MP3 RECORDER AND PLAYBACK

Anish Churi
Virag Doshi

Final Project Report
ECEN 5613 Embedded System Design
December 12, 2015

TABLE OF CONTENTS

1 INTRODUCTION

MP3 music compression format is an extremely popular choice for digital audio compression. A high compression ratio is achieved using MP3 encoding, yet maintaining a great audio quality. Space and bandwidth are important criteria for maintaining and distributing files over internet. The compression achieved using MP3 while maintaining a great audio quality makes it a logical choice for storing and distributing audio files. As a result of this, a variety of portable MP3 players are developed with an intention to capitalize on the growing market demand. The quality of audio is improving everyday with lots of additional features being added to customize music as per the user's needs. Although the MP3 player market is currently dominated by lot of well established companies, understanding the process of how audio files are transmitted, compressed and decoded, and how different file types are detected, seemed very fascinating. Hence an attempt has been made to design a basic MP3 player and recorder.

2 TECHNICAL DESCRIPTION

The following section contains detailed design and implementation of the MP3 Player and Recorder using MSP430. The Board Design section contains details about all the hardware components of the system, Firmware design covers all the driver code for LCD, VS1063 encoder/decoder IC, SD card interfacing, and the Software Design describes the file access system displayed on LCD, and the PC development tools.

2.1 Board Design

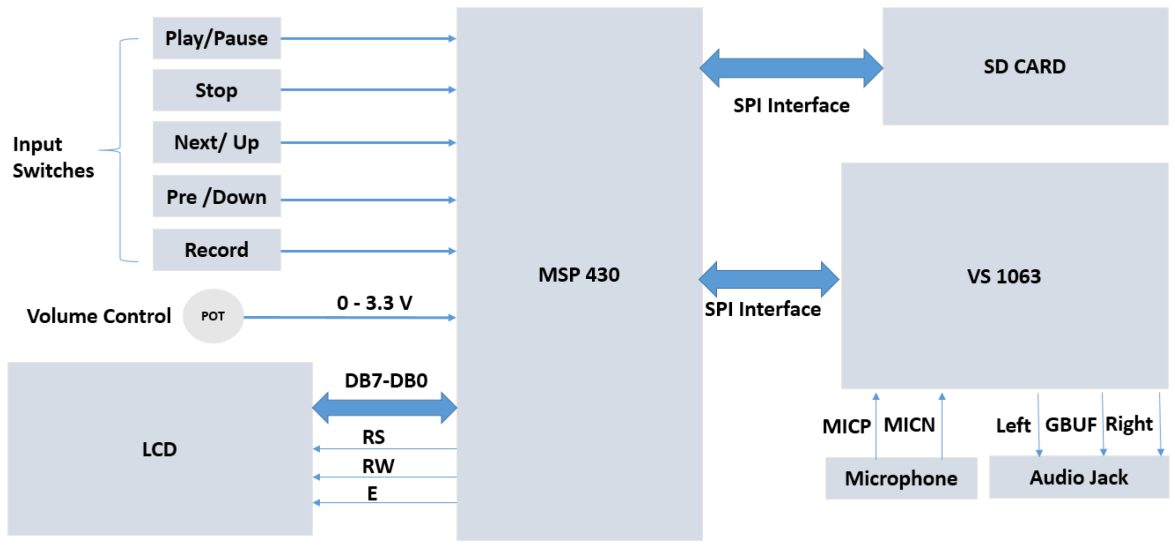

Figure 2.1: System Block Diagram

As shown in Figure 2.1, this system is based on the MSP430F5529 [1] development board. The MSP430 acts as a master controller for all the other devices on the board. The SD card is used to store the MP3 files to be played and recorded, and communicates with the Master Controller using the SPI interface. The other hardware component of the system is the breakout board for VS1063 IC. The primary purpose of the VS1063 IC is to encode and decode the MP3 audio files. As shown in the Figure 2.1 this VS1063 IC also communicates with MSP430 via the SPI interface. The decoded audio data is then played through the audio jack, which is directly interfaced with the VS1063 breakout board.

The Optrex DMC20434 LCD is added on the board for the user interface. The LCD is used to display the list of files on the SD card, the current file playing, and other user options. The board also contains a number of input switches for play/pause, stop, and an analog control for volume of the audio.

2.1.1 MSP430 Development Board

The complete system is based on the MSP430F5529 development board from Texas Instruments. It offered a quick way to start developing on MSP430 with on board emulation for programming and debugging. The development board has two crystal oscillator circuits, one with crystal frequency 4 MHz for internal clock and one with crystal frequency 32 KHz for the internal Real Time Clock which is used in the system to associate the time with the recorded files stored on the SD Card. The development board also comes with some important features such as push buttons and LEDs for user interface. Figure 2.2 shows the functional block diagram of the development board.

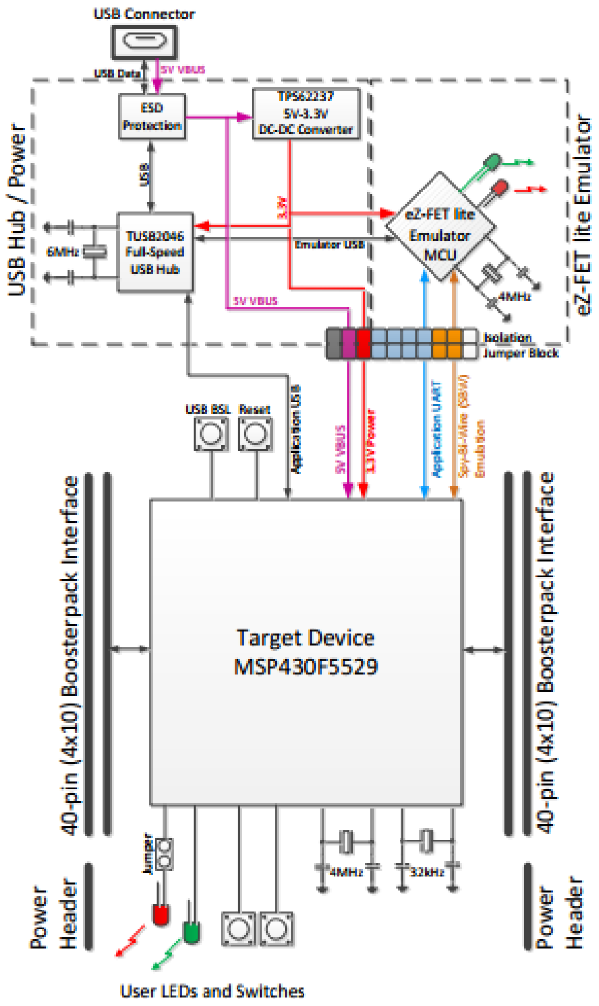

Figure 2.2: MSP430 Development Board Block diagram [3]

2.1.2 SD Card Interface

The SD hardware interface consists of a micro SD card adapter. The adapter pins are soldered on to the board and then wire wrapped to the MSP430 microcontroller port pins. NC pin was pulled up and GND pins are connected to ground, and VCC is connected to 3.3 V output of the MSP430 breakout board. A 0.1 µF decoupling capacitor is soldered between the ground and VCC pins.

The communication between the Master controller and the SD card is achieved using the SPI interface. The inputs to the SD card are the slave select, MISO and SCLK signals. The output of the chip is the MISO signal. MSP430 ports have internal pull-ups and pull-down resisters which can be enabled or disabled using software. Registers were programmed to enable these pull up registers for the SPI communication.

Table 2.1 lists chip signals and their connection to MSP430 controller. The voltage values for all the signals for the SD card and MSP430 microcontroller is 3.3 V, so no logic level translation is required when interfacing the SD card [9].

Table 2.1: SD Card SPI Signals

Signals	SD card Pin Number	MSP430 Port
/SS	1	P1.6
SCK	5	P4.3
MOSI	2	P4.1
MISO	7	P4.2

2.1.3 *VS1063 Encoder and Decoder IC Interface*

The hardware interface for the encoder and decoder circuit uses the breakout board for VS1063, TRRS audio jack and a small electret microphone. The VS1063 [4] is a versatile encoder and decoder capable of decoding various audio file formats such as OGG Vorbis, MP3, AAC, and WMA. The IC operates at 3.3 V and so no additional voltage level conversion was needed. The IC operates at a frequency of 12.288 MHz. The VCC of the IC was connected to 3.3 V output of the MSP430 breakout board.

Table 2.2: VS1063 SPI Signals

Signals	Pin Description	VS1063 Pin Number	MSP430 Port
xCS	Chip Select Input	23	P8.1
SCK	Clock for Serial Bus	28	P6.4
MOSI	Serial Input	29	P3.0
MISO	Serial Output	30	P3.1
DREQ	Data Request Input Bus	8	P7.4
BSYNC/xDCS	Byte Sync	13	P2.0
xRST	Active Low Reset	3	P8.2

The communication between the Master controller and VS1063 IC is achieved using SPI interface. The inputs to the IC are the Chip select, MISO, SCLK and BSYNC. The output of the chip is the MISO signal and DREQ. BSYNC signal is generated by the master to ensure correct bit alignment of input bit stream. The first sampling edge of the clock depending on the selected polarity during which BSYNC is logic high, marks the first bit of byte.

DREQ is an output pin which is used to signal if the 2048 byte buffer FIFO of VS1063 is capable of receiving data. If DREQ is high VS1063 can take at least 32 bytes of data. Again, for the SPI communication the internal pull-ups of the MSP430 port pins were enabled. VS1063 IC also has a UART for debugging purpose that wasn’t used so the RX pin of the IC was pulled high on the development board and TX was left unconnected. Table 2.2 lists chip signals and their connection to MSP430 controller.

The hardware interfacing of VS1063 IC also included the interfacing of a small electret microphone, and a 3.5 mm TRRS audio jack. Differential microphone inputs, MICP and MICN were used for this purpose. The VS1063 datasheet was referred, and accordingly the hardware circuit was designed to interface the microphone with the IC [5]. Similarly the TRSS 3.5 audio jack was interfaced. Table 2.3 lists the pin connections for microphone and audio jack.

Table 2.3: VS1063 SPI Signals

VS1063 Signals	**Pin Description**	**Connections**
MICP	Positive Differential mic input	Microphone Terminal 1
MICN	Negative Differential mic input	Microphone Terminal 2
Left	Left channel output	Audio Jack Tip
Right	Right channel output	Audio Jack Ring 1
GBUF	Common Buffer for headphones	Audio Jack Sleeve

2.1.4 LCD and User Interface

The Optrex DMC 20434 LCD was added to the system to provide a user interface. Along with the LCD, various input switches were also added to the system. The switches were used to play/pause, stop, access files and to record the audio from the microphone. The data pins and control signals of the LCD were connected directly to port pins of the microcontroller. The VCC of the LCD was connected to 5V supply from the MSP430 board.

The major concern was whether a signal level translation would be required. The LCD module and the controller datasheet [13] was referred and it was found out, that the minimum voltage required for the signal to be detected as logic high is 2.2V. Thus the need for logic level translation was eliminated. A potentiometer was also added to the system, to provide an option for the user to control the volume of the audio output. Table 2.3 shows the connections for the above with the MSP430 microcontroller.

Table: 2.4 LCD pins and User Inputs

Signals	**Pin Description**	**MSP430 Port**
DB7-DB0	LCD data bus	P6.7 –P6.0
E	Control Signal LCD Enable	P2.3
RS	Control signal Register Select	P3.7
RW	Control signal LCD read/write	P2.6
Pause/Play	Input switch	P1.2
Stop/Home	Input switch	P1.3
Next/Up	Input switch	P1.4
Previous/Down	Input switch	P1.5
Record Audio	Input switch	P2.4
Volume Control	Analog Input	P7.0

2.2 Firmware Design

Code Composer Studio (CCS) by Texas Instruments is the preferred IDE for MSP430 code development, thus all the code was written in C on Code Composer Studio 6.1.1. Firmware for this project involved initialization of MSP430 operating attributes such as the core voltage and the core operating frequency, and it also involved the drivers for interfacing the SD card and VS1063. Apart from this, the firmware also has the functions for initialization and for reading the ADC and RTC values.

The firmware may be divided into different parts, and the division is closely related to the different hardware elements which are detailed in section 2.1.

2.2.1 MSP430 Configuration Drivers

These drivers contain the initialization functions for the MSP430 core parameters, buttons, ADC and RTC and are present in the config.c file. The description of these functions are given below.

Table 2.5 MSP430 Configuration Driver functions

Function	**Description**
set_MCLK_25MHz()	Sets the core frequency to 25 MHz, also sets core voltage to 3.3V.
ctrlBtn_portInit()	Initializes all the port pins interfaced with buttons.
ADC_init()	Initializes the ADC.
ADC_read()	Reads ADC values.
RTC_init()	Initializes the RTC, and set its initial parameters.
RTC_read()	Reads the RTC value.

To play MP3 files without any glitches, the VS1063 needs to receive data at a very fast rate. Thus, we needed the SPI communication between the microcontroller and SD card, and also in between the microcontroller and the VS1063 to be fast. To achieve this, we had to increase the MCLK, the clock on which the MSP430 core operates, to an operating frequency of 25 MHz.

For the core to operate at 25 MHz, the core voltage first needs to be set to 3.3 V. This needs to be done in a number of steps, since the initial operating voltage is set at 1.6 V. Once that is done, the DCO reference is set to REFO, and the DCO range is set for 50 MHz operation. The clock frequency is then divided by the factor of 2, using the FLL. The fault bits are then polled to let the oscillator stabilize.

For all the port initializations, first the I/O function needs to be selected. Then, in case the port pin is an input, the corresponding bit in the PxDIR register needs to be reset. Internal pull-up or pull-down can be enabled using the PxREN register, and whether a pull-up is used or a pull-down is used depends on the value inside the PxOUT register. Finally, the value can be read from the PxIN register.

For controlling the volume levels, the internal ADC with the resolution of 12 bits was used. The ADC was set to use external reference voltages, which are set to be 3.3 V and 0 V for the positive and the negative reference respectively.

The RTC inside the MSP430 was used in the calendar mode. Since the device has no battery to keep the RTC running, it was initialized while compiling to start at the date of the project presentation, and the initial time was set to be 6:30 PM.

2.2.2 SD Card and FatFs Drivers

While the SPI drivers written for the communication with the SD card were written by us, the SD card FatFs drivers contain some code which was leveraged.

We used the generic FatFs library by Chan [10]. The library is divided into number of separate files – ff.h, ff.c, diskio.h, ffconfig.h, integer.h and mmc.h and mmc.c. The application note given on the FatFs webpage [11] also provides details on how to use the library for interfacing with SD card. We also found some example codes for microcontrollers such as LCP23xx, PIC24 and STM32 online [10].

The different library files used for the SD card interfacing are as given below.

Table 2.6: SD card firmware Parts

No.	Name of the file	Description
1	SD_spi.h, SD_spi.c	Contains the port initializations for all the SD card pins. Also contains the SPI drivers needed for sending and receiving data.
2	integer.h	Integer type definitions for the FatFs module.
3	mmc_driver.h, mmc_driver.c	Contains the timeout functions and functions for sending commands to the SD card. Also contains functions to select and deselect the SD card.
4	mmc.h, mmc.c	Contains the functions used by the FatFs library to initialize and check the status of the SD card. Also contains the functions required to send and receive data to and from the SD card.
5	ffconf.h	Contains the different definitions of parameters used by the FatFs library.
6	diskio.h	Contains the low disk interfacing parameter definitions used by the FatFs library.
7	ff.h, ff.c	Contains the functions required to implement a FAT file system on the SD card.

SD Card SPI communication

The SD protocol is byte oriented. The SD card, when being initialized, needs to operate at the SCLK frequency of 100 KHz to 400 KHz. Later, we can increase the operating frequency for fast operation. In our code, we have increased it later, to 12.5 MHz, instead of about 397 KHz. The SD card protocol defines that the SPI clock be at logic 0, when idle, and that the data must be gated out by the microcontroller at the falling edge of the clock.

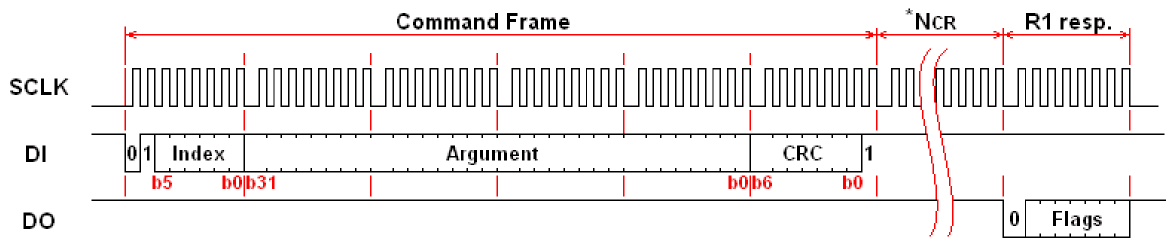

Figure 2.3: SD card SPI communication [12]

FatFs drivers

For the SD card implementation, since we did not have enough time to completely build the whole library, we decided to make use of the FatFs filesystem drivers written by Chan. When working on the provided library, we found out that the mmc.c file had some errors.

One of the errors present in the function send_cmd(), was that it did not send any CRC bytes for some of the commands. This resulted in the SD card sometimes not getting detected.

The library, specifically the functions disk_status() and disk_initialize(),involved some checks to determine whether the card is SDv2, SDv1 or an MMC. Since the SDv1 and MMC protocols are rarely used nowadays, we rewrote parts of the code, and optimized the said functions for SDv2.

Also, the ff.c file needs the function get_fattime(), which was written originally to take a fixed arbitrary constant time. Since the MSP430 has an inbuilt RTC, we decided to use it to our advantage, and changed the code so that get_fattime() was now a macro, which called RTC_read(), which is explained in the previous section.

We also decided to split the mmc.h file into two parts – mmc_driver.h and mmc.h. While mmc_driver.h part, present in mmc.c as defined by Chan, does not have any errors, and our versions did not involve many major optimizations, we rewrote the functions to improve readability and usability.

The block diagram in Figure 2.4 shows the different function calls and include files to explain the structure and abstraction levels in the FatFs library.

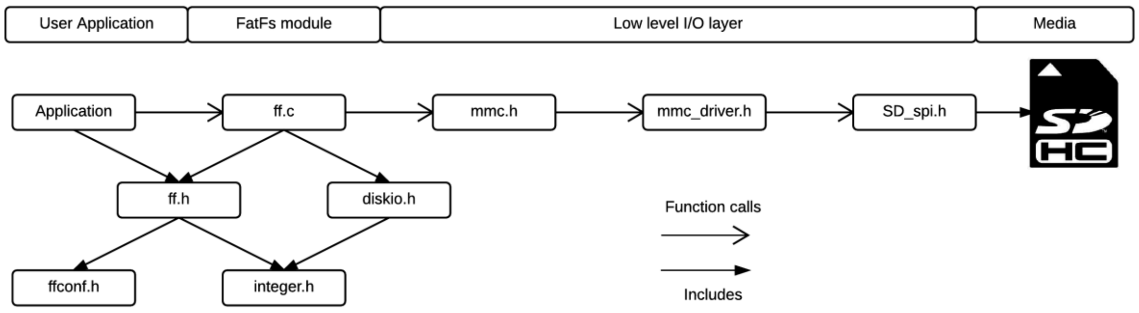

Figure 2.4: SD card library file structure

2.2.3 VS1063 Drivers

The VS1063 mp3 encoder and decoder is an integral part of our project. We found a sample code provided by VLSI Solutions [6, 8], when we tried to compile their code, we found out that it exceeded out code size by more than 1.2 KB. And hence, we decided to write our own drivers for interfacing the VS1063 IC.

We leveraged the register definitions for the VS1063 IC present in the VS1063_reg.h file, from the sample code from VLSI Solution, since redefining these registers would have been a time consuming mechanical task and would not have offered much learning.

The files developed for the VS1063 drivers are as given below:

Table 2.7: VS1063 firmware Parts

No.	Name of the file	Description
1	VS1063_spi.h, VS1063_spi.c	Contains the port initializations for all the pins used for the VS1063 interface and SPI drivers to send a receive data.
2	VS1063_reg.h	Contains the register definitions of the VS1063 internal registers.
3	VS1063_driver.h, VS1063_driver.c	Contains the functions needed to reset the VS1063, and also to write and read different registers, and to send data to the VS1063. It also has functions needed to write to or read from the VS1063 memory.
4	VS1063_handler.h, VS1063_handler.c	Contains the functions for playing and recording the audio file.

The VS1063 communicates with the microcontroller using the SPI protocol. The VS1063 documentation [4] divides the communication into two types: SCI (Serial Command Interface) and SDI (Serial Data Interface).

The SDI interface uses BSYNC/xDCS line as the signaling input to show that data is being sent over from the microcontroller, while the SCI interface uses xCS to signal that a command operation is occurring. These signals must stay asserted until the transmission is over.

During SCI, the first byte transmitted determines whether the operation is a read or a write. For a read the value 3 must be written, whereas for a write a 2 must be written. After this, the address of the register must be sent on which the operation is being performed.

Also, before transmitting any data whether in SCI or SDI, the microcontroller must make sure that the DREQ line is high.

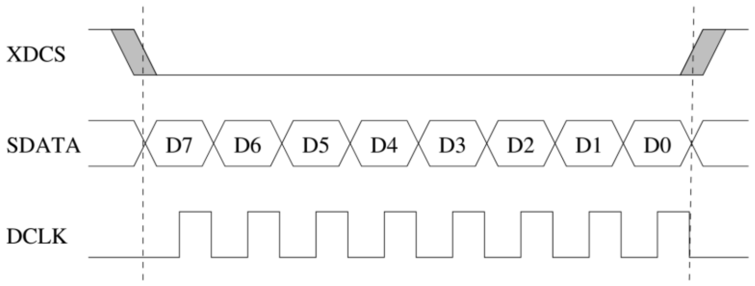

Figure 2.5: SDI data transfer [4]

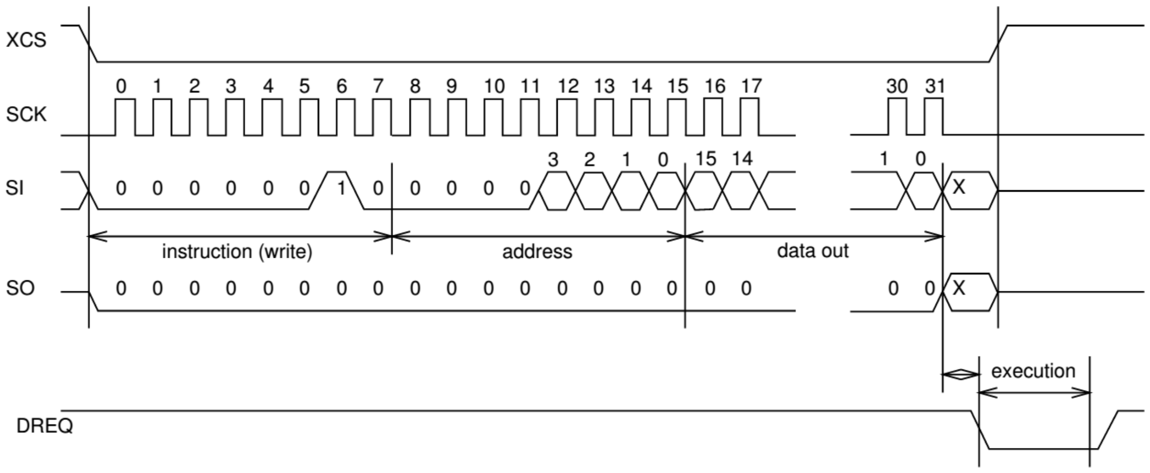

Figure 2.6: SCI data transfer [4]

At startup, the VS1063 needs to receive a hardware reset pulse. This is done by asserting the xRESET pin for minimum 2 clock cycles of the VS1063 main clock, which initially operates at 12.288 MHz. After the reset, the VS1063 asserts the xDREQ line.

Then, a software reset must be performed, which is done by setting the bit SM_RESET, inside the SCI_MODE register. After that, we write a value to a register inside the chip, and read the value again to verify that the proper communication takes place. Then, other registers such as the SCI_CLOCKF, which controls the clock multiplier, and SCI_VOL register, which controls the volume of the output, can be written to. We set the clock multiplier at 5x, and additional clock boost to 2x.

During playback, first, the SCI_DECODE_TIME register needs to be written to with the value 0. This register holds the amount of time for which the file has been playing. After this, the sound data can be sent to the chip, 32 bytes at a time. Also, after every 1024 bytes, something known as the endFillByte needs to be read from the register END_FILL_BYTE. This is done so that, in case the file ends abruptly, or, if it is stopped suddenly, the sound does not get distorted. So, when such a situation occurs, the endFillByte must be sent as data to the VS1063 32 times.

The volume of the playback can be changed anytime, by writing to the SCI_VOL register. Also, playback speed shifting can be performed by changing the value in the SPEED_SHIFTER register after setting the bit PLAY_MODE_SPEED_SHIFTER_ENABLE bit inside the playMode register. But this function was not implemented, since due to switch bounce, the speed change tended to stay between the two extremes of 1.68x and 0.68x of the playback speed.

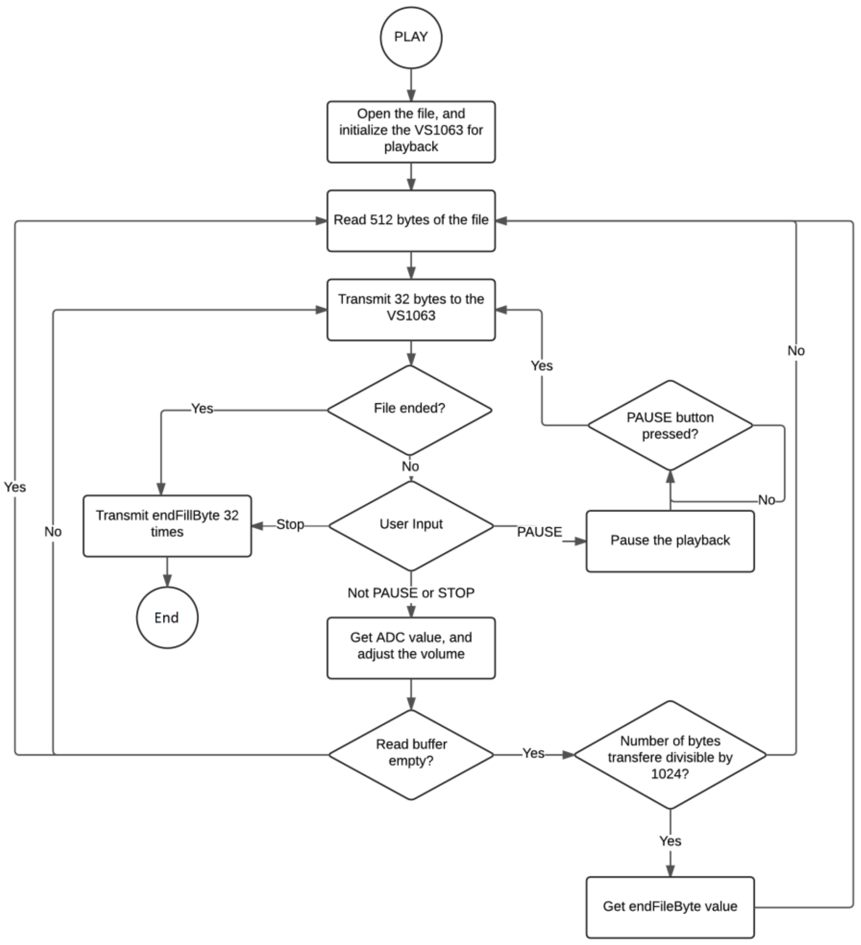
PLAY
Open the file, and initialize the VS1063 for playback
Read 512 bytes of the file
Transmit 32 bytes to the VS1063
File ended?
Yes
No
Transmit endFillByte 32 times
End
Stop
User Input
PAUSE
Pause the playback
PAUSE button pressed?
Yes
No
Not PAUSE or STOP
Get ADC value, and adjust the volume
Read buffer empty?
Yes
No
Number of bytes transfere divisible by 1024?
Yes
No
Get endFileByte value

Figure 2.7: Playback mode flowchart

To cancel the playback, we must set the SM_CANCEL bit in the SCI_MODE register. To pause playback, the PLAY_MODE_PAUSE_ENABLE bit in the playMode register must be set.

For recording, we first have to set up the sampling rate in the register SCI_AICTRL0, the gain in the register SCI_AICTRL1, the recording format and the recording mode (mono or stereo) in SCI_AICTRL3 and the bitrate in the register SCI_WRAMADDR. Then, to start recording, the SM_ENCODE bit in SCI_MODE register must be set, followed by writing the value 0x0050 in the SCI_AIADDR register. The data that is recorded can be found in the SCI_RECDATA register, which needs to be read using an SCI read, and written to the SD card.

To stop recording the SM_CANCEL bit in the SCI_MODE register has to be set. When the playback stops, in case the number of bytes written to the file are not even, then as in the playback mode, the endFillByte must be read from the equivalent register to maintain sound continuity.

The flow charts in figures 2.7 and 2.8 explain VS1063 playback and record operation.

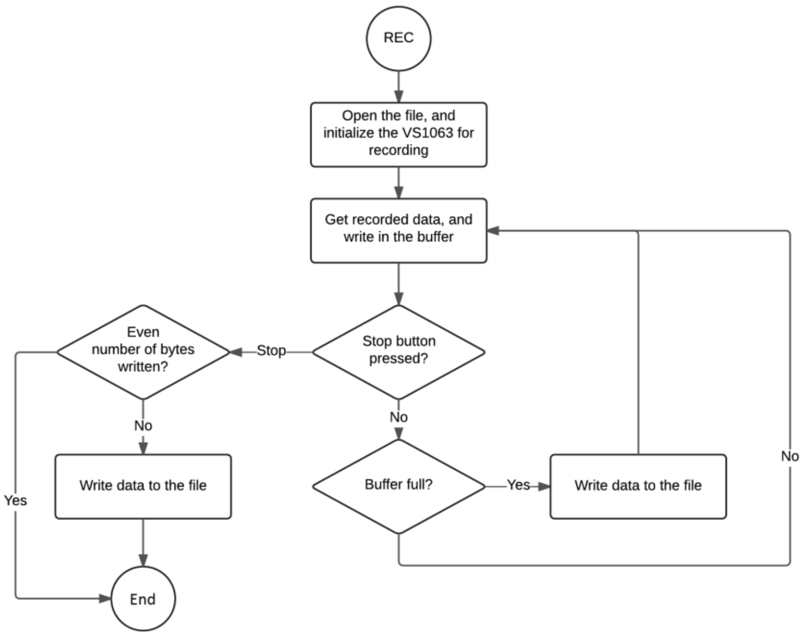
REC
Open the file, and initialize the VS1063 for recording
Get recorded data, and write in the buffer
Stop button pressed?
Stop
Even number of bytes written?
No
Yes
Write data to the file
Buffer full?
Yes
Write data to the file
No
End

Figure 2.8: Record mode flowchart

The SPI protocol used by VS1063 states that when the bus is idle, SCLK must be at a logic low, and data must be sent at the falling edge of the clock. When we first wrote the drivers for the VS1063, we wrote it such that the data would be transferred at the rising edge of the clock instead. This caused the song while being played to sound extremely distorted.

While programming initially, we were writing to one of the registers inside the VS1063, and reading the value back to verify that the SPI communication was taking place properly. While doing this, we did not see any difference in the written and read driver, which lead us to believe that the SPI bus is not at fault.

Then we tried to test the communication, by setting the SM_TESTS bit inside the SCI_MODE register, and then sending bytes representing a sine wave. This did not produce any distortion, which strengthened our belief that the SPI communication was working perfectly. Thus, we then moved to writing to different registers inside the VS1063, and tried to see whether some wrong value which was written by default by the chip at reset produced the distortion.

We finally realized our mistake when we compared the logic analyzer output with the waveforms in the datasheet, and upon changing the bit that changed the clock polarity, we could hear undistorted sound at the output.

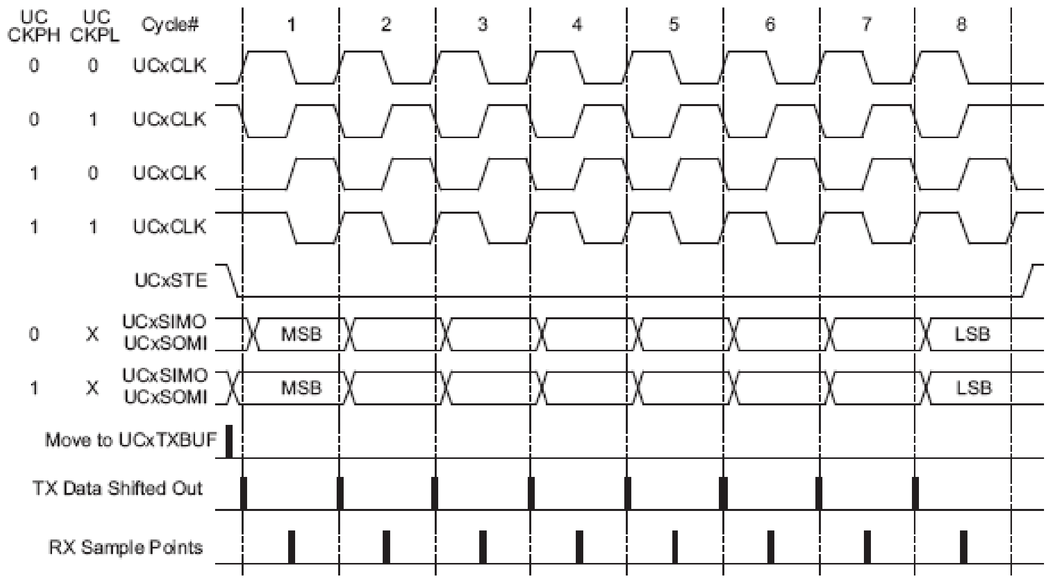

Figure 2.9: Different SPI modes [15]

Another problem faced, was because of the fact that we were attempting to record audio in the stereo format, but since we only had one microphone, the output produced was only audible on one side of the earphones, and not on the other. This problem was brought to our attention by the professor, when we showed him our project on the day prior to the final presentation. The solution was to record instead in mono, which produced the same output on both the sides.

2.2.4 LCD Drivers

Even though we had initially planned to use a graphics LCD with touch feature to display the GUI of our device, due to the problems faced with the VS1063 implementation, we had very little time left on our hands. Hence we decided to use the same LCD as the one used in ESD lab # 4. Much of the drivers written for the lab were reused for the project with some functions which were not required, such as the function to get the DDRAM dump, and the function to create a special character, were removed to save code space.

The only other changes that were required, were due to the fact that the MSP430 does not have any data and address bus. Thus, a memory mapped I/O was not possible, and the LCD had to be connected to port pins. Also, since the MSP430 works at 25 MHz, and takes only 1 clock cycle for execution of most instructions, as compared to 12 clock cycles on the 11.0592 MHz 8051, the macros used earlier for sending commands, writing and reading data from memory, and for busy flag polling had to be changed to functions, and delays had to be added to meet the timing constraints. The following code shows the difference between the two versions:

```
#define LCD_Check_BF() {while((*LCD_Read_BF_Addr) >= LCD_POS_BF);}  // macro to
check for the busy flag
```

The above code on the 8051 translates to the following code on the MS430:

```
// This function checks the busy flag, and only returns when its 0
// Parameters: - void
// Returns: - void
void LCD_Check_BF(void)
{
	set_DATA_BUS_input();				// set bus as an input
	while(1)
	{
		RW_high();					// make RW high
		RS_low();					// make RS low
		__delay_cycles(TAS_DELAY_CYCLES);
		EN_high();					// enable the LCD
		__delay_cycles(PWEH_DELAY_CYCLES);
		if(!read_BF())				// is the busy flag 0
		{
			EN_low();				// disable the LCD
			return;
		}
		EN_low();					// disable the LCD
		__delay_cycles(TCYC_DELAY_CYCLES);	// to satisfy tCYC
	}
}
```

2.3 Software Design

The following section describes the software flow of our program. The next section details our GUI design.

2.3.1 Software Flow

The flowcharts show the basic operation flow of our device. The flow is as follows:

1. Initialize all hardware.
2. Initialize SD card
3. If there is an error in the SD card initialization, wait for the user to press play, and once pressed, retry initialization.
4. Initialize VS1063, if there is an error, display error message and stop.
5. Prompt user to select between the recordings and music folders. If the user presses REC, start recording. (Step 9)
6. Open the selected folder and get the file names.
7. Get user input, if play, play the file, if up or down, scroll the display, or move the cursor. If the user presses stop, go back. If REC is pressed, start recording. (Step 9)
8. Wait for the file to end, or for the user to play stop.
9. If the user presses REC, start recording.
10. Wait for the user to press stop. Once stop is pressed, finish writing to the file.

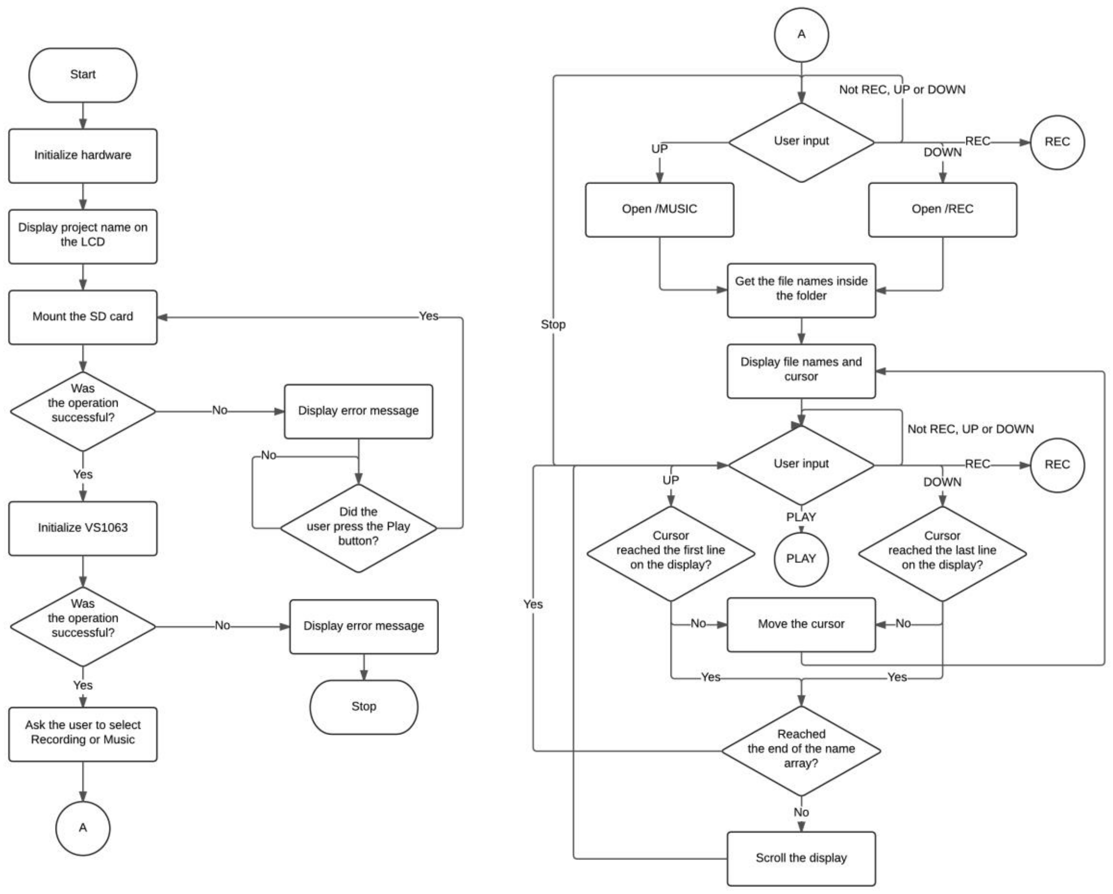
Start
Initialize hardware
Display project name on the LCD
Mount the SD card
Was the operation successful?
No
Display error message
Did the user press the Play button?
Yes
No
Initialize VS1063
Was the operation successful?
No
Display error message
Stop
Yes
Ask the user to select Recording or Music
A
A
Not REC, UP or DOWN
User input
UP
DOWN
REC
REC
Open /MUSIC
Open /REC
Get the file names inside the folder
Stop
Display file names and cursor
Not REC, UP or DOWN
User input
UP
DOWN
REC
REC
PLAY
PLAY
Cursor reached the first line on the display?
Cursor reached the last line on the display?
Move the cursor
No
No
Yes
Yes
Yes
Reached the end of the name array?
No
Scroll the display

Figure 2.10: Software flow chart

2.3.2 GUI Development

The development of the GUI on the LCD was an important part of the project. For this purpose, the files file_handler.h and file_handler.h were written. The file_handler header only has two functions which can be accessed globally:

Table 2.8: file_handler functions

Function	Description
file_play()	Function to list, select and play a file.
file_record()	Function to generate a file name, create a file with the generated name and record.

The FatFs library provides the function f_readdir() which can be used to read the names of the files present inside a directory. The problem is, this function iterates through the directory, and only gives one name at a time, according to the date when the file was last edited, until it reaches the end of the directory, when it returns a null pointer. Also, the function can only move in one direction. Thus, once a file name has been read, to read it again would require starting the whole process all-over again. This posed a problem for us, since we needed for the user to have the ability to move through the file list and select the desired file.

Hence, we decided to implement the function file_data(), which returns the names of all the files inside the directory in an array. The f_readdir() function then lists the file names, and then the user could select the desired file using the input buttons.

Another potential problem was the generation of file names for recording purposes. If a fixed file name was to be used, only one recording could be present in the SD card at a time, which was undesirable. While we considered the possibility of generating the file names dependent on the system date and time, this posed a couple of problems. One problem was, that the time was changed back to a particular time once the device was turned off, since no battery back-up was present. Other problem was due to the constraints in the FatFs library. The f_readdir() library function, could only read file names of length of 13 characters. This meant, that in case a file name was longer than that, the file could not be read on the device, and hence any recordings must be played using another device.

The solution we came up with, was to make use of the f_open() function with the file access specifier of FA_CREATE_NEW. This means, that if the provided file name already exists, the function will return an error. We used this to our advantage, and decided to give the recorded files a name of the form “RECx.MP3”, where x stands for the character generated dynamically to prevent clashes by us. We started with the character ‘0’ and moved up to ‘9’, after which the characters from ‘A’ to ‘Z’ were used to try and open the file. If the file threw an error, it meant that the file already existed, after which, the file was closed, and a new file name was tried, until a file name was found which generated the error FR_EXIST. After this, the said file name was used to create a file and record.

2.4 Testing Process

The project development was done using an incremental approach. After completion of the project, the whole system was tested a number of times for any erroneous operation. We also tried to play different music file formats on the device. We tried recording different sounds in a variety of different settings. We changed the bit rates, sampling rate, and also tried keeping the speakers producing the output at various distances from the microphone, to test whether any noise is induced due to the distance between the microphone and the sound source.

3 RESULTS AND ERROR ANALYSIS

The MP3 recording and playback was implemented successfully. We were able to achieve most of the features as planned.

A good sound quality was achieved while playback. Also, several basic features such as pause, play, and Stop, along with volume control were successfully implemented. For MP3 audio recording, mono audio format was used, and files were successfully recorded at a variety of bit rates and sampling rates in the both the constant and variable bit rate format. The recorded audio files were successfully stored and played from the SD card.

We are able to successfully access all the stored music files and recorded files using the GUI displayed on the LCD. The user was provided with an option to select and play any random file from the SD card. An RTC clock was implemented so that current date and time could be associated with the recorded files stored on SD card.

We encountered few errors during the project development, some of which have been listed in the firmware development section of the report. Some other errors, which we were unable to completely resolve, along with the probable causes are listed below.

3.1 Inability to play FLAC audio files

During testing process we tried to test the MP3 player to play different file formats. During these tests we realized that the IC was not able to decode FLAC file formats even when the datasheet shows that this file format is supported by the IC. We later figured out that this was probably due to the fact that the FLAC file format does not implement compression and thus has a high bit rate. This resulted in insufficient bandwidth during data transfer from the microcontroller to the VS1063 IC.

3.2 Volume control using Analog Input

We encountered some issues in controlling the volume using the potentiometer. After a certain point, a slight change in the potentiometer value drastically changed the volume level. On referring to the VS1063 datasheet we noticed, that changing the volume control register value by 1 actually reduced the volume output by 0.5 dB.

When we attempted to convert the potentiometer reading to produce a linear volume change by calculating an antilog of the ADC reading, the calculation took too much time and produced an output with glitches due to the reduction in transfer rate. We partially solved this problem by dividing the ADC output reading with different values in different ranges, and by trial and error found values that produced a relatively better output.

4 CONCLUSION

The project involved using the MSP430 microcontroller, which was a new architecture for us. We also had the opportunity to work with different hardware interfaces as a part of this project.

This project helped us explore different areas of Embedded Systems. We understood a great deal about SD card working and FAT file systems, as our project was based on an SD card interface. We also developed an understanding of different audio format and their decoding an encoding.

Apart from all this, to debug our various mistakes, we gained some invaluable experience in debugging.

5 FUTURE DEVELOPMENT IDEAS

One of the improvements that we could do in our projects, would be to develop our own PCB. This would make the device very small in size, and also enable us to remove some of the hardware elements present on the MSP430 development board which were not used.

Another possible improvement could be to make use of a touchscreen graphic LCD, instead of a 16x4 LCD. This would remove the need of keeping any buttons, and thus solve some of the switch bounce issues, and also improve the user interface. USB was another feature that we had planned on implementing with our project. But due to the lack of time, we were unable to implement the said feature.

Some other features such as stereo recording, line input, audio equalizers, and speed shifting could also be the possible improvements with the project. As the current system does not have any battery backup, we could have added a small battery to the system, and made the MSP430 go into an extremely low power mode, so as to keep the RTC running.

6 ACKNOWLEDGEMENTS

We would like to thank the course instructor, Prof. Linden McClure, for conducting a great course, which we thoroughly enjoyed studying. We are thankful to him and the teaching assistants, for solving our doubts, and guiding us in the process of component selection.

We would like to express our gratitude to Chan, whose extensive FatFs documentation online helped us achieve our goal.

We would also like to thank the authors of all the references cited below.

7 REFERENCES

[1] MSP430x5xx and MSP430x6xx Family User guide
http://www.ti.com/lit/ug/slau208o/slau208o.pdf

[2] MSP430F552x, MSP430F551x Mixed-Signal Microcontrollers Datasheet
http://www.ti.com/lit/ds/symlink/msp430f5529.pdf

[3] MSP430F5529 Launchpad development kit
http://www.ti.com/lit/ug/slau533c/slau533c.pdf

[4] VS1063a Datasheet
http://www.vlsi.fi/fileadmin/datasheets/vs1063ds.pdf

[5] VS1063a Hardware Guide
http://www.vlsi.fi/fileadmin/manuals_guides/vs1063hg.pdf

[6] VS1063a Programmer's Guide
http://www.vlsi.fi/fileadmin/manuals_guides/vs1063pg.pdf

[7] VS1063a Patches
http://www.vlsi.fi/fileadmin/software/VS10XX/vs1063a-patches.pdf

[8] VS1063 Application Notes
http://www.vlsi.fi/en/support/applicationnotes.html

[9] SD Memory Specifications
https://www.sdcard.org/downloads/pls/simplified_specs/archive/part1_101.pdf

[10] FatFs Generic FAT File System Module by Chan
http://elm-chan.org/fsw/ff/00index_e.html

[11] FatFs Application Note by Chan
http://elm-chan.org/fsw/ff/en/appnote.html

[12] How to Use MMC/SDC by Chan
http://elm-chan.org/docs/mmc/mmc_e.html

[13] LCD module specifications
http://ecee.colorado.edu/~mcclurel/20434aje.pdf

[14] SparkFun Electret Microphone Datasheet
http://cdn.sparkfun.com/datasheets/Sensors/Sound/CEM-C9745JAD462P2.54R.pdf

[15] Texas Instruments Forum TI E2E Community
http://e2e.ti.com/support/applications/high_reliability/f/30/p/364803/1369391

8 APPENDICES

Several appendices have been attached to this report in the order shown below.

8.1 Appendix - Bill of Materials

Part Description	Source		Cost
MSP430F5529 Launchpad	TI	www.ti.com	$12.99
VS1063 Breakout Board	Proto Central	www.protocentral.com	$22.89
8 GB SD Card with Adapter	Amazon	www.amazon.com	$3.92
PCB	CU Electronics Store		$6.92
Potentiometer	CU Electronics Store		$3.27
LCD	Embedded Systems parts kit		$0.00
Trim Potentiometer	Embedded Systems Lab		$1.00
Push Buttons(5)	Embedded Systems Lab		$1.25
1uF ceramic capacitors(2)	Embedded Systems Lab		$0.20
47uF Electrolytic capacitors(5)	Embedded Systems parts kit		$0.00
10uF Electrolytic capacitors(2)	Embedded Systems Lab		$0.20
10uF ceramic capacitors(2)	Embedded Systems Lab		$0.20
47uF ceramic capacitors(2)	Embedded Systems Lab		$0.20
Electret Microphone	Sparkfun	www.sparkfun.com	$0.95
TRRS 3.5mmAudio Jack Breakout	Sparkfun	www.sparkfun.com	$3.95
Resistors 1k ohms(7)	Embedded Systems Lab		$0.00
TOTAL			$57.94

Note: If we were doing the project again, we would use a graphic LCD with touch, instead of using the 16x4 LCD and buttons.

8.2 Appendix - Schematics

8.3 Appendix - Firmware Source Code

8.4 Appendix - Software Source Code

8.5 Appendix - Data Sheets and Application Notes