11-2009

Touch-screen SD Card Text Reader

Samia Rahman

Touch-screen SD Card Text Reader

by

Samia Rahman

Undergraduate honors thesis under the direction of

Dr. Jerry Trahan

Department of Electrical and Computer Engineering

Submitted to the LSU Honors College in partial fulfillment of
the Upper Division Honors Program.

November, 2009

Louisiana State University
& Agricultural and Mechanical College
Baton Rouge, Louisiana

Abstract

This paper deals with the design and implementation of a touch-screen text reader that reads text files from an SD card and displays them on an LCD screen that the user can touch to select a button on the menu displayed to read a text file. The touch screen was designed using infrared technology. The BASIC Atom -40M 40-pin module microcontroller was used as the host controller to interface with the LCD screen, infrared detectors and transmitters and the SD card. The File Allocation Table structure was used to read text files from the SD card. This paper gives an overview of the FAT32 structure, the format used on the SD card and the Serial Peripheral Interface protocol of the SD card used by the host controller to read data from it. Detailed algorithms of how to send a command and read data from the card have been presented in this paper. The algorithms can be used to read files of any size. This paper is a good reference for anyone trying to read data from an SD card using a microcontroller. The project was successfully finished with the user able to touch the screen to read text files from the SD card on an LCD screen.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

1. Introduction

A touch-screen enables its user to interact with information on a display area by detecting a touch on it. One way to detect a touch is to use infrared technology. Infrared is part of the electromagnetic spectrum with wavelength longer than light that is not visible to the human eye. An infrared detector placed across from an infrared transmitter will give a high voltage while it receives infrared transmission and a low voltage when it does not receive a transmission. Using this information, a touch screen can be designed by placing an array of infrared light-emitting diodes (LEDs) across from infrared detectors so that a line of infrared beams is across the display. When an object touches the screen, it disturbs the infrared beams causing a decrease in voltage across the corresponding infrared detectors. The output change at the infrared detector can be measured to determine the touch event location.

The project is to design a touch-screen device that should provide the following features to the user:

- A menu displayed on a screen, with buttons to allow the selection of text files;
- A touch-screen mechanism that allows the user to select a text file by simply touching a menu item on the screen;
- If the text file does not fit on the screen, then the user should be able to get to the next page of the text file and also be able to return to the menu once the text file is read to the end; and
- The user should be able to read text files that they save on a memory storage device.

The project should be realized using a microcontroller as the host controller to control the screen to display menu and text files, to measure the voltage across the infrared detectors, to set up the touch-screen mechanism and finally to control a memory storage device to read text files.

The display area of the touch-screen can simply be an LCD screen since it will not interfere with the grid of infrared beams. The LCD screen can be used to display a menu for user interaction and provide a surface that the user can touch to interrupt the infrared beams. It can also be used to display text from text files. In this project the GLK12232-25-SM LCD screen with a 122 x 32 pixel graphics display is used [1]. It can be interfaced using RS-232 asynchronous communication with a microcontroller to display a menu and display texts when needed.

A Secure Digital (SD) card is a non-volatile memory card that is used as a portable storage device by users. It can store files in the Gigabyte (GB) range and can be used for reading large amounts of data or storing large amounts of data in projects using a microcontroller. This is possible because SD cards can be communicated with synchronously using Serial Peripheral Interface Bus transfer mode using a microcontroller [2]. The microcontroller becomes the host controller of the SD card since it sends read and write commands to it while receiving and sending commands.

The BASIC Atom -40M microcontroller is a powerful programmable device that can be used to measure voltage drops across infrared detectors and determine a touch event. It can also communicate with the LCD screen model mentioned earlier using RS-232 communication. Additionally, it is able to synchronously communicate with an SD card to read data from it. Therefore the BASIC Atom -40M microcontroller can be used to realize the touch-screen SD card text reader. It uses a programming language modeled after BASIC allowing a modular approach needed to write the code for the project [3]. The schematic in Figure 1 shows an overview of the project.

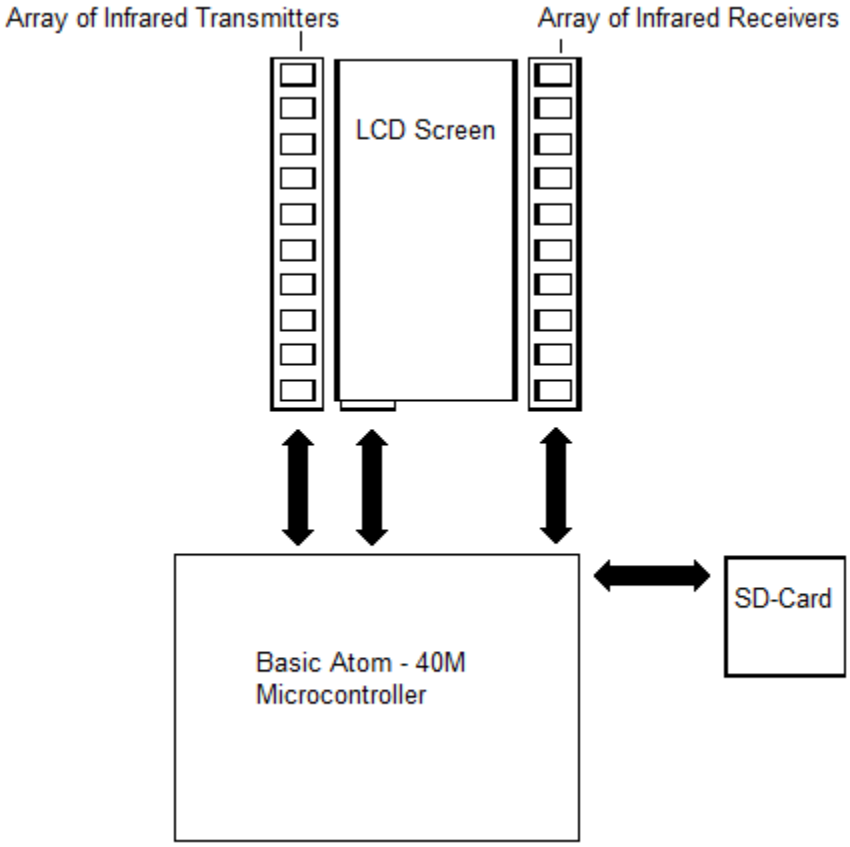

Figure 1: Overview of design of touch screen SD-card text reader

The rest of the paper presents the design of the touch screen SD-card text reader. The features of BASIC Atom -40M microcontroller are summarized followed with how the microcontroller controls the GLK12232-25-SM LCD screen, the infrared detectors and transmitters and the SD card. Algorithms are presented for the modules used by the BASIC Atom -40M microcontroller to control the whole system. Results are presented with a link to a demo of the project and a discussion of how the project can help other projects is presented in the conclusion.

2. Design

In this section the design of the product along with the design considerations are reported. Figure 2 below shows how the various components were setup together to get the final touch-screen SD card text reader.

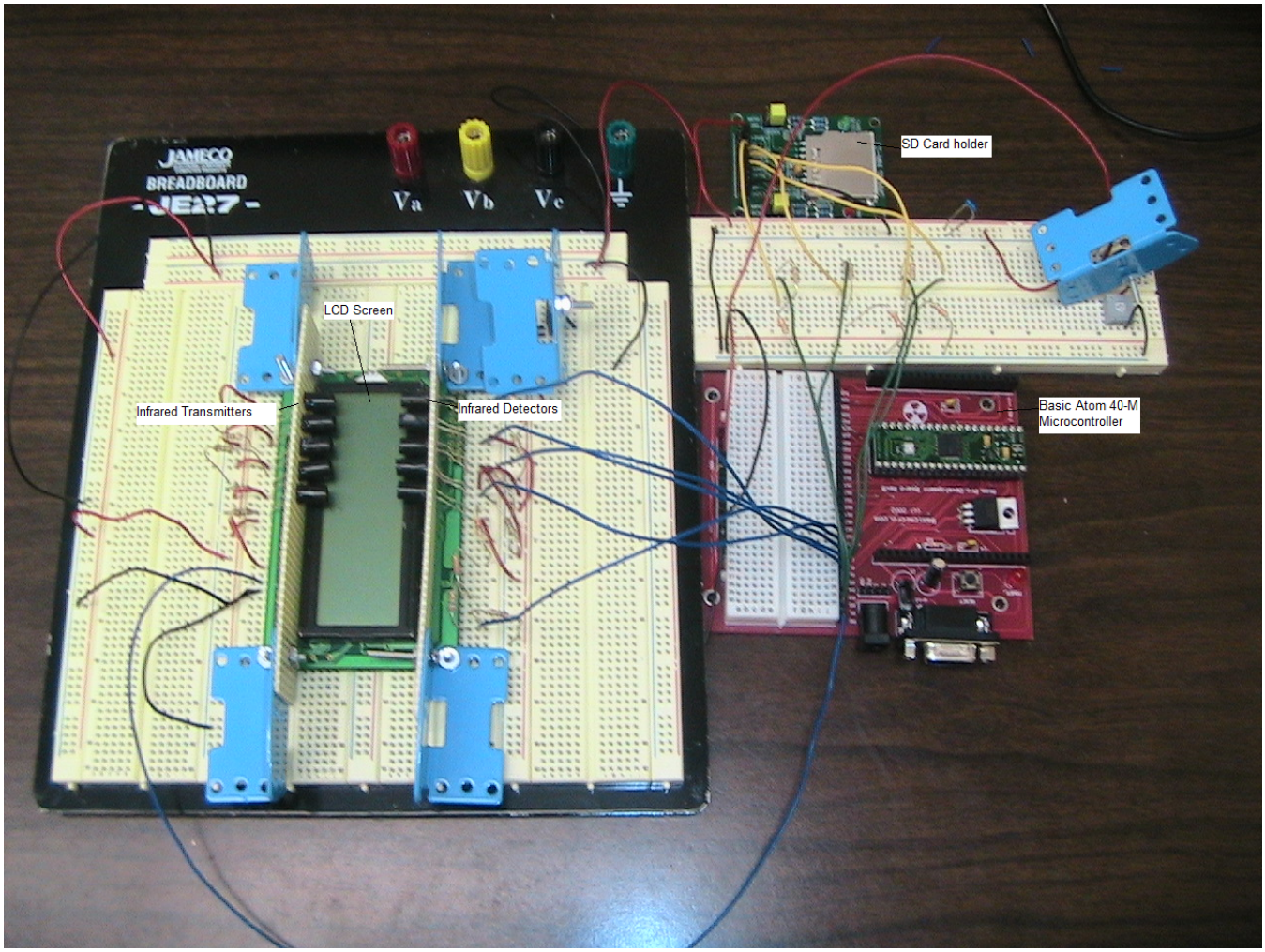

Figure 2: The final touch-screen SD Card Text Reader

2.1 Basic Atom -40M microcontroller

The BASIC Atom -40M is a 40-pin microcontroller that can be programmed using a high level language based on BASIC provided by the manufacturer of the microcontroller. The microcontroller has the following important features needed to implement the project [4]:

- Programmable through a computer using serial communication cable,
- 5V regulated input/output pins,
- Supports RS-232 communication to interface with devices, and
- Supports SPI communication to interface with devices.

This microcontroller will be used as the main host controller to control the LCD screen to display a menu and text files using RS-232 communication. It will be used to control the SD card to read

text files in the card using SPI communication. It will also be used to measure the voltage across the photo-detector in order to determine a touch-event. This is the central component of the whole product as it controls all the other devices.

2.2 Interfacing with the GLK12232-25-SM LCD Screen

The GLK12232-25-SM is an LCD screen with a 122 x 32 pixel graphics display that allows display of text or images loaded in its built-in flash-ROM at specific coordinates of the screen. It can display up to 80 characters. It can be interfaced with the BASIC Atom -40M host controller to display a menu by sending simple commands asynchronously using the "SEROUT" command from the host controller using RS-232 communication [1]. The flash-ROM of the LCD screen was first loaded from a computer with images of buttons using the software provided with the LCD screen. These images of buttons will be placed in a list on the LCD screen to provide a menu with which the user can interact to select text files.

The electrical interface between the LCD screen and the BASIC Atom -40M host controller is shown in Figure 3 below.

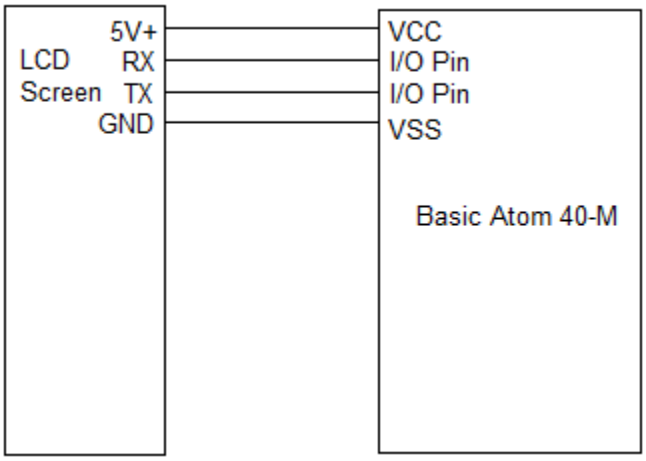

Figure 3: Electrical interface of GLK12232-25-SM LCD Screen with the BASIC Atom 40-M

After connecting the LCD to the BASIC Atom -40M host controller using the electrical interface in Figure 3, the BASIC Atom -40M host controller was programmed to display the menu by sending commands out to the LCD screen to display each button image at specific coordinates so that the buttons appear in a list. RS-232 communication is used by the BASIC Atom -40M host controller to send commands to the LCD screen which involves sending commands asynchronously through the I/O Pin of the host controller connected to the RX pin on the LCD screen. Figure 4 below shows how the menu looks in the final product. The buttons show numbers 1, 2, 3 and 4 which correspond to the first four text files found in the SD card.

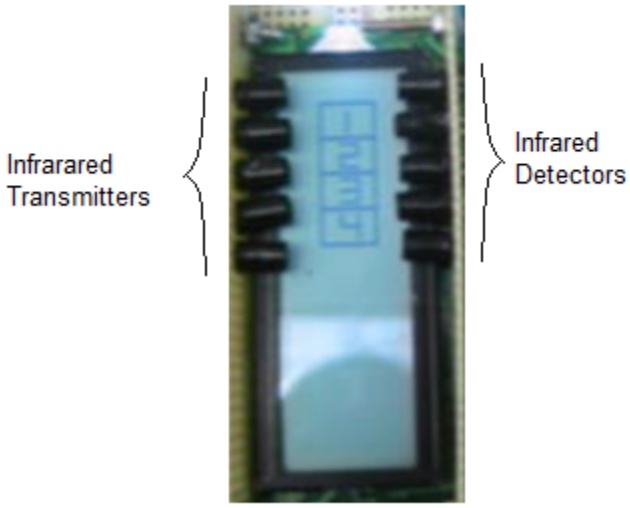

Figure 4: Menu displayed on the LCD screen by the microcontroller.

2.3 Infrared Detectors and Transmitters

To provide the feature of a touch event for the user an array of infrared detectors and transmitters were placed opposite each other. The menu on the LCD screen was positioned in a manner that each button on the menu corresponded with a pair of infrared detector and emitter. In search of the right infrared detector and transmitter for the project, the radiation angle of the infrared detector and the response curve of the transmitter were considered so that there is no overlap in the reception areas of adjacent receivers and in the transmission areas of adjacent transmitters. If there is overlap, then a line of infrared beams cannot be realized because of the possibility of the following circumstances:

1. Adjacent transmitters' transmission areas may overlap at a detector and therefore when a finger is placed between a pair of transmitter and detector, the detector will still receive infrared transmission from the adjacent transmitter and no touch-event will be detected.
2. Adjacent detectors' detection areas may overlap and therefore the detection area is too wide and the detectors' will detect transmission from more than one transmitter and no touch-event will be detected.

QED523 is used as the infrared transmitter and paired with QSD723 as the infrared detector since their radiation and reception angles do not exceed 40° [5, 6] which is also enough to provide detection of a touch event over the button area in the menu on the display. Figure 5 shows the circuit diagram used to connect QED523, the infrared transmitter, to a voltage supply. The maximum current that can flow through QED523 is 30 mA and the threshold voltage is about 2V [5]. Therefore to prevent it from burning out, a 200 Ω resistor is connected in series to keep the current below 30 mA. Figure 6 shows the circuit diagram used to connect QSD723, the infrared receiver, to the BASIC Atom – 40M host controller. The maximum current that can flow through QSD723 is 25 mA [6] and therefore a 10 kΩ resistor is placed in series with it to keep

the current low. Figure 4 from the earlier section shows how the infrared transmitters and detectors were placed around the LCD screen in the final product.

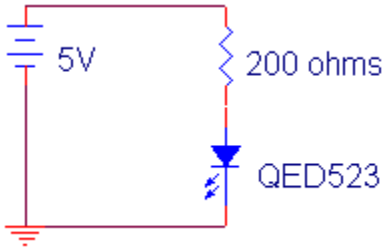

Figure 5: Circuit Diagram for the connection of QED523 to emit infrared

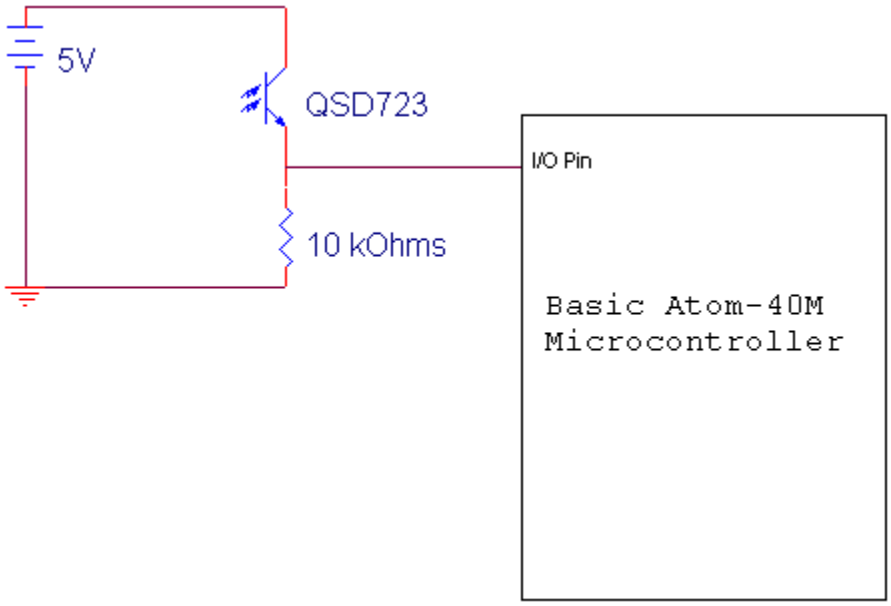

Figure 6: Circuit diagram for the connection of QSD723 to the BASIC Atom -40M

The module TOUCHEVENT below is run in the BASIC Atom -40M host controller in a continuous loop to check for a touch-event on the display. The output pin number corresponds to the button 1, 2, 3 or 4 on the menu and indicates which button was touched on the display. An I/O pin connected to an infrared detector measures the voltage across it and has a value of 1 when no touch-event occurs since the infrared detector received infrared transmission. On the occurrence of a touch-event, the detector corresponding to the touch-event position will have a low voltage that will be measured as a zero at the I/O pin of the BASIC Atom -40M host controller.

Module: TOUCHEVENT
Inputs: Value at I/O pins connected to infrared detectors

Output: Pin number 1, 2, 3 or 4 if touch-event occurs; else, 0 is returned
Algorithm:
IF Pin 1 has value of 0(Low Voltage) THEN
    Return 1
ELSEIF Pin 2 has value of 0(Low Voltage)
    Return 2
ELSEIF Pin 3 has value of 0(Low Voltage)
    Return 3
ELSEIF Pin 4 has value of 0(Low Voltage)
    Return 4
ELSE
    Return 0

The BASIC Atom -40M host controller cannot measure voltage on more than one pin simultaneously. The module TOUCHEVENT measures the voltage at each pin sequentially. This is not a problem for the BASIC Atom -40M host controller to detect a touch, though, because the time it takes the user to touch the screen and then lift his or her finger from it is longer than it takes for the module TOUCHEVENT to run.

2.4 SD Card

To read data from the SD card, the SD card must first be connected to the BASIC Atom -40M host controller so that the Serial Peripheral Interface (SPI) Bus communication system can be used as shown in Figure 7 below. SD card has an operational voltage range of 2.7-3.6V for commands and memory access. Therefore, a 3.3 V regulator is used to power up the SD card. The 3.3 k and 1.8 k resistors are used to create voltage dividers to reduce the BASIC Atom-40M's 5 V signals to 3 V signals on the SD card. The data output pin of the SD card is connected directly to the data input pin of the BASIC Atom -40M and there is no issue in reading the 3 V signals as a high value since the I/O pins use TTL logic in which 3V falls in the input high-voltage range. Voltage signal below 0.8V is considered input low voltage by BASIC Atom -40 M [7].

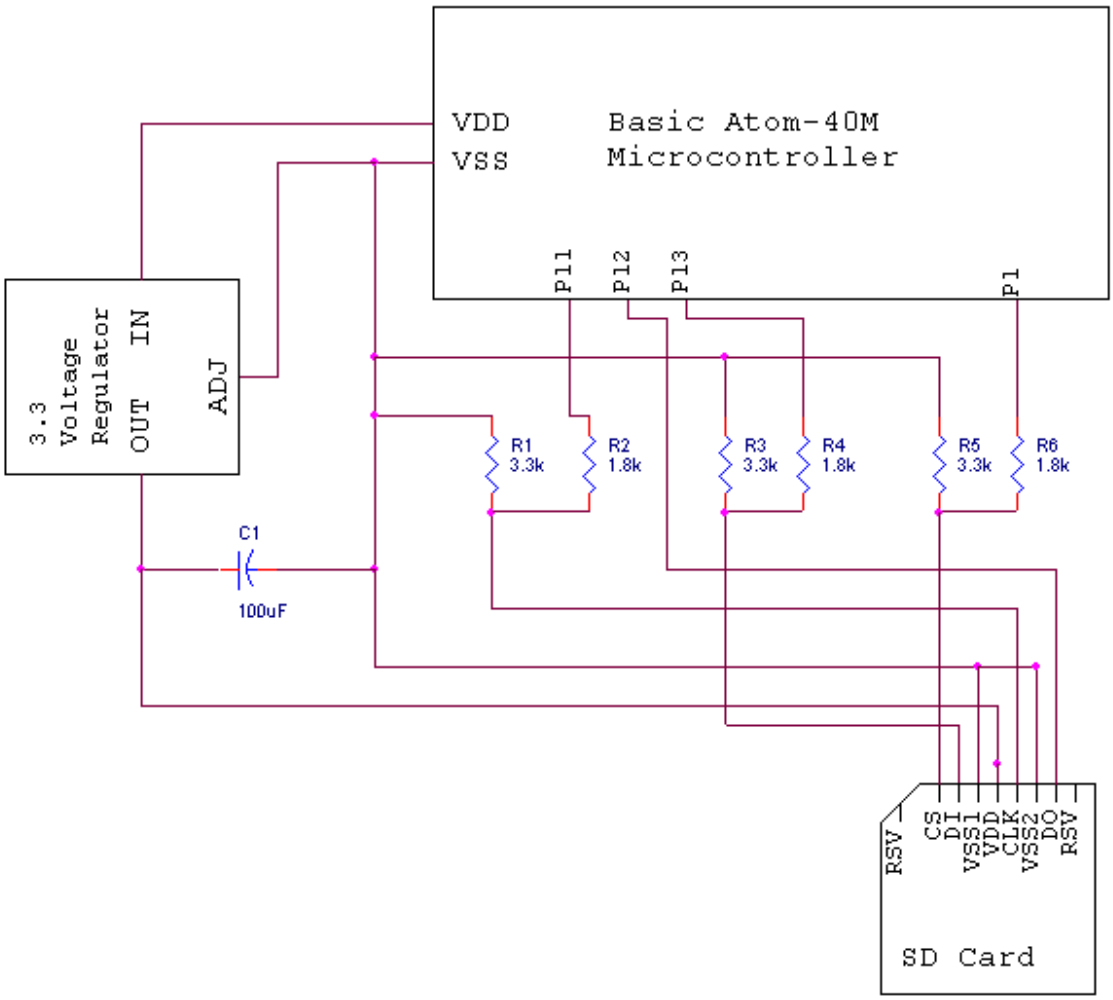

Figure 7: Electrical interface of SD card with the BASIC Atom -40M

SPI communication is the only way data can be read from an SD card by a BASIC Atom -40 M host controller using synchronous communication. Bytes are shifted out to the DataIn pin of the SD card by the BASIC Atom -40M host controller to send commands. Bytes are shifted in from the DataOut pin of the SD card to read data and responses of the command by the BASIC Atom -40M host controller. SPI Bus mode provides a command response mechanism of reading from and writing to an SD card [2]. The card always responds to a command and will give error responses if it encounters a data retrieval problem [2].

2.4.1 Command – Response algorithms

To send a command from the BASIC Atom -40M controller to the SD card, the following SEND_COMMAND module was used. All SD card commands are 6 bytes long and transmitted Most Significant Bit (MSB) first. SD card commands are constructed using the format provided in Figure 8 below.

Byte 1			Bytes 2—5	Byte 6	
7	6	5 0	31 0	7	0
0	1	Command	Command Argument	CRC	1

Figure 8: SD Card Command structure

When there is no command argument 4 bytes of zero values are sent as the command argument. In SPI mode CRC values are treated as don't cares by the SD Card and therefore any value can be sent as the CRC value.

Module: SEND_COMMAND
Inputs: Command (1 byte), Command Argument (4 bytes), CRC Value (1 byte)
Output: No output // the response of the command is read in a different module to determine the success of sending a command to the SD card
Algorithm:
Low CS
Shift out an idle byte 0xFF to waste 8 clock cycles so that change of the CS signal has been recognized by the SD card.
Send the command byte
Send 4 bytes of the command argument
Send the CRC value

To receive a response from the SD card following the sending of a command to it, the following RECEIVE_RESPONSE module was used by the BASIC Atom -40M host controller.

Module: RECEIVE_RESPONSE
Inputs: Valid byte; that is, the expected response byte
Output: True, if response received matched valid byte; otherwise, False
Algorithm:
counter = 0
While counter < 0xFFF //0xFFF retries are given to receive the response
{
    Shift in 1 byte from SD Card as Temp
    If Temp = Valid Byte Then
        Return True
    End If
    counter = counter +1
}
Return False

The response byte received when sending commands to the SD card has the following format.

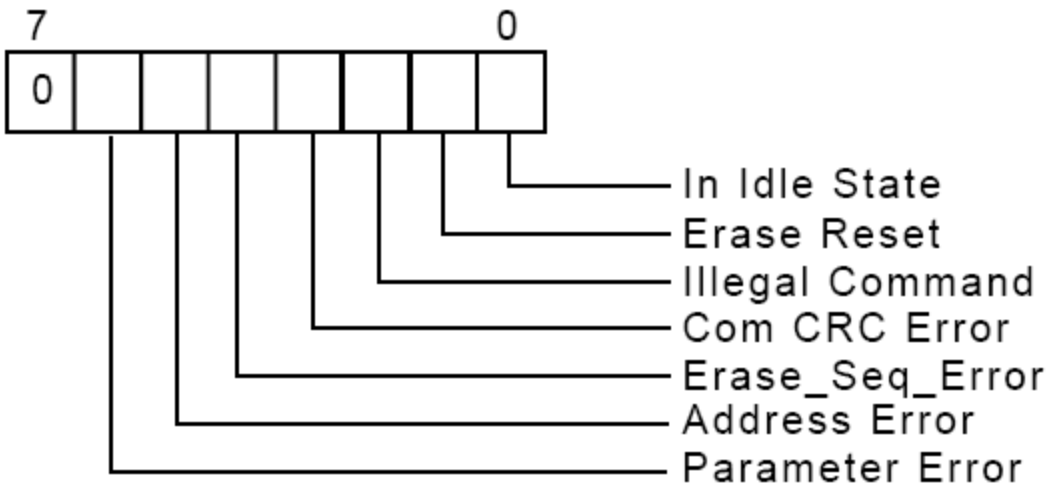

Figure 9: Response byte format

The bits in the response byte are 1's when the labels in Figure 9 are true.

2.4.2 Initializing the SD Card for SPI mode

As soon as the SD card is turned on, its default mode of communication is the SD Bus mode. The BASIC Atom -40M host controller must run the algorithm given below to initialize SPI Bus mode on the SD card.

Module: INITIALIZATION_SPI_MODE
Input: No inputs
Output: True, if initialization to SPI mode is successful; otherwise, False
Algorithm:
High CS
//Shift out 0xFF bytes as idle bytes to waste about 80 clock cycles to allow the card to power up and initialize
For i= 0 to 9
{
    Shift out 0xFF
}
// Send CMD0 to reset the card with CRC value of 0x95 as given in the documentation
SEND_COMMAND (Command = 0x40, Argument = 0x00000000, CRC value = 0x95)
//Expect 0x01 as the expected response since the idle bit should be true
If RECEIVE_RESPONSE (valid = 0x01) == False Then
    High CS to deselect the card
    Return False
End If
// From now on the CRC value are defined as don't cares by the SD card
//Send CMD1 to activate the initialization process of the SD card in SPI mode

```
SEND_COMMAND (Command = 0x41, Argument = 0x00000000, CRC value = 0xFF)
If RECEIVE_RESPONSE (valid = 0x00) == False Then
        High CS to deselect the card
        Return False
End If
Shift out idle byte 0xFF to waste 8 clock cycles
High CS
Shift out idle byte 0xFF to waste 8 clock cycles
Return True
```

2.4.3 Reading a single block of data from an SD Card

The default block length set on the SD card is 512 bytes. The BASIC Atom -40M must first send the READ_SINGLE_BLOCK command to the SD card to read a block. The SD card responds with a Data Error Token byte for which the expected value is 0x00 as we want no error. The SD card then sends 0xFE as the start block byte followed by 512 bytes of data and 2 bytes of don't care CRC value. The BASIC Atom -40M uses two modules: OPEN_SECTOR to start the reading of a single block of data and CLOSE_SECTOR to end the reading of single block of data. After OPEN_SECTOR is called 512 bytes of data need to be read and then CLOSE_SECTOR needs to be called by the BASIC Atom -40M for a successful read of single block of data.

```
Module: OPEN_SECTOR
Input: Sector value; that is, the sector number to open or read
Output: True, if sector opened successfully; otherwise, False
Algorithm:
SEND_COMMAND(Command = 0x51, Argument = sector*512, CRC = 0xFF)
If RECEIVE_RESPONSE (valid = 0x00) == False Then
        High CS to deselect the card
        Return False
End If
Low CS
If RECEIVE_RESPONSE (valid = 0xFE) == False Then
        High CS to deselect the card
        Return False
End If
Return True


Module: CLOSE_SECTOR
Input: No inputs
Output: No outputs
```

Algorithm:
Shift in 1 byte of CRC value
Shift in 1 byte of CRC value
Shift out an idle byte 0xFF to waste 8 clock cycles for the change in CS signal to take place
High CS
Shift out an idle byte 0xFF to waste 8 clock cycles for the SD card to recognize change in CS signal

2.4.4 FAT32 system overview

To read or write to any storage device, it must be formatted with a File Allocation Table (FAT) file system. A FAT file system is an organized lookup method to store and retrieve data from a storage device. When storing data on a memory storage device, the file name and location are stored in the FAT, and when reading data from a memory storage device, the FAT is used to look up the location of the data of a particular file. The FAT file system is found in the first couple of sectors of a memory storage device. In this project, a 1Gigabyte SD Card is used which was formatted using a computer to have 512 allocation unit size per sector to match the default block length size of the SD card and have the FAT 32 file system. The SD card was found to have a Master Boot Record (MBR) since it was formatted just like a hard drive. Therefore, the first sector of the memory of the SD card is the MBR from which the boot record address of the FAT needs to be found. The boot record has the address of the root directory on the SD card where names of files are saved along with their start addresses. The root directory is fixed in size and is allocated 1 cluster. A cluster is made up of 512 byte sectors and the number of sectors in it is defined in the boot record.

Table 1 below summarizes the meanings of the values read from the MBR and Table 2 summarizes the meanings of the values read from the boot record which will be used in the initialization algorithm of the FAT in the next section.

Table 1: Summary of values read from MBR [8]

Byte Offset	Field Length	Field Name and Definition
0x01C6	4 bytes	**Relative Sectors** - The offset from the beginning of the disk to the beginning of the volume, counting by sectors.
0x1FE	2 bytes	end of sector marker always set to 0xaa55

Table 2: Boot Record structure [9]

Name	Offset (byte)	Size (bytes)	Description
BPB_BytsPerSec	11	2	Count of bytes per sector. This should be 512 bytes for the formatting used on the SD card used.
BPB_SecPerClus	13	1	Number of sectors per allocation unit.
BPB_RsvdSecCnt	14	2	Number of reserved sectors in the Reserved region of the volume starting at the first sector of the volume.
BPB_NumFATs	16	1	The count of FAT data structures on the volume.
BPB_HiddSec	28	4	Count of hidden sectors preceding the partition that contains this FAT volume.
BPB_TotSec32	32	4	This field is the total count of sectors on the volume.
BPB_FATSz32	36	4	This field is the FAT32 32-bit count of sectors occupied by ONE FAT.
BPB_RootClus	44	4	This is set to the cluster number of the first cluster of the root directory, usually 2 but not required to be 2.
BPB_Sign	510	2	End of sector marker is 0xaa55

2.4.5 Initializing the FAT 32 structure

The algorithm FAT_INIT is run by the BASIC Atom -40M to get the address of the root directory and other values from the FAT-32 structure on the SD card for future use in reading text files from the SD card. FAT_INIT uses small modules called Get4bytes, which shifts in 4 bytes of data, Get2bytes, which shifts in 2 bytes of data, and Get1byte, which shifts in 1 byte of data from the data output pin of the SD card into the BASIC Atom -40M host controller.

```
Module: FAT_INIT
Input: No inputs
Output: True; if FAT initialization is successful; otherwise false
Algorithm:
//Open sector 0 to get location of root cluster of the boot sector
If (OPEN_SECTOR (Sector = 0) == False) Then
        CLOSE_SECTOR ();
        Return False;
End If
//Read 512 bytes of data of the MBR
For  i= 0 to 511
{
        If i=0x1c6 Then
                RootCluster = Get4bytes() // root cluster is address of the boot sector
                i=i+3
```

```
        Else If i=0x1fe
                CAS = Get2bytes(); // CAS is the marker value of the MBR
                i=i+1
                If CAS != 0xaa55 Then
                        Display on LCD Screen "Error0: Unable to get 0xaa55 marker"
                        Return False
                End If
        Else
                Get1byte();
        End If
}
CLOSE_SECTOR();

//Open boot record of FAT using address stored in RootCluster
If (OPEN_SECTOR(Sector = RootCluster) == False) Then
        CLOSE_SECTOR()
        Return False
End If
//Read 512 bytes of boot record data and store values that are needed with the corresponding
variable names as given in Table 2
For i=0 to 511
{
        If i = 0x0B Then
                Bytes_per_Sector = Get2bytes()
                i=i+1
                If Bytes_per_sector != 0x0200 Then
                        Display on LCD Screen "Error1: not correct bytes per sector"
                        Return False
                End If
        Else If i=0x0d
                sectorPerCluster = Get1byte()
        Else If i=0x0e
                ReservedSectorCount = Get2bytes()
                i=i+1
        Else If i=0x10
                BPB_NumFATs = Get1byte()
                If BPB_NumFATs != 0x02 Then
                        Display on LCD Screen "Error2: number of fats”
                        Return False
                End If
```

```
        Else If i=0x1c
                BPB_hiddensectors = Get4bytes()
                i=i+3;
        Else If i = 0x20
                BPB_TotSec32 = Get4bytes()
                i=i+3
        Else If i = 0x24
                BPB_FATsz32 = Get4bytes()
                i=i+3
        Else If i = 0x2c
                rootCluster = Get4bytes() // Address of the root directory
                i=i+3
        Else If i = 0x1fe
                BPB_Sign = Get2bytes()
                i=i+1
                If BPB_Sign != 0xaa55 Then
                        Display on LCD Screen "Error3: Not able to get partition marker 0xaa55"
                        Return False
                End If
        Else
                Get1byte()
        End If
}

CLOSE_SECTOR()

//Calculations as given by the FAT 32 specification of Microsoft [9]
firstDataSector = BPB_NumFATs * BPB_Fatsz32
firstDataSector += BPB_hiddenSectors
firstDataSector += ReservedSectorCount
dataSectors = BPB_NumFATs * BPB_Fatsz32
dataSectors = dataSectors - ReservedSectorCount
totalCluster = dataSectors /sectorPerCluster

Return True
```

2.4.6 Filtering the first four text files

Before the algorithm to find the addresses and names of the first text files in the SD card is presented, it is important to know the 32-byte root directory entry structure; also called the file name structure. The root directory is found in a cluster which is made up of sectors. Therefore

each sector of the root cluster must be read until four text files have been found. Table 3 below summarizes the root directory entry structure.

Table 3: FAT 32 Byte Directory Entry Structure [9]

Name	Offset (byte)	Size (bytes)	Description
DIR_Name	0	11	Short name.
DIR_Attr	11	1	File attributes: ATTR_READ_ONLY 0x01 ATTR_HIDDEN 0x02 ATTR_SYSTEM 0x04 ATTR_VOLUME_ID 0x08 ATTR_DIRECTORY 0x10 ATTR_ARCHIVE 0x20 ATTR_LONG_NAME ATTR_READ_ONLY | ATTR_HIDDEN | ATTR_SYSTEM | ATTR_VOLUME_ID The upper two bits of the attribute byte are reserved and should always be set to 0 when a file is created and never modified or looked at after that.
DIR_NTRes	12	1	Reserved for use by Windows NT. Set value to 0 when a file is created and never modify or look at it after that.
DIR_CrtTimeTenth	13	1	Millisecond stamp at file creation time. This field actually contains a count of tenths of a second. The granularity of the seconds part of DIR_CrtTime is 2 seconds so this field is a count of tenths of a second and its valid value range is 0-199 inclusive.
DIR_CrtTime	14	2	Time file was created.
DIR_CrtDate	16	2	Date file was created.
DIR_LstAccDate	18	2	Last access date.
DIR_FstClusHI	20	2	High word of this entry's first cluster number
DIR_WrtTime	22	2	Time of last write.
DIR_WrtDate	24	2	Date of last write.
DIR_FstClusLO	26	2	Low word of this entry's first cluster number.
DIR_FileSize	28	4	32-bit DWORD holding this file's size in bytes.

Based on the root directory structure, the BASIC Atom -40M runs the FIND_TEXT_FILES algorithm to find the first four text files with .txt extension and saves the file names and addresses in its internal Flash memory. This algorithm requires that there are at least four text files present in the SD card memory. Before the FIND_TEXT_FILES algorithm is presented, the helper module called GET_FIRST_SECTOR_OF_CLUSTER is given.

Module: GET_FIRST_SECTOR_OF_CLUSTER
Input: Cluster number
Output: First sector of cluster
Algorithm:
FirstSector = (Cluster Number – 2) * sectorPerCluster + firstDataSector
Return FirstSector

```
Module: FIND_TEXT_FILES
Input: No inputs
Output: True, if four text files have been found; otherwise, False
Algorithm:
FileCounter = 0;
First_Sector_Of_RootDirectory = GET_FIRST_SECTOR_OF_CLUSTER(rootCluster)
For each Sector in the Root Directory Cluster
{
        If OPEN_SECTOR(Sector) == False Then
            Display on LCD Screen “Error Opening sector in Find_Text_Files”
        Else
        {
            //read sector
            For i=0 to 511
            {   //read each file entry structure of 32 bytes
                For j = 0 to 31
                {       Name = Get 11 bytes of data
                        Attribute = Get1byte()
                        NTreserved = Get1byte()
                        timeTenth = Get1byte()
                        createTime = Get2bytes()
                        createData = Get2bytes()
                        lastAccessDate = Get2bytes()
                        firstClusterHI = Get2bytes()
                        writeTime = Get2bytes()
                        writeDate = Get2bytes()
                        firstClusterLO= Get2bytes()
                        fileSize = Get4bytes()
                }
                If name != 0xe5(check if empty) and attribute != ATTR_DIRECTORY Then
                {
                        If Last 3 bytes of Name contains TXT Then
                        {
                            If FileCounter <= 3 Then
                            {
                                Save Name in memory of BASIC ATOM
                                Save firstClusterHI in memory of BASIC ATOM
                                Save firstClusterLO in memory of BASIC ATOM
                                Save FileSize in memory of BASIC ATOM
                                FileCounter ++
```

```
                        }
                        End If
                    }
                    End If
                }
                End If
            }
        }
        End If
        CLOSE_SECTOR()
        //Check if 4 files have been found in the sector if not continue reading more sectors
        If FileCounter < 3 Then
            Return False
        Else Return True
}
//Check if four files have been found after the entire cluster has been read
If FileCounter < 3 Then
    Return False
Else Return True
```

The algorithm FIND_TEXT_FILES is run before loading the menu screen on the LCD screen so that, when the buttons are touched, the text file can be retrieved easily based on the addresses stored in the internal memory of the BASIC Atom -40M microcontroller.

2.4.7 Reading text files from the SD card and displaying them on the LCD screen

The BASIC Atom -40M runs the algorithm READ_FILE to read a text file from the SD Card and to display the text on the LCD screen. If the entire text does not fit into the LCD screen, then the READ_FILE module calls the INTERRUPT module so that the user can touch the screen to read the next set of characters in the text file. The INTERRUPT module is called every time there are more characters of the text file to be displayed on the LCD screen. The LCD screen can display a maximum of 80 characters. Once the complete text file has been displayed, the INTERRUPT module is called again so that the user can touch the LCD screen to return to the menu.

The GET_NEXT_CLUSTER module is presented first as it is used by the READ_FILE module to get the next cluster of the file if the file is larger than the size of one cluster. The INTERRUPT module is presented followed by the READ_FILE cluster.

Module: GET_NEXT_CLUSTER [9]
Inputs: Current Cluster address

```
Outputs: Next Cluster address if successful; otherwise, False
Algorithm:
FATEntrySector = (CurrentClusterAddress * 4)/bytesPerSector + unusedSectors +
                    reservedSectorCount
FATEntryOffset = (CurrentClusterAddress *4)%bytesPerSector
If OPEN_SECTOR(FATEntrySector) == False Then
        Return False
Else
{
        For i=0 to 511
        {
                If i=FATEntryOffset then
                        FATEntryValue = Get4bytes()
                        i=i+3
                Else
                        Get1Byte()
                End If
        }
        CLOSE_SECTOR()
        Return FATEntryValue & 0x0fffffff
}
End If

Module: INTERRUPT
Inputs: Value at I/O pins connected to infrared detectors
Outputs: True if screen was touched; otherwise, False
While True
{

        If Pin 1 has value of 0(Low Voltage) Then
                Return True
        Else If Pin 2 has value of 0(Low Voltage)
                Return True
        Else If Pin 3 has value of 0(Low Voltage)
                Return True
        Else If Pin 4 has value of 0(Low Voltage)
                Return True
        End If
}
End While
```

```
Module: READ_FILE
Inputs: File Number; that is the file number from the list stored in the memory of the BASIC
Atom -40M after FIND_TEXT_FILES is run
Outputs: Display file on screen if successfully able to read; otherwise, False
ByteCounter = 0 //total bytes read from the SD card
CharacterCounter = 0 //Counts the number of characters displayed
DisplayMenu= False
ClusterOfFile = firstClusterHI << 16 | firstclusterLO //firstClusterHI and firstClusterLO values
are retrieved from the memory of the BASIC Atom -40M from the values stored in it during
FIND_TEXT_FILES module execution.
Sector = GET _FIRST_SECTOR_OF_CLUSTER(ClusterOfFile)
While True
{
        For SectorCounter = 0 to sectorPerCluster-1
        {
                If OPEN_SECTOR(Sector + SectorCounter) == True Then
                {
                        For k=0 to 511
                        {
                                If CharacterCounter < 80 then
                                        If !DisplayMenu Then
                                                Display Get1byte()
                                                CharacterCounter++
                                                ByteCounter++
                                        End If
                                Else
                                        CharacterCounter = 0
                                        If !DisplayMenu Then
                                                Pause 6 seconds
                                                If Interrupt() == True Then
                                                        Clear LCD Screen
                                                        Display Get1byte()
                                                        CharacterCounter++
                                                        ByteCounter++
                                                End If
                                        End If
                                End If
                                If !DisplayMenu Then
                                        If ByteCounter >= FileSize Then
                                                DisplayMenu = True
```

```
                                End If
                        End If
                }
        }
        End If
        CLOSE_SECTOR()
        If DisplayMenu Then
                Pause 6 seconds
                If Interrupt() == True Then
                        Return True
                End If
        End If
    }
    ClusterOfFile = GET_NEXT_CLUSTER(ClusterOfFile)
    Sector = GET_FIRST_SECTOR_OF_CLUSTER(ClusterOfFile)
}
```

2.5 Overall Algorithm

The overall algorithm run by the BASIC Atom -40M to accomplish the requirements of the project is given below.

```
INITIALIZATION_SPI_MODE()
FAT_INIT()
While True
{
    Display Menu on LCD Screen
    FileNumber = 0
    While FileNumber == 0
    {
        FileNumber = TouchEvent()
    }
    READ_FILE(FileNumber)
}
```

3. Results

The video at http://www.dailymotion.com/relevance/search/touch-screen/video/xas5jk_touchscreen-sd-card-reader_tech demonstrates the final product. From the video we can see that the user is able to touch the buttons in the menu screen to read different text files. Since the LCD screen limits 80 characters per screen view, the user is also able to touch the screen to read the next 80 characters and so on until the entire text file is read. Once the text file is read, the user is able to touch the screen to return to the menu and touch another button to read a different file. In the video, the

user also showed that if a text file was modified from a computer and then read from touch-screen SD card reader, then the changes can be seen accordingly on the screen when that particular text file is selected for viewing. The SD card was also stored with a mixture of different kinds of files with different extensions. The touch-screen SD card reader was able to filter out the text files proving the success of the FIND_TEXT_FILES module.

4. Conclusion

The project was successful as the design requirements were met. The user is able to touch the screen and make the necessary events happen. The user is able to touch a button on the menu screen to read a text file. The four buttons correspond to the first four text files in the SD card. There is no limitation on the size of the text file that the touch-screen SD-card reader can read. If the text file exceeds the 80 character size limit of the display screen, then the user is able to touch the touch-screen to continue reading the text file. Once the end of the text file is reached, the user is able to touch the screen to return to the menu. This project was fruitful in showing how to access the memory of an SD card from a host controller. This is very useful for realizing other projects that require a microcontroller like the BASIC Atom -40M to read a lot of data. SD cards can be used as the memory to read data instead of being limited to the small memory space 250 MB of space available on the BASIC Atom -40M.

References

[1] Matrix Orbital. GLK12232-25-SM User Manual,
<http://www.jameco.com/Jameco/Products/ProdDS/281391Ver1.4.pdf>
[2] SanDisk. Secure Digital Card Product Manual, Revision 1.9, December 2003.
[3] Basic Micro. BasicATOM Syntax Manual,
<http://basicmicro.com/downloads/docs/atom.pdf>
[4] Basic Micro. BasicATOM 40-M Data Sheet,
<http://basicmicro.com/downloads/datasheets/B0091.pdf>
[5] FairChildSemiconductors. Plastic Infrared Light Emitting Diode datasheet,
<http://www.fairchildsemi.com/ds/QE/QED523.pdf>
[6] FairChildSemiconductors. Plastic Silicon Infrared Phototransistor datasheet,
<http://www.datasheetcatalog.org/datasheet/fairchild/QSD723.pdf>
[7] Davis, Leroy. “Logic Threshold Voltage Levels”, 6 Sept. 2009
<http://www.interfacebus.com/voltage_threshold.html>
[8]Microsoft. “Disk Concepts and Troubleshooting”, <http://technet.microsoft.com/en-us/library/cc977219.aspx>
[9] Microsoft Extensible Firmware Initiative FAT32 File System Specification, Revision 1.03, December 6, 2000