Final Project Report

Embedded Garageband

By:
Maulik Kapuria
Niket Shah
Vishal Verma

Embedded Systems Design
ECEN-5613
Spring 2011

30th April, 2011
University of Colorado, Boulder

Index

1. Introduction

The project is aimed at developing an embedded Garage band that uses an 8051 microcontroller. The user can access the instruments via the touchscreen+graphics LCD or use the instruments that we have constructed. In this document we will cover the complete detail of the project including its construction, hardware design, software design and debugging.

1.1 Project Overview

The Embedded Garageband was aimed to play music notes on a speaker without using any Actual instruments. This project was intended to use touchscreen to take user input and use graphics LCD to display various "Virtual instruments". This opens a tremendous opportunity to re-create wide variety of instruments without any construction efforts. We used the graphics LCD for displaying instruments like Guitar, Drums and Piano. After we take the user interface, we play corresponding notes using MIDI commands. The music is played on either the PC or on the speaker connected to a MIDI decoder. In addition to this we also had a Keyboard from which we can simulate as piano and we built a piezo element based discs that we can use as drumsets. Thus we have two types of instruments

- "Virtual" i.e the instruments created using the touchscreen and the LCD (eg: guitar, piano, drums)
- "Real" i.e. All other instruments that are not virtual. (eg. Keyboard as Piano, Piezo discs as Drums)

1.2 Project Objective

The objective of the project was to build a system capable of:

i. Taking input from user using Touchscreen.
ii. Display Instruments on the graphics LCD.
iii. Play music notes using MIDI commands.
iv. Give option to user to play the notes either on the connected computer via a serial port or on the speakers connected to the onboard MIDI decoder.
v. Give the ability to store the MIDI notes on the SDcard and give the ability to reproduce the same MIDI tune again (also have to store the timing delays between notes for reproducibility)
vi. Give the user an option to select between Real instruments and Virtual Instruments.

1.3 Selecting the System configuration – Tradeoff's

In any system design tradeoffs have to be made between quantity A and quantity B. It is crucial that when starting a new system design that we weigh these components and understand the tradeoffs. While selecting our system we had to

think of the tradeoff between various factors that influenced how our system configuration looked.

When we chalked out the block diagram of the system we had a lot of interfaces and peripherals. So we required a processor that had a lot of interfaces and peripherals. We also required large memory space in the CPU to support the huge code and drivers modules. At the same time the system should be robust & cost effective. With all the requirements it was tough to get a cheaper system working in the given time frame.

We had a few 8051 based wireless SOC modules lying around. They were powerful, had large internal memory, consumed low power, had reconfigurable pins, and had many peripherals but they only had 14 CPU pins that we can use. Those numbers of pins were definitely not enough for the amount of interface we had. Hence we decided to separate the system in different modules. Now we had two such modules, each accomplishing a specific function. They communicated to each other using a wireless protocol that we defined (and will be discussed later). The main question was how to decide what external hardware interface goes on which module. We had to make sure that we do not overload the peripheral on one module and also that we distribute the computational requirement between both the modules.

We created two basic lists on paper:

[i] LIST 1: It contained a description of all the external hardware interfaces we were required to do and the computational requirement of each.

[ii] LIST2: It contained the list of peripherals available on each module.

We mapped the interfaces from LIST1 to the available pins and peripherals on the LIST2. This we way utilized most of the on chip peripherals without overloading them. For example, we required six timers for our piezo drums. Each module only had three timers to use. Hence we split the drums between the modules that solved the problem of resource scarcity & overloading.

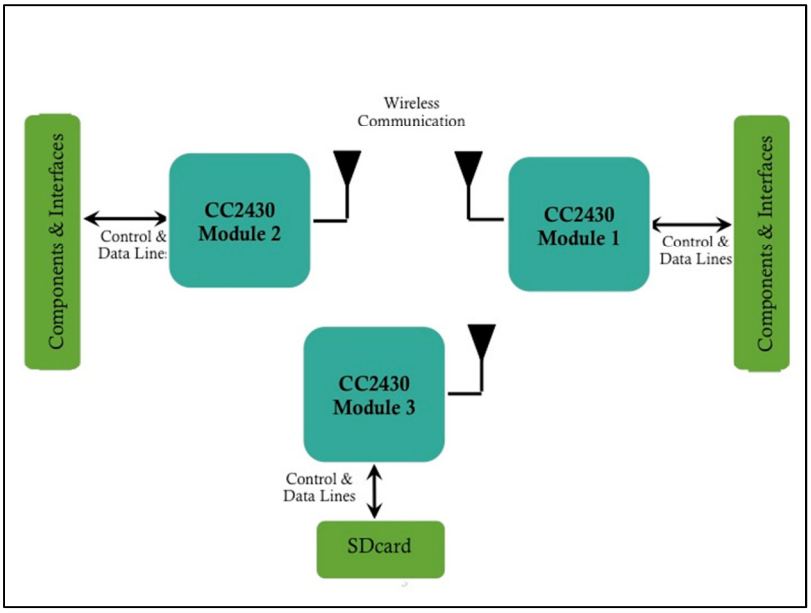


Figure1-1 A Very High Level Diagram of the system

1.4 High Level System Diagram

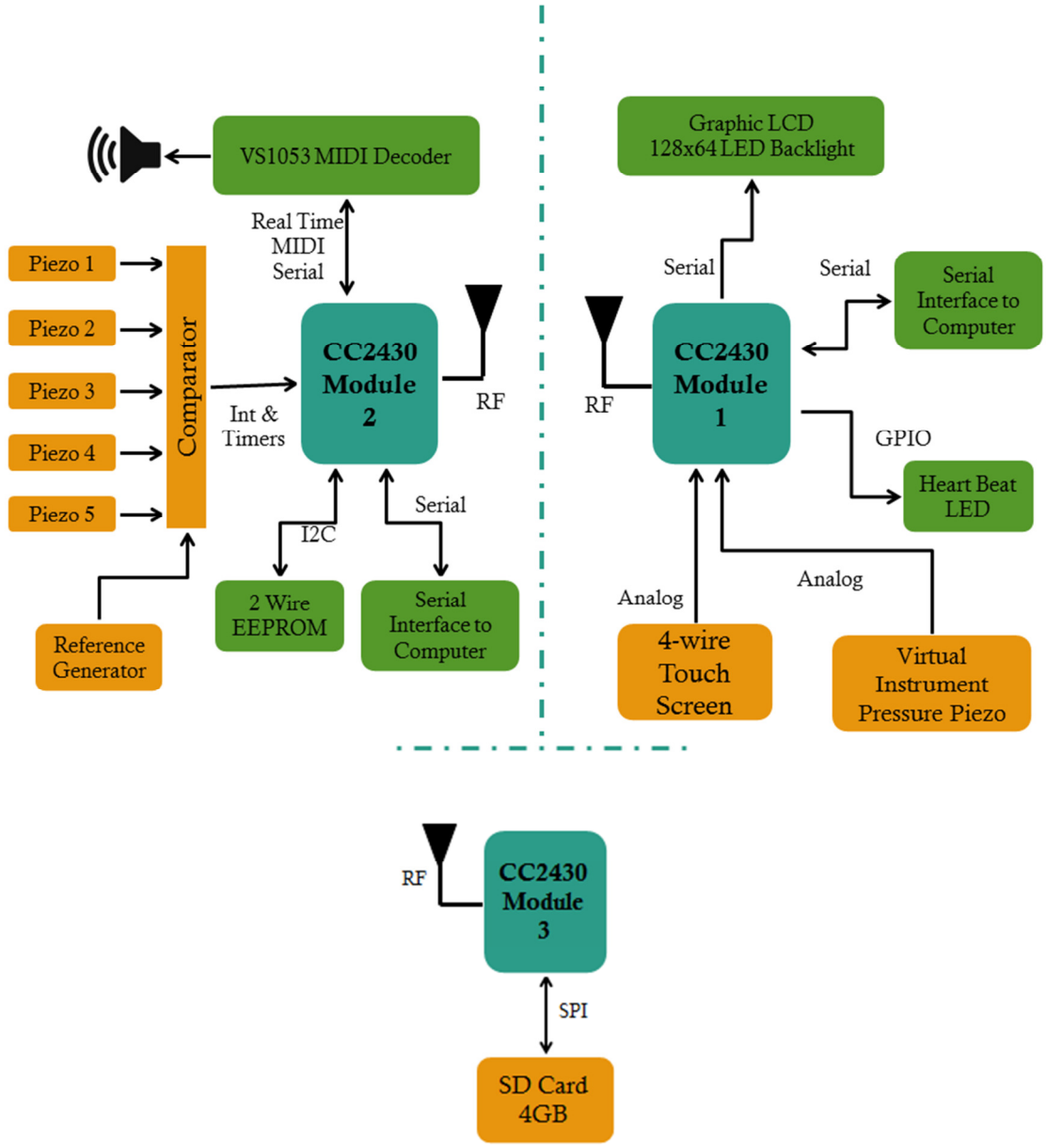
As we have already discussed in the previous section that we had to make a lot of trade offs while we selected the system configuration. A very high-level system diagram looks like shown in figure 1-1.

As we can see that in the figure we have shown different modules, but instead of the two discussed before we now have three modules. The third 8051 SOC module was introduced a few days before the final demo since we got the SD card working using SPI, and the SPI bus was already occupied in the reconfigurable modules. And moreover we were also out of pins on both the modules. So we thought of having a third module that will have SD card interface to it, and will store and playback using wireless stream.

All the other hardware interfaces are connected to Module1 and Module2. We have designed our own wireless initialization, a wireless packet format and a handshake protocol to communicate between the modules. We will discuss the system configuration and the wireless protocol as we dive deep into the project details.

2. Hardware Description

2.1 Hardware Block Diagram



Description:

- Module 2 is the master module which gets inputs from the user and sends commands to Modules 1 and 3.
- It interfaces the MIDI controller which is an integral part of the project. This is a serial, 31250 baud interface.
- It also has the UART to connect to the computer – for user interface, debugging as well as playing MIDI notes on the computer.
- Module-2 also uses five digital inputs configured as interrupts to interface the 'real' drums.
- Module-1 interfaces the graphic LCD and the 4-wire touchscreen.
- It is used only for virtual instruments.
- Module-3 interfaces the SD card, and is powered by a separate source in the form of a battery. Module-3 is an off-board module.

2.2 CC2430 and Zigbee Modules used in Project

2.2.a Introduction to TI's CC2430 SoC – 8051 core with 2.4 Ghz RF Transciever

The CC2430 combines the excellent performance of the leading CC2420 RF transceiver with an industry-standard enhanced 8051 MCU, 128 KB flash memory, 8 KB RAM and many other powerful features.

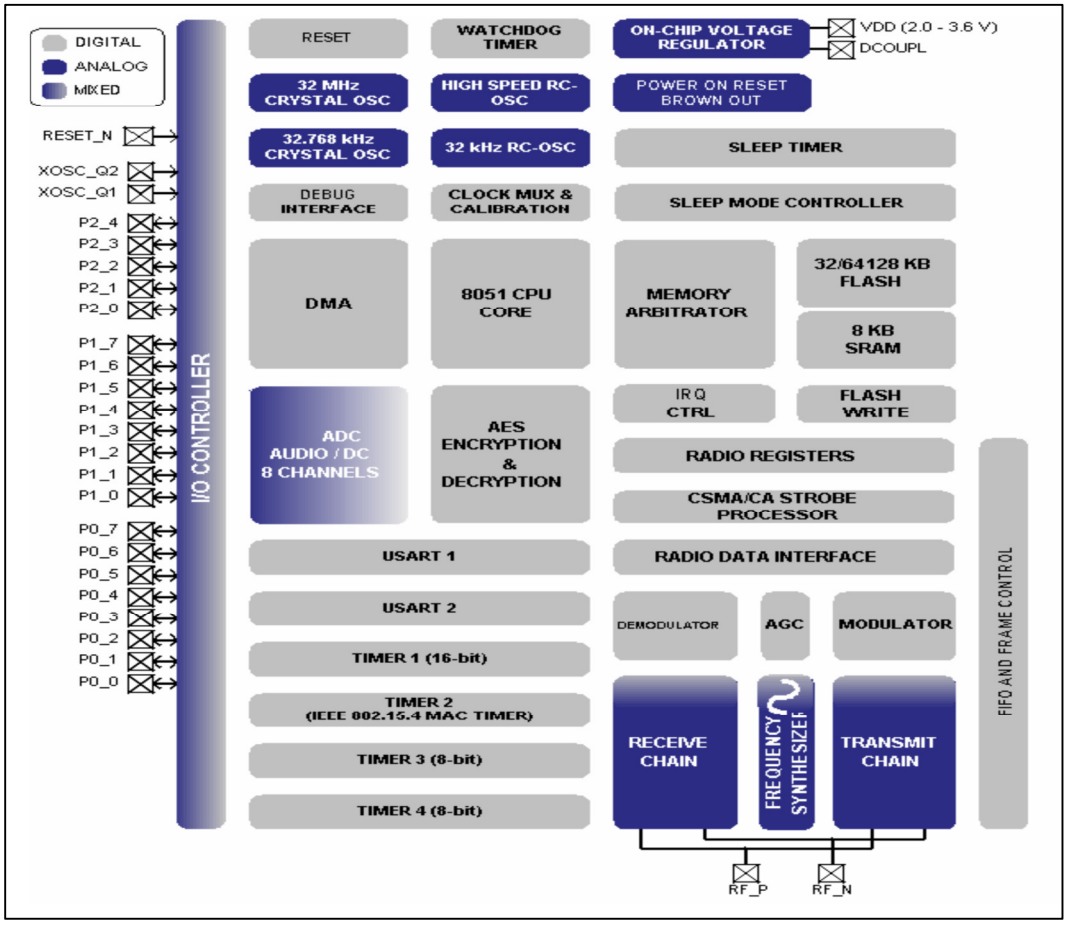


Figure2-1 CC2430 Architecture (http://focus.ti.com/lit/ds/symlink/cc2430.pdf)

The CC2430 includes an 8-bit CPU core, which is an enhanced version of the industry standard 8051 core. The enhanced 8051 core uses the standard 8051 instruction set. Instructions execute faster than the standard 8051 due to the following:

- One clock per instruction cycle is used as opposed to 12 clocks per instruction cycle in the standard 8051.
- Wasted bus states are eliminated.

Since an instruction cycle is aligned with memory fetch when possible, most of the single byte instructions are performed in a single clock cycle. In addition to the speed improvement, the enhanced 8051 core also includes architectural enhancements.

2.2.b Features

RF/Layout

- 2.4 GHz IEEE 802.15.4 compliant RF transceiver (industry leading CC2420 radio core)
- Excellent receiver sensitivity and robustness to interferers
- Very few external components
- Only a single crystal needed for mesh network systems

Low Power

- Current Consumption: (RX/ TX: 27 mA, microcontroller running at 32 MHz)
- Only 0.5 µA current consumption in powerdown mode, where external interrupts or the RTC can wake up the system
- Very fast transition times from low-power modes to active mode enables ultra low average power consumption in low dutycycle systems
- Wide supply voltage range (2.0V - 3.6V)

Microcontroller

- High performance and low power 8051 microcontroller core
- 128 KB in-system programmable flash
- 8 KB RAM, 4 KB with data retention in all power modes
- Powerful DMA functionality
- Watchdog timer
- One IEEE 802.15.4 MAC timer, one general 16-bit timer and two 8-bit timers
- Hardware debug support
- Power On Reset/Brown-Out Detection
- Eight channel ADC with configurable resolution
- Four timers: one general 16-bit timer, two general 8-bit timers.
- Two programmable USARTs for master/slave SPI or UART operation
- 21 configurable general-purpose digital I/O-pins

2.2.c Memory Map

The 8051 CPU architecture has four different memory spaces. The 8051 has separate memory spaces for program memory and data memory. The 8051 memory spaces are the following:

CODE: A read-only memory space for program memory. This memory space addresses 64 KB.

DATA: A read/write data memory space, which can be directly or indirectly, accessed by a single cycle CPU instruction, thus allowing fast access. This memory space addresses 256 bytes. The lower 128 bytes of the DATA memory space can be addressed either directly or indirectly, the upper 128 bytes only indirectly.

XDATA: A read/write data memory space access to which usually requires 4-5 CPU instruction cycles, thus giving slow access. This memory space addresses 64 KB. Access to XDATA memory is also slower in hardware than DATA access as the CODE and XDATA memory spaces share a common bus on the CPU core and instruction pre-fetch from CODE can thus not be performed in parallel with XDATA accesses.

SFR: A read/write register memory space, which can be directly accessed by a single CPU instruction. This memory space consists of 128 bytes. For SFR registers whose address is divisible by eight, each bit is also individually addressable.

2.2.d CC2430 Zigbee Modules

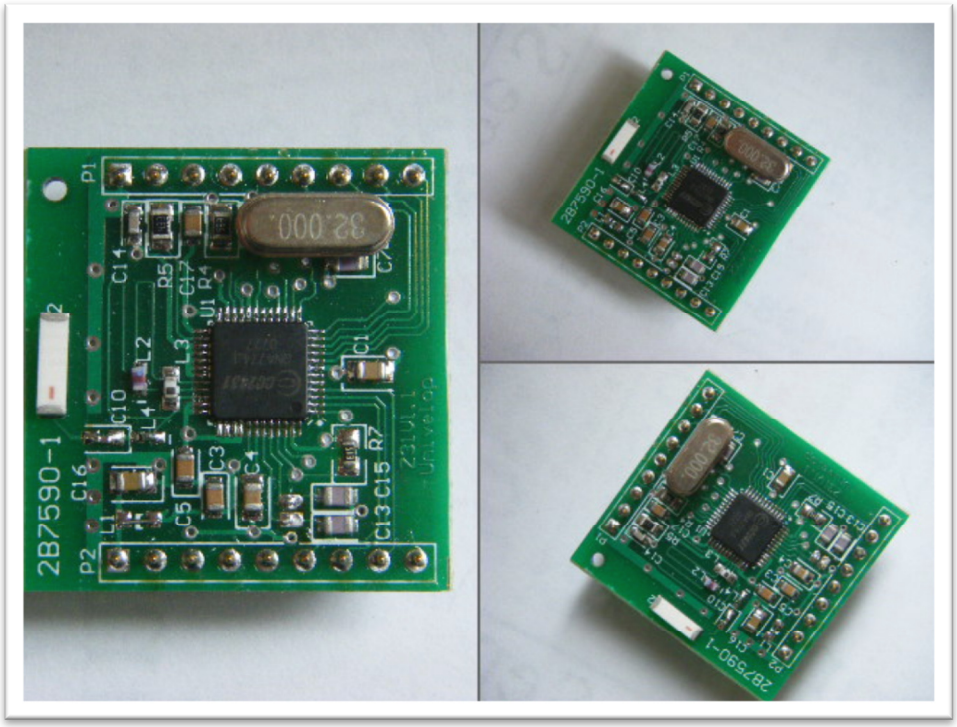


Figure2-2: CC2430 based Zigbee Modules used for the project

2.2.e Reasons for selection of CC2430 based Zigbee Modules

- **Powerful 8051 core CPU, runs at 32 MHz**: Sufficient to use several resources (Graphics LCD, 2/3 UARTs, SPI, ADC etc.) simultaneously.

- **2 dedicated USART modules per chip**: Our application requires more than 3 dedicated UART intensive peripherals like GLCD, MIDI Decoder, SD card, User Interface etc.
- **In-built 12-bit ADC pins reconfigurable to digital GPIOs**: Required for Touchscreen, External ADC chip like (ADC0808N) cannot be used to sense Touch Data.
- **In-built RF Modem for Wireless Communication**: Enables Wireless Packet Communication between two or more CC2430 modules. Proved to be extremely helpful for distribution of computation load, processing power, sensor interface and eliminates requirement of single Microcontroller with exceptional processing power, more GPIOs/Peripherals)
- **Reconfigurable Peripheral I/O Ports**: The CC2430 has 21 digital input/output pins that can be configured as general purpose digital I/O or as peripheral I/O signals connected to the ADC, Timers or USART peripherals. The usage of the I/O ports is fully configurable from user software through a set of configuration registers. Peripheral units have two alternative locations for their I/O pins, as shown in following Table.

Periphery / Function	P0								P1								P2				
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	4	3	2	1	0
ADC	A7	A6	A5	A4	A3	A2	A1	A0													
USART0 SPI			C	SS	M0	MI															
Alt. 2											MO	MI	C	SS							
USART0 UART			RT	CT	TX	RX															
Alt. 2											TX	RX	RT	CT							
USART1 SPI			MI	M0	C	SS															
Alt. 2									MI	M0	C	SS									
USART1 UART			RX	TX	RT	CT															
Alt. 2									RX	TX	RT	CT									
TIMER1				2	1	0															
Alt. 2														0	1	2					
TIMER3												1	0								
Alt. 2									1	0											
TIMER4															1	0					
Alt. 2																		1			0
32.768 kHz XOSC																	Q2	Q1			
DEBUG																			DC	DD	

Figure2-3: IO Pin Re-configurability (http://focus.ti.com/lit/ds/symlink/cc2430.pdf)

- **In-System Debug Interface**: The CC2430 includes a debug interface that provides a two-wire interface to an on-chip debug module. The debug interface allows programming of the on-chip flash and it provides access to memory and register contents and debug features such as breakpoints, single-stepping and register modification. This feature really helped to debug the code on run time using breakpoints on IDE.

2.3 Graphics LCD

We used the 128x64 LED backlit LCD display from Sparkfun. The 128x64 LCD is available in two different interfaces i.e Parallel and Serial. In the parallel interface the amount of complexity involved in displaying complicated shapes was high. Also it required many IO pins and critical timing, due to which we went for the Serial Interface Graphic LCD module. The serial Interface Graphics LCD has a Serial Backpack connected to the back end of the Parallel Interface LCD. It has an ATMEGA168 on the backpack as shown in figure 2-4.

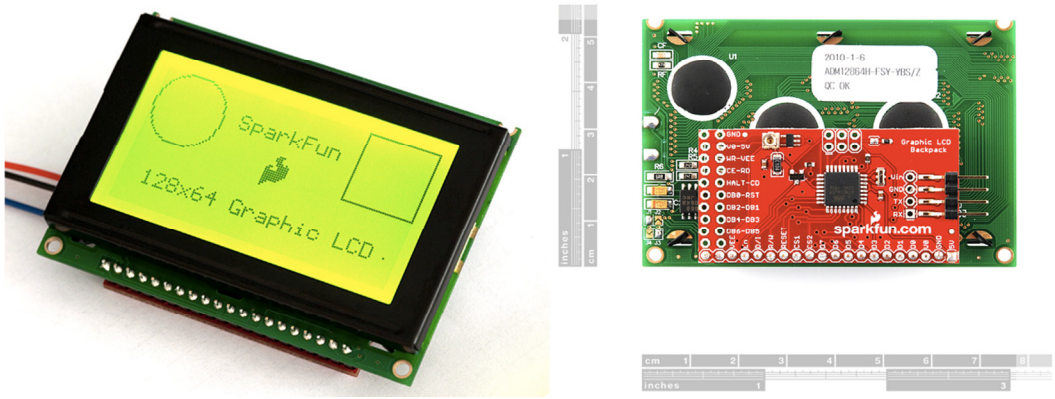
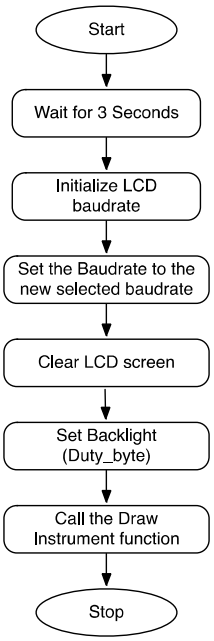


Figure2-4: 128x64 Graphics LCD (Image Courtsey: www.sparkfun.com)

Besides writing text, this serial graphic LCD allows the user to draw lines, circles and boxes, set or reset individual pixels, erase specific blocks of the display, control the backlight and adjust the baud rate. Additionally, all source code for the ATMega168 processor is compiled using the free WinAVR compiler and the firmware is open source. We used the default firmware preinstalled in the LCD.



The LCD was powered at 5V which consumed appx 250mA of current. We had the display to the brightest. This LCD was interfaced via a Serial port (at 3.3V level) to Module 1. The LCD only requires data coming on the RX pin and hence the TX pin was left not connected. The serial port was accepting commands at 115200bps (8N1).

Figure2-5:LCD Co-ordinates

Operation:

ASCII characters are printed to the screen with respect to two user-changeable settings, x_offset and y_offset. These two settings define the top-left corner bit of a character space, which is a 6x8 bit. By changing x_offset and y_offset the user can place test anywhere on the screen. Printing characters to the screen happens left to right, top to bottom, without adjusting x_offset and y_offset.

The serial LCD has an input buffer of 416 bytes, the number of characters that fill the screen. Streaming more than 416 characters at a time may cause a buffer overrun, and the time it takes for the firmware to process the entire buffer is dependent on what combination of characters and special commands that the user sends. Hence we injected appropriate amount of delay between commands in order to ensure that we do not over fill the buffer.

All the commands sent to the LCD are serial and have a specific format. They all start with a start byte i.e. 0x7C followed by a command identifier.

Depending on the command identifier there can be zero or more bytes. The table below shows some important commands.

Start Byte	Identifier	Data	Purpose
0x7C	0x00	NO data	Clear Screen
0x7C	0x04	NO data	Runs Demo mode
0x7C	0x12	NO data	Toggles between Black background and white background display
0x7C	0x02	<duty_cycle>	Duty cycle from (0 – 100) changes the backlit
0x7C	0x07	<baud_byte>	Changes baudrate at which the LCD accepts data
0x7C	0x18	<x_cord> <y_cord> <Set_reset>	Sets x and y co-ordinates
0x7C	0x0C	<x1_cord> <y1_cord> <x2_cord> <y2_cord> <set_reset>	Sets or resets line from (x1,y1) to (x2,y2)
0x7C	0x03	<x_cord> <y_cord> <radius> <Set_reset>	Draws circle of radius centered at (x,y)
0x7C	0x0F	<x1_cord> <y1_cord> <x2_cord> <y2_cord> <set_reset>	Draws box with opposite vertices at (x1,y1) (x2,y2)
0x7C	0x05	<x1_cord> <y1_cord> <x2_cord> <y2_cord>	Erases Block from (x1,y1) (x2,y2)

Table 1 – LCD Command Table

2.4 Drumset

One of the Real Instruments are the drums set. In order to make a drum set we had to come up with a circuit that demonstrated change in output when the pressure on it changes. We came up with two solutions, Force sensitive resistor and a piezo element buzzer. The force sensitive resistors are expensive and are not circular (which is essential property for our drum construction). Also using force sensitive resistor means that we have to provide power to get some

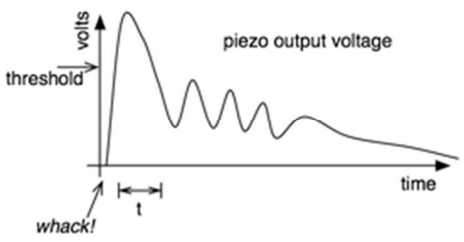


Figure 2-6: Output of piezo element when it is struck

valuable output to the microcontroller (since Force sensitive resistors like all resistors are passive components). Hence we dropped the idea of using a Force sensitive resistor and went for a piezo element. When piezo elements are struck, their output voltage rings, sort of like a bellas shown in fig 2-6

One of the ways to use this output is to use an A/D converter to read the values continuously, but we end up with large amount of processing overhead, and hence we decided to convert this to digital form that microcontroller can easily process. Moreover we wanted the signal to be such that it would interrupt the microcontroller when the pulse crossed the threshold. We used the circuit shown below to do that.

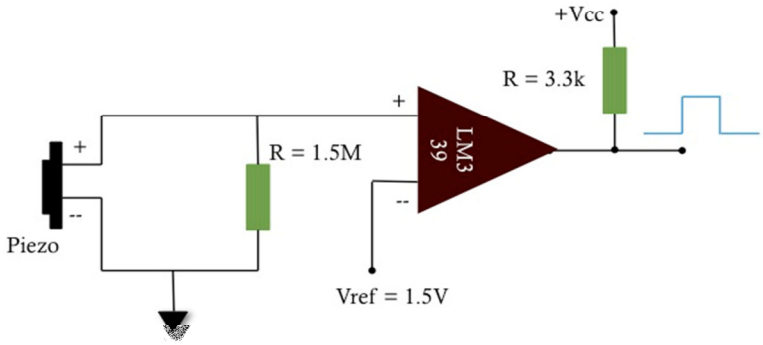


Figure 2-7: Piezo Interface circuit

The circuit as shown is the interface circuit that provides the necessary pulse (pulses) to the microcontroller in case the piezo is struck. The 1Mega resistor is used to protect the LM339 from overvoltage in case the piezo generates very high voltage. The reference voltage is set to 1.5V(threshold) thus whenever the voltage generated form piezo crosses 1.5 volts the output of the comparator (LM339 is a quad comparator) will go from Low to High and the microcontroller will be interrupted.

Then when the output of piezo falls below 1.5V the output again goes from High to Low. The piezo output oscillates for some times, in which it crosses the threshold of 1.5V few times, and the output of the comparator produces some edges that can be used with timer to measure the equivalent force.

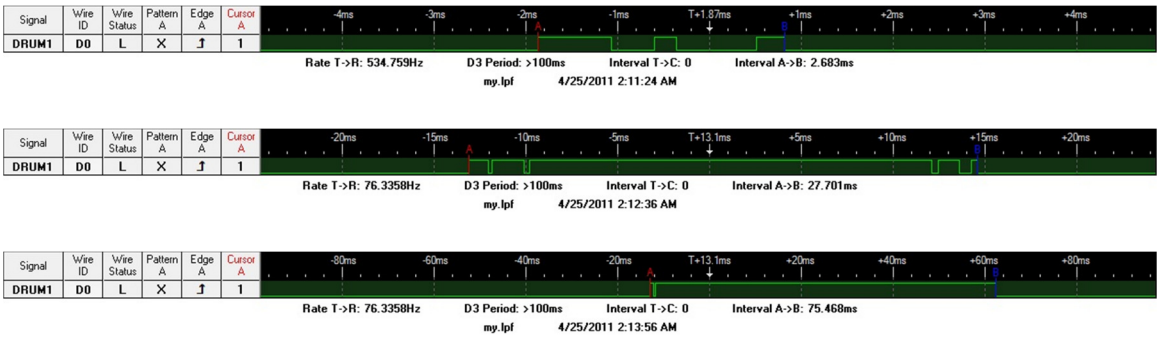


Figure 2-8: The topmost figure is when a light tap is experienced on the piezo. The Time interval between extreme edges is 2.683ms. For a medium amount of force, the time interval between extreme edges is 27.701ms and for a very hard tap, the interval is the maximum i.e. 75.486 ms

Thus we can see that the amount of force on the piezo is proportional to the difference between the extreme edges of the comparator output.

The flow chart of the software use on one of the six piezo pins is shown as in figure 2-9 . It can be seen that we require a very unique property here, that the pins must be initially be set as a positive edge triggered interrupt so see when the even starts (the first time when the piezo output crosses the threshold) this way we save the unnecessary computation wasted in polling the pin.

After this the pin must have the ability to be configured as a negative edge triggered interrupt until the timer overflows. On each negative edge we capture the current value of the low resolution timer in a capture variable and overwrite it on every negative edge. When the timer overflows (after 150ms), the value in the capture variable (16bit) corresponds the amount of force. This can be used to decide the MIDI note velocity.

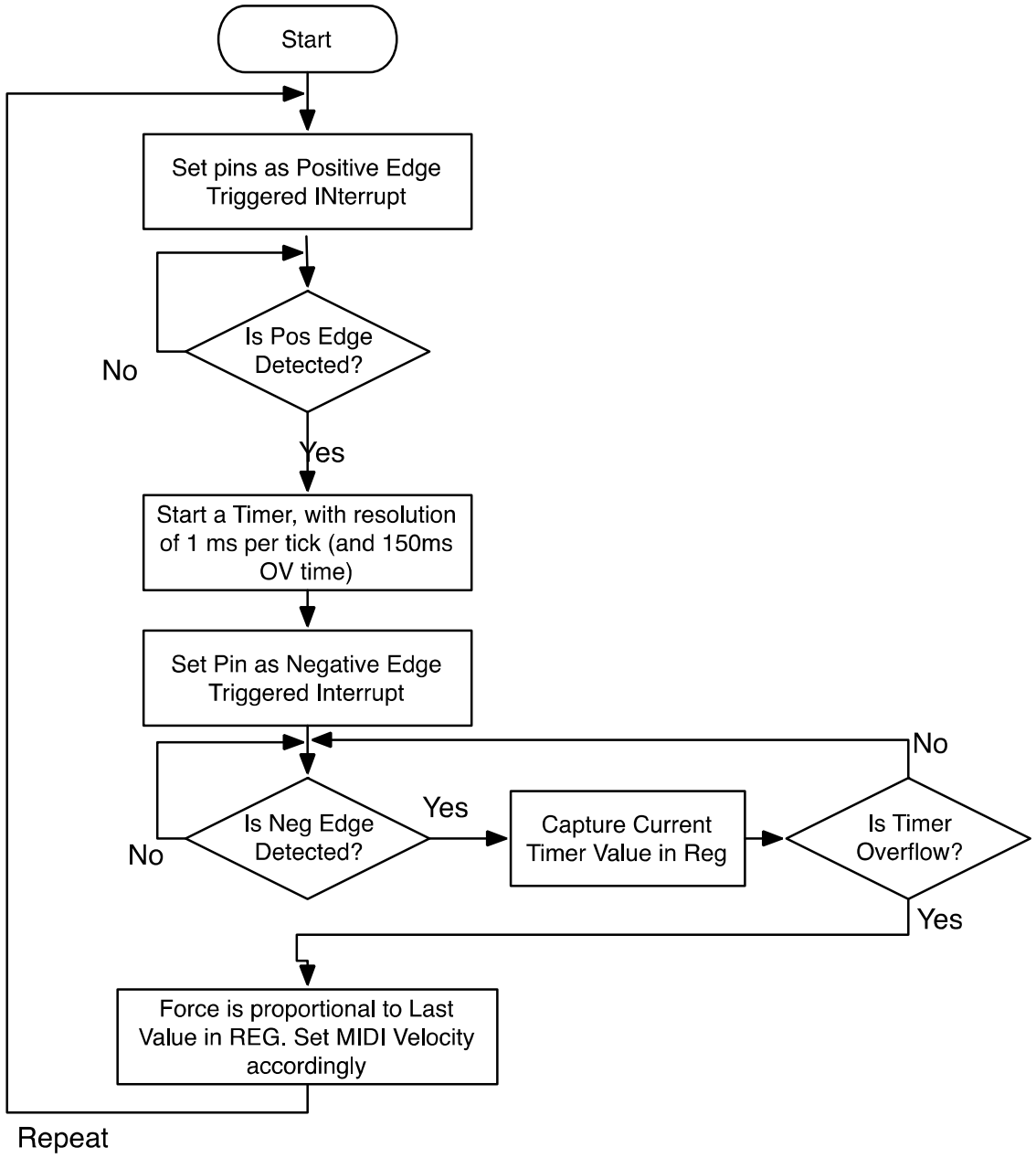
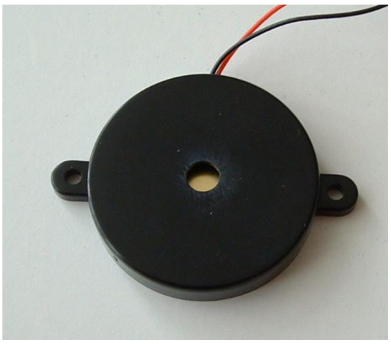


Figure 2-9: Flowchart to detect the timing event on the interrupt pin to detect the piezo force

CONSTRUCTION:

Each pad is a circle of Duralar (i.e. thick mylar) sandwiched between two pieces of foam (from Pearl Art Store). Attached to the Duralar circle is a piezo sensor extracted from its plastic casing. Everything is then just glued together. Each pad cost about $3.00 to make.

We bought the piezo elements at 3100hz from Radioshack and carefully removed the inner piezo sensor from the plastic case. It's a very delicate process and we broke a few in the process.

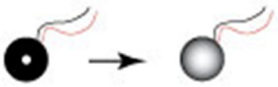


After Careful Extraction Process

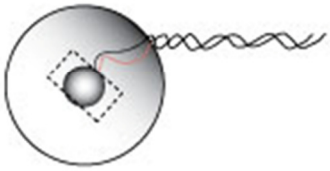
Assembly Instructions

1 Cut 2 circles of foam & 1 circle of Duralar

2 Carefully extract the piezo sensor from the plastic case



3 Solder longer wires onto the piezo leads and tape to the back of the Duralar circle



4 Glue two pieces of foam to the top and bottom of the Duralar pad. Cut a hole in the lower foam circle and route the wire out the bottom.

2.5 The MIDI specification and Implementation

2.5.a The General Midi (GM) Protocol

- MIDI (Musical Instrument Digital Interface) is a digital, non-proprietary hardware and software protocol for data communications among electronic musical instruments and computers.
- GM 1.0 specifies a hardware level specification as well as a software/firmware level specification. We deal only with the software portion – which deals with message types and their meanings. A comprehensive list of the different types of MIDI messages is provided in the appendices.

2.5.b Some Important MIDI Commands

- There are 8 categories of status messages.
- The left four bits indicates the type of message
    - 8 = Note Off
    - 9 = Note On
    - A = AfterTouch (i.e. key pressure)
    - B = Control Change
    - C = Program (patch) change
    - D = Channel Pressure
    - E = Pitch Wheel
- The right four bits indicate the MIDI channel (0-15).
- The following table lists some important MIDI messages we used and a brief description of their function.

Function	**MIDI commands**
Channel Volume Set (max 127)	0xB0, 0x07, <vol>
Bank set (Implementation dependent)	0xB0, 0x00, <bank>
Instrument/Patch select	0xC0, <patch>
Note On	0x90, <note>, <attack_velocity>
Note Off	0x80, <note>, <attack_velocity>

2.5.c The VS1053 MIDI Decoder

- The Instrument Shield from Sparkfun houses a VS1053 MIDI/MP3 decoder chip that we use for real-time MIDI decoding.

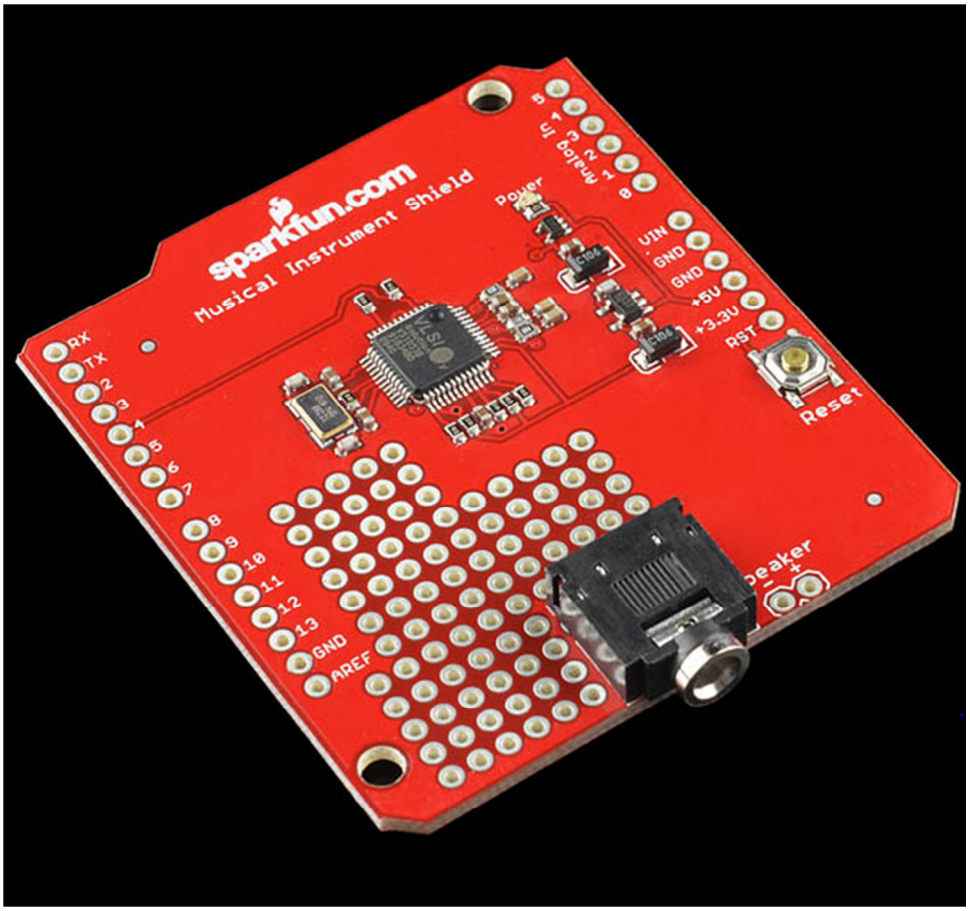


Figure 2-10: MIDI Instrument Shield (Courtsey: www.sparkfun.com)

- The interface to the board is really simple; it expects serial data at 31250 baud.
- It has a reset line which is active high, and can be used to get the co-processor out of a bad state.
- The board runs on a supply voltage of 5V and expects signals only at 3.3V. This necessitated the use of a voltage level shifter which we soldered on the prototyping area of the board.
- The board also has a 3.5mm audio connector which can drive earphones and amplified speakers.

2.5.d Real-Time MIDI

- Real time MIDI is a mode in the VS1053 decoder in which the chip accepts data on the UART at 31250 baud.
- In this, it does real time decoding of the MIDI commands and outputs them to the audio termination.
- It is enabled by holding GPIO0 is low and GPIO1 high during boot, which is implemented in the instrument shield we have.

2.5.e Selection of MIDI Notes for our instruments

We used the following set of notes for our different instruments:

Piano:

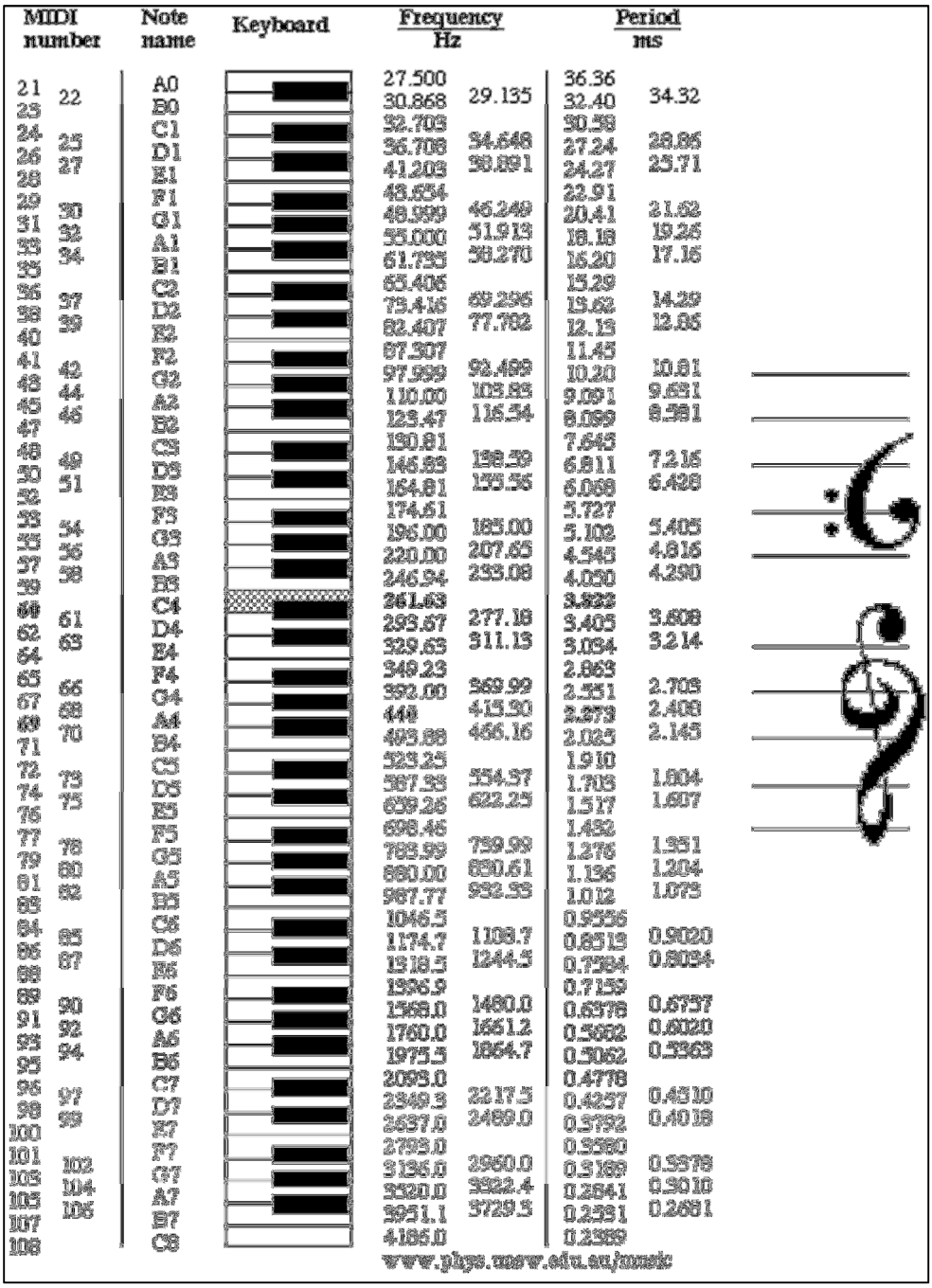


Figure 2-11: Piano MIDI Notes (courtesy: http://www.phys.unsw.edu.au/jw/notes.html)

Drums:

- 40 – Snare Drum
- 36 – Bass Drum
- 48 – Tom Tom
- 41 – Low Floor Tom
- 51 – Ride Cymbal 1

Guitar:

- 52 – String 1
- 57 – String 2
- 62 – String 3
- 67 – String 4
- 71 – String 5
- 76 – String 6

2.5.f Playing MIDI notes on a PC using a serial port

- We wanted to provide an additional termination option for the audio apart from the onboard analog audio generator on the MIDI decoder.
- We looked into playing MIDI notes on a computer transferred via a serial line.
- The project described here: http://www.spikenzielabs.com/SpikenzieLabs/ turned out to be just what we needed for this.
- We installed the software and sent it the same MIDI data which we were sending to the hardware MIDI decoder and it worked perfectly out-of-the-box
- Hence this is what we ended up implementing for out alternate audio termination.
- The software uses a virtual MIDI port emulated over a COM port, and hence we did not need to buy a specialized MIDI port for the computer.

2.6 4-Wire Resistive TouchScreen

2.6.a Construction of Resistive TouchScreen

A resistive touch screen is constructed with two transparent layers coated with a conductive material stacked on top of each other. When a finger or a stylus on the screen applies pressure, the top layer makes contact with the lower layer. When a voltage is applied across one of the layers, a voltage divider is created. The coordinates of a touch can be found by applying a voltage across one layer in the Y direction and reading the voltage created by the voltage divider to find the Y coordinate, and then applying a voltage across the other layer in the X direction and reading the voltage created by the voltage divider to find the X coordinate.

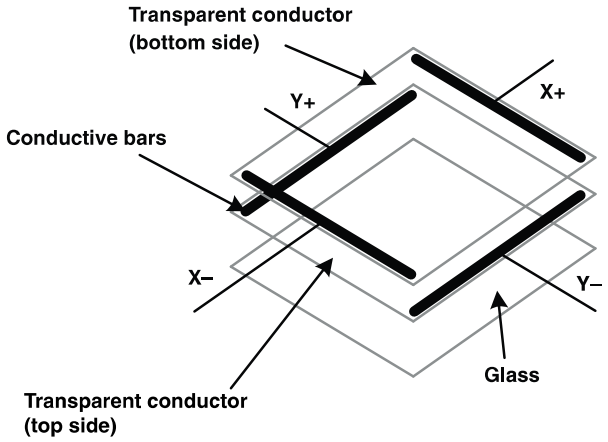


Figure 2-12 http://focus.ti.com/lit/an/slaa384a/slaa384a.pdf

2.6.b Reading X-Y Co-ordinates when from TouchScreen

$$y = \frac{V_{X+}}{V_{Drive}} \times height_{screen} \qquad x = \frac{V_{Y+}}{V_{Drive}} \times width_{screen}$$

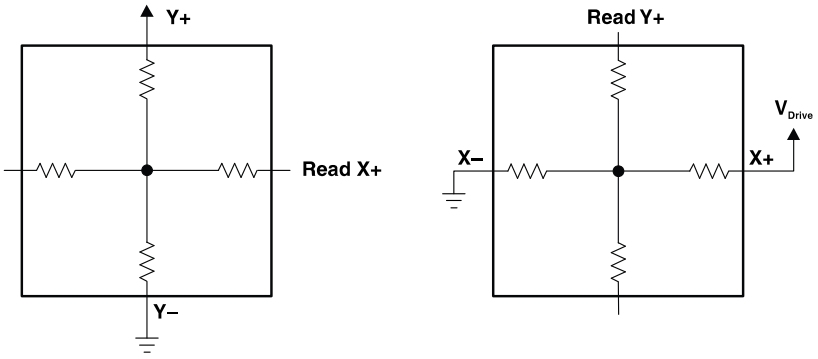


Figure 2-13 http://focus.ti.com/lit/an/slaa384a/slaa384a.pdf

2.7 SD Card and SPI Interface

The CC2430 Module is used to communicate with the 4 GB SDHC card via the SPI interface. SPI is a fast and efficient protocol that allows for simultaneous bidirectional data transfer. Serial data is transmitted and received by the CC2430's dedicated USART1 peripheral in SPI mode. The hardware interconnection for the master-slave configuration operating on a single supply voltage is shown in following figure.

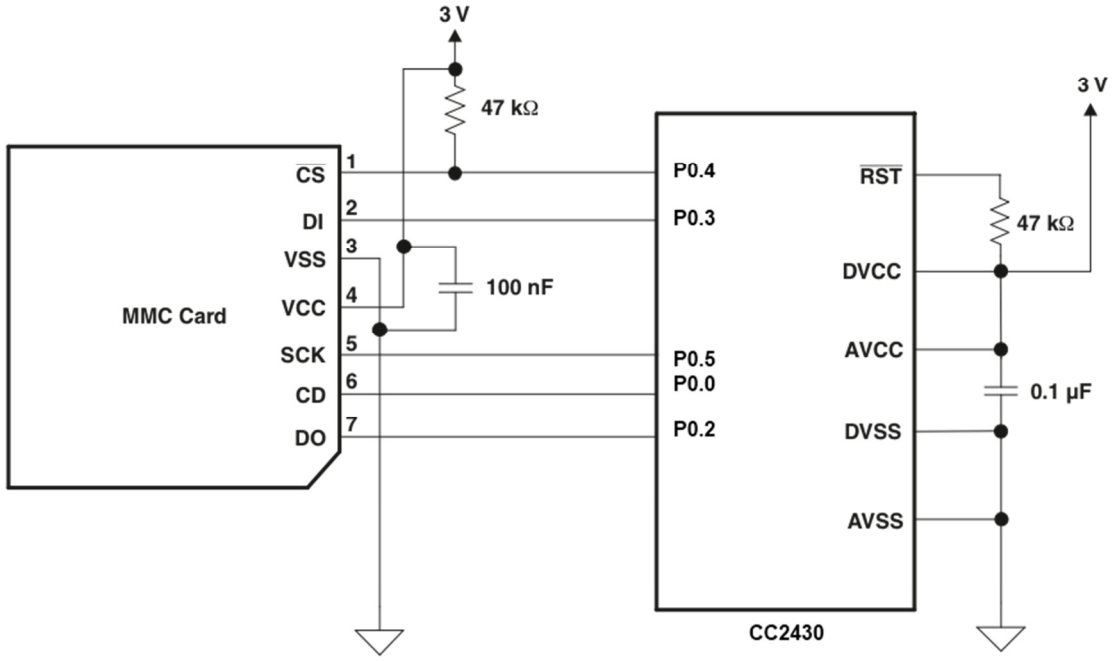


Figure 2-14 Connection Details

2.7.a SD card SPI Protocol

SD Card SPI messages are built from command, response and data-block tokens. All communication between host and cards is controlled by the host (master). The host starts every bus transaction by asserting the CS signal low. Every command or data block is built of eight bit bytes and is byte aligned (multiples of eight clocks) to the CS signal.

In addition to the command response, every data block sent to the card during write operations will be responded with a special data response token. A data block may be as big as one card write block (WRITE_BL_LEN) and as small as a single byte.

The SD Card wakes up in the SD Bus mode. It will enter SPI mode if the CS signal is asserted (negative) during the reception of the reset command (CMD0). If the card recognizes that the SD Bus mode is required it will not respond to the command and remain in the SD Bus mode. If SPI mode is required, the card will

switch to SPI mode and respond with the SPI mode R1 response. Timing sequence for Resetting/Initializing SD Card is shown below.

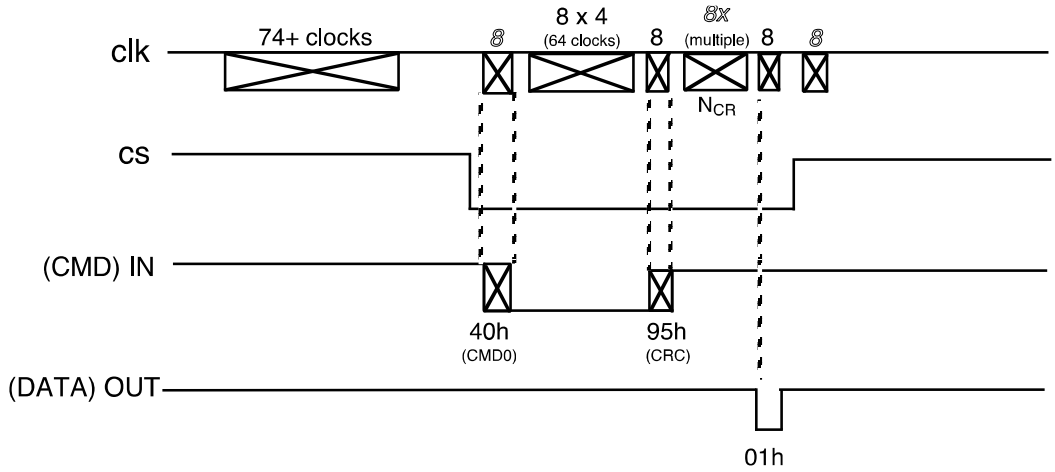
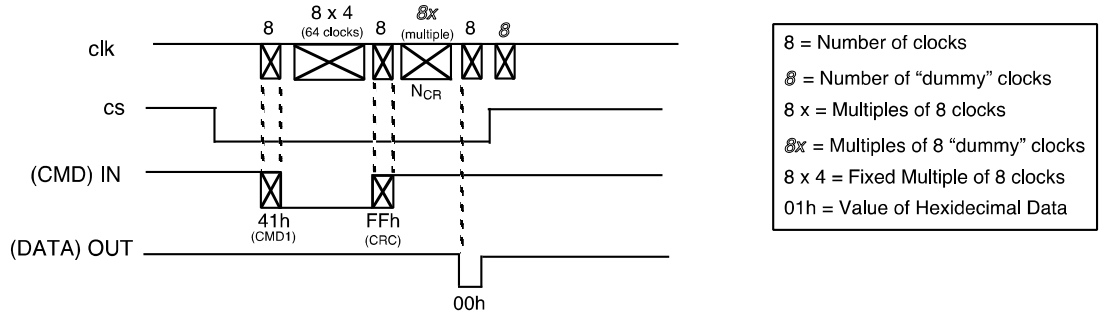


Figure 2-15 SD card Reset and Initialization Sequence
http://mfb.nopdesign.sk/datasheet/cat_d/MMC/spitiming.pdf

Init (CMD 1)



2.7.b SD card Block Read

SPI mode supports single block and multiple block read operations (SD Card CMD17 or CMD18). Upon reception of a valid read command the card will respond with a response token followed by a data token in the length defined in a SET_BLOCK_LENGTH (CMD16) command.

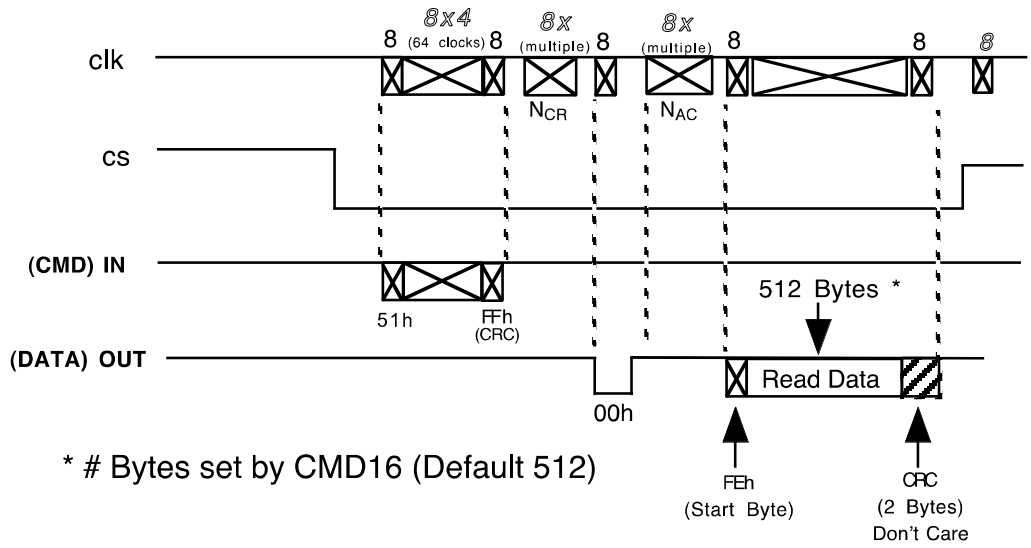


Figure 2-16 :http://mfb.nopdesign.sk/datasheet/cat_d/MMC/spitiming.pdf

2.7.c SD card Block Write

In SPI mode, the SD Card supports single block or multiple block write operations. Upon reception of a valid write command (SD Card CMD24 or CMD25), the card will respond with a response token and will wait for a data block to be sent from the host. The only valid block length, however, is 512 bytes. Setting a smaller block length will cause a write error on the next write command.

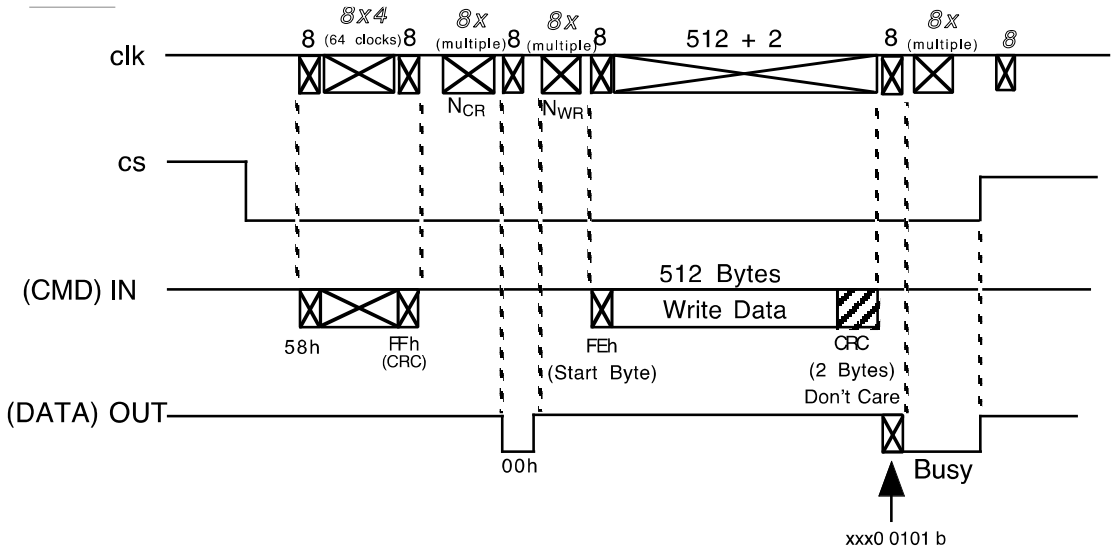
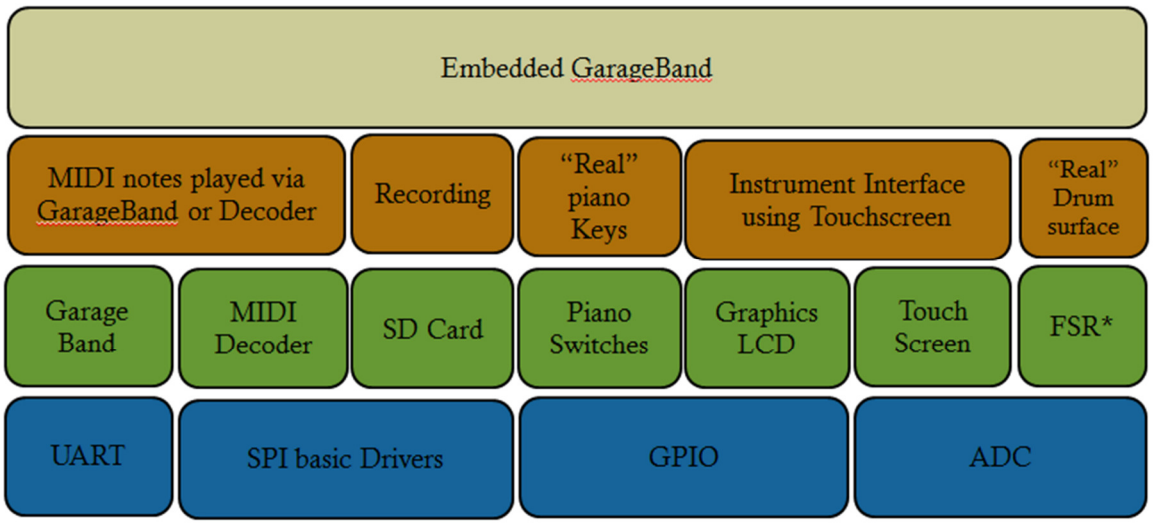


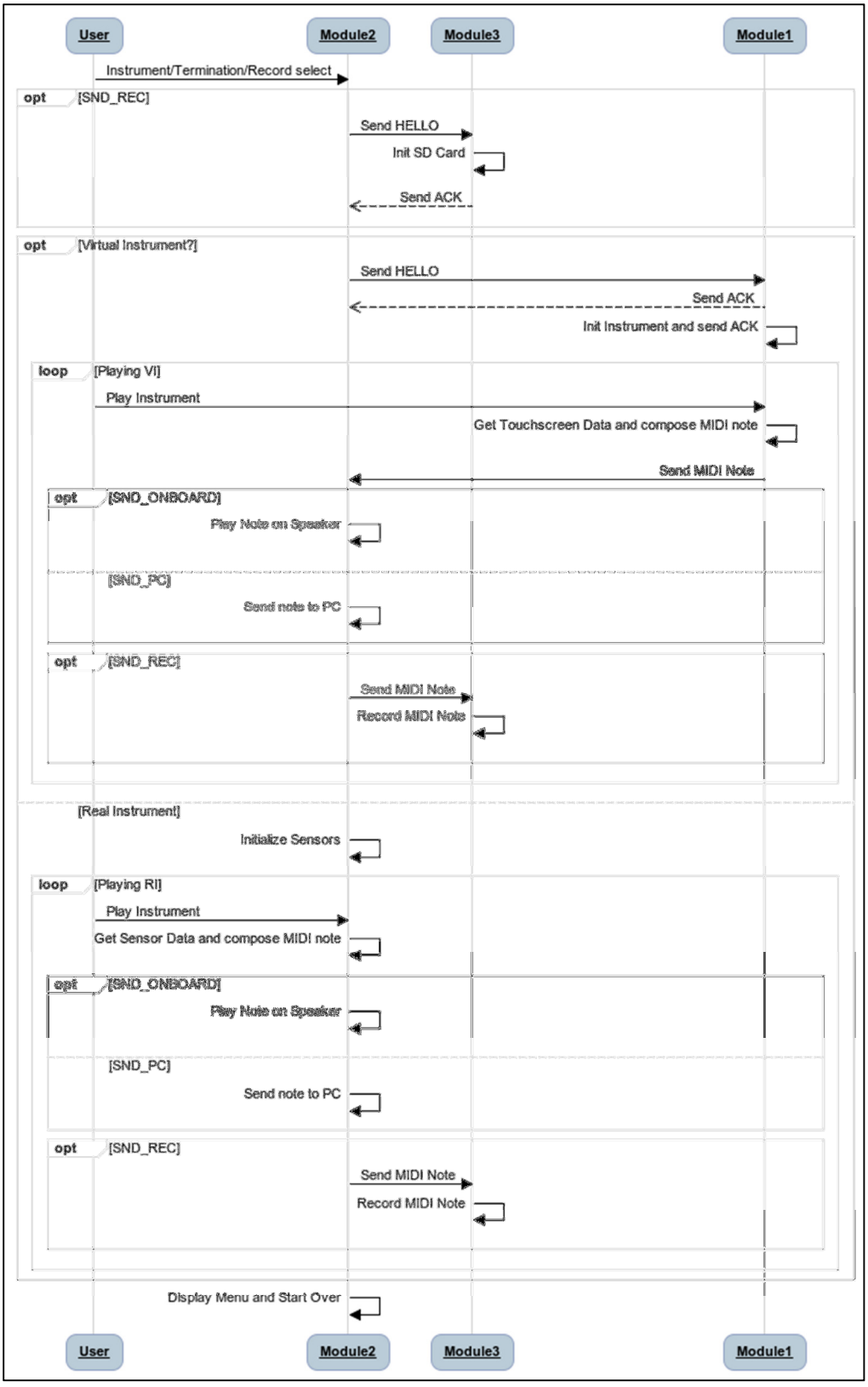
Figure 2-17: http://mfb.nopdesign.sk/datasheet/cat_d/MMC/spitiming.pdf

3. Software Description

3.1 Software Modules



3.2 System Sequence Diagram



Description of terms used:

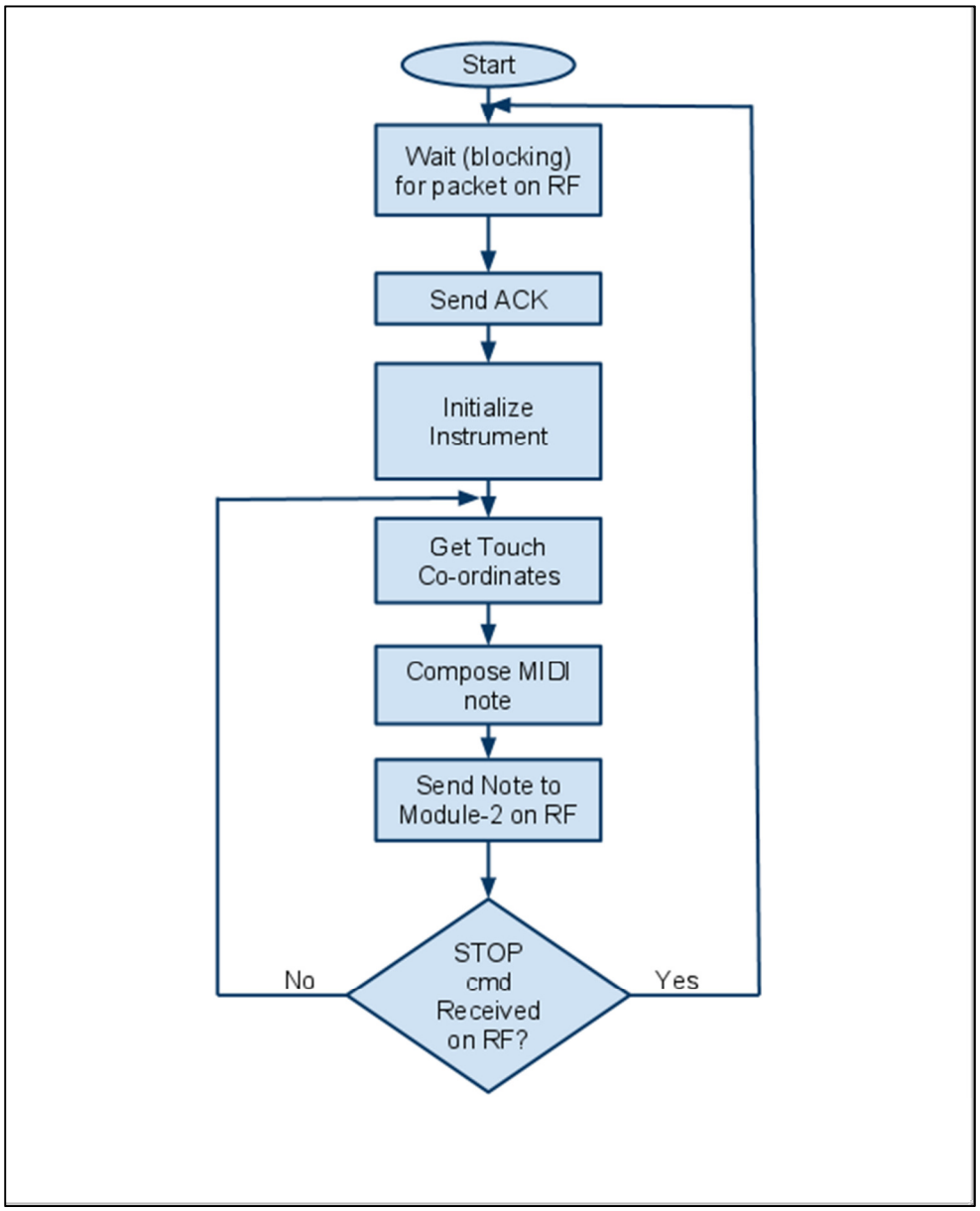
Term	Description
SND_REC	Record track to SD card
SND_ONBOARD	Play on onboard speaker
SND_PC	Play MIDI notes on PC
VI	Virtual Instrument
HELLO/ACK	Inter-module Communication Handshake

3.3 Software flow for Each Module

3.1.a Module 1 Software Flow

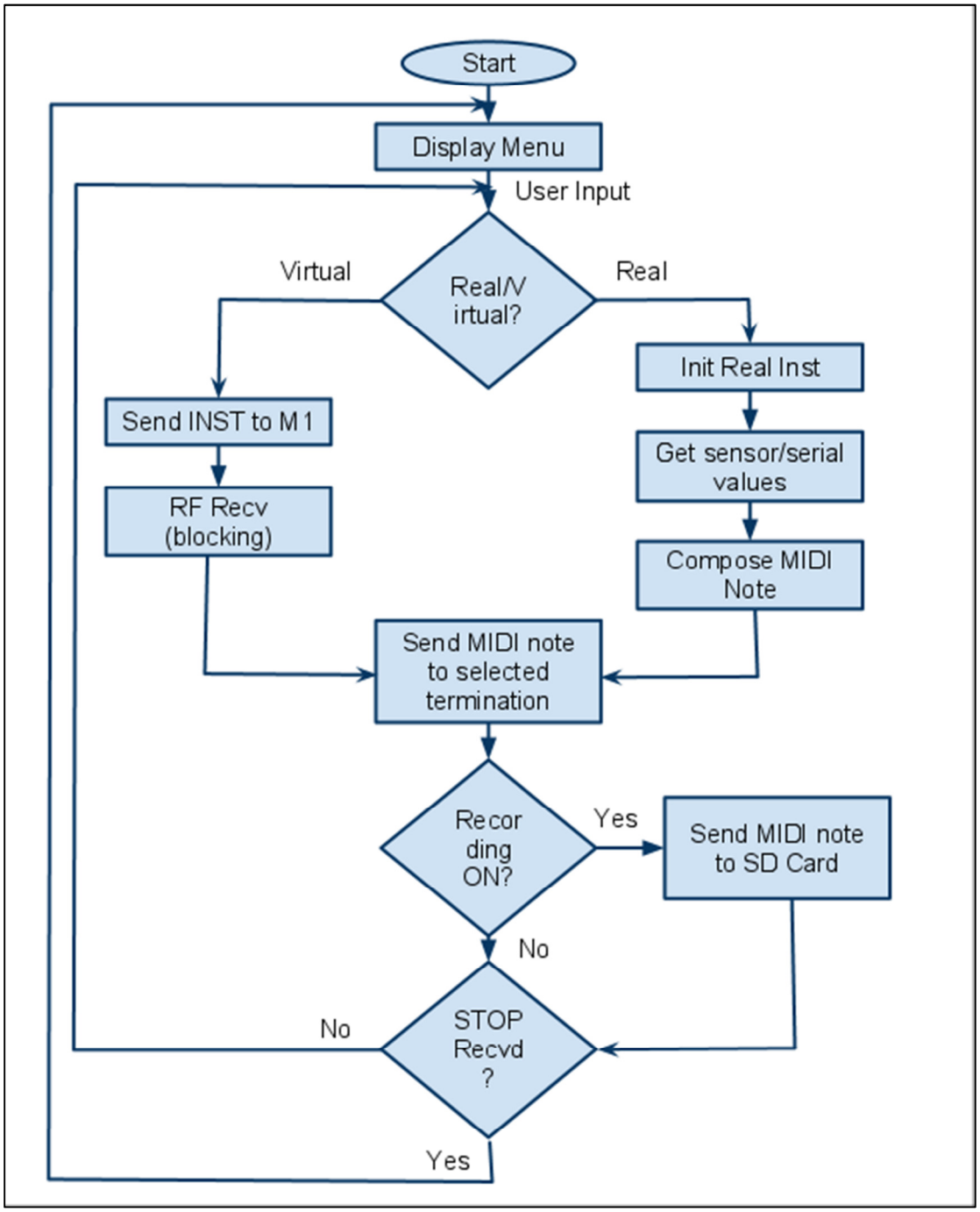
Module 1 interfaces the Graphics LCD, the touchscreen. There is a P/S2 connector and a board pressure sensor (using a Piezo Buzzer) also connected in the hardware but we have not implemented the software for either of those.

The graphic LCD is connected via a UART operating at 115200 baud. The touchscreen is connected to four of the module's multi-purpose pins which switch between pulled-high, pulled-low, and Analog sense for correct touchscreen operation. There is a serial UART for debugging.



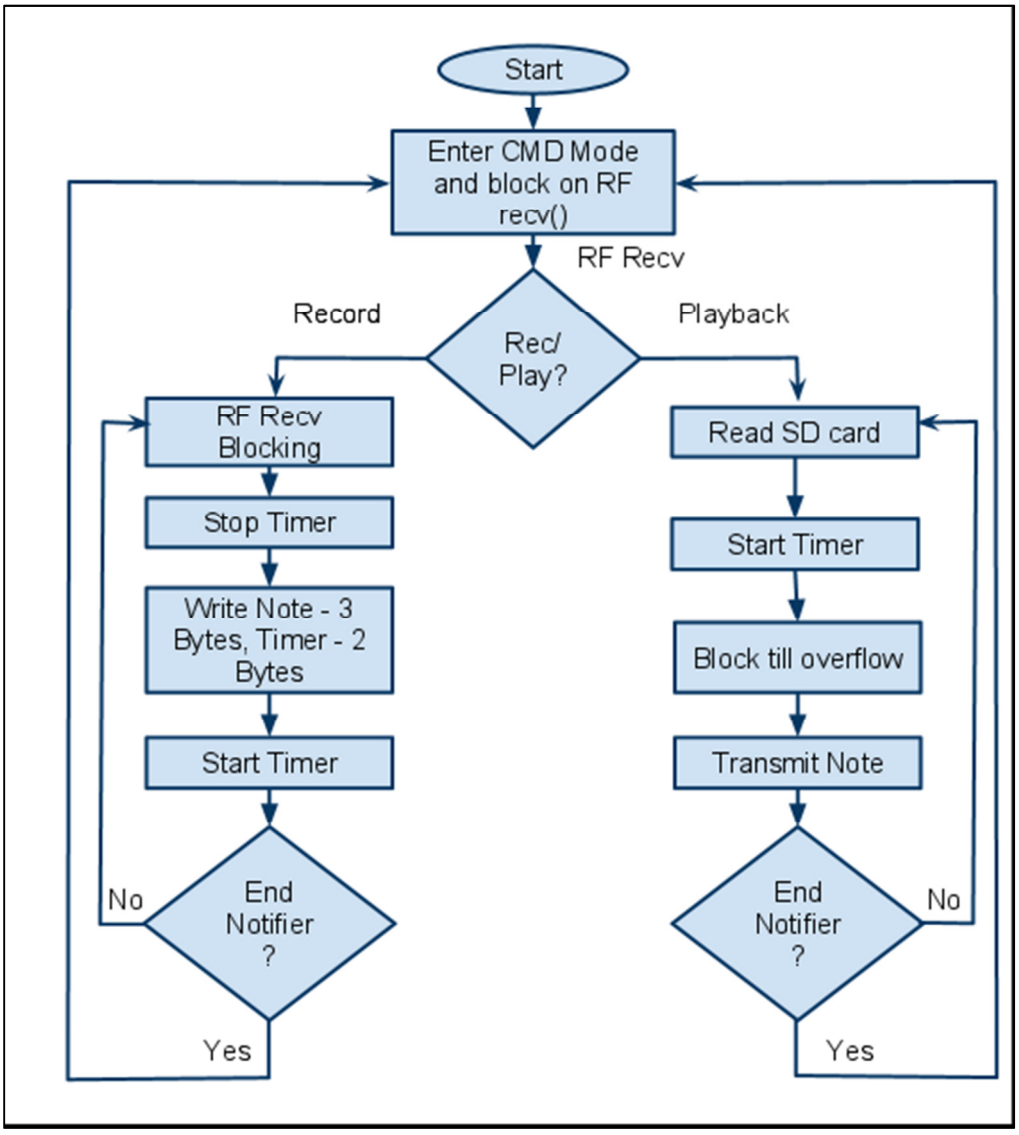
3.1.b Module 2 Software Flow

Module 2 is the master module which talks to the user via a UART, and commands Modules 1 and 3, and the MIDI controller. It also Interfaces the 'Real' Drums (via foam pads) and the 'Real" Piano (via serial data from a keyboard).



3.1.c Module-3 Software Flow

Module-3 is an off-board module dedicated for the SD card. It is powered using a separate battery and uses the SPI protocol to communicate with the SD card. The SD card we used is a 4GB card with 512 sectors. Using our driver we can only read/write at sector level granularity, i.e. in 512 Byte chunks. We use a timer to store a 16 bit delay between notes while storing them. This is used while playback; we delay for the amount of time associated with each note.



SD Card Storage format:

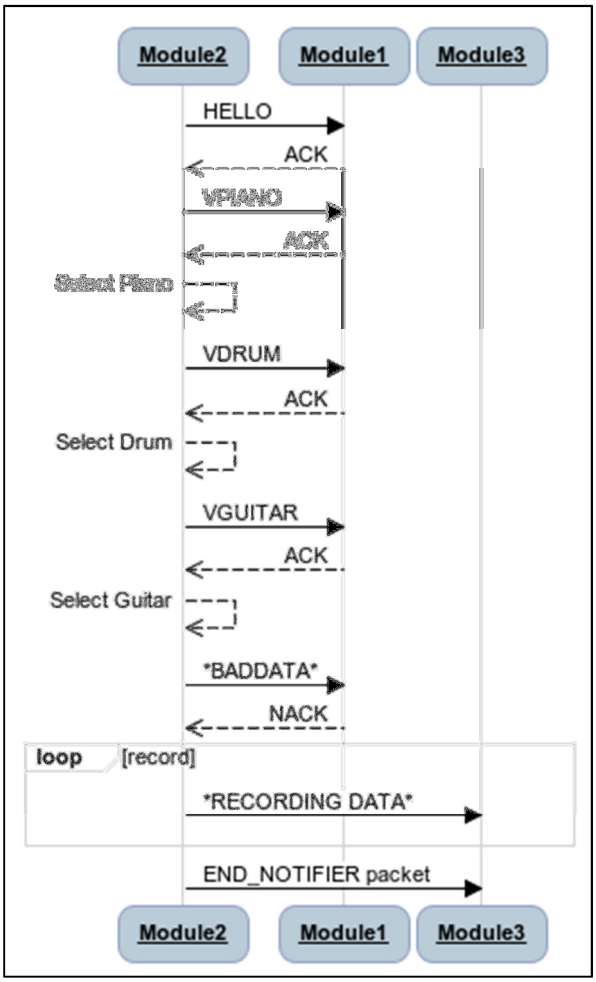
MIDI Note (3 Bytes)			Delay (2 Bytes)	

3.4 Inter Module Communication Protocol

Inter module communication manifests itself in the following possibilities:

1. M2 sending M1 “HELLO”
2. M1 replying to M2 with “ACK” or “NACK”
3. M2 sending M1 “VPIANO”
4. M2 sending M1 “VDRUM”
5. M2 sending M1 “VGUITAR”
6. M2 sending M1 “STOP” (Asynchronous End Notifier)
7. M2 sending M1 3-byte MIDI notes
8. M1 sending M3 Recording data (for SD card) to M3
9. M1 sending M3 three bytes of 0xFF (Asynchronous End Notifier)
10. M3 sending M1 three bytes of 0xFF (Track End Notifier)

This is a high level simplified communication we developed to allow ease of programming. The lower level wireless driver takes care of data integrity and assurance of reception for us, and we don’t need to account for that in out higher level protocol.



4. Testing & Development

We had a lot of basic modules/components that needed to be tested and we went about testing them one by one before we jumped in the final system integration. Component level testing enabled us to verify and understand the functionality of the newer components easily.

4.1 CC2430

It was very essential to test the CC2430 modules since they are complex SOC. We had the following tests/steps for testing the CC2430:

4.1.a. *Basic Drivers*

We wrote low level drivers & built infrastructure for basic peripherals like serial port, timers, counters, interrupts, ADC, SPI, I2C. We wrote small application layer tests to verify their functionality.

4.1.b. *Test Applications*

We wrote basic low level applications like Blinking LED and basic serial Transmit receive to test the application layer.

4.1.c. *Enhancements*

We then transformed the normal serial drivers to FIFO based serial drivers for greater efficiency.

4.1.d. *RF Drivers*

We then wrote basic drivers for RF communication between two modules. It took us long time to figure out the basic RF setting for communication. We now had all the basic drivers working with the new CC2430 modules and we could go ahead and test the rest of the components.

4.2 Graphics LCD

Another crucial part of the system was the graphics LCD. In order to testthe LCD we did not wish to use a microcontroller, since then we ould end up debugging two things if they did not work as expected. So initially we powered up the LCD from an external 5V supply and connected the TX from the Computer (via a RS232 circuit) to the RX of the Graphics LCD.

Using Advanced Serial Terminal Programs, we created scripts and macros to write a small test code that helped us understand the functionality in depth. Code for displaying the instruments was algorithmically developed on the macros and then ported over to work with the CC2430 modules. The prior testing and algorithm development using PC as the command master helped us in reducing the time to get the LCD working.

4.3 Touch Screen

A very important element of the project was the touchscreen interface. As already described the touch screen is an analog interface. We were working on the process of driver development on CC2430 and getting the touchscreen ready, simultaneously. Hence we could not use the CC2430 to test the Touchscreen interface.

We connected an external A/D converter along with an 8051 microcontroller (and some tweaks in the circuit) and tested the 4-wire touchscreen. It was very useful for us to understand the Touchscreen interface early since the code for touchscreen is a bit complex. Also we had the drivers for driving an external A/D using the 89C51. This previous testing eased the process of porting the touchscreen interface to the CC2430.

4.4 MIDI Decoder

The heart of the project is the VS1053 MIDI decoder IC. It was very essential that we understand and test all the functionality of the MIDI decoder. One of the major hurdles was to generate an exact baudrate of 31250bps that can be fed to MIDI. We tried to do this using a microcontroller but the baudrate generated was not precise but 10% off. While we develop drivers for CC2430, we hooked up the VS1053 to the computer via a logic converter.

Using Advanced Serial Terminal Programs, we created scripts and macros to write a small test code that helped us understand the functionality in depth. Code for initializing the MIDI using MIDI command was algorithmically developed on the macros and then ported over to work with the CC2430 modules. Modifying the macros is much less time consuming rather then testing every musical note by microcontroller code modification & ISP.

5. Debugging

Problems	Debugging steps/Problems
Generating 31250 baudrate for Real time MIDI	- MIDI requires precise 31250 baudrate. Generating this baudrate was a challenge. - To test it on the 89C51 board, we had to replace the crystal from 11.0592Mhz to 12.000Mhz (since 12Mhz has 31250 in its standard baud rate) - We could not test it on CC2430 Embedded Board unless we got the 31250 baudrate working. We had to research a lot to get this baudrate on CC2430. - The 31250 baud generated by the CC2430 never printed proper data on the serial terminal also set at 31250, so we never knew that we are getting the correct baudrate generated. - We connected the Logic analyzer and measured the baudrate to find that it was 31250 and the custom baudrate in terminal was messed up.
MIDI board reset switch is not as per specs	- There is a MIDI reset switch on the MIDI board, which is connecting to the reset of the board according to the datasheet. We used this switch to reset the MIDI board in case of any problems. - But we always faced weird problems. When we actually saw on the board of MIDI, the switch was not connected anywhere. The Switch was in reset mode only if you have it interfaced with an Arduino. The specs never said that.
Logic converter generated 3.3V instead of 5V	- The logic converters that we had to convert between 5V to 3.3V works best one way. When converting from 3.3V to 5V it works great. But when converting from 5V to 3.3V it is supposed to generated 3.3V for every 5V at input. - We had problems with the other way communication. When we probed the entire data path on CRO we found that the 5V is not converted to 3.3V but only to 2.5V. When we look at the internal circuit diagram it was evident why it was happening (since they used resistor divider with equal value resistors to step down) - We had to tweak the circuit to generate 3.3V
Touchscreen Interface needs strong pull ups.	- The touchscreen interface requires that a single pin supply voltage at one time and at other time it gives the analog voltage proportional to x/y coordinate. - The sequence said that a weak pull up will work, but it did not work exactly. - It would work but decrease the range of the output affecting the area of the touchscreen. We took a long time to debug this. - CC2430 GPIO can be configured to have strong pull-ups. When testing with CC2430, the problem was solved.

Graphic LCD Delay	- The graphical LCD we were using has a serial to parallel converter on its backpack that uses ATMEGA128 running a firmware. When trying to write multiple bytes to the LCD it would go crazy and the display would be messed up. - It was couple of hours later we realized that the buffer of the ATMEGA got filled and hence we had to put delays. We tried small delays but the problem persisted. When we put delays of appx 10-12ms it did work well.
Graphic LCD Baudrate	- The Graphics LCD accepted commands at 115200 Baudrate. Using a PC terminal we set the LCD to operate at 115200. Since while testing with 89C51, both the LCD and the ISP uses the same serial port, the LCD got some commands during ISP that changed its configuration. - We were unaware of this, and continued doubting our software. We directly hooked up the LCD to computer to see if it worked at the expected baudrate and it did not. We reset the LCD using the PC serial terminal. - The problem was in using the same port for ISP and serial LCD. During ISP, some of the ISP commands changed the baudrate configuration on the LCD.
PS2 Noise	- The PS2 keyboard we had was very noisy. We debugged it for a long time to get it working. When we hooked it up to a Logic analyzer, we found that between the expected edges, we had few other edges. - When we looked at the signal on the Oscilloscope our doubt was confirmed. It was lot of noise in the system that was triggering the interrupt falsely.

6. Results and Error Analysis

We got all the things that we planned working perfectly. It was a great achievement. We would have loved to get the PS2 keyboard working too but we did not have a keyboard that had less noise in it. I wish that PS2 keyboard were not so scares to find replacement for. The results were pretty impressive. Though we discussed debugging in the previous section few things are worth mentioning:

1. Overheated LCD regulator: Due to some glitch in the design g the power supply circuit we had overheating of the LM7805 regulator which forced us to use a heatsink on it which solved the problem.

2. Delay in the playback: During the playback of the MIDI notes from the SD card we have to read an entire 512 bytes of sector from the SD card which takes a large amount of time. This time is not visible when we are playing large number of MIDI notes but it is clearly a huge overhead when playing very few notes.

3. Drum to sound delay: We had considerable delay in the amount of time it takes from the time the drum is hit to the time the drum sound is played. This was primarily due to the fact that piezo element takes some time to settle below the oscillation threshold value. Before it settles down we cannot determine exactly the amount of force. This forces us to wait for a few milliseconds. Adding to this is the RF link delay and the MIDI playback delay. This causes a lot of error which cannot be compensated for so easily.

7. Conclusion

The project was a very ambitious one that taught us a lot. It was very crucial to work in a team and communicate with each other very often. Working on complex project in a team teaches you a lot. In this project we realized the power SOC's possess and the simplicity of using them. The project taught us many concepts in system integration and programming(from a team's perspective). We used many new tools like IAR embedded systems, MIDI synthesizer software, debugger etc. We also ended up using many administrative tools like Version control on Google code, Issue Tracker, Gantt chart, etc. Overall it was a great learning experience for all of us. We are glad that the project and the process of learning through it was a fun process.

8. Future Development Ideas

There are definitely few things that everyone wishes that should be done differently. As an extension to this we would like to wirelessly stream MIDI notes to play them. Also we would love to add the support to play MIDI files directly from the

SD card for which we need to have the file system implemented on the SD card. If it would not have been for monetary reasons we would have certainly gone for a bigger graphics LCD. May be purchase one that has a capacitive touchscreen inbuilt.

As a music player system we would love to extend the system to include the decoding and playing of ogg, flac, wav, mp3 files also. The current decoder we have does support these formats but due to lack of time we were not able to add thise extra features.

9. Acknowledgements

First of all we would like to thank Prof.Linden McClure for such an outstanding course, extensive labs, tools, logic analyzer, for answering our questions, being helpful thought the semester.

We would also like to thank sparkfun for their prompt customer service and quick responses. We also acknowledge Nathan Seidle, from Sparkfun for such a motivating presentation during class.

We are also deeply grateful to all the authors of the various sources cited in the references. Without their support & guidance this would not have been possible.

10. References

[1] Wikipedia article on MIDI - http://en.wikipedia.org/wiki/Musical_Instrument_Digital_Interface
[2] MIDI Tutorial on arduino forums - http://arduino.cc/en/Tutorial/Midi
[3] Support for VS1053 - http://www.vlsi.fi/en/support/software/microcontrollersoftware.html
[4] Resource for Drums - http://code.google.com/p/ardrumo/
[5] Serial_MIDI - http://www.spikenzielabs.com/SpikenzieLabs/Serial_MIDI.html
[6] MIDI Tutorial - http://www.indiana.edu/~emusic/etext/MIDI/chapter3_MIDI4.shtml
[7] LCD Tutorial - http://www.sparkfun.com/tutorials/120
[8] SD Card Interfacing Application note - http://www.cyantechnology.com/public/AN037InterfacingtoanMMCorSDCardviaSPI.pdf
[9] MIDI Spec - http://www.srm.com/qtma/davidsmidispec.html
[10] 4-Wire touchscreen project - http://kalshagar.wikispaces.com/Arduino+and+a+Nintendo+DS+touch+screen
[11] CC2430 official documentation - http://focus.ti.com/docs/prod/folders/print/cc2430.html
[12] FTDI USB-Serial documentation - http://www.ftdichip.com/
[13] Piezo Element information - http://www.radioshack.com/product/index.jsp?productId=2062402
[14] Chipcon CC2430 General help - http://e2e.ti.com/
[15] EDAboard forums for misc topics - http://www.edaboard.com/forum.php
[16] 8051 basics – http://www.keil.com/
[17] 8051 code examples - http://www.8052.com/codelib

11. Appendix

11.1 Appendix – Bill of Material

Product	Part no	Vendor	Nos	Cost/unit	cost
Piezo Element 1500-3000Hz	273-073	Radioshack	7	$1.99	$17.91
VS1053B MIDI Decoder	DEV-10529	sparkfun	1	$29.95	$29.95
Level Shifter	BOB-08745	Sparkfun	NA	$1.95	$9.75
SDcard holder	PRT-00136	Sparkfun	1	$3.95	$3.95
Graphics LCD	LCD-09351	Sparkfun	1	$34.95	$34.95
touchscreen	PRT-0133	Sparkfun	**1**	$9.95	$9.95
PS2 connector	CON-09333	Locally	**1**	$5.45	$5.45
Zener	3.3 OR 5V	Locally	7	NA	NA
Resistor	1M	Locally	7	NA	NA
Quad comparator	LM339	Locally	2	NA	NA
16:4 encoder	NA	Locally	1	NA	NA
Push Buttons		Locally			
				TOTAL:	**$111.91**

11.2 Appendix – Schematics

Please see the CD

11.3 Appendix – Firmware Source Code

Please see the CD

11.4 Appendix – Software Source Code

Please see the CD

11.5 Appendix – Datasheet & application notes

Please see the CD