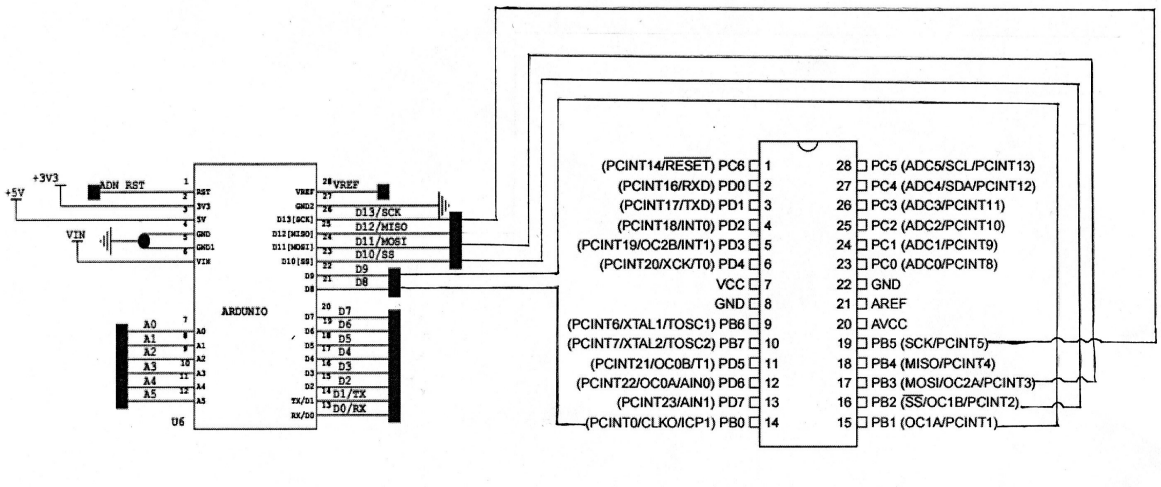

Figure 4: GLCD to Atmega328p Connections

There are a few connections to note from Figure 4. First, there is no need for a MISO (master in, slave out) connection because the LCD (the slave) will never give information to the microcontroller. Only the MOSI line (master out, slave in) is necessary for the microcontroller to give instructions to the LCD.

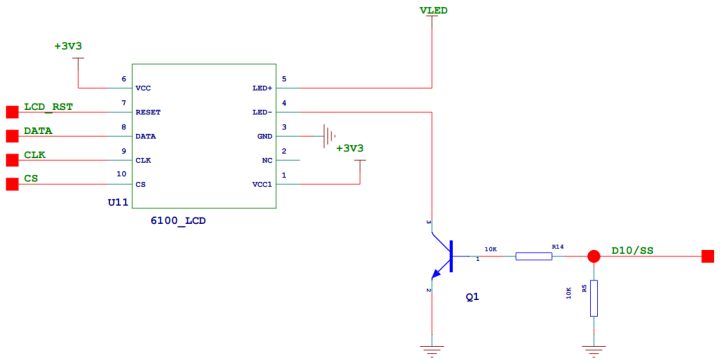

Figure 5: Nokia 6100 LCD Schematic

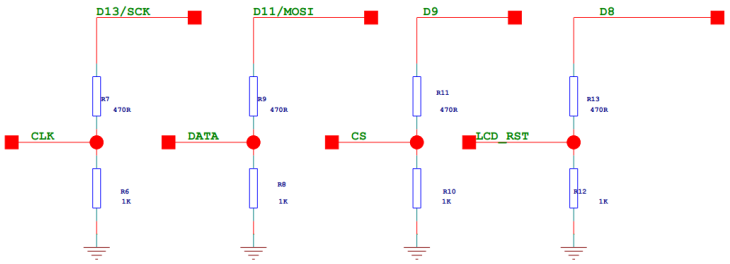

Figure 6: Pins D8, D9, D11, and D13 of LCD

Figures 5 and 6 give us a closer look at a few of the LCD pins. Of note is D8, which in Figure 4 is connected to PB0 on the Atmega328p. This is the LCD's reset pin. D9 is the chip select, connected to PB1 on the microcontroller. The last pin to note is D10, slave select, in Figure 5. This pin drives the backlight of the LCD.

To implement the page turning functionality of an e-reader, the joystick attached to the this shield was used, schematic shown in Figure 7.

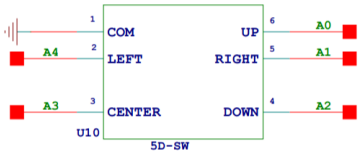

Figure 7: LCD Schematic for Joystick Connections

The schematic helpfully lays out which joystick command connects to which pin (A0-A4) which, looking at the Arduino schematic in Figure 4, can be matched to the appropriate pins on the microcontroller. In our case, to advance the page forward, we will connect A0 to PC0 and to move backwards, A2 to PC2 on the microcontroller. These will function as buttons, so we will set Port C to inputs then look for when they are pressed and then initiate the proper software control.

2.2.2 Embedded Software Design and Control

In order to communicate to the LCD through the microcontroller, the datasheets of both the MCU and the LCD must be carefully examined. For many LCD shields, the company that makes the product and the drivers typically offers starter code to make the initialization process as simple as possible. Elecfreaks, the company that distributes the Nokia 6100 LCD shield, offers a bit of

starter code on their website that includes tables for how to draw pixels into letters and numbers, how to draw lines, circles, and more.

The most important part in adapting this starter code to the appropriate microcontroller is to change declarations such as 'LCD_RST_PIN' to the appropriate numbered port based on the datasheet of the microcontroller.

```
#define LCD_DDR                 DDRB
#define LCD_PORT                PORTB
#define LCD_RST_PIN             0
#define SPI_MOSI_PIN            3

set_bit(LCD_PORT,LCD_RST_PIN)
set_bit(LCD_DDR, SPI_MOSI_PIN)
```

DDR is a data directional port of the AVR MCU. By setting this bit to high, we declare it as an output port, whereas low indicates an input port. By setting a PORT to high or low, we are setting the pin itself to either high or low. Therefore, the code above sets the reset pin to high so that the LCD does not reset, and also makes the SPI MOSI pin an output pin.

The specifics of initialization is entirely dependent on the driver which should be detailed in its datasheet. On a low level, this involves sending commands in the form of individual bits through the microcontroller's MOSI line.

```
1  void LCDShield::LCDCommand(unsigned char data)
2  {
3    char jj;

4    cbi(LCD_PORT_CS, CS);      // enable chip [LCD_CE_PORT]
5    cbi(LCD_PORT_DIO, DIO);    // output low on data out (9th bit low = command)

6    cbi(LCD_PORT_SCK, SCK_PIN);    // send clock pulse
7    delayMicroseconds(1);
8    sbi(LCD_PORT_SCK, SCK_PIN);

9    for (jj = 0; jj < 8; jj++)
10   {
11     if ((data & 0x80) == 0x80)
12       sbi(LCD_PORT_DIO, DIO);
13     else
14       cbi(LCD_PORT_DIO, DIO);
```

```
15      cbi(LCD_PORT_SCK, SCK_PIN); // send clock pulse
16      delayMicroseconds(1);
17      sbi(LCD_PORT_SCK, SCK_PIN);
18      data <<= 1;
19   }
20    sbi(LCD_PORT_CS, CS);       // disable
21 }
```

Above is the code for sending a command to the LCD. Line 4 first enables the LCD using the designated CS (chip select) port. Lines 6-8 send an initial clock pulse by clearing then setting the clock pin. Because in this case the microcontroller is the master and the LCD the slave, we use the microcontroller's clock. The for loop sends 8 clock pulses through MOSI to go through the length of the data on the SPI bus (detailed in Figure 8). The full initialization code is included in the appendix, but involves many steps listed in the datasheet for this LCD, such as starting the internal oscillator, setting color mode, and instructions on memory access.

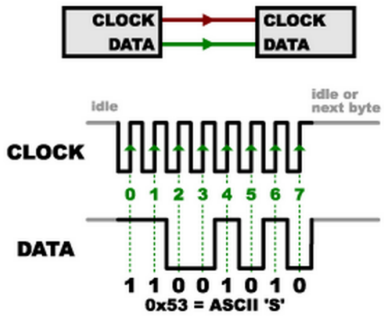

Figure 8: Sending Synchronous Data bits through SPI

2.3 Accessing SD Card Memory

To easily access memory on an SD card, the memory can be saved as raw data on the card. However, because an e-reader should be able to use a computer to add book files to it, the SD card should be more than just clumps of raw data. In the case where an SD card needs to be user friendly with a computer (instead of just, for example, using the card as a data logger), a file system needs to be used and parsed.

2.3.1 Hardware Design

The hardware connections for the SD card are straightforward. For this e-reader, we will be using an SD breakout board so that the SD can be easily removed and inserted back in with a push. Even without a breakout board the SD card can be directly soldered to a microcontroller and work, though we would like to add more files and so being able to remove the card is a valuable feature.

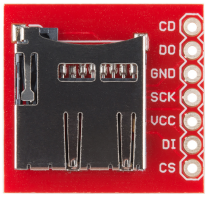

Figure 9: MicroSD Breakout Board

Figure 9 shows the connections from the microSD card and Table 1 details the exact connections needed to route to the Atmega328p.

SD card	Atmega328p
CD	None
DO	PB4 (MISO)
GND	GND
SCK	PB5 (SCK)
VCC	3.3V
DI	PB3 (MOSI)
CS	PD0

Table 1: SD to Atmega328p Connections

The CD option on the SD card is to determine whether or not a card has been inserted. CS (chip select) can be connected to any unused pin, in this case PD0 since none of the pins on PORTD of the microcontroller were being used. Of note is VCC which must be connected to 3.3 volts, since anything higher could burn out the card. The Atmega328p runs on 5 volts, so some level converting needs to be done to power the SD card. As we can see by the connections through MISO and MOSI, we are once again using SPI to send data between the microcontroller and the SD card.

In terms of the hardware for SD cards, in theory, any SD card, whether FAT16 or FAT32 (see Section 2.3.2 for definitions), can be used for this project with proper software modifications and no hardware modifications needed. Recently SDHC (high capacity) cards have become common in

the market and many software libraries have been able to adapt to both SD and SDHC cards. One notable difference between SD and SDHC cards besides memory capacity is their read lengths. SD cards can read any size block length, whereas SDHC cards are fixed to a 512-byte block length.[1] Regarding microSD cards versus regular sized SD cards, both can be used without software modifications and the hardware connections such as VCC, GND, MOSI, MISO, will be on all sizes of cards.

2.3.2 Embedded Software Design and Control

An SD card uses a FAT (File Allocation Table) file system which allows files to be partitioned and saved through a computer running on an operating system such as Microsoft Windows. Most SD cards are formatted in either FAT16 or FAT32, file systems that differ mainly in memory capacity and partition sizes. Typically, smaller SD cards (1-2GB) will automatically be formatted to FAT16 while larger ones to FAT32. A FAT partition will typically have[2]:

- Boot sector
- FAT #1
- FAT #2 (duplicate of FAT #1)
- root directory (only in FAT16)
- "data area" with files and subdirectories

When formatting an SD card with a FAT file system, the sizes of the sections listed above become fixed. FAT #2 serves as a backup for FAT #1 in case of corruption. One interesting difference between FAT32 and FAT16 is that FAT16 has a root directory while FAT32 does not– instead it stores its root directory with the rest of its "data area".

Zooming out from the FAT partition, we find that SD cards hold many partitions whose information is held by the MBR- Master Boot Record. The MBR holds the partition table and takes up one sector (512 bytes). Because there are many partitions, when trying to read memory from an SD card, one must look through the partition table to find the start of a partition. Only then can the FAT and root directory be decoded.

Files are saved in *clusters* onto the SD card, for example, saving a text file to an SD card may allocate Clusters 2-13, while a smaller one could allocate only Cluster 14. Files can be expanded

[1] `https://www.sdcard.org/downloads/pls/simplified_specs/part1_410.pdf`
[2] Credit to: `http://www.compuphase.com/mbr_fat.htm`

or deleted though so files often become separated through many nonconsecutive clusters. The file allocation table is used to "map" these clusters together for SD card reads[3].

The basic outline for reading a file is:

1. Find and go to start of first data cluster
2. Read cluster contents
3. Go to FAT entry for this cluster
4. Read number of next cluster
5. If there are more clusters, go to start of next cluster
6. Repeat!

Because SD cards are common in embedded systems projects with limited memory, many libraries have been written to read and write to SD cards. Most notable is FatFS and its subset Petit FatFS which have become commonly used in the embedded systems world. Petit FatFS is used in this project due to its very small RAM consumption. Only two files of code need to be added to a project to get the SD card running! Petit FatFS can read from SD cards but is more limited than FatFS in its writing capabilities. However, because the e-reader does not need to write to the SD card, Petit FatFS will be suitable for this project.

2.4 Battery

To make the e-reader portable, battery power is needed to replace the power supplies that are used in the lab or in the development system. From the datasheets, we can see that the Atmega328p needs 5 volts to operate, the SD card 3.3 volts, and the LCD needs both 3.3 and 5 volts (3.3 for the LCD, 5 for its switching regulator). Therefore, to acquire the necessary 5 volts, we can use 4 AA or 4 AAA batteries at 1.5 volts each to get to 6 volts. Once at 6 volts, we can use voltage regulators to get to 5 volts and 3.3 volts.

CMOS Low dropout (LDO) voltage regulators were used in both cases. Though voltage dividers can also reduce a voltage, voltage regulators also have the advantage of being to regulate current. Voltage regulators also are reliable for providing a fixed voltage level, normally achieved through feedback methods. The voltage drop is often given off as heat in a voltage regulator, so one should be sure to allow room for this component.

[3]Analogy from Code and Life Tutorial: `http://codeandlife.com/2012/04/07/simple-fat-and-sd-tutorial-part-2/`

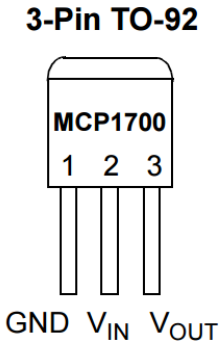

Figure 10: MCP1700 LDO Schematic

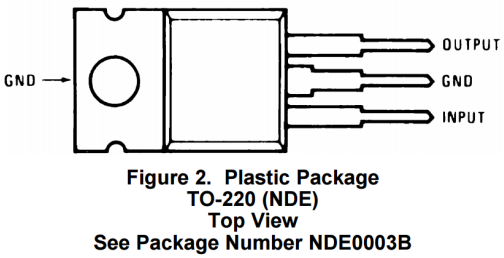

Figure 11: LM78XX Series Voltage Regulator

Figure 10 shows the schematic of the 3.3 volt regulator. The leftmost pin goes to ground and either the 5v or the 6v can go into V_{in} so that 3.3v comes out of V_{out}. Of note is the difference between the 3.3v regulator and the 5v in Figure 11 where GND and V_{in} are switched, an important difference between the 78XX and LDO regulators.

2.5 Integration

Because we are using SPI for both the LCD and the SD card, integrating the two is not entirely straightforward, though is not too difficult either. To choose between the SPI peripherals, we use the slave select pin. Each time we need to talk to either the SD card or the LCD, we turn on their respective slave select pins. When the slave select pins is set low, then the device is enabled. When the slave is not in use, the pin is set back to high. Therefore, in a function such as `LCDCommand()` from the code snippet in section 2.2.2, every time we want to send a command, we first enable the LCD by setting its slave select low, and we are done with it, turning it back on to high. This ensures that we are not sending SPI data to the wrong slave. Another detail to consider is the SPI Control Register (SPCR).

In the code for initializing and reading/writing to the SD card, we set the SPCR to 0x52. This can be decoded as shown in Figure 12.[4] We set SPCR to 0x52 which translates to 0101 0010 in binary, meaning that we enable the SPI interface and set the microcontroller to master mode. We also choose a relatively fast clock rate.

There are also SPSR (SPI Status Register) and SPDR (SPI Data Register). SPSR lets us know when a serial transfer is complete, while SPDR serves as the read/write register during data transfer. To write, we sent SPDR data through MOSI, while to read, SPDR gets data from MISO.

[4]Oregon State Engineering: `http://web.engr.oregonstate.edu/~traylor/ece473/lectures/spi.pdf`

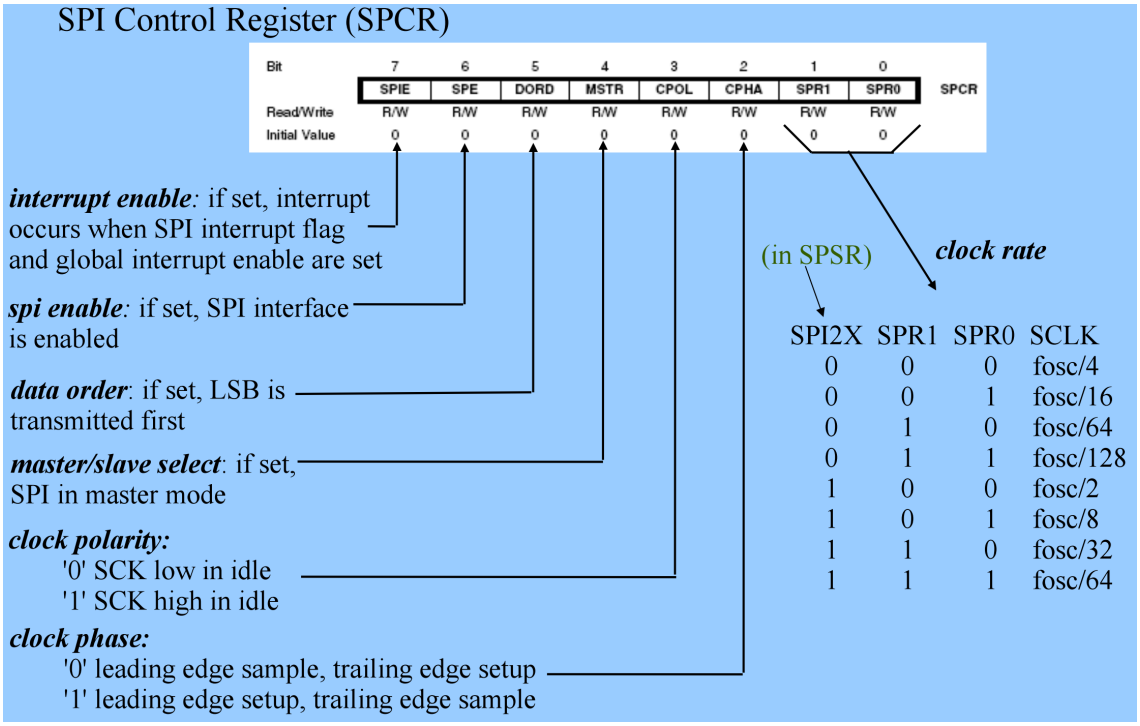

Figure 12: SPCR Breakdown

2.6 Saving to EEPROM

The Atmega328p boasts 1024 bytes of EEPROM (electrically erasable programmable read-only memory). EEPROM is a type of non-volatile memory, meaning that it can store data that will still be available after power down. This is necessary for implementing a feature where the page that the reader left off on can be accessed even after the e-reader has been powered down. One limitation of EEPROM is that there is a limited number of writes that can be done to it. In the case of most AVR's, this number is around 100,000 writes but unlimited reads. Therefore, it is important to only write to the EEPROM when necessary.

WinAVR includes an eeprom library where different size data can be written and accessed.

In Figure 13, the int 1 was written to the EEPROM flash at an arbitrarily chosen address. The three bytes to the right are set to zero because right before this screenshot was taken, the variable 0 was stored in the same place. Because EEPROM writes are limited, new variables will only be written if they differ from the old ones. Integers are typically 4 bytes, so when we switched from saving the integer 0 to saving the integer 1, only one of the bytes needed to be changed, highlighted in red. Figure 14 shows the EEPROM memory at start up and shows that the integer successfully saved even through power down.

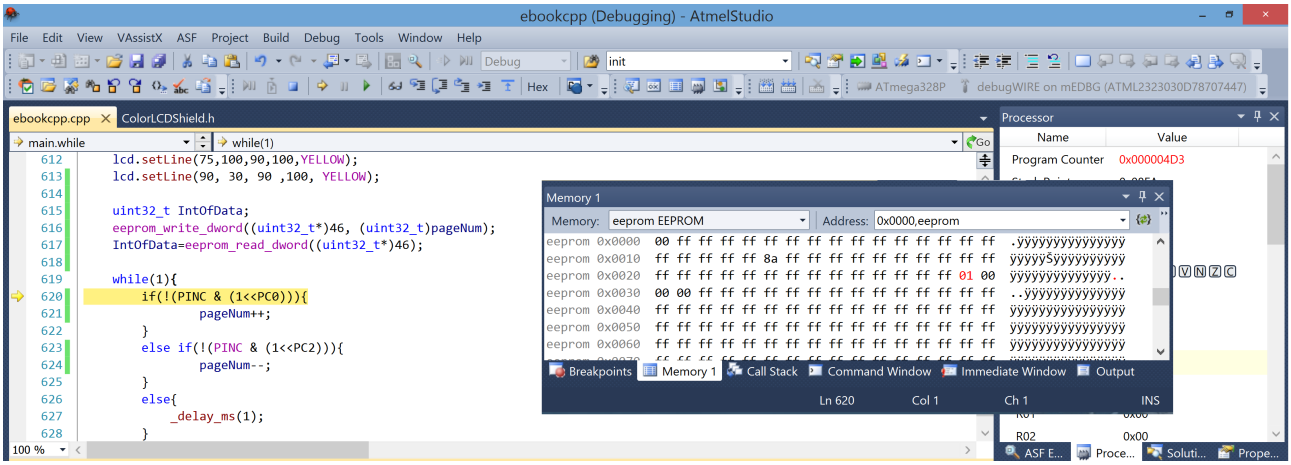

Figure 13: Writing Page Number 1 to EEPROM

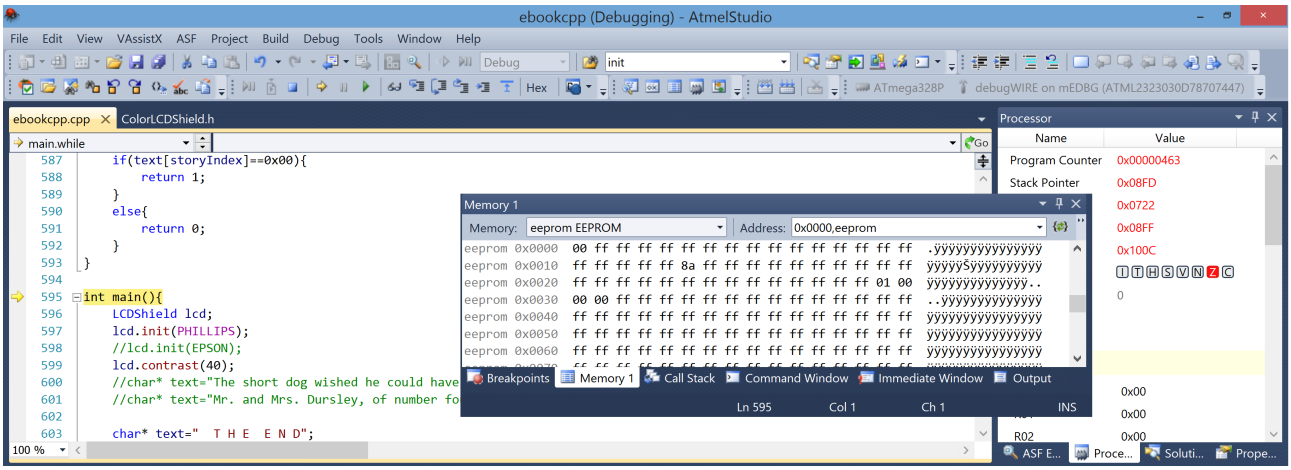

Figure 14: EEPROM Data still available after power down

3 Materials

3.1 Materials used for physical e-reader

- Atmega328p Microcontroller
- Color LCD Shield SHD-CLS (Philips PCF8833 controller, Nokia 6100 LCD)
- Kingston 16GB SDHC microSD card
- microSD Transflash Breakout board
- 4 AA batteries
- 3.3 LDO voltage regulator
- 5V LM78 voltage regulator

3.2 Materials necessary for development system

- USBtinyISP– USB AVR Programmer
- Target Board materials:
  - Prototyping perfboard
  - 2x3 DIP header
  - 28 pin DIP socket
- Atmega328p Xplained Mini Evaluation Kit (development board)
- Global Specialties Proto-Board

4 Design

To learn a new embedded development system and to make the debugging process easier, this project was completed in incremental steps. First, a trivial blinking LED hex file was programmed onto the microcontroller to understand how to program AVR in C. Next, code was written to be able to communicate with the driver of the LCD shield. This code was programmed and debugged through the Xplained Mini Evaluation Kit, explained further in Section 4.2. Once the LCD could be initialized, small chunks of text were written to it and functions for "turning a page" were written using text that was hard coded into the program. The last step of making the e-reader was to incorporate the SD card. Code was written and debugged once again through the Xplained to retrieve chunks of data from the SD card via SPI. These buffers of text replaced the hard coded text so that full books could be processed.

The code for this project was written in C, however to get the code onto the microcontroller, it had to be translated through the compiler to an ELF file and then to Intel Hex. The Hex file was then saved onto the flash of the microcontroller. Then, once the MCU is powered, the programmed code will take over and execute as desired.

4.1 Using the Microcontroller

The microcontroller was programmed to using the USBtinyISP and a target board, both assembled in the lab.

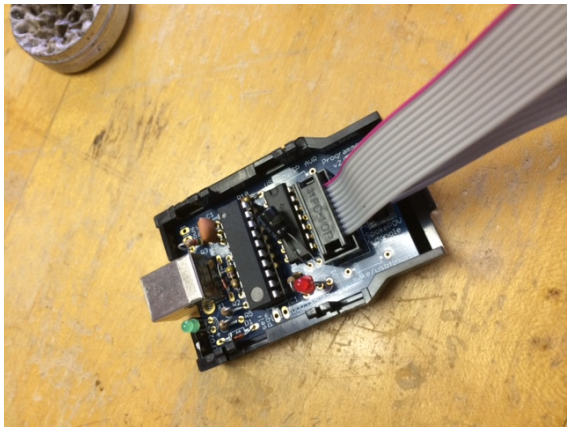

Figure 15: USBtinyISP after Assembly without Case

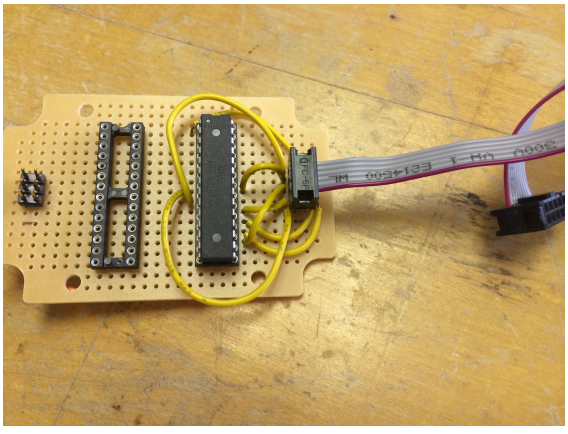

Figure 16: Programmer Connected to Microcontroller via Target Board

After a connection is established between the programmer and the microcontroller on the target board, the program must be compiled. In this case, both C and C++can be compiled either through

Atmel Studio or directly through the command line using avrdude. Once compiled, the code is turned into an ELF (executable and linkable format) file, then into an Intel Hex file. It is the Hex file that gets loaded onto the microcontroller.

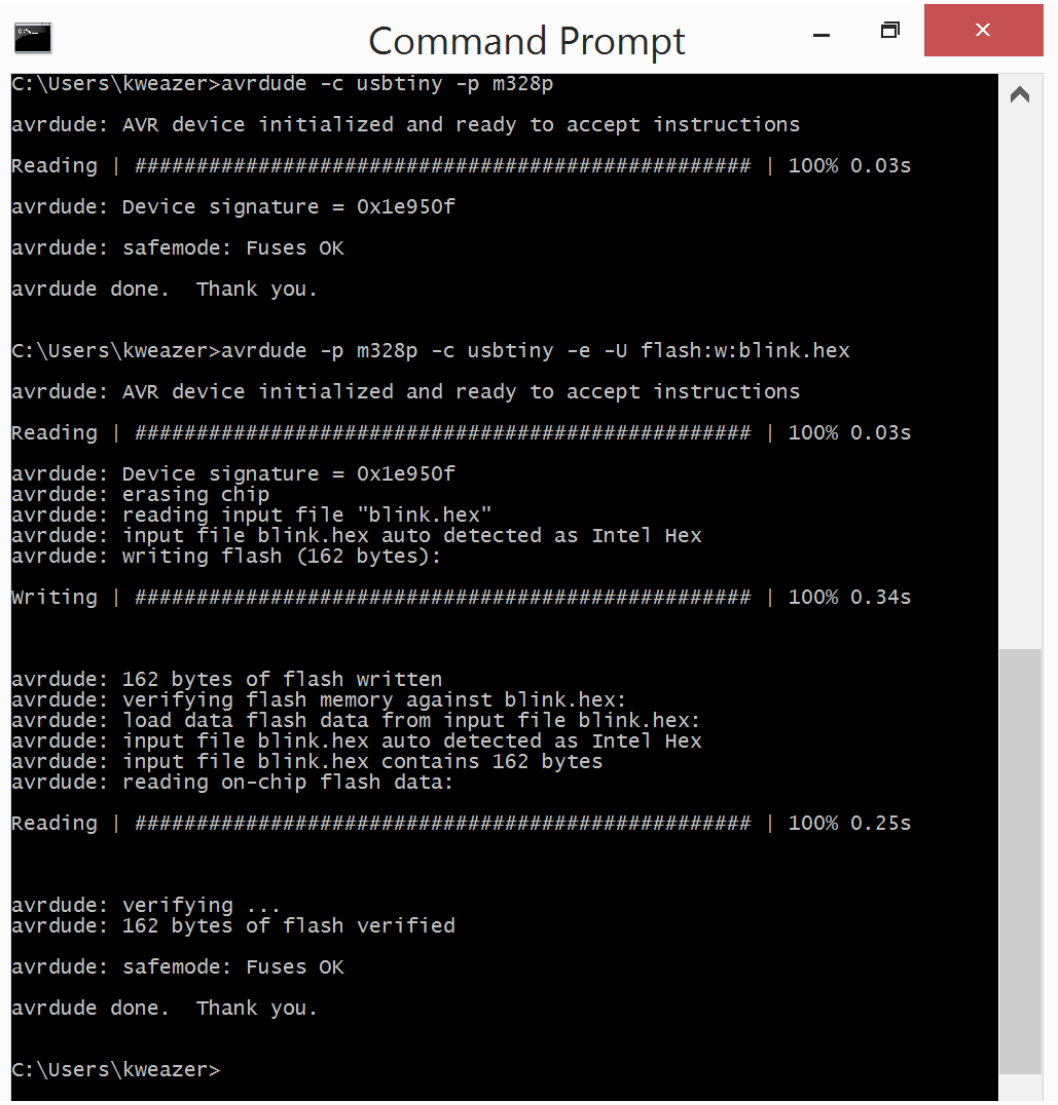

Figure 17: Programming blink.hex onto Microcontroller

Figure 17 shows the process for programming the file `blink.hex` onto a microcontroller. In the first command, the computer looks for a connection to the microcontroller and returns that a connection was established assuming the programmer is connected properly. The computer then uses the programmer to erase the current contents of the chip then writes the `blink.hex` file onto it.

4.2 Development Board and Breadboard

In order to effectively test and debug throughout this project, a development board also using the Atmega328p was purchased. Although an in system debugger would have been ideal, these debuggers tend to be very expensive. Atmel's Xplained Mini evaluation kit was chosen because of its on-board debugger and because it also uses the Atmega328p microcontroller. It also conveniently has Arduino header footprints which works well with the GLCD chosen.

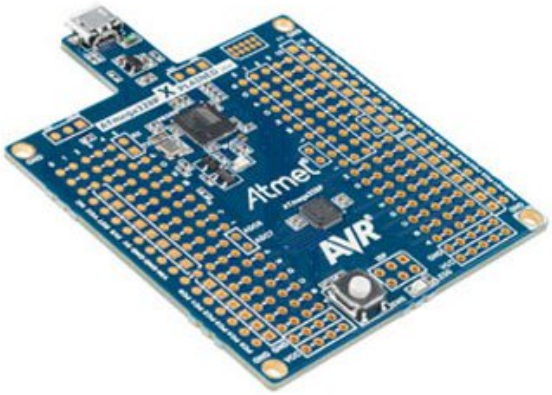

Figure 18: Atmel Xplained Mini

There are many benefits with working with the Xplained Mini, the debugger being a main one. Another benefit is that the GLCD Shield fits into the footprints of the Xplained easily without any wires, only soldering of launchpad connectors necessary. Figures 19 and 20 show how the Xplained could be adapted to connect to a GLCD. The Xplained is powered through a computer's USB port.

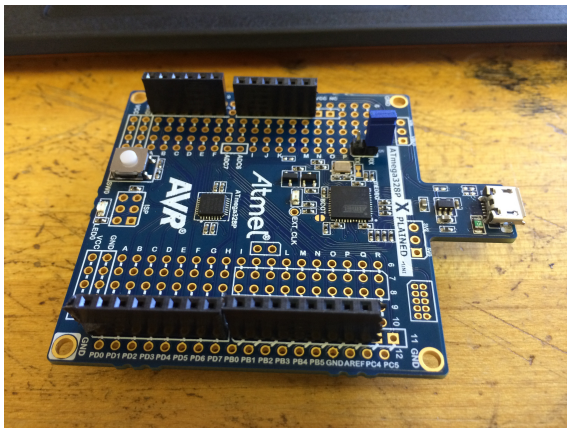

Figure 19: Launchpad Connectors for GLCD Shield on Xplained

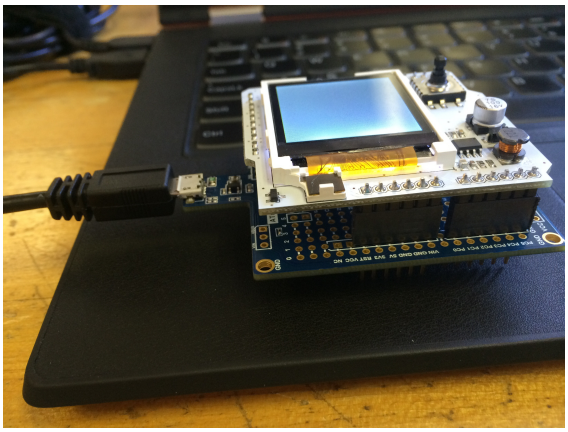

Figure 20: GLCD Connection to Xplained and Computer

All of the software code was tested and debugged on the Xplained using Atmel Studio. Once the software worked on the Xplained, the same hex file was programmed onto the Atmega328p and the same connections made through a breadboard. In theory, since the hex files were the same, as long as the breadboard connections matched, the outcome should be the same on both the Xplained and the breadboard prototype. The development board was not used in the final product since the cost of having the debugger on board would increase the total cost, defeating the purpose of a low cost e-reader. Supplying the correct power to the board and the other components (LCD and SD card) becomes more of a problem on the breadboard since the Xplained draws all of its power, both 5v and 3.3v, from the computer's USB port so these voltages have to be supplied separately to the breadboard.

The breadboard used for this project was capable of supplying 5 volts which ran along the top row. To get 3.3 volts, an external voltage supplier (Figure 21) was used and set to 3.3 volts.

4.3 Debugging Process

A majority of the time working on this project was spent on debugging the code. On the hardware side, a multimeter was used. To confirm solder connections, the multimeter would beep upon detecting a connection. The multimeter was also used to confirm voltage outputs from different sources, as well as to be able to tell if a pin was set high or low. At times when a multimeter was not available, a small LED was used as an indicator.

On the software side, the Atmel Xplained Mini's debugger was heavily used. The debugger allowed for stepping through the code one line at a time while also showing changes to flash memory and variables. By using the debugger, it became clear exactly what each line of code

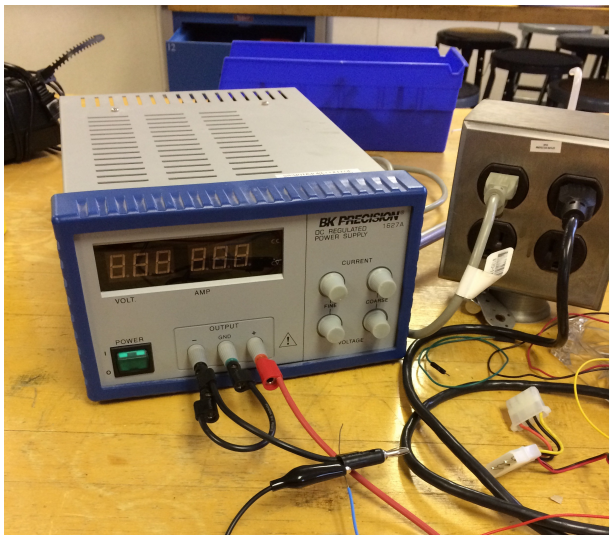

Figure 21: External 3.3 Voltage Supply for Breadboard

did since the hardware output would reflect it. For instance, when the backlight pin for the LCD was set high, the LCD screen would turn white. Many 'watches' were set, meaning the value of a variable would be shown in a separate window. This was especially useful for debugging the SD card interfacing since it became easy to tell which error flags the code was throwing. Watches were also unexpectedly useful to detect when the compiler optimized certain variables away that were actually needed.

Combining the step by step debugging with the abilities of the multimeter or debugging LED was also useful to see if pins were turning high or low when expected.

4.4 Connecting the LCD

The connections to the LCD were done via the schematic in Figure 4. Figure 22 shows these connections on the breadboard. The connections on the breadboard are much messier than those on the development board because of all of the wires to connect individual pins together. However, we would rather not use the development board in the final product because of its superflous expense– for instance, a debugger is not needed in the final product.

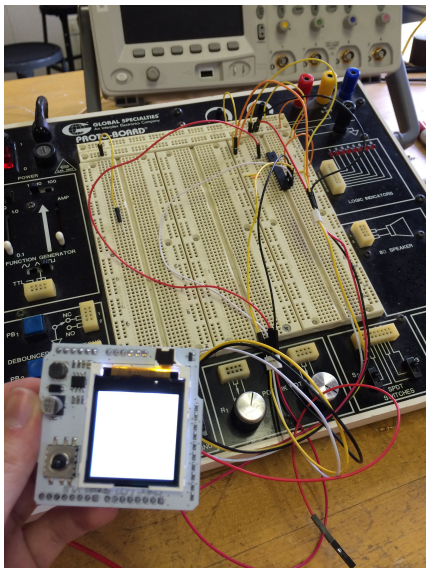

Figure 22: Atmega328p Connected to LCD via Breadboard

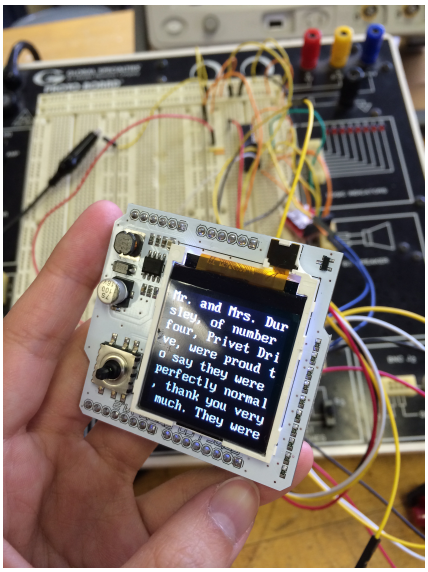

Figure 23: Hard Coded Text Displayed on LCD Screen

Figure 22 shows the LCD screen when connected to the breadboard with only the backlight turned on through the slave select pin. Once more sophisticated code was programmed onto the chip with the ability to write text to the screen, we get the output shown in Figure 23.[5] Once text that was hard coded could be displayed on the LCD, the next step was to read text from memory– in this case, our SD card.

4.5 Interfacing the SD Card

The connections to the SD card were made as described in Table 1. The library Petit FS was added to the existing AVR to LCD code. The software involved first initializing (`disk_initialize()`) the SD card to identify what kind of card it was (SD or SDHC), then checking if the clusters were where they were expected to be(`pf_mount()`). In this case, the code looks for SD cards formatted to FAT16. The next instruction, `pf_open()`, opens the file specified on the SD card by name. `pf_read()` takes three arguments– the first being the buffer of text that was read, the second the number of bytes to read, and the third an output of how many bytes were read.

```
1        FATFS fat;

2        fs_res=disk_initialize(1);
3        if(!fs_res){
```

[5]Text from *Harry Potter and the Sorcerer's Stone* by J.K. Rowling

```
4                   pf_mount(&fat);
5           }

6           fs_res=pf_open(``file_name.txt'');
7           char file_buf[16*8];
8           word bytes_read=0;
9           fs_res=pf_read(file_buf,16*8,&bytes_read);
```

The code above shows the entire initialization process on a high level. `fs_res` stores a flag as to whether or not a process was successful. Checks should be made at each step to make sure `fs_res` is still 0 and to handle telling users the error if one does come up. Lines 7 and 8 declare two of the arguments we pass into `pf_read`– the file buffer that will store our text, and the number of bytes read. We pass in $16 * 8$ for number of bytes to read since this is how many characters the LCD screen can see at one time.

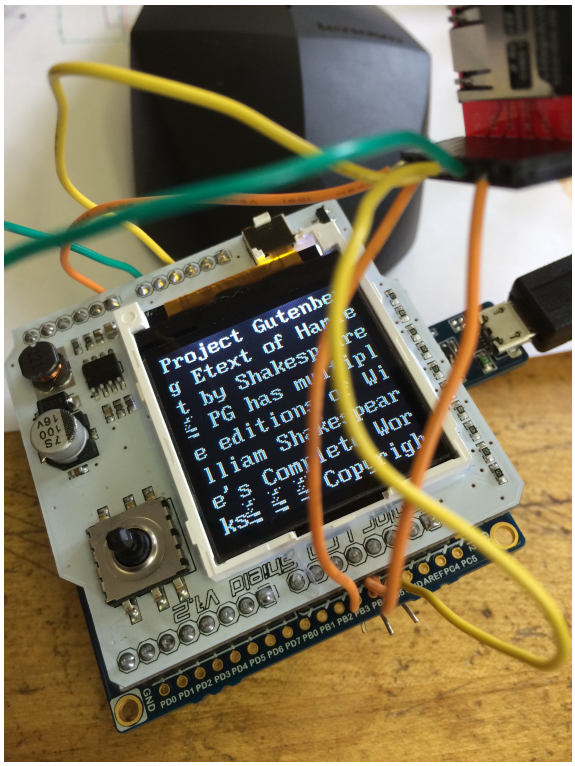

Figure 24: Reading from Project Gutenberg's copy of *Hamlet* by William Shakespeare

4.5.1 Page Turning Functionality

Because we read a buffer of $16 * 8$ characters at a time, this reading buffer by buffer lends itself well to page turning. To progress in a book, we would want to read the next $16 * 8$ characters after the ones we just read. However, `pf_read` always reads from where the file read/write pointer is. Without modifying it, the pointer will always be at the beginning of the file. Therefore, we use `pf_lseek` to move the pointer accordingly.

```
1       int pageNum=0;
2       fs_res=pf_lseek(16*8*pageNum);
3       fs_res=pf_read(file_buf,16*8, &bytes_read);
4       // display to the LCD
5       if page turning forward{
6               memset(&file_buf,0,sizeof(file_buf));
7               pageNum++;
8               fs_res=pf_lseek(16*8*pageNum);
9               fs_res=pf_read(file_buf,16*8,&bytes_read);
10              // display to the LCD
11      }
12      if page turning backward{
13              // same as page turning forward, but pageNum--;
14      }
```

From the pseudocode above, when a page turn is detected, we first clear the file buffer on line 6, increment the variable `pageNum`, then seek to the next unread section (line 8). The same behavior is necessary for going backwards in a book. The hardware connections are detailed in section 2.2.1.

4.6 Integrating and Porting

Once the software worked on the development board, it was programmed onto the microcontroller and the same connections were made through wires on a breadboard. A few differences between the development board and the breadboard versions were noted though without a reason or solution yet. The first problem noticed was the significant decrease in clock speed of writing to the LCD on the breadboard, where each character can be seen being written. On the development board, the words could appear and clear very quickly without noticeable time. Changing the clock speed in the code did not seem to alleviate this problem at all. The second problem was a problem where sometimes the LCD would show a white screen instead of the proper text. This could be

fixed by either resetting the microcontroller after all power was supplied, or by turning on the 3.3 voltage power supply first. This may be a power sequencing problem, though the exact problem has not been deduced yet. Because the Xplained Mini and the breadboard both used the exact MCUs and hex files, these problems should not be software related.

Besides these two problems, because the software was proven to work on the development board, transferring to the breadboard was done with minimal problems. The transition from breadboard power and voltage regulator power to a battery pack was also simple once the 5 volt and 3.3 volt regulators were implemented.

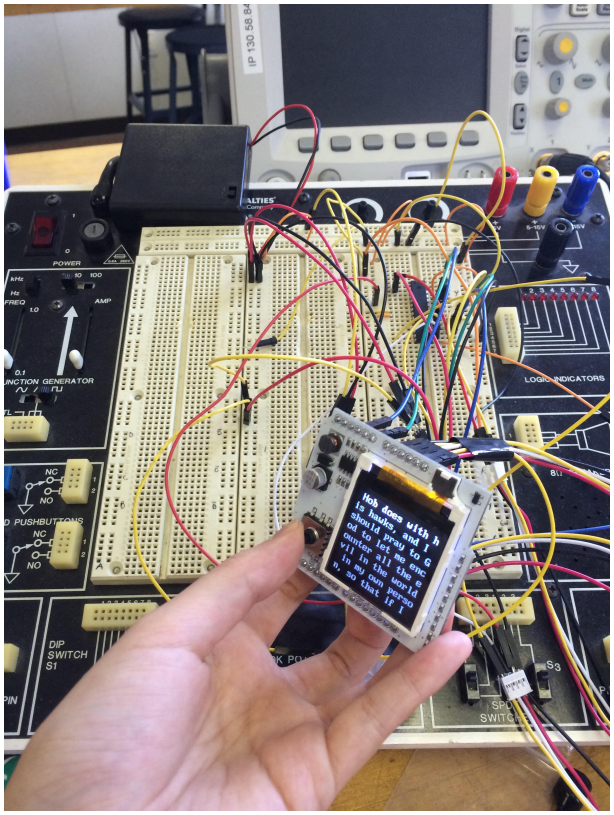

Figure 25: Breadboard with Battery Power Supply

Figure 25 shows the breadboard implementation without the 3.3 voltage power supply or the 5 volts from the breadboard. Instead, a battery pack replaced it shown in the top left corner of the figure.

Finally, Figure 26 shows everything disconnected from a breadboard and soldered together with wires instead. Ideally, a PCB would be used though given the limited time, the connections were simply made manually and were proven to work and could be carried around anywhere!

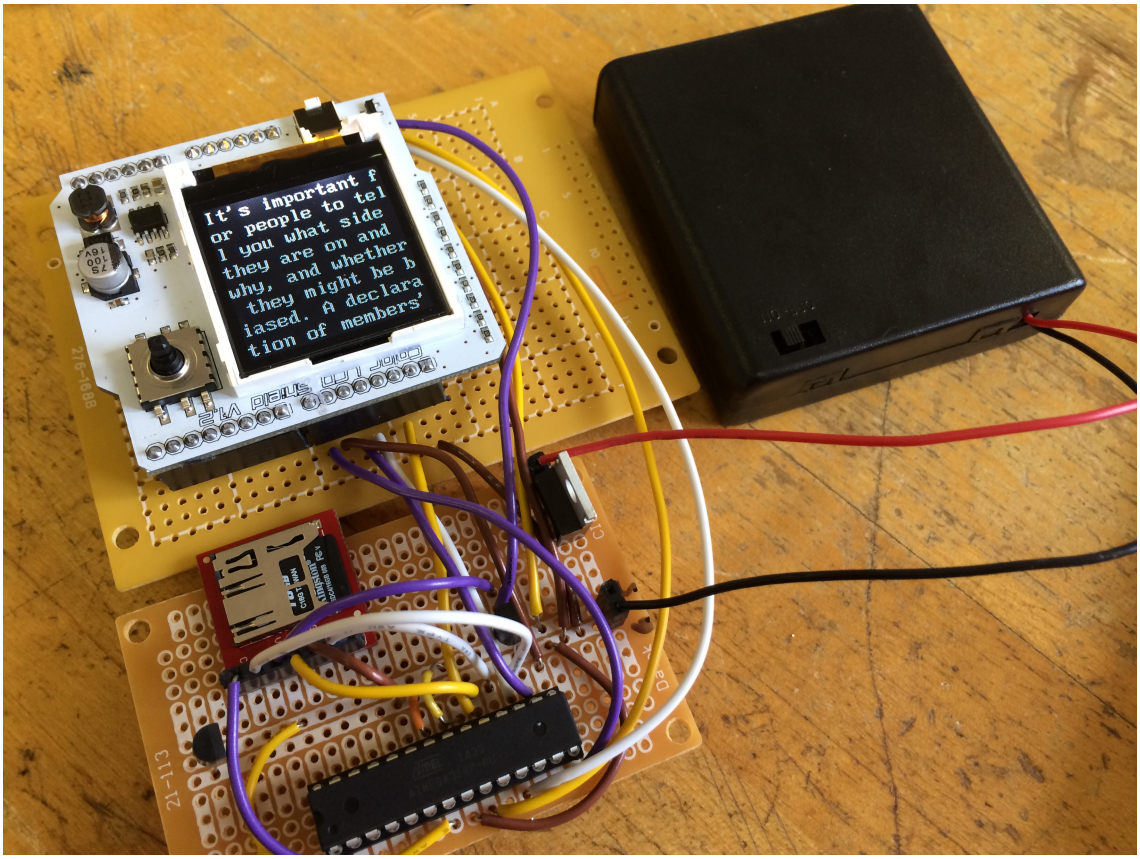

Figure 26: Making Portable

5 Results

5.1 Videos

The following videos show the steps taken in the development of this e-reader:

1. Atmel Xplained Mini Development Board test: `https://youtu.be/39xntkavszk`
2. Breadboard test: `https://youtu.be/blS-AOrP4rQ`
3. Standalone test (battery powered): `https://youtu.be/YBD-gafEGBU`

Though the same hex file code was programmed on each microcontroller and the connections were the same in theory, we can see that the product on the breadboard and the standalone are much slower than on the development board. In all of these videos, the files are being read from the SD card. However, it is hypothesized that the slower speed is not from the SD card but rather the LCD screen code, possibly due to clock speeds.

5.2 Final Cost

The total cost:

1. Atmega328p Microcontroller	$3.70 (Digi-Key)
2. Color LCD Shield	$22.00 (Elecfreaks)
3. Kingston Digital 16GB SDHC MicroSD Card	$5.99 (Amazon)
4. MicroSD Transflash Breakout Board	$9.95 (Sparkfun)
5. Duracell 4 AA Batteries	$4.29 (Amazon)
6. 5 Volt Regulator	$0.95 (Sparkfun)
7. 3.3 Volt Regulator	$1.95 (Sparkfun)
	Total: $48.83

The cost of wires and soldering connections were assumed to be trivial.

These costs were all calculated from exact prices to buy one unit of each product. $48.83 is within the hypothesized price and still rather expensive, though it should be kept in mind that these are the price to buy one unit and in practice should be much cheaper. The SD card purchased

had a much higher capacity than needed, though this was the cheapest one that could be found on Amazon from a trusted vendor at the time. It seems that the price of an e-reader remains largely determined by the cost of its screen, with e-paper screens being even more expensive.

6 Future Work

There is still more work that could be done to enhance this project. In the final implementation, the page number was not saved through the EEPROM though the theory was understood. Because of limited writes to EEPROM, a more efficient way than rewriting the EEPROM every time a page is advanced is needed. For instance, having a power down routine before the power is completely cut off that would save the page number only when the e-reader was about to be turned off, not every time the page is turned. However, right now the switch is on the power supply and having to move it somewhere else to give time to save to the EEPROM would require much more research. A workaround for this would be to have the page number save after the user presses a button on the joystick of the LCD shield, though this would require user input every time before turning the e-reader off and so would not be ideal.

More work could be done on the software side to make the e-reader more interactive, such as having options to choose which book to read out of the files stored on the SD card, the option to switch books through a main menu, etc. Right now the device can only read plain ASCII text files, though with a lot more coding ePub files should be able to be read as well. However, making a main menu is a software implementation that would not further learning much since we would simply have to call already in place SD card and LCD functions. Parsing ePub files is out of the scope of this project, though would certainly be a useful feature.

In theory, this project could be ported to a larger screen, though the datasheet of the larger screen would have to be carefully examined and much of the LCD code would need to be rewritten depending on the driver. The hardware connections would be similar through connecting the SPI pins. A larger screen would be better for reading.

A PCB (printed circuit board) could also be developed to connect the LCD screen, microcontroller, and SD cards more cleanly.

Finally, different battery sources could be investigated such as a lithium polymer battery, a common battery for embedded systems due to its light weight and small size. LiPo batteries also have the advantage of having very low internal discharge rates[6] and so can last for days in low power systems. They are most commonly foundin cell phones. Solar powered cells could also be investigated, with one cell costing about $1.50 and claiming to provide 0.5 volts. Solar would be a useful option especially for a low cost e-reader that could potentially be used in underdeveloped countries to encourage reading.

[6] `https://learn.sparkfun.com/tutorials/battery-technologies`

7 Conclusion

Overall, the most valuable skills learned through this project were understanding how to program to an AVR microcontroller, interfacing established components such as the LCD screen and the SD card to the microcontroller, and understanding on a low level how the SD card memory is read and manipulated by both the computer and the microcontroller. These three components were all successfully interfaced together and behaved more or less as expected, thanks to careful debugging on the development board and then porting to the breadboard.

Regarding cost, this low-cost e-reader is still a bit expensive at around $50 for a sum of all of the individual parts since it does much less than the Kindle of the Nook do. However, it is important to keep in mind that these are unit costs, and in an actual manufacturing stage each item should be much cheaper. Since all of the components except the LCD screen cost less than $10 on an individual basis, their costs should be nearly trivial on a large scale. This leads to the deduction that the screen is the most expensive part of the e-reader currently. Much like how early e-readers did not do well in the market due to limited battery life and memory capacity, current e-reader prices are high largely due to the expensiveness of screens, whether LCD or e-paper. Once we see the cost of screens go down, it is predicted that the price of e-readers will also decrease dramatically since then all of the components would be very inexpensive.

Therefore, it is difficult to say whether companies such as Amazon and Barnes & Noble overcharge their devices from this project. However, the cost to output text from a memory storage device to a screen is pretty low with a relatively small amount of code.

While technology to make screens remains expensive and large companies such as Amazon and Barnes & Noble dominate the e-reading market, e-readers may remain a luxury item. However, remembering that both memory and battery life have increased dramatically in the last few years, it is probable that the cost of an e-reader will one day be affordable to many more, and hopefully incorporated into schools and given to those who would otherwise not be able to read a good book.

> *"I don't believe in the kind of magic in my books. But I do believe something very magical can happen when you read a good book."*
>
> *-J.K. Rowling*

8 References

Bakelaar, B. "E-readers behind the Scenes." Princeton Library, 25 July 2012. Web. 30 Apr. 2015.

Bridle, James. "From Books to Infrastructure." *Domusweb.it.* Domus, 4 June 2012. Web. 30 Apr. 2015.

Camera, Dean. "Using the EEPROM memory in AVR-GCC" *Four Walled Cubicle* N.p, 15 March 2015. Web. 6 May 2015.

Carrick, Micah. "Getting Started: Blinking an LED." *AVR Microcontroller Tutorial.* Web. 27. Jan. 2015.

ChaN, Elm. "Petit FAT File System Module." *Elm-chan.* N.p., 10 June 2014. Web. 30 Apr. 2015.

"Color LCD Shield." *ElecFreaks.* N.p., 30 Nov. 2012. Web. 30 Apr. 2015.

Corporation, Atmel. "ATMEL 8-BIT MICROCONTROLLER WITH 4/8/16/32KBYTES IN-SYSTEM PROGRAMMABLE FLASH." *Datasheet Summary* (n.d.): n. pag. Web.

Fried, Limor. "USBtinyISP." *Overview.* Adafruit, 10 June 2013. Web. 30 Apr. 2015.

Gaiman, Neil. "Why Our Future Depends on Libraries, Reading and Daydreaming." *The Guardian.* N.p., 15 Oct. 2013. Web.

Grusin, Mike. "Serial Peripheral Interface (SPI)" *Sparkfun* N.p, Web. 6 May 2015.

Inc., Microchip Technology. "Low Quiescent Current LDO." *MCP1700 Data Sheet* (n.d.): n. pag. Web.

Inc., Texas Instruments. "LM78XX Series Voltage Regulators." *Texas Instruments Datasheets* (Rev. D) (n.d.): n. pag. May 2000. Web.

Kanda Admin. "What Is AVR Microcontroller?" *Kanda Electronics Blog.* N.p., 20 Dec. 2012. Web. 30 Apr. 2015.

Litvinovich, Artyom. "DIY Ebook Reader." *Orbides.* N.p., 6 Feb. 2015. Web. 30 Apr. 2015.

Oskay, Windell. "The Adafruit Industries USBtinyISP Kit." *Evil Mad Scientist Laboratories.* N.p., 13 June 2007. Web. 30 Apr. 2015.

Pihlajamaa, Joonas. "Simple FAT and SD Tutorial Part 1." *Code and Life.* N.p., 2 Apr. 2012. Web. 30 Apr. 2015.

Rowling, J. K. *Harry Potter and the Sorcerer's Stone.* New York: A.A. Levine, 1998. Print.

Seidle, Nate. "Battery Technologies." *Sparkfun.* N.p., n.d. Web. 30 Apr. 2015.

SD Card Association. "SD Specifications." *SD Card Association.* SDCard.org, 22, Jan. 2013. 8

May 2015.

Traylor, Roger. "Serial Peripheral Interface." *College of Engineering.* Oregon State University, Web. 30 Apr. 2015.

White, T. H. *The Once and Future King.* New York: Putnam, 1958. Print.

Frank, Zhao. "MMC/SD Card and FAT Tutorial." *Eleccelerator.* N.p., 01 Jan. 2013. Web. 30 Apr. 2015.

A C Source Code

A.1 main.c (includes LCD source code)

```
#define F_CPU 800UL
#include <stdlib.h> // substitute for Arduino.h
#include <string.h>
#include <math.h>
#include <util/delay.h>

#include <avr/pgmspace.h> // fun stuff
#include <avr/io.h>
#include <avr/interrupt.h>

/* SD Card Ports */
#define SD_CS 0
#define SD_PORT PORTD
#define SD_DDR DDRD


/* LCD stuff */
#define LCD_RES                 0
#define CS                              1
#define DIO                             3
#define SCK_PIN                 5
#define SS                              2

#define LCD_PORT_SS             PORTB
#define LCD_PORT_RES    PORTB
#define LCD_PORT_CS             PORTB
#define LCD_PORT_DIO    PORTB
#define LCD_PORT_SCK    PORTB

#include "sdlib/utils.h"
#include "sdlib/mmc.h"
#include "sdlib/pff.h"
#include "ColorLCDShield.h"

/*
  LCDShield.cpp - Arduino Library to control a Nokia 6100 LCD,
  specifically that found on SparkFun's Color LCD Shield.
  This code should work for both Epson and Phillips display drivers
  normally found on the Color LCD Shield.

  License: CC BY-SA 3.0: Creative Commons Share-alike 3.0. Feel free
  to use and abuse this code however you'd like. If you find it useful
  please attribute, and SHARE-ALIKE!

  This is based on code by Mark Sproul, and Peter Davenport.
  Thanks to Coleman Sellers and Harold Timmis for help getting it to work with the Phillips Driver 7-31-2011
*/

#include "ColorLCDShield.h"

/*extern "C" {
        #include "wiring.h"
}*/


//#define F_CPU 8000000UL // clock speed
#include <stdlib.h> // substitute for Arduino.h
#include <string.h>
#include <math.h>
#include <util/delay.h>

static char x_offset = 0;
static char y_offset = 0;


/* Constructor */
void LCDconstruct(void)
{
```

```
        /* Specific to Atmega328p */
        DDRB = ((1<<CS)|(1<<DIO)|(1<<SCK_PIN)|(1<<LCD_RES)|(1<<SS)); // Set the control pins as outputs for DDR
        //DDRD  =       0x00;   // makes all pins of PORTD as input
        DDRC    =       0X00;   // makes all pins of PORTC as input
        //PORTD =       0xFF;
        PORTC = 0xFF;
        //DDRD &= ~(1<<PD1);
}

void LCDCommand(unsigned char data)
{
        char jj;

        cbi(LCD_PORT_CS, CS);      // enable chip [LCD_CE_PORT]
        cbi(LCD_PORT_DIO, DIO);    // output low on data out (9th bit low = command)

        cbi(LCD_PORT_SCK, SCK_PIN);    // send clock pulse
        _delay_ms(1);
        sbi(LCD_PORT_SCK, SCK_PIN);

        for (jj = 0; jj < 8; jj++)
        {
                if ((data & 0x80) == 0x80)
                        sbi(LCD_PORT_DIO, DIO);
                else
                        cbi(LCD_PORT_DIO, DIO);

                cbi(LCD_PORT_SCK, SCK_PIN); // send clock pulse
                _delay_ms(1);
                sbi(LCD_PORT_SCK, SCK_PIN);

                data <<= 1;
        }

        sbi(LCD_PORT_CS, CS);      // disable
}

void LCDData(unsigned char data)
{
        char j;

        cbi(LCD_PORT_CS, CS);      // enable chip
        sbi(LCD_PORT_DIO, DIO);    // output high on data out (9th bit high = data)

        cbi(LCD_PORT_SCK, SCK_PIN);    // send clock pulse
        _delay_ms(1);
        sbi(LCD_PORT_SCK, SCK_PIN);    // send clock pulse

        for (j = 0; j < 8; j++)
        {
                if ((data & 0x80) == 0x80)
                        sbi(LCD_PORT_DIO, DIO);
                else
                        cbi(LCD_PORT_DIO, DIO);

                cbi(LCD_PORT_SCK, SCK_PIN); // send clock pulse
                _delay_ms(1);
                sbi(LCD_PORT_SCK, SCK_PIN);

                data <<= 1;
        }

        LCD_PORT_CS     |=      (1<<CS);  // disable
}

void LCDinit(int type)
{
        driver = type;

        _delay_ms(200);

        sbi(LCD_PORT_SS,SS);                    // turn on backlight through SS!
        cbi(LCD_PORT_SCK, SCK_PIN);     //CLK = LOW
        cbi(LCD_PORT_DIO, DIO);     //DIO = LOW
```

```
        _delay_ms(1000);
        sbi(LCD_PORT_CS, CS);         //CS = HIGH
        _delay_ms(1000);
        cbi(LCD_PORT_RES, LCD_RES); //RESET = LOW
        _delay_ms(20000);
        sbi(LCD_PORT_RES, LCD_RES); //RESET = HIGH
        _delay_ms(20000);
        sbi(LCD_PORT_SCK, SCK_PIN);       // SCK_PIN = HIGH
        sbi(LCD_PORT_DIO, DIO);       // DIO = HIGH
        _delay_ms(1000);

        LCDCommand(DISCTL);    // display control(EPSON)
        LCDData(0x0C);         // 12 = 1100 - CL dividing ratio [don't divide] switching period 8H (default)
        LCDData(0x20);
        LCDData(0x00);
        LCDData(0x01);

        LCDCommand(COMSCN);    // common scanning direction(EPSON)
        LCDData(0x01);

        LCDCommand(OSCON);     // internal oscialltor ON(EPSON)

        LCDCommand(SLPOUT);    // sleep out(EPSON)
        LCDCommand(SLEEPOUT); //sleep out(PHILLIPS)

        LCDCommand(PWRCTR);    // power ctrl(EPSON)
        LCDData(0x0F);         //everything on, no external reference resistors
        LCDCommand(BSTRON);    //Booset On(PHILLIPS)

        LCDCommand(DISINV);    // invert display mode(EPSON)

        LCDCommand(DATCTL);    // data control(EPSON)
        LCDData(0x03);         // correct for normal sin7
        LCDData(0x00);         // normal RGB arrangement
        LCDData(0x02);         // 16-bit Grayscale Type A

        LCDCommand(COLMOD);    // Set Color Mode(PHILLIPS)
        LCDData(0x03);

        LCDCommand(MADCTL);    // Memory Access Control(PHILLIPS)
        LCDData(0xC0);


        LCDCommand(VOLCTR);    // electronic volume, this is the contrast/brightness(EPSON)
        LCDData(0x24);         // volume (contrast) setting - fine tuning, original
        LCDData(0x03);         // internal resistor ratio - coarse adjustment
        LCDCommand(SETCON);    // Set LCDcontrast(PHILLIPS)
        LCDData(0x30);

        LCDCommand(NOP);       // nop(EPSON)
        LCDCommand(NOPP);      // nop(PHILLIPS)

        _delay_ms(200);

        LCDCommand(DISON);     // display on(EPSON)
        LCDCommand(DISPON);    // display on(PHILLIPS)
}

void LCDclear(int color)
{
        if (driver) // if it's an Epson
        {
                LCDCommand(PASET);
                LCDData(0);
                LCDData(131);

                LCDCommand(CASET);
                LCDData(0);
                LCDData(131);

                LCDCommand(RAMWR);
        }
        else // otherwise it's a phillips
        {
```

```
                LCDCommand(PASETP);
                LCDData(0);
                LCDData(131);

                LCDCommand(CASETP);
                LCDData(0);
                LCDData(131);

                LCDCommand(RAMWRP);
        }

        for(unsigned int i=0; i < (131*131)/2; i++)
        {
                LCDData((color>>4)&0x00FF);
                LCDData(((color&0x0F)<<4)|(color>>8));
                LCDData(color&0x0FF);
        }

        x_offset = 0;
        y_offset = 0;
}

void LCDcontrast(char setting)
{
        LCDCommand(VOLCTR);         // electronic volume, this is the contrast/brightness(EPSON)
        LCDData(setting);           // volume (contrast) setting - course adjustment, -- original was 24

        LCDCommand(NOP);            // nop(EPSON)
}

void LCDsetPixel(int color, unsigned char x, unsigned char y)
{
        y       =       (COL_HEIGHT - 1) - y;
        x = (ROW_LENGTH - 1) - x;

        if (driver) // if it's an epson
        {
                LCDCommand(PASET);  // page start/end ram
                LCDData(x);
                LCDData(ENDPAGE);

                LCDCommand(CASET);  // column start/end ram
                LCDData(y);
                LCDData(ENDCOL);

                LCDCommand(RAMWR);  // write
                LCDData((color>>4)&0x00FF);
                LCDData(((color&0x0F)<<4)|(color>>8));
                LCDData(color&0x0FF);
        }
        else  // otherwise it's a phillips
        {
                LCDCommand(PASETP); // page start/end ram
                LCDData(x);
                LCDData(x);

                LCDCommand(CASETP); // column start/end ram
                LCDData(y);
                LCDData(y);

                LCDCommand(RAMWRP); // write

                LCDData((unsigned char)((color>>4)&0x00FF));
                LCDData((unsigned char)(((color&0x0F)<<4)|0x00));
        }
}


void LCDsetChar(char c, int x, int y, int fColor, int bColor)
{
        y       =       (COL_HEIGHT - 1) - y; // make display "right" side up
        x       =       (ROW_LENGTH - 2) - x;

        int             i,j;
```

```
unsigned int     nCols;
unsigned int     nRows;
unsigned int     nBytes;
unsigned char    PixelRow;
unsigned char    Mask;
unsigned int     Word0;
unsigned int     Word1;
unsigned char    *pFont;
unsigned char    *pChar;

// get pointer to the beginning of the selected font table
pFont = (unsigned char *)FONT8x16;
// get the nColumns, nRows and nBytes
nCols = *pFont;
nRows = *(pFont + 1);
nBytes = *(pFont + 2);
// get pointer to the last byte of the desired character
pChar = pFont + (nBytes * (c - 0x1F)) + nBytes - 1;

if (driver)      // if it's an epson
{
        // Row address set (command 0x2B)
        LCDCommand(PASET);
        LCDData(x);
        LCDData(x + nRows - 1);
        // Column address set (command 0x2A)
        LCDCommand(CASET);
        LCDData(y);
        LCDData(y + nCols - 1);

        // WRITE MEMORY
        LCDCommand(RAMWR);
        // loop on each row, working backwards from the bottom to the top
        for (i = nRows - 1; i >= 0; i--) {
                // copy pixel row from font table and then decrement row
                PixelRow = *pChar++;
                // loop on each pixel in the row (left to right)
                // Note: we do two pixels each loop
                Mask = 0x80;
                for (j = 0; j < nCols; j += 2)
                {
                        // if pixel bit set, use foreground color; else use the background color
                        // now get the pixel color for two successive pixels
                        if ((PixelRow & Mask) == 0)
                                Word0 = bColor;
                        else
                                Word0 = fColor;
                        Mask = Mask >> 1;
                        if ((PixelRow & Mask) == 0)
                                Word1 = bColor;
                        else
                                Word1 = fColor;
                        Mask = Mask >> 1;
                        // use this information to output three data bytes
                        LCDData((Word0 >> 4) & 0xFF);
                        LCDData(((Word0 & 0xF) << 4) | ((Word1 >> 8) & 0xF));
                        LCDData(Word1 & 0xFF);
                }
        }
}
else
{
        // Row address set (command 0x2B)
        LCDCommand(PASETP);
        LCDData(x);
        LCDData(x + nRows - 1);
        // Column address set (command 0x2A)
        LCDCommand(CASETP);
        LCDData(y);
        LCDData(y + nCols - 1);

        // WRITE MEMORY
        LCDCommand(RAMWRP);
        // loop on each row, working backwards from the bottom to the top
```

```
                pChar+=nBytes-1;  // stick pChar at the end of the row - gonna reverse print on phillips
                for (i = nRows - 1; i >= 0; i--) {
                        // copy pixel row from font table and then decrement row
                        PixelRow = *pChar--;
                        // loop on each pixel in the row (left to right)
                        // Note: we do two pixels each loop
                        Mask = 0x01;  // <- opposite of epson
                        for (j = 0; j < nCols; j += 2)
                        {
                                // if pixel bit set, use foreground color; else use the background color
                                // now get the pixel color for two successive pixels
                                if ((PixelRow & Mask) == 0)
                                        Word0 = bColor;
                                else
                                        Word0 = fColor;
                                Mask = Mask << 1; // <- opposite of epson
                                if ((PixelRow & Mask) == 0)
                                        Word1 = bColor;
                                else
                                        Word1 = fColor;
                                Mask = Mask << 1; // <- opposite of epson
                                // use this information to output three data bytes
                                LCDData((Word0 >> 4) & 0xFF);
                                LCDData(((Word0 & 0xF) << 4) | ((Word1 >> 8) & 0xF));
                                LCDData(Word1 & 0xFF);
                        }
                }
        }
}

void LCDsetStr(char *pString, int x, int y, int fColor, int bColor)
{
        x = x + 16;
        y = y + 8;

        // loop until null-terminator is seen
        while (*pString != 0x00) {
                /*if(x>128 && (PIND & ((1<<PD0))==1)){ // page turn functionality...
                        //_delay_ms(262);
                        LCDclear(bColor);
                        x=16;
                        y=0;
                }*/
                /*if(y==128 && *pString!=' '){ // if we are at the end of the line and the word is not finished
                        LCDsetChar('-',x,y,fColor,bColor);                // draw a dash
                }
                else if(y==120 && *pString==' '){ // space on second to last character so just go to next line- no dash nee
                        y=8;
                        x=x+16;
                        *pString++;
                        charNum++; // I think
                }*/
                //else{                                                 // otherwise write the character!
                        LCDsetChar(*pString++, x, y, fColor, bColor);
                        //charNum++;
                //}
                // advance the y position
                y = y + 8;
                // go to the next line
                if (y > 131){
                        y=8; // reset to start
                        x=x+16; // advance x position down one line
                }
        }
        //return charNum;
}

void LCDoff(void)
{
        if (driver)      // If it's an epson
                LCDCommand(DISOFF);
        else // otherwise it's a phillips
                LCDCommand(DISPOFF);
}
```

```
void LCDon(void)
{
        if (driver)      // If it's an epson
                LCDCommand(DISON);
        else // otherwise it's a phillips
                LCDCommand(DISPON);
}

void LCDsetPage(char *text){
        LCDclear(BLACK);
        int x=16;
        int y=8;
        while(*text != 0x00){
                LCDsetChar(*text++, x, y, WHITE, BLACK);
                // advance the y position
                y = y + 8;
                // go to the next line
                if (y > 131){
                        y=8; // reset to start
                        x=x+16; // advance x position down one line
                }
        }
}
```

```
int main(){
        int pageNum=0;
        /* Initialization of LCD stuff */
        DDRB = ((1<<CS)|(1<<DIO)|(1<<SCK_PIN)|(1<<LCD_RES)|(1<<SS)); // Set the control pins as outputs for DDR
        DDRC     =        0X00;    // makes all pins of PORTC as input
        PORTC = 0xFF;

        FATFS fat;
        set_bit(PORTB,CS); // set_bit(LCD_DDR,LCD_CE_PIN)
        set_bit(DDRD,SD_CS);
        set_bit(PORTD,SD_CS);

        SPCR=0x52; // SPI control register
        SPSR=0x01; // SPI status register

        byte fs_res = disk_initialize(1);
        if(!fs_res){
                pf_mount(&fat);
        }

        /* open file */
        fs_res = pf_open("ALLISON.txt");

        /* seek within file*/
        fs_res=pf_lseek(16*8*pageNum);

        /* read data to the memory */
        char file_buf[16*8];
        word bytes_read=0;
        fs_res=pf_read(file_buf,16*8,&bytes_read);
        /* testing */
        //pageNum++;
        //fs_res=pf_lseek(16*8*pageNum);
        //fs_res=pf_read(file_buf,16*8,&bytes_read);
        //pageNum++;
        //fs_res=pf_lseek(16*8*pageNum);
        //fs_res=pf_read(file_buf,16*8,&bytes_read);


        //pageNum++;
        //fs_res=pf_lseek(16*8*pageNum);
        //fs_res=pf_read(file_buf,16*8,&bytes_read);

        /* end of testing- if fs_res=0, problem with switching SPCR? */
```

```
        SPCR=0x00;

        LCDinit(PHILLIPS);
        LCDclear(BLACK);
        LCDcontrast(40);
        //char* text="Mr. and Mrs. Dursley, of number four, Privet Drive, were proud to say they were perfectly normal, tha
        //LCDsetPage(text,pageNum);
        LCDsetPage(file_buf);
        //int end=LCDsetPage(file_buf,pageNum);
        while(1){
                if(!(PINC & (1<<PC0))){
                        memset(&file_buf,0,sizeof(file_buf));
                        pageNum++;
                        // seek, then read
                        SPCR = 0x52;
                        fs_res=pf_lseek(16*8*pageNum);
                        fs_res=pf_read(file_buf,16*8,&bytes_read);
                        SPCR=0x00;
                        LCDsetPage(file_buf);
                }
                else if(!(PINC & (1<<PC2))){
                        if(pageNum>0){
                                memset(&file_buf,0,sizeof(file_buf));
                                pageNum--;
                                SPCR=0x52;
                                fs_res=pf_lseek(16*8*pageNum);
                                fs_res=pf_read(file_buf,16*8,&bytes_read);
                                SPCR=0x00;
                                LCDsetPage(file_buf);
                        }
                }
                else{
                        _delay_ms(1);
                }
        }

        return 0;
}
```

A.2 ColorLCDShield.h

```
/*
  ColorLCDShield.h - Arduino Library to control a Nokia 6100 LCD,
  specifically that found on SparkFun's Color LCD Shield.
  This code should work for both Epson and Phillips display drivers
  normally found on the Color LCD Shield.

  License: CC BY-SA 3.0: Creative Commons Share-alike 3.0. Feel free
  to use and abuse this code however you'd like. If you find it useful
  please attribute, and SHARE-ALIKE!

  This is based on code by Mark Sproul, and Peter Davenport.
*/

#ifndef ColorLCDShield_H
#define ColorLCDShield_H

#define PHILLIPS        0
#define EPSON           1

//#include <WProgram.h>

#include <inttypes.h>

//*************************************************
//                                          SD Card Ports
//*************************************************
//#define SD_CS 0
//#define SD_PORT PORTC
//#define SD_DDR DDRC


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

```
//                                                  Macros
//*******************************************************
#define sbi(var, mask)   ((var) |= (uint8_t)(1 << mask))
#define cbi(var, mask)   ((var) &= (uint8_t)~(1 << mask))

//*****************************************************************
//
//                                      LCD Dimension Definitions
//
//*****************************************************************
#define ROW_LENGTH      132
#define COL_HEIGHT      132
#define ENDPAGE     132
#define ENDCOL      130

//*****************************************************************
//
//                                      EPSON Controller Definitions
//
//*****************************************************************
#define DISON       0xAF        // Display on
#define DISOFF      0xAE        // Display off
#define DISNOR      0xA6        // Normal display
#define DISINV      0xA7        // Inverse display
#define SLPIN       0x95        // Sleep in
#define SLPOUT      0x94        // Sleep out
#define COMSCN      0xBB        // Common scan direction
#define DISCTL      0xCA        // Display control
#define PASET       0x75        // Page address set
#define CASET       0x15        // Column address set
#define DATCTL      0xBC        // Data scan direction, etc.
#define RGBSET8     0xCE        // 256-color position set
#define RAMWR       0x5C        // Writing to memory
#define RAMRD       0x5D        // Reading from memory
#define PTLIN       0xA8        // Partial display in
#define PTLOUT      0xA9        // Partial display out
#define RMWIN       0xE0        // Read and modify write
#define RMWOUT      0xEE        // End
#define ASCSET      0xAA        // Area scroll set
#define SCSTART     0xAB        // Scroll start set
#define OSCON       0xD1        // Internal oscillation on
#define OSCOFF      0xD2        // Internal osciallation off
#define PWRCTR      0x20        // Power control
#define VOLCTR      0x81        // Electronic volume control
#define VOLUP       0xD6        // Increment electronic control by 1
#define VOLDOWN     0xD7        // Decrement electronic control by 1
#define TMPGRD      0x82        // Temperature gradient set
#define EPCTIN      0xCD        // Control EEPROM
#define EPCOUT      0xCC        // Cancel EEPROM control
#define EPMWR       0xFC        // Write into EEPROM
#define EPMRD       0xFD        // Read from EEPROM
#define EPSRRD1     0x7C        // Read register 1
#define EPSRRD2     0x7D        // Read register 2
#define NOP         0x25        // No op

//*****************************************************************
//
//                      PHILLIPS Controller Definitions
//
//*****************************************************************
//LCD Commands
#define NOPP            0x00    // No operation
#define BSTRON          0x03    // Booster voltage on
#define SLEEPIN     0x10        // Sleep in
#define SLEEPOUT        0x11    // Sleep out
#define NORON           0x13    // Normal display mode on
#define INVOFF          0x20    // Display inversion off
#define INVON           0x21    // Display inversion on
#define SETCON          0x25    // Set contrast
#define DISPOFF     0x28        // Display off
#define DISPON      0x29        // Display on
#define CASETP      0x2A        // Column address set
#define PASETP      0x2B        // Page address set
#define RAMWRP      0x2C        // Memory write
```

```
#define RGBSET       0x2D          // Color set
#define MADCTL           0x36      // Memory data access control
#define COLMOD           0x3A      // Interface pixel format
#define DISCTR       0xB9          // Super frame inversion
#define EC                         0xC0     // Internal or external oscillator

//*****************************************************
//                            12-Bit Color Definitions
//*****************************************************
#define BLACK            0x000
#define NAVY             0x008
#define BLUE             0x00F
#define TEAL             0x088
#define EMERALD          0x0C5
#define GREEN            0x0F0
#define CYAN             0x0FF
#define SLATE            0x244
#define INDIGO           0x408
#define TURQUOISE        0x4ED
#define OLIVE            0x682
#define MAROON           0x800
#define PURPLE           0x808
#define GRAY             0x888
#define SKYBLUE          0x8CE
#define BROWN            0xB22
#define CRIMSON          0xD13
#define ORCHID           0xD7D
#define RED                        0xF00
#define MAGENTA          0xF0F
#define ORANGE           0xF40
#define PINK             0xF6A
#define CORAL            0xF75
#define SALMON           0xF87
#define GOLD             0xFD0
#define YELLOW           0xFF0
#define WHITE            0xFFF

//*****************************************************
//                            Circle Definitions
//*****************************************************
#define FULLCIRCLE 1
#define OPENSOUTH  2
#define OPENNORTH  3
#define OPENEAST   4
#define OPENWEST   5
#define OPENNORTHEAST 6
#define OPENNORTHWEST 7
#define OPENSOUTHEAST 8
#define OPENSOUTHWEST 9

#define LCD_RES                    0
#define CS                                 1
#define DIO                                3
#define SCK_PIN                    5
#define SS                                 2

#define LCD_PORT_SS                PORTB
#define LCD_PORT_RES     PORTB
#define LCD_PORT_CS                PORTB
#define LCD_PORT_DIO     PORTB
#define LCD_PORT_SCK     PORTB

const unsigned char FONT8x16[97][16] = {
{0x08,0x10,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},       // columns, rows, bytes, ...
{0x08,0x10,0x10,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},       // columns, rows, bytes, ...
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},       //  ' '
{0x00,0x00,0x18,0x3C,0x3C,0x3C,0x18,0x18,0x18,0x00,0x18,0x18,0x00,0x00,0x00,0x00},       //  '!'
{0x00,0x63,0x63,0x63,0x22,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},       //  '"'
{0x00,0x00,0x00,0x36,0x36,0x7F,0x36,0x36,0x36,0x7F,0x36,0x36,0x00,0x00,0x00,0x00},       //  '#'
{0x0C,0x0C,0x3E,0x63,0x61,0x60,0x3E,0x03,0x03,0x43,0x63,0x3E,0x0C,0x0C,0x00,0x00},       //  '$'
{0x00,0x00,0x00,0x00,0x00,0x61,0x63,0x06,0x0C,0x18,0x33,0x63,0x00,0x00,0x00,0x00},       //  '%'
{0x00,0x00,0x00,0x1C,0x36,0x36,0x1C,0x3B,0x6E,0x66,0x66,0x3B,0x00,0x00,0x00,0x00},
{0x00,0x30,0x30,0x30,0x60,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x0C,0x18,0x18,0x30,0x30,0x30,0x30,0x18,0x18,0x0C,0x00,0x00,0x00,0x00},
```

```
{0x00,0x00,0x18,0x0C,0x0C,0x06,0x06,0x06,0x06,0x0C,0x0C,0x18,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x42,0x66,0x3C,0xFF,0x3C,0x66,0x42,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x18,0x18,0x18,0xFF,0x18,0x18,0x18,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x18,0x30,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00},
{0x00,0x00,0x01,0x03,0x07,0x0E,0x1C,0x38,0x70,0xE0,0xC0,0x80,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x63,0x63,0x6B,0x6B,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00},   // '0'
{0x00,0x00,0x0C,0x1C,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3F,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x03,0x06,0x0C,0x18,0x30,0x61,0x63,0x7F,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x03,0x03,0x1E,0x03,0x03,0x03,0x63,0x3E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x06,0x0E,0x1E,0x36,0x66,0x66,0x7F,0x06,0x06,0x0F,0x00,0x00,0x00,0x00},
{0x00,0x00,0x7F,0x60,0x60,0x60,0x7E,0x03,0x03,0x63,0x73,0x3E,0x00,0x00,0x00,0x00},   // '5'
{0x00,0x00,0x1C,0x30,0x60,0x60,0x7E,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x7F,0x63,0x03,0x06,0x06,0x0C,0x0C,0x18,0x18,0x18,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x63,0x63,0x3E,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x3F,0x03,0x03,0x06,0x3C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x00},   // ':'
{0x00,0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,0x00,0x18,0x18,0x18,0x30,0x00,0x00},
{0x00,0x00,0x00,0x06,0x0C,0x18,0x30,0x60,0x30,0x18,0x0C,0x06,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x7E,0x00,0x00,0x7E,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x60,0x30,0x18,0x0C,0x06,0x0C,0x18,0x30,0x60,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x63,0x06,0x0C,0x0C,0x0C,0x00,0x0C,0x0C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x63,0x6F,0x6B,0x6B,0x6E,0x60,0x60,0x3E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x08,0x1C,0x36,0x63,0x63,0x63,0x7F,0x63,0x63,0x63,0x00,0x00,0x00,0x00},   // 'A'
{0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x33,0x33,0x33,0x33,0x7E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x1E,0x33,0x61,0x60,0x60,0x60,0x60,0x61,0x33,0x1E,0x00,0x00,0x00,0x00},   // 'C'
{0x00,0x00,0x7C,0x36,0x33,0x33,0x33,0x33,0x33,0x33,0x36,0x7C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x7F,0x33,0x31,0x34,0x3C,0x34,0x30,0x31,0x33,0x7F,0x00,0x00,0x00,0x00},
{0x00,0x00,0x7F,0x33,0x31,0x34,0x3C,0x34,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00},
{0x00,0x00,0x1E,0x33,0x61,0x60,0x60,0x6F,0x63,0x63,0x37,0x1D,0x00,0x00,0x00,0x00},
{0x00,0x00,0x63,0x63,0x63,0x63,0x7F,0x63,0x63,0x63,0x63,0x63,0x00,0x00,0x00,0x00},   // 'H'
{0x00,0x00,0x3C,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x0F,0x06,0x06,0x06,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x73,0x33,0x36,0x36,0x3C,0x36,0x36,0x33,0x33,0x73,0x00,0x00,0x00,0x00},
{0x00,0x00,0x78,0x30,0x30,0x30,0x30,0x30,0x30,0x31,0x33,0x7F,0x00,0x00,0x00,0x00},
{0x00,0x00,0x63,0x77,0x7F,0x6B,0x63,0x63,0x63,0x63,0x63,0x63,0x00,0x00,0x00,0x00},
{0x00,0x00,0x63,0x63,0x73,0x7B,0x7F,0x6F,0x67,0x63,0x63,0x63,0x00,0x00,0x00,0x00},
{0x00,0x00,0x1C,0x36,0x63,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x30,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x63,0x63,0x6B,0x6F,0x3E,0x06,0x07,0x00,0x00},
{0x00,0x00,0x7E,0x33,0x33,0x33,0x3E,0x36,0x36,0x33,0x33,0x73,0x00,0x00,0x00,0x00},
{0x00,0x00,0x3E,0x63,0x63,0x30,0x1C,0x06,0x03,0x63,0x63,0x3E,0x00,0x00,0x00,0x00},
{0x00,0x00,0xFF,0xDB,0x99,0x18,0x18,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x63,0x63,0x36,0x1C,0x08,0x00,0x00,0x00,0x00},
{0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x6B,0x6B,0x7F,0x36,0x36,0x00,0x00,0x00,0x00},
{0x00,0x00,0xC3,0xC3,0x66,0x3C,0x18,0x18,0x3C,0x66,0xC3,0xC3,0x00,0x00,0x00,0x00},
{0x00,0x00,0xC3,0xC3,0xC3,0x66,0x3C,0x18,0x18,0x18,0x18,0x3C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x7F,0x63,0x43,0x06,0x0C,0x18,0x30,0x61,0x63,0x7F,0x00,0x00,0x00,0x00},   // 'Z'
{0x00,0x00,0x3C,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x30,0x3C,0x00,0x00,0x00,0x00},
{0x00,0x00,0x80,0xC0,0xE0,0x70,0x38,0x1C,0x0E,0x07,0x03,0x01,0x00,0x00,0x00,0x00},   // '\'
{0x00,0x00,0x3C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x3C,0x00,0x00,0x00,0x00},
{0x08,0x1C,0x36,0x63,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xFF,0x00,0x00,0x00},   // '^'
{0x18,0x18,0x0C,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x3C,0x46,0x06,0x3E,0x66,0x66,0x3B,0x00,0x00,0x00,0x00},   // '_'
{0x00,0x00,0x70,0x30,0x30,0x3C,0x36,0x33,0x33,0x33,0x33,0x6E,0x00,0x00,0x00,0x00},   // '`'
{0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x60,0x60,0x60,0x63,0x3E,0x00,0x00,0x00,0x00},   // 'a'
{0x00,0x00,0x0E,0x06,0x06,0x1E,0x36,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x7E,0x60,0x63,0x3E,0x00,0x00,0x00},
{0x00,0x00,0x1C,0x36,0x32,0x30,0x7C,0x30,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x66,0x3C,0x00,0x00},
{0x00,0x00,0x70,0x30,0x30,0x36,0x3B,0x33,0x33,0x33,0x33,0x73,0x00,0x00,0x00,0x00},
{0x00,0x00,0x0C,0x0C,0x00,0x1C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x06,0x06,0x00,0x0E,0x06,0x06,0x06,0x06,0x66,0x66,0x3C,0x00,0x00,0x00},
{0x00,0x00,0x70,0x30,0x30,0x33,0x33,0x36,0x3C,0x36,0x33,0x73,0x00,0x00,0x00,0x00},
{0x00,0x00,0x1C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x0C,0x1E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x6E,0x7F,0x6B,0x6B,0x6B,0x6B,0x6B,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x33,0x33,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x63,0x63,0x63,0x63,0x3E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x6E,0x33,0x33,0x33,0x33,0x3E,0x30,0x30,0x78,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x3B,0x66,0x66,0x66,0x66,0x3E,0x06,0x06,0x0F,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x6E,0x3B,0x33,0x30,0x30,0x30,0x78,0x00,0x00,0x00,0x00},
```

```
{0x00,0x00,0x00,0x00,0x00,0x3E,0x63,0x38,0x0E,0x03,0x63,0x3E,0x00,0x00,0x00,0x00},
{0x00,0x00,0x08,0x18,0x18,0x7E,0x18,0x18,0x18,0x18,0x1B,0x0E,0x00,0x00,0x00,0x00},      // 't'
{0x00,0x00,0x00,0x00,0x00,0x66,0x66,0x66,0x66,0x66,0x66,0x3B,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x36,0x36,0x1C,0x1C,0x08,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x6B,0x6B,0x7F,0x36,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x63,0x36,0x1C,0x1C,0x1C,0x36,0x63,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x63,0x63,0x63,0x63,0x63,0x3F,0x03,0x06,0x3C,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x7F,0x66,0x0C,0x18,0x30,0x63,0x7F,0x00,0x00,0x00,0x00},      // 'z'
{0x00,0x00,0x0E,0x18,0x18,0x18,0x70,0x18,0x18,0x18,0x18,0x0E,0x00,0x00,0x00,0x00},      // '{'
{0x00,0x00,0x18,0x18,0x18,0x18,0x18,0x00,0x18,0x18,0x18,0x18,0x18,0x00,0x00,0x00},      // '|'
{0x00,0x00,0x70,0x18,0x18,0x18,0x0E,0x18,0x18,0x18,0x18,0x70,0x00,0x00,0x00,0x00},      // '}'
{0x00,0x00,0x3B,0x6E,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}       // '~'
};



        void LCDCommand(unsigned char data);
        void LCDData(unsigned char data);
        uint8_t driver;

        void LCDconstruct(void);
        void LCDinit(int type);
        void LCDclear(int color);
        void LCDcontrast(char setting);

        void LCDsetPixel(int color, unsigned char x, unsigned char y);

        void LCDsetChar(char c, int x, int y, int fColor, int bColor);
        void LCDsetStr(char *pString, int x, int y, int fColor, int bColor);
        void LCDsetPage(char *pString);

        void LCDsetLine(int x0, int y0, int x1, int y1, int color);


        void LCDon(void);
        void LCDoff(void);

#endif  // ColorLCDShield_H
```

A.3 utils.h

```
//************************************************************************//
#include <avr/pgmspace.h>
#include <avr/io.h>
#include <avr/power.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <string.h>
//************************************************************************//
#ifdef __AVR_ATmega328__
#define TIMSK TIMSK1
#endif
//************************************************************************//
#define NOP asm("nop")
#define byte  unsigned char
#define word uint16_t
#define dword uint32_t
//unsigned long int
#define qword uint64_t
#define int64 int64_t
#define int32 int32_t
//************************************************************************//
#define set_bit(port,bit) port|=(1<<bit)
#define clear_bit(port,bit) port&=~(1<<bit)
#define get_bit(port,bit)  ((port&(1<<bit))>>bit)
//************************************************************************//
```

A.4 pff.h

```
//#############################################################################//
#define _WORD_ACCESS 1
/* The _WORD_ACCESS option defines which access method is used to the word
/  data in the FAT structure.
/
/   0: Byte-by-byte access. Always compatible with all platforms.
/   1: Word access. Do not choose this unless following condition is met.
/
/  When the byte order on the memory is big-endian or address miss-aligned
/  word access results incorrect behavior, the _WORD_ACCESS must be set to 0.
/  If it is not the case, the value can also be set to 1 to improve the
/  performance and code efficiency. */
//#############################################################################//
#define CLUST word
//#############################################################################//
//File system object structure
typedef struct{
 byte flag;         //File status flags
 byte csize;        //Number of sectors per cluster
 byte pad1;
 word n_rootdir;    //Number of root directory entries (0 on FAT32)
 CLUST n_fatent;    //Number of FAT entries (= number of clusters + 2)
 dword fatbase;     //FAT start sector
 dword dirbase;     //Root directory start sector (Cluster# on FAT32)
 dword database;    //Data start sector
 dword fptr;        //File R/W pointer
 dword fsize;       //File size
 CLUST org_clust;   //File start cluster
 CLUST curr_clust; //File current cluster
 dword dsect;       //File current data sector
}FATFS;
//#############################################################################//
//Directory object structure
typedef struct{
 word index;     //Current read/write index number
 byte* fn;       //Pointer to the SFN (in/out) {file[8],ext[3],status[1]}
 CLUST sclust;   //Table start cluster (0:Static table)
 CLUST clust;    //Current cluster
 dword sect;     //Current sector
}DIR;
//#############################################################################//
//File status structure
typedef struct {
 dword fsize;     //File size
 word fdate;      //Last modified date
 word ftime;      //Last modified time
 byte fattrib;    //Attribute
 char fname[13]; //File name
}FILINFO;
//#############################################################################//
//File function return code
#define FR_OK               0
#define FR_DISK_ERR         1
#define FR_NOT_READY        2
#define FR_NO_FILE          3
#define FR_NO_PATH          4
#define FR_NOT_OPENED       5
#define FR_NOT_ENABLED      6
#define FR_NO_FILESYSTEM 7
//#############################################################################//
//File status flag (FATFS.flag)
#define FA_OPENED 0x01
#define FA_WPRT    0x02
#define FA__WIP    0x40
//#############################################################################//
//File attribute bits for directory entry
#define AM_RDO  0x01 //Read only
#define AM_HID  0x02 //Hidden
#define AM_SYS  0x04 //System
#define AM_VOL  0x08 //Volume label
#define AM_LFN  0x0F //LFN entry
#define AM_DIR  0x10 //Directory
#define AM_ARC  0x20 //Archive
#define AM_MASK 0x3F //Mask of defined bits
```

```
//------------------------------------------------------------------------//
//Multi-byte word access macros
#if _WORD_ACCESS==1 //Enable word access to the FAT structure
#define LD_word(ptr)  (word)(*(word*)(byte*)(ptr))
#define LD_dword(ptr)  (dword)(*(dword*)(byte*)(ptr))
#define ST_word(ptr,val) *(word*)(byte*)(ptr)=(word)(val)
#define ST_dword(ptr,val) *(dword*)(byte*)(ptr)=(dword)(val)
#else    //Use byte-by-byte access to the FAT structure
#define LD_word(ptr)  (word)(((word)*((byte*)(ptr)+1)<<8)|(word)*(byte*)(ptr))
#define LD_dword(ptr)  (dword)(((dword)*((byte*)(ptr)+3)<<24)|((dword)*((byte*)(ptr)+2)<<16)
       |((word)*((byte*)(ptr)+1)<<8)|*(byte*)(ptr))
#define ST_word(ptr,val) *(byte*)(ptr)=(byte)(val); *((byte*)(ptr)+1)=(byte)((word)(val)>>8)
#define ST_dword(ptr,val) *(byte*)(ptr)=(byte)(val); *((byte*)(ptr)+1)=(byte)((word)(val)>>8);
       *((byte*)(ptr)+2)=(byte)((dword)(val)>>16); *((byte*)(ptr)+3)=(byte)((dword)(val)>>24)
#endif
//------------------------------------------------------------------------//
#define LD_CLUST(dir)LD_word(dir+DIR_FstClusLO)
/*
//cp1251
#define _EXCVT { \
 0x80,0x81,0x82,0x82,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x80,0x91,0x92,0x93,0x94,0x95,0x96,0x97,
0x98,0x99,0x8A,0x9B,0x8C,0x8D,0x8E,0x8F,\
 0xA0,0xA2,0xA2,0xA3,0xA4,0xA5,0xA6,0xA7,0xA8,0xA9,0xAA,0xAB,0xAC,0xAD,0xAE,0xAF,0xB0,0xB1,0xB2,0xB2,0xA5,0xB5,0xB6,0xB7,
0xA8,0xB9,0xAA,0xBB,0xA3,0xBD,0xBD,0xAF,\
 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,
0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF,\
 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,
0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF \
}
*/
//dos866
#define _EXCVT { \
 0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0x90,0x91,0x92,0x93,0x94,0x95,0x96,0x97,
0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F,\
 0x80,0x81,0x82,0x83,0x84,0x85,0x86,0x87,0x88,0x89,0x8A,0x8B,0x8C,0x8D,0x8E,0x8F,0xB0,0xB1,0xB2,0xB3,0xB4,0xB5,0xB6,0xB7,
0xB8,0xB9,0xBA,0xBB,0xBC,0xBD,0xBE,0xBF,\
 0xC0,0xC1,0xC2,0xC3,0xC4,0xC5,0xC6,0xC7,0xC8,0xC9,0xCA,0xCB,0xCC,0xCD,0xCE,0xCF,0xD0,0xD1,0xD2,0xD3,0xD4,0xD5,0xD6,0xD7,
0xD8,0xD9,0xDA,0xDB,0xDC,0xDD,0xDE,0xDF,\
 0x90,0x91,0x92,0x93,0x9d,0x95,0x96,0x97,0x98,0x99,0x9A,0x9B,0x9C,0x9D,0x9E,0x9F,0xF0,0xF0,0xF2,0xF2,0xF4,0xF4,0xF6,0xF6,
0xF8,0xF9,0xFA,0xFB,0xFC,0xFD,0xFE,0xFF \
}
//------------------------------------------------------------------------//
//Character code support macros
#define IsUpper(c)(((c)>='A')&&((c)<='Z'))
#define IsLower(c)(((c)>='a')&&((c)<='z'))

#define IsDBCS1(c)0
#define IsDBCS2(c)0
//------------------------------------------------------------------------//
//FatFs refers the members in the FAT structures with byte offset instead
//of structure member because there are incompatibility of the packing option
//between various compilers.

#define BS_jmpBoot    0
#define BS_OEMName    3
#define BPB_BytsPerSec   11
#define BPB_SecPerClus   13
#define BPB_RsvdSecCnt   14
#define BPB_NumFATs    16
#define BPB_RootEntCnt   17
#define BPB_TotSec16   19
#define BPB_Media    21
#define BPB_FATSz16    22
#define BPB_SecPerTrk   24
#define BPB_NumHeads   26
#define BPB_HiddSec    28
#define BPB_TotSec32   32
#define BS_55AA     510

#define BS_DrvNum    36
#define BS_BootSig    38
#define BS_VolID    39
#define BS_VolLab    43
#define BS_FilSysType   54
```

```
#define BPB_FATSz32    36
#define BPB_ExtFlags   40
#define BPB_FSVer      42
#define BPB_RootClus   44
#define BPB_FSInfo     48
#define BPB_BkBootSec  50
#define BS_DrvNum32    64
#define BS_BootSig32   66
#define BS_VolID32     67
#define BS_VolLab32    71
#define BS_FilSysType32  82

#define MBR_Table    446

#define DIR_Name     0
#define DIR_Attr     11
#define DIR_NTres    12
#define DIR_CrtTime    14
#define DIR_CrtDate    16
#define DIR_FstClusHI  20
#define DIR_WrtTime    22
#define DIR_WrtDate    24
#define DIR_FstClusLO  26
#define DIR_FileSize   28
//#############################################################################//
//Pointer to the file system object(logical drive)
static FATFS *FatFs;
//#############################################################################//
//Fill memory
static void mem_set(void* dst,int val,int cnt)
{
 char *d=(char*)dst;
 while(cnt--)*d++=(char)val;
}
//#############################################################################//
//Compare memory to memory
static int mem_cmp(const void* dst,const void* src,int cnt)
{
 const char *d=(const char *)dst,*s=(const char *)src;
 int r=0;
 while(cnt-- &&(r=*d++-*s++)==0);
 return r;
}
//#############################################################################//
//FAT access-Read value of a FAT entry
static CLUST get_fat(CLUST clst)
{
 byte buf[4];
 FATFS *fs=FatFs;

 if(clst<2 || clst>=fs->n_fatent)return 1;

 if(disk_readp(buf,fs->fatbase+clst/256,(word)(((word)clst%256)*2),2))return 1;
 return LD_word(buf);
}
//#############################################################################//
//Get sector# from cluster #
//!=0: Sector number,0: Failed-invalid cluster #
static dword clust2sect(CLUST clst)
{
 FATFS *fs=FatFs;
 clst -= 2;
 if(clst >=(fs->n_fatent-2))return 0;  //Invalid cluster #
 return (dword)clst*fs->csize+fs->database;
}
//#############################################################################//
//Directory handling-Rewind directory index
static byte dir_rewind(DIR *dj)
{
 CLUST clst;
 FATFS *fs=FatFs;
```

```
 dj->index=0;
 clst=dj->sclust;
 if(clst==1 || clst >= fs->n_fatent)return FR_DISK_ERR; //Check start cluster range
 dj->clust=clst;       //Current cluster
 dj->sect=clst ? clust2sect(clst): fs->dirbase; //Current sector

 return FR_OK; //Seek succeeded
}
//#########################################################################//
//Directory handling-Move directory index next
//FR_OK: Succeeded
//FR_NO_FILE: End of table
static byte dir_next(DIR *dj)
{
 CLUST clst;
 word i;
 FATFS *fs=FatFs;


 i=dj->index+1;
 if(!i || !dj->sect)/* Report EOT when index has reached 65535 */
  return FR_NO_FILE;

 if(!(i % 16)){  /* Sector changed? */
  dj->sect++;   /* Next sector */

  if(dj->clust==0){ /* Static table */
   if(i >= fs->n_rootdir)/* Report EOT when end of table */
    return FR_NO_FILE;
  }
  else{      /* Dynamic table */
   if(((i / 16)&(fs->csize-1))==0){ /* Cluster changed? */
    clst=get_fat(dj->clust);  /* Get next cluster */
    if(clst <= 1)return FR_DISK_ERR;
    if(clst >= fs->n_fatent) /* When it reached end of dynamic table */
     return FR_NO_FILE;    /* Report EOT */
    dj->clust=clst;     /* Initialize data for new cluster */
    dj->sect=clust2sect(clst);
   }
  }
 }

 dj->index=i;

 return FR_OK;
}




/*-----------------------------------------------------------------------*/
/* Directory handling-Find an object in the directory                    */
/*-----------------------------------------------------------------------*/

static
byte dir_find(
 DIR *dj, /* Pointer to the directory object linked to the file name */
 byte *dir  /* 32-byte working buffer */
)
{
 byte res;
 byte c;


 res=dir_rewind(dj);   /* Rewind directory object */
 if(res != FR_OK)return res;

 do{
  res=disk_readp(dir,dj->sect,(word)((dj->index % 16)* 32),32)/* Read an entry */
   ? FR_DISK_ERR : FR_OK;
  if(res != FR_OK)break;
  c=dir[DIR_Name]; /* First character */
  if(c==0){ res=FR_NO_FILE; break; }/* Reached to end of table */
  if(!(dir[DIR_Attr] & AM_VOL)&& !mem_cmp(dir,dj->fn,11))/* Is it a valid entry? */
```

```
    break;
   res=dir_next(dj);      /* Next entry */
  }while(res==FR_OK);

  return res;
}
//#########################################################################//
//Read an object from the directory
static byte dir_read(DIR *dj,byte *dir)
{
 byte res;
 byte a,c;


 res=FR_NO_FILE;
 while(dj->sect){
  res=disk_readp(dir,dj->sect,(word)((dj->index % 16)* 32),32)/* Read an entry */
   ? FR_DISK_ERR : FR_OK;
  if(res != FR_OK)break;
  c=dir[DIR_Name];
  if(c==0){ res=FR_NO_FILE; break; }/* Reached to end of table */
  a=dir[DIR_Attr] & AM_MASK;
  if(c != 0xE5 && c != '.' && !(a & AM_VOL))/* Is it a valid entry? */
   break;
  res=dir_next(dj);   /* Next entry */
  if(res != FR_OK)break;
 }

 if(res!=FR_OK)dj->sect=0;

 return res;
}
//#########################################################################//


/* Pick a segment and create the object name in directory form           */
/*-----------------------------------------------------------------------*/

#ifdef _EXCVT
 static const byte cvt[]=_EXCVT;
#endif

static
byte create_name(
 DIR *dj,  /* Pointer to the directory object */
 const char **path /* Pointer to pointer to the segment in the path string */
)
{
 byte c,d,ni,si,i,*sfn;
 const char *p;

 /* Create file name in directory form */
 sfn=dj->fn;
 mem_set(sfn,' ',11);
 si=i=0; ni=8;
 p=*path;
 for(;;){
  c=p[si++];
  if(c <= ' ' || c=='/')break; /* Break on end of segment */
  if(c=='.' || i >= ni){
   if(ni != 8 || c != '.')break;
   i=8; ni=11;
   continue;
  }
#ifdef _EXCVT
  if(c >= 0x80)     /* To upper extended char(SBCS)*/
   c=cvt[c-0x80];
#endif
  if(IsDBCS1(c)&& i<ni-1){ /* DBC 1st byte? */
   d=p[si++];     /* Get 2nd byte */
   sfn[i++]=c;
   sfn[i++]=d;
  }else{       /* Single byte code */
   if(IsLower(c))c -= 0x20; /* toupper */
```

```
    sfn[i++]=c;
   }
  }
  *path=&p[si];       /* Rerurn pointer to the next segment */

  sfn[11]=(c <= '␣')? 1 : 0;  /* Set last segment flag if end of path */

  return FR_OK;
}



/*-----------------------------------------------------------------------*/
/* Get file information from directory entry                             */
/*-----------------------------------------------------------------------*/
static
void get_fileinfo(  /* No return code */
 DIR *dj,  /* Pointer to the directory object */
 byte *dir,  /* 32-byte working buffer */
 FILINFO *fno   /* Pointer to store the file information */
)
{
 byte i,c;
 char *p;


 p=fno->fname;
 if(dj->sect){
  for(i=0; i<8; i++){ /* Copy file name body */
   c=dir[i];
   if(c=='␣')break;
   if(c==0x05)c=0xE5;
   *p++=c;
  }
  if(dir[8] != '␣'){  /* Copy file name extension */
   *p++='.';
   for(i=8; i<11; i++){
    c=dir[i];
    if(c=='␣')break;
    *p++=c;
   }
  }
  fno->fattrib=dir[DIR_Attr];     /* Attribute */
  fno->fsize=LD_dword(dir+DIR_FileSize); /* Size */
  fno->fdate=LD_word(dir+DIR_WrtDate);  /* Date */
  fno->ftime=LD_word(dir+DIR_WrtTime);  /* Time */
 }
 *p=0;
}



/*-----------------------------------------------------------------------*/
/* Follow a file path                                                    */
/*-----------------------------------------------------------------------*/

static
byte follow_path( /* FR_OK(0): successful,!=0: error code */
 DIR *dj,  /* Directory object to return last directory and found object */
 byte *dir,  /* 32-byte working buffer */
 const char *path /* Full-path string to find a file or directory */
)
{
 byte res;


 while(*path=='␣')path++;  /* Skip leading spaces */
 if(*path=='/')path++;   /* Strip heading separator */
 dj->sclust=0;       /* Set start directory(always root dir)*/

 if ((byte)*path <= '␣'){    /* Null path means the root directory */
  res=dir_rewind(dj);
  dir[0]=0;
```

```
 }else{        /* Follow path */
  for(;;){
   res=create_name(dj,&path); /* Get a segment */
   if(res != FR_OK)break;
   res=dir_find(dj,dir);   /* Find it */
   if(res != FR_OK){     /* Could not find the object */
    if(res==FR_NO_FILE && !*(dj->fn+11))
     res=FR_NO_PATH;
    break;
   }
   if(*(dj->fn+11))break;  /* Last segment match. Function completed. */
   if(!(dir[DIR_Attr] & AM_DIR)){ /* Cannot follow because it is a file */
    res=FR_NO_PATH; break;
   }
   dj->sclust=LD_CLUST(dir);
  }
 }

 return res;
}
//#########################################################################//
//Check a sector if it is an FAT boot record
//0: The FAT boot record
//1: Valid boot record but not an FAT
//2: Not a boot record
//3: Error
static byte check_fs(byte *buf,dword sect)
{
 //Read the boot sector
 if (disk_readp(buf,sect,510,2))return 3;
 //Check record signature
 if (LD_word(buf)!=0xAA55)return 2;
 //Check FAT116
 if (!disk_readp(buf,sect,BS_FilSysType,2)&& LD_word(buf)==0x4146)return 0;
 return 1;
}
//#########################################################################//
//#########################################################################//
//Mount/Unmount a Locical Drive
byte pf_mount(FATFS *fs)
{
 volatile byte fmt,buf[36];
 dword bsect,fsize,tsect,mclst;

 FatFs=0;
 if (!fs)return FR_OK;     //Unregister fs object

 if (disk_initialize(0)& STA_NOINIT)return FR_NOT_READY; //Check if the drive is ready or not

 //Search FAT partition on the drive
 bsect=0;
 fmt=check_fs(buf,bsect);    //Check sector 0 as an SFD format
 if(fmt==1){       //Not an FAT boot record,it may be FDISK format
  //Check a partition listed in top of the partition table
  if(disk_readp(buf,bsect,MBR_Table,16)){ //1st partition entry
   fmt=3;
  }else if(buf[4]){      //Is the partition existing?
   bsect=LD_dword(&buf[8]); //Partition offset in LBA
   fmt=check_fs(buf,bsect); //Check the partition
  }
 }
 if(fmt==3)return FR_DISK_ERR;
 if (fmt)return FR_NO_FILESYSTEM;  //No valid FAT patition is found

 //Initialize the file system object
 if (disk_readp(buf,bsect,13,sizeof(buf)))return FR_DISK_ERR;

 fsize=LD_word(buf+BPB_FATSz16-13);     //Number of sectors per FAT
 if (!fsize)fsize=LD_dword(buf+BPB_FATSz32-13);

 fsize*=buf[BPB_NumFATs-13];       //Number of sectors in FAT area
 fs->fatbase=bsect+LD_word(buf+BPB_RsvdSecCnt-13); //FAT start sector(lba)
 fs->csize=buf[BPB_SecPerClus-13];      //Number of sectors per cluster
```

```
 fs->n_rootdir=LD_word(buf+BPB_RootEntCnt-13);  //Nmuber of root directory entries
 tsect=LD_word(buf+BPB_TotSec16-13);     //Number of sectors on the file system
 if (!tsect)tsect=LD_dword(buf+BPB_TotSec32-13);
 mclst=(tsect-LD_word(buf+BPB_RsvdSecCnt-13)-fsize-fs->n_rootdir/16)/fs->csize+2;
 fs->n_fatent=(CLUST)mclst;

 if(mclst<0xFF7)return FR_NO_FILESYSTEM;        //Number of clusters<0xFF5
 if(mclst>=0xFFF7)return FR_NO_FILESYSTEM;      //Number of clusters>=0xFFF5

 fs->dirbase=fs->fatbase+fsize;     //Root directory start sector(lba)
 fs->database=fs->fatbase+fsize+fs->n_rootdir/16; //Data start sector(lba)

 fs->flag=0;
 FatFs=fs;

 return FR_OK;
}
//////////////////////////////////////////////////////////////////////////////
//Open or Create a File
byte pf_open(const char *path)
{
 byte res;
 DIR dj;
 byte sp[12],dir[32];
 FATFS *fs=FatFs;

 if (!fs)return FR_NOT_ENABLED;      //Check file system

 fs->flag=0;
 dj.fn=sp;
 res=follow_path(&dj,dir,path);                                  //Follow the file path
 if(res!=FR_OK)return res;                                       //Follow failed
 if(!dir[0] ||(dir[DIR_Attr] & AM_DIR))return FR_NO_FILE; //It is a directory

 fs->org_clust=LD_CLUST(dir);           //File start cluster
 fs->fsize=LD_dword(dir+DIR_FileSize); //File size
 fs->fptr=0;                            //File pointer
 fs->flag=FA_OPENED;

 return FR_OK;
}
//////////////////////////////////////////////////////////////////////////////
//Read File
//void *buff Pointer to the read buffer(NULL:Forward data to the stream)
//word btr Number of bytes to read
//word *br  Pointer to number of bytes read
byte pf_read(void *buff,word btr,word *br)
{
 byte dr;
 CLUST clst;
 dword sect,remain;
 word rcnt;
 byte cs,*rbuff=buff;
 FATFS *fs=FatFs;

 *br=0;
 if (!fs)return FR_NOT_ENABLED;  //Check file system
 if(!(fs->flag & FA_OPENED))return FR_NOT_OPENED; //Check if opened

 remain=fs->fsize-fs->fptr;
 if(btr>remain)btr=(word)remain;   //Truncate btr by remaining bytes

 while (btr){          //Repeat until all data transferred
  if((fs->fptr % 512)==0){     //On the sector boundary?
   cs=(byte)(fs->fptr/512 &(fs->csize-1)); //Sector offset in the cluster
   if(!cs){         //On the cluster boundary?
    clst=(fs->fptr==0)?fs->org_clust:get_fat(fs->curr_clust);    //On the top of the file?
    if(clst<=1)goto fr_abort;
    fs->curr_clust=clst;     //Update current cluster
   }
   sect=clust2sect(fs->curr_clust);  //Get current sector
   if(!sect)goto fr_abort;
   fs->dsect=sect+cs;
  }
```

```
  rcnt=(word)(512-(fs->fptr%512));  //Get partial sector data from sector buffer
  if(rcnt>btr)rcnt=btr;
  dr=disk_readp(!buff?0:rbuff,fs->dsect,(word)(fs->fptr%512),rcnt);
  if(dr)goto fr_abort;
  fs->fptr+=rcnt;
  rbuff+=rcnt;   //Update pointers and counters
  btr-=rcnt;
  *br+=rcnt;
 }
 return FR_OK;

fr_abort:
 fs->flag=0;
 return FR_DISK_ERR;
}
//////////////////////////////////////////////////////////////////////////////
byte pf_write(const void* buff,word btw,word* bw)
{
 CLUST clst;
 dword sect,remain;
 const byte *p=buff;
 byte cs;
 word wcnt;
 FATFS *fs=FatFs;


 *bw=0;
 if (!fs) return FR_NOT_ENABLED;  //Check file system
 if (!(fs->flag & FA_OPENED)) return FR_NOT_OPENED;  //Check if opened

 //Finalize request?
 if (!btw){
  if ((fs->flag & FA__WIP) && disk_writep(0, 0)) goto fw_abort;
  fs->flag &= ~FA__WIP;
  return FR_OK;
 }

 //Write data request
 if (!(fs->flag & FA__WIP))fs->fptr &= 0xFFFFFE00;  //Round-down fptr to the sector boundary

 remain=fs->fsize-fs->fptr;
 if (btw > remain) btw=(word)remain;   //Truncate btw by remaining bytes

 while (btw) {          //Repeat until all data transferred
  if (((word)fs->fptr % 512)==0) {   //On the sector boundary?
   cs=(byte)(fs->fptr / 512 & (fs->csize-1)); //Sector offset in the cluster
   if (!cs) {         //On the cluster boundary?
    clst=(fs->fptr==0) ? fs->org_clust : get_fat(fs->curr_clust);  //On the top of the file?
    if (clst <= 1) goto fw_abort;
    fs->curr_clust=clst;     //Update current cluster
   }
   sect=clust2sect(fs->curr_clust);  //Get current sector
   if (!sect) goto fw_abort;
   fs->dsect=sect + cs;
   if (disk_writep(0, fs->dsect)) goto fw_abort; //Initiate a sector write operation
   fs->flag |= FA__WIP;
  }
  wcnt=512-((word)fs->fptr % 512);  //Number of bytes to write to the sector
  if (wcnt > btw) wcnt=btw;
  if (disk_writep(p, wcnt)) goto fw_abort; //Send data to the sector
  fs->fptr += wcnt; p += wcnt;     //Update pointers and counters
  btw -= wcnt; *bw += wcnt;
  if (((word)fs->fptr % 512)==0) {
   if (disk_writep(0, 0)) goto fw_abort; //Finalize the currtent secter write operation
   fs->flag &= ~FA__WIP;
  }
 }

 return FR_OK;

fw_abort:
 fs->flag=0;
 return FR_DISK_ERR;
}
```

```
//#############################################################################//
//Seek File R/W Pointer
byte pf_lseek(dword ofs)
{
 CLUST clst;
 dword bcs,sect,ifptr;
 FATFS *fs=FatFs;

 if (!fs)return FR_NOT_ENABLED;                          //Check file system
 if(!(fs->flag & FA_OPENED))return FR_NOT_OPENED;  //Check if opened

 if(ofs>fs->fsize)ofs=fs->fsize; //Clip offset with the file size
 ifptr=fs->fptr;
 fs->fptr=0;

 if(ofs>0){
  bcs=(dword)fs->csize * 512; //Cluster size(byte)
  if(ifptr>0 && (ofs-1)/bcs >=(ifptr-1)/bcs){ //When seek to same or following cluster, start from the current cluster
   fs->fptr=(ifptr-1)& ~(bcs-1);
   ofs-=fs->fptr;
   clst=fs->curr_clust;
  }else{        //When seek to back cluster, start from the first cluster
   clst=fs->org_clust;
   fs->curr_clust=clst;
  }

  while(ofs>bcs){     //Cluster following loop
   clst=get_fat(clst);  //Follow cluster chain
   if(clst<=1 || clst>=fs->n_fatent)goto fe_abort;
   fs->curr_clust=clst;
   fs->fptr+=bcs;
   ofs-=bcs;
  }
  fs->fptr+=ofs;
  sect=clust2sect(clst);  //Current sector
  if(!sect)goto fe_abort;
  fs->dsect=sect+(fs->fptr/512 & (fs->csize-1));
 }

 return FR_OK;

fe_abort:
 fs->flag=0;
 return FR_DISK_ERR;
}
//#############################################################################//
byte pf_opendir(DIR *dj,const char *path)
{
 byte res;
 byte sp[12],dir[32];
 FATFS *fs=FatFs;

 if (!fs)return FR_NOT_ENABLED;

 dj->fn=sp;
 res=follow_path(dj,dir,path);  //Follow the path to the directory
 if(res==FR_OK){       //Follow completed
  if(dir[0]){       //It is not the root dir
   if(dir[DIR_Attr] & AM_DIR)dj->sclust=LD_CLUST(dir); //The object is a directory
   else res=FR_NO_PATH;       //The object is not a directory
  }
  if(res==FR_OK)res=dir_rewind(dj);    //Rewind dir
 }
 if(res==FR_NO_FILE)res=FR_NO_PATH;

 return res;
}
//#############################################################################//
//Read Directory Entry in Sequense
byte pf_readdir(DIR *dj,FILINFO *fno)
{
 byte res;
 byte sp[12],dir[32];
 FATFS *fs=FatFs;
```

```
 if (!fs)return FR_NOT_ENABLED;
 dj->fn=sp;
 if (!fno)return dir_rewind(dj);

 res=dir_read(dj,dir);
 if(res==FR_NO_FILE){dj->sect=0;res=FR_OK;}

 if(res==FR_OK){     //A valid entry is found
  get_fileinfo(dj,dir,fno); //Get the object information
  res=dir_next(dj);    //Increment index for next
  if(res==FR_NO_FILE){
   dj->sect=0;
   res=FR_OK;
  }
 }

 return res;
}
//#########################################################################//
```

A.5 mmc.h

```
//#########################################################################//
#include <avr/io.h>
//#########################################################################//
/*
#define SD_CS 0
#define SD_PORT PORTC
#define SD_DDR DDRC
*/
//#########################################################################//
#define CS_LOW()   clear_bit(SD_PORT,SD_CS) //CS=L
#define CS_HIGH()    set_bit(SD_PORT,SD_CS) //CS=H

#define STA_NOINIT  0x01 //Drive not initialized
#define STA_NODISK  0x02 //No medium in the drive

//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 ACMD23  (0xC0+23)        //SET_WR_BLK_ERASE_COUNT (SDC)
#define CMD8    (0x40+8)  //SEND_IF_COND
#define CMD12   (0x40+12) //STOP_TRANSMISSION
#define CMD16   (0x40+16) //SET_BLOCKLEN
#define CMD17   (0x40+17) //READ_SINGLE_BLOCK
#define CMD18     (0x40+18)       //READ_MULTIPLE_BLOCK
#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
#define CMD9    (9)         //Send CSD
//#########################################################################//
//Card type flags
#define CT_MMC    0x01 //MMC ver 3
#define CT_SD1    0x02 //SD ver 1
#define CT_SD2    0x04 //SD ver 2
#define CT_BLOCK 0x08 //Block addressing
//#########################################################################//
#define RES_OK 0  //0: Function succeeded
#define RES_ERROR 1 //1: Disk error
#define RES_NOTRDY 2  //2: Not ready
#define RES_PARERR 3  //3: Invalid parameter
//#########################################################################//
#define GET_SECTOR_COUNT 1
#define CTRL_SYNC 2
//#########################################################################//
static byte card_type;
static byte mmc_inited=0;
//#########################################################################//
byte xchg_spi(byte dat)
{
```

```
 SPDR=dat;
 loop_until_bit_is_set(SPSR,SPIF);
 return SPDR;
}
//###########################################################################//
void xmit_spi_multi(const byte *p,word cnt)
{
        do{
                SPDR=*p++;loop_until_bit_is_set(SPSR,SPIF);
                SPDR=*p++;loop_until_bit_is_set(SPSR,SPIF);
        }while(cnt-=2);
}
//###########################################################################//
void rcvr_spi_multi(byte *p,word cnt)
{
 do{
  SPDR=0xFF;loop_until_bit_is_set(SPSR,SPIF);*p++ =SPDR;
  SPDR=0xFF;loop_until_bit_is_set(SPSR,SPIF);*p++ =SPDR;
 }while(cnt-=2);
}
//###########################################################################//
int wait_ready(void)
{
 byte d;
 word t;

 t=25000;
 do{
  d=xchg_spi(0xFF);
 }while(d!=0xFF&&(t--));

 return(d==0xFF)?1:0;
}
//###########################################################################//
int rcvr_datablock(byte  *buff,word btr)
{
 byte token;
 word t;

 //Wait for data packet in timeout of 200ms
 t=200;
 do{
  token=xchg_spi(0xFF);
  if(token==0xFF)_delay_ms(1);
 }while((token==0xFF)&&(t--));

 if(token!=0xFE)return 0; //If not valid data token, retutn with error

 rcvr_spi_multi(buff, btr);  //Receive the data block into buffer
 xchg_spi(0xFF);      //Discard CRC
 xchg_spi(0xFF);

 return 1;       //Return with success
}
//###########################################################################//
int xmit_datablock(const byte *buff,byte token)
{
        byte resp;

        //if (!wait_ready(500)) return 0;
        if (!wait_ready()) return 0;

        xchg_spi(token);                                        //Xmit data token
        if (token!=0xFD){       //Is data token
                xmit_spi_multi(buff,512);               //Xmit the data block to the MMC
                xchg_spi(0xFF);                                 //CRC (Dummy)
                xchg_spi(0xFF);
                resp=xchg_spi(0xFF);                    //Reveive data response
                if ((resp&0x1F)!=0x05)return 0;         //If not accepted, return with error
        }
        return 1;
}
//###########################################################################//
void deselect(void)
```

```
{
 CS_HIGH();
 xchg_spi(0xFF);
 //xchg_spi(0x15);
}
//////////////////////////////////////////////////////////////////////////////
int select(void)
{
 CS_LOW();
 xchg_spi(0xFF);

 if (wait_ready())return 1;
 deselect();
 return 0;
}
//////////////////////////////////////////////////////////////////////////////
//Send a command packet to MMC
//1st byte(Start + Index)
//Argument(32 bits)
byte send_cmd(byte cmd,dword arg)
{
 byte n,res;

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

 //Select the card
        if (cmd!=CMD12){
  deselect();
  if (!select())return 0xFF;
 }

 //Send a command packet
 xchg_spi (0x40 | cmd);       //Start + Command index
 xchg_spi ((byte)(arg >> 24));  //Argument[31..24]
 xchg_spi ((byte)(arg >> 16));  //Argument[23..16]
 xchg_spi ((byte)(arg >> 8));   //Argument[15..8]
 xchg_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)
 xchg_spi(n);

 //Receive a command response
 if(cmd==CMD12)xchg_spi(0xFF);
 n=10;          //Wait for a valid response in timeout of 10 attempts
 do{
  res=xchg_spi(0xFF);
 }while((res & 0x80)&& --n);

 return res;    //Return with the response value
}
//////////////////////////////////////////////////////////////////////////////
//Initialize Disk Drive
byte disk_initialize(byte redo)
{
 volatile byte n,cmd,ty,ocr[4];
 volatile word tmr;

 if (mmc_inited && !redo)return 0;
 mmc_inited=0;

 deselect();
 for(n=10;n;n--)xchg_spi(0xFF); //80 dummy clocks with CS=H

 ty=0;
 if (send_cmd(CMD0,0)==1){    //Enter Idle state
  if(send_cmd(CMD8,0x1AA)==1){ //SDv2
   for(n=0;n<4;n++)ocr[n]=xchg_spi(0xFF);   //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
```

```
    for(tmr=10000;tmr && send_cmd(ACMD41,1UL<<30);tmr--)_delay_us(100); //Wait for leaving idle state(ACMD41 with HCS bit)
    if(tmr && send_cmd(CMD58,0)== 0){   //Check CCS bit in the OCR
     for(n=0; n<4; n++)ocr[n]=xchg_spi(0xFF);
     ty=(ocr[0] & 0x40)? CT_SD2 | CT_BLOCK : CT_SD2; //SDv2(HC or SC)
    }
   }
   }else{        //SDv1 or MMCv3
    if(send_cmd(ACMD41,0)<= 1) {
     ty=CT_SD1; cmd=ACMD41; //SDv1
    }else{
     ty=CT_MMC; cmd=CMD1; //MMCv3
    }
    for(tmr=10000; tmr && send_cmd(cmd,0); tmr--)_delay_us(100); //Wait for leaving idle state
    if(!tmr || send_cmd(CMD16,512)!= 0)   //Set R/W block length to 512
     ty=0;
   }
  }
  card_type=ty;
  deselect();

  if (ty)mmc_inited=1;
  return ty?0:STA_NOINIT;
}
//-----------------------------------------------------------------------//
byte disk_ioctl(byte ctrl,void *buff)
{
  byte res;
  byte n,csd[16];
  dword csize;

  res=RES_ERROR;


  switch (ctrl){
   //Get number of sectors on the disk (dword)
   case GET_SECTOR_COUNT:{
    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) + ((dword)(csd[7] & 63) << 16) + 1;
      *(dword*)buff=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=csize << (n - 9);
     }
     res=RES_OK;
    }
    break;
   }
   //Make sure that no pending write process. Do not remove this or written sector might not left updated.
                case CTRL_SYNC:
                        if (select()) res=RES_OK;
                        break;
/*
  case GET_BLOCK_SIZE : //Get erase block size in unit of sector (dword)
   if (CardType & CT_SD2) { //SDv2?
    if (send_cmd(ACMD13, 0) == 0) { //Read SD status
     xchg_spi(0xFF);
     if (rcvr_datablock(csd, 16)) {     //Read partial block
      for (n=64 - 16; n; n--) xchg_spi(0xFF); //Purge trailing data
      *(dword*)buff=16UL << (csd[10] >> 4);
      res=RES_OK;
     }
    }
   } else {      //SDv1 or MMCv3
    if ((send_cmd(CMD9, 0) == 0) && rcvr_datablock(csd, 16)) { //Read CSD
     if (CardType & CT_SD1) { //SDv1
      *(dword*)buff=(((csd[10] & 63) << 1) + ((word)(csd[11] & 128) >> 7) + 1) << ((csd[13] >> 6) - 1);
     } else {      //MMCv3
      *(dword*)buff=((word)((csd[10] & 124) >> 2) + 1) * (((csd[11] & 3) << 3) + ((csd[11] & 224) >> 5) + 1);
     }
     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++=xchg_spi(0xFF);
   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
   xchg_spi(0xFF);
   if (rcvr_datablock(ptr, 64))
    res=RES_OK;
  }
  break;
*/
   default: res=RES_PARERR;
 }
 deselect();
 return res;
}
//////////////////////////////////////////////////////////////////////////////
//Read partial sector
byte disk_readp(byte *buff,dword lba,word ofs,word cnt)
{
 byte res;
 byte rc;
 word bc;

 if (!(card_type&CT_BLOCK))lba*=512;

 res=RES_ERROR;
 if (!buff)return res;

 if (send_cmd(CMD17,lba)==0){
  bc=40000;
  do{        //Wait for data packet
   rc=xchg_spi(0xFF);
  }while(rc == 0xFF && --bc);

  if(rc==0xFE){     //A data packet arrived
   bc=514-ofs-cnt;

   //Skip leading bytes
   if(ofs){
    do xchg_spi(0xFF); while(--ofs);
   }

   //Receive a part of the sector
   do{
    *buff++=xchg_spi(0xFF);
   }while(--cnt);


   //Skip trailing bytes and CRC
   do xchg_spi(0xFF); while(--bc);

   res=RES_OK;
  }
 }
```

```
 //for(int jj=0; jj<10 && send_cmd(CMD0,0)!=0;jj++){}; // Allison: idle state??

 deselect();
 int i=0;
for(i =0; i<100 && xchg_spi(0xFF)!=0; i++){}; // wait for 0
 return res;
}
//#######################################################################################//
byte disk_read(byte *buff,dword sector,word count)
{
 byte res;
 res=RES_ERROR;
 if (!buff)return res;
 if (!(card_type&CT_BLOCK))sector*=512;

        if (count==1){  //Single block read
                if ((send_cmd(CMD17,sector)==0)&& 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
                }
        }
        deselect();

        return count ? RES_ERROR : RES_OK;
}
//#######################################################################################//
byte disk_writep(const byte *buff, dword sa)
{
        byte res;
        word bc;
        static word wc;

        res=RES_ERROR;

        if (buff){              //Send data bytes
                bc=(word)sa;
                while (bc && wc){               //Send data bytes to the card
                        xchg_spi(*buff++);
                        wc--;
                        bc--;
                }
                res=RES_OK;
        }else{
                if (sa){        //Initiate sector write process
                        if (!(card_type & CT_BLOCK))sa*=512;    //Convert to byte address if needed
                        if (send_cmd(CMD24, sa)==0){                    //WRITE_SINGLE_BLOCK
                                xchg_spi(0xFF);
                                xchg_spi(0xFE);         //Data block header
                                wc=512;                                                 //Set byte counter
                                res=RES_OK;
                        }
                }else{  //Finalize sector write process
                        bc=wc + 2;
                        while (bc--) xchg_spi(0);       //Fill left bytes and CRC with zeros
                        if ((xchg_spi(0xFF) & 0x1F)==0x05) {    //Receive data resp and wait
                                // for end of write process in timeout of 500ms
                                for (bc=5000; xchg_spi(0xFF)!=0xFF && bc;bc--) _delay_us(100);  //Wait ready
                                if (bc)res=RES_OK;
                        }
                        deselect();
                        xchg_spi(0xFF);
                }
        }

        return res;
}
//#######################################################################################//
byte disk_write(const byte *buff,dword sector,word count)
{
```

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

        if (count==1){  //Single block write
                if ((send_cmd(CMD24, sector)==0) && xmit_datablock(buff,0xFE)) count=0;
        }else{                                  //Multiple block write
                if (card_type&(CT_SD1|CT_SD2)) 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)) count=1;
                }
        }
        deselect();

        return count ? RES_ERROR : RES_OK;
}
//#############################################################################//
```