

# EE 478 Final Project

*Final Report*

---

*June 8, 2010*

*Jonathan Feucht*

*Whitney James*

## ***Abstract***

---

This document describes the prototyping process of a 49-key digital keyboard instrument. This instrument is intended to be a foundation or reference to create other synthesized instruments. One could add more keys, velocity sensitive keys, analog sensors as input, or even skip the traditional piano key all together in favor of something more creative or application specific. This particular keyboard implementation is equipped with an infrared distance sensor to demonstrate the potential functions that analog sensors can perform in music composition and performance. An LCD screen and buttons module is used as a user interface which shows a navigation menu, and is navigated with pushbuttons. The navigation menu allows the user to change musical settings, such as instrument, reverb, chorus and volume. The system is designed to be low-power, low-cost, and simple to replicate. The 49-key musical keyboard consists of an 8×7 button matrix. The microcontroller continually polls the state of the keys, and outputs key events as MIDI serial commands to the synthesizer. A second microcontroller is

responsible for interfacing with LCD and pushbuttons for musical configuration settings. The result is a keyboard musical instrument.

## Table of Contents

Introduction .....	2
Discussion .....	2
Requirement Specifications .....	2
Inputs .....	2
Outputs .....	2
User Interface .....	2
Design Specification.....	3
Operating Specifications .....	3
Reliability and Safety .....	3
Specification of External Environment .....	3
Use Cases .....	3
System Inputs and Outputs.....	3
System Description.....	4
Hardware Implementation .....	6
Software Implementation.....	9
Keyboard Controller.....	9
Navigation menu .....	9
LCD driver.....	10
I <sup>2</sup> C driver.....	12
Testing and results .....	13
Test Specification .....	13
Test plan.....	13
Test Cases.....	13
Results.....	14
Key latency .....	14
I <sup>2</sup> C setting transfer latency.....	14
Power consumption.....	14
Functionality.....	15
Conclusion.....	15
Appendix 1. Bill of materials .....	16
Appendix 2. Schedule .....	16
References .....	17

## Introduction

For our final project, we took apart a Radio Shack keyboard, reused the mechanical key portion with the connecting electronics, and replaced the electronics with our own embedded circuit. We made use of the ATSAM2195 Atmel synthesizer chip in our design, and used an LCD display with a keypad to allow the user to adjust settings. In this paper, we detail the specifications, design and testing procedure, and results for our design of this keyboard.

## Discussion

### Requirement Specifications

This section details the requirements specification for the synthesizer. The system consists of a 49-key keyboard, a keyboard controller, LCD driver, a serial LCD screen equipped with a 6-button control panel, and a synthesizer chip. The keyboard is continually polled for key events, and on key changes, MIDI commands are sent in parallel to the synthesizer chip using standard MIDI messages. The keyboard controller also outputs MIDI messages at 31,250-baud for use with any serial MIDI device. The LCD screen communicates with the keyboard controller using 19,200 baud 8-bit RS-232, no parity. The LCD display consists of 2 rows of 16 characters.

### Inputs

A 49-key touch-insensitive keyboard

Six navigation buttons for: Up, Down, Left, Right, Okay, and Cancel

### Outputs

LCD screen with navigable menu

- Adjustable volume
- Adjustable reverb / chorus
- Adjustable instrument sound (from library of 128 instruments)

Stereo output for dual 4-ohm 3 watt speakers

MIDI standard output via CBI MIDI cable

### User Interface

Menu items used in the hierarchical menu:

- ❖ Effects
  - Reverb
  - Reverb type
  - Chorus
  - Chorus type
  - Pitch bend (on / off)
  - Transposition
- ❖ Volume
- ❖ Instrument
- ❖ Save settings as presets
- ❖ Restore presets to factory default
- ❖ The UP and DOWN buttons navigate the menu among items of the same node.

- ❖ The RIGHT button navigates to the child node associated with the menu item selected.
- ❖ The LEFT button navigates to the parent node.
- ❖ The OKAY button applies the currently selected item in the menu, or navigates to the child node
- ❖ The CANCEL button returns to the root menu

## Design Specification

### Operating Specifications

5V Supply Voltage, maximum power consumption of 2.5 watts

Keyboard controller and synthesizer chip operate off of 3.3 volts, maximum 495 milliwatts

Maximum power consumption by LCD screen with backlight on, audio amplifier, and speakers: 750 milliwatts

Maximum 0.8 ms latency from time of keystroke until time of sound

The Keyboard Controller communicates with the synthesizer using 31,250 baud 8-bit RS-232 with 1 stop bit, no parity.

The LCD screen communicates with the Keyboard Controller with 19,200 baud 8-bit RS-232 with 1 stop bit, no parity.

### Reliability and Safety

The MTBF will be a minimum of 20,000 hours. The counter shall comply with the following safety standards:

- UL 61010-1
- CSA 61010-1
- IEC 1010

### Specification of External Environment

The keyboard is to operate in a consumer environment in with traditional commercial grade temperature environment (0 °C to 70 °C). The unit can be powered by USB supplying 5V, maximum 2.5 watts. The operating humidity is 0 to 93%, non-condensing.

### Use Cases

The user of this device has three use cases:

- Play keyboard music
- Change instrument sound
- Pitch bend sound by use of infrared optoelectronic distance sensor (D-beam)
- Change volume / chorus / reverb for the instrument

## System Inputs and Outputs

### Inputs

- Minimum key hold time to trigger a key event is 0.8 milliseconds.
- A pushbutton has to be held down for at most 1 millisecond in order to trigger a navigation menu event.

- Menu allows users to select instrument, and modify volume, chorus, and reverb settings.

### **Outputs**

- Signal latency of 0.8 ms latency, with no more than 500  $\mu$ s jitter.
- Sound is played mono on 4 $\Omega$  3W speakers.
- Sound output is supplied on 3.5 mm stereo audio jack.
- Navigation menu is displayed on 2 $\times$ 16 character LCD screen.

### **User Interface**

- The keyboard has 49 keys, spanning from C2 to C7.
- The keyboard is touch insensitive.
- A hierarchical menu is displayed on a LCD screen, and is navigated with a 6-button keypad.
- The navigation allows the user to select from 128 musical instruments, and modify values for reverb, chorus, and volume on a scale of 0 to 127.
- Included are eight types of reverb and chorus settings.
- User could save settings as preset values, or recall factory settings.

### **System Description**

An operator of the system is to interface with the device via a 2x16 character display and pushbuttons to select musical settings, such as instrument, reverb, chorus and volume. After the operator is satisfied with their selections they can use the musical keyboard as they would a traditional piano keyboard. The system will output sound corresponding to the operator's keyboard playing and the musical setting made using the LCD and pushbutton interface. In addition to playing the instrument like a traditional piano keyboard, the user may utilize a "pitch bend" feature, which uses an infrared optoelectronic distance sensor located above and to the left of the piano keys. With the pitch bend option enabled in the instrument settings the sensor will respond to hand gestures in the sensors field of view by modulating the instruments overall pitch.

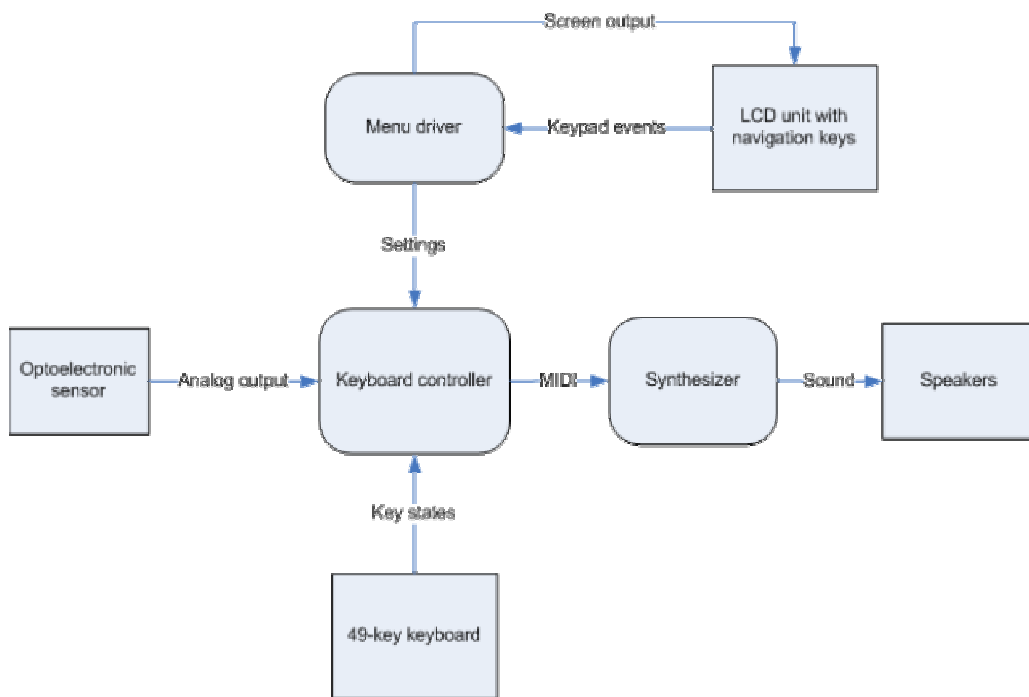


Figure 1. Functional decomposition of keyboard

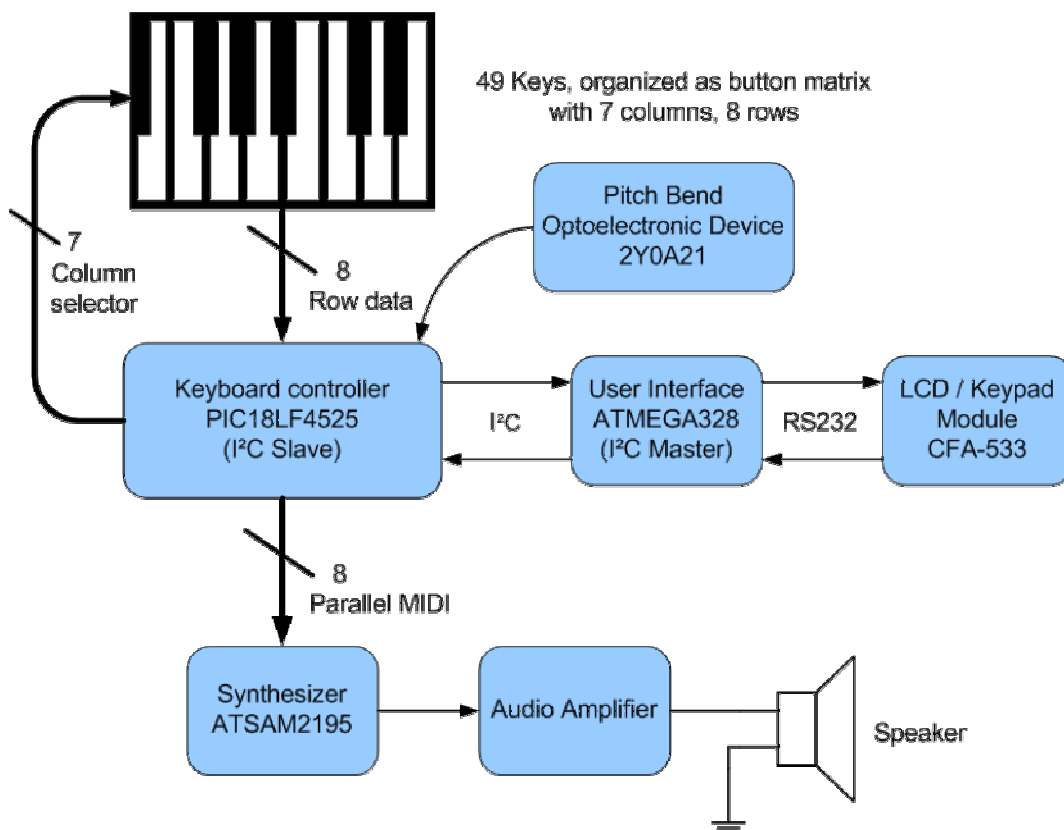


Figure 2. Block diagram of keyboard

## Hardware Implementation

The synthesizer utilizes the ATSAM2195 synthesizer chip, mounted on a QFN-44 to DIP-44 mount. A LCD screen is used as a hierarchical navigation menu for various system options, including volume, reverb, chorus, and instrument sound. A 49-key touch-insensitive keyboard provides key data to a keyboard controller, which converts the note information to MIDI standard format, and sends the information in parallel to the data bus on the synthesizer chip. The keyboard controller also outputs MIDI note commands using 31,250-baud RS-232.

This synthesizer takes in input from a 49-key keyboard, organized as a button matrix with 8 columns and 7 rows. Each group of seven keys is connected to the same node with a diode, where all the anodes are connected together. The keys are arranged as a matrix with rows and columns. On the keyboard I got, the notes are probably grouped by rows of 8, not 16. The keyboard controller polls only one of the rows of eight keys at a time by asserting high one of the row selector pins. When a row is selected, the values of the notes on the row will appear on the column pins for the keyboard controller (pins 2-9, and 19-30 in the image). For 49 keys, you need 7 columns of notes to select out the individual row of eight notes ( $49 / 8 = 7$ , rounded up). Doing the math, 8 rows + 7 columns = 15, which accounts for all the wires in the ribbon. Our next step was to figure out which wire corresponds to which row or column on the key matrix, which we did by using a digital multimeter.

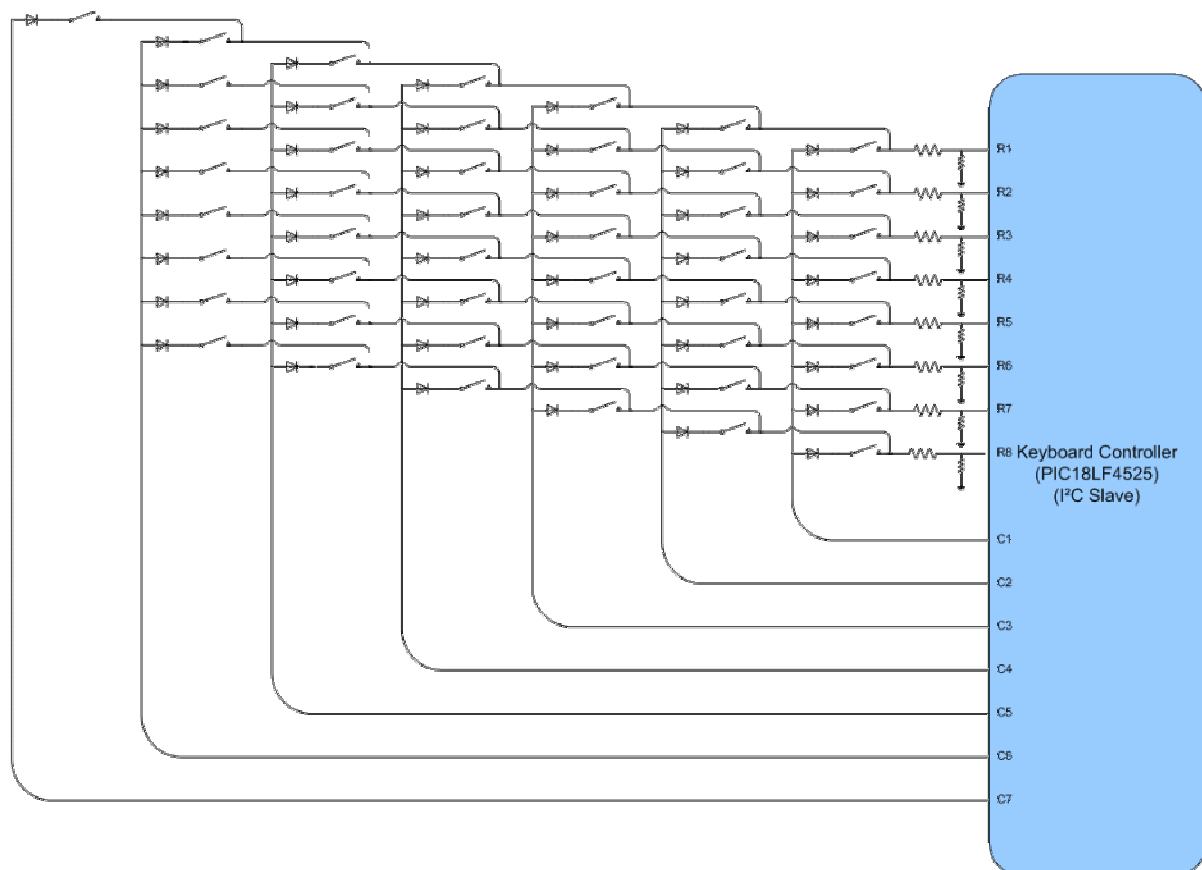


Figure 3. Keyboard schematic

The keyboard controller continuously loops through all the rows of keys and collects data on the state of all the keys. When one key is noticed to have changed, an on/off key event will be triggered, depending

on the new state of the key. The keyboard controller outputs MIDI commands to the synthesizer chip when a key value changes via transmitting. These MIDI commands are transmitted to the synthesizer chip via its parallel input (D0 – D7, \_CS, \_RD, \_WR, and A0). Figure 4 shows the schematic borrowed from the ATSAM2195-EK datasheet. This schematic was referenced while wiring the synth chip on a solderless breadboard. All connections and component values from this schematic were followed accurately. One exception though is that digital and analog ground planes were NOT separated as recommended by the schematic. For the purpose of prototyping the circuit this proved adequate.

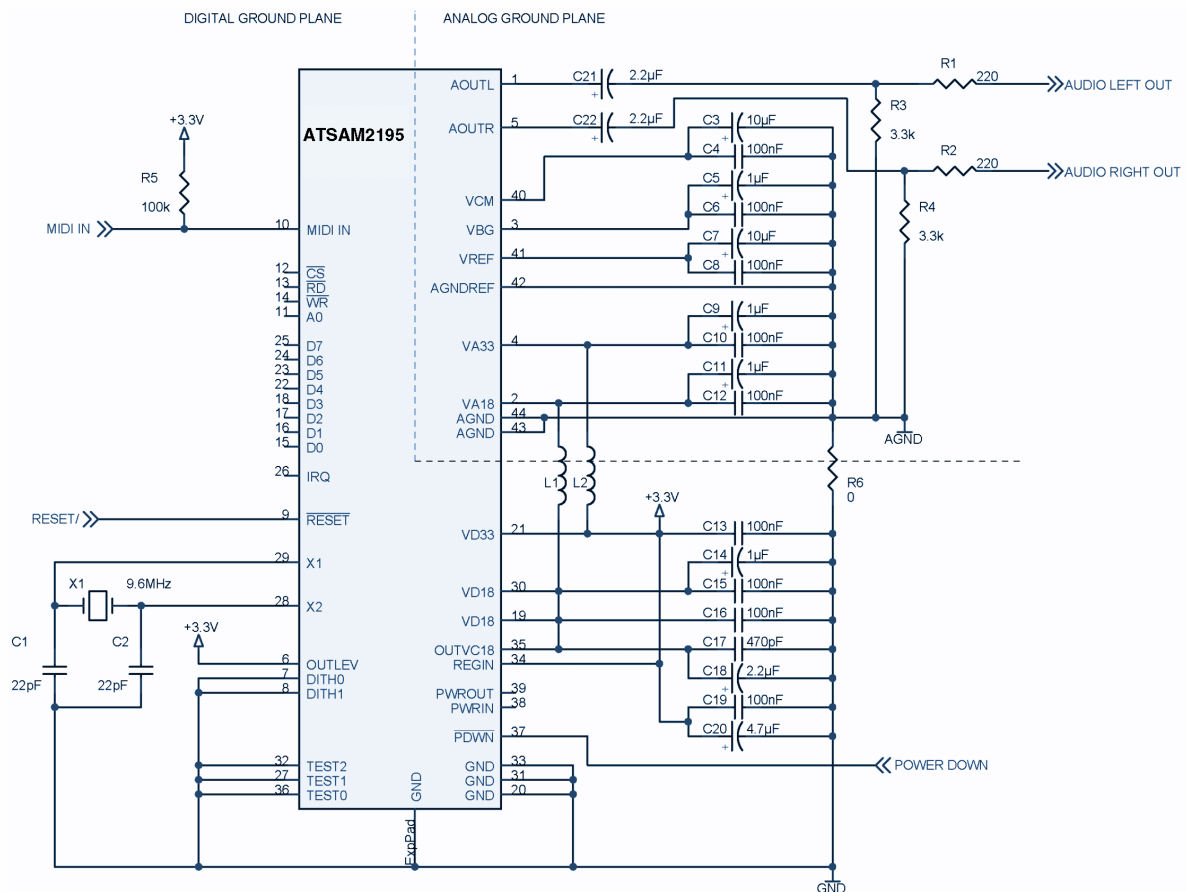


Figure 4. Schematic of example circuit in 3.3 V operation mode (taken from page 17 of ATSAM2195 data sheet)

Pins 1 and 5 (AOUTL and AOUTR respectively) were connected to a 3.5mm headphone jack for the user to easily connect their personal headphones or an external amplifier. One of these audio output lines from the synthesizer were also connected to a simple single stage audio amplifier which powered the original internal speakers that came with the hacked music keyboard. A rough schematic is shown of the amplifier topology used since the component values are not curtail for this application and only produce about 4dB amplitude gain anyway. Much more detail can be found by searching the internet for “opamp amplifier”. This simple amp proved adequate to drive the two internal 4ohm speakers in parallel.

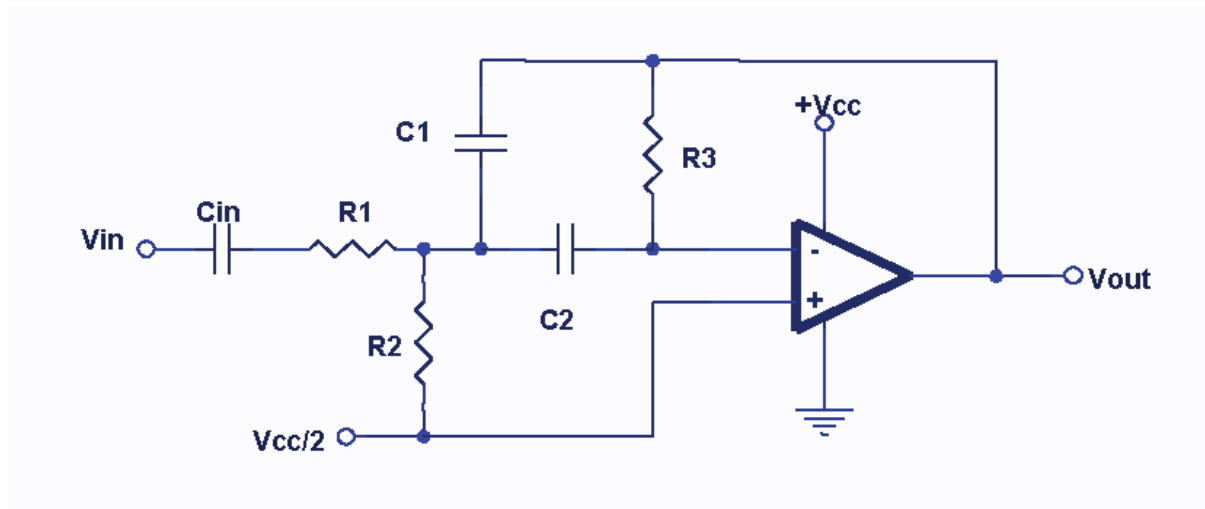


Figure 5. Schematic for multiple-feedback band-pass amplifier (Texas Instruments)

For user to interface with the device settings and parameters an Arduino microcontroller was used in conjunction with a Crystalfontz LCD display/buttons module. The user can select parameters which the Arduino then sends to the Keyboard Controller via the I<sup>2</sup>C communication protocol and the UART pins available on both microcontrollers. The Arduino to display/buttons module communication is accomplished through TTL serial UART interfaces available on both devices. Power to the entire system is conveniently provided by the Arduino microcontroller board. A variant of the Arduino was used called the Seeeduino. The Seeeduino is completely compatible with the Arduino development environment and hardware peripherals (Shields). The added advantage the Seeeduino provided is its ability to operate on 3.3V logic. This makes it easy interface with external 3.3V devices. The 3.3V devices in this keyboard system consist of the PIC18 Microcontroller, ATMEL synthesizer chip, LCD/Buttons module, and the Seeeduino. Since the Seeeduino has its own on-board 5V and 3.3V dedicated voltage regulators, powering the rest of the circuit was convenient and organized.

One of the challenging aspects of this project was interfacing with the ATMEL synthesizer chip. The chip is very powerful and provides much of the keyboard systems functionality and it is relatively inexpensive. However, it only comes in a QFN package which makes it a surface-mount component only. Generally requires expensive equipment to reflow solder and fabrication of a custom PCB. These two requirements were far beyond the scope and time constraint of this project, so alternatives were found. First, a QFN breakout board kit was purchased from [protovantage.com](http://protovantage.com). This provided all the necessary supplies needed to reflow solder the synthesizer chip onto a breadboard compatible breakout board. The other hurdle was how to reflow the solder without access to an expensive reflow oven. A \$30 electric skillet was purchased from Target and proved to work very well. Simply stencil the solder paste, place the synthesizer chip on the pasted pads, and stick it in the skillet until the solder paste turns shiny and has flowed to cover the pads completely. More information on this process can be found by searching the internet for "skillet reflow solder".

## Software Implementation

### Keyboard Controller

Messages are sent to the ATSAM2195 chip in parallel mode for this application. When the keyboard controller boots up, the ATSAM2195 chip is initialized in parallel mode by sending the value 0x3F as a control message (A0 = 1). Our boot method continually sends this value to the chip, until the IRQ pin on the synthesizer chip goes high, indicating a byte is available on the synthesizer chip for read. A byte is then read from the synthesizer chip as a data message (A0 = 0). If the returned byte equals 0xFE, then the synthesizer chip was successfully initialized in parallel mode. Also during the boot procedure for the keyboard controller, the pitch bend optoelectronic sensor is calibrated by taking one distance measurement. When pitch bend is enabled on the keyboard, pitch bend messages are continually sent to the synthesizer chip based on distance measurements taken from the sensor.

The keyboard controller driver loops through each column pin on the key matrix asserting it high. The row pins are then read into the microcontroller. After the states of all buttons on the keyboard are read into the system, the button states are compared to previous button states. The keyboard controller then checks for changes to the buttons, and sends out MIDI NOTE ON/NOTE OFF messages to the synthesizer chip as data messages (A0 = 0). All NOTE ON/NOTE OFF messages are also transmitted on the PIC TX port at 31250 baud 8-bit no parity, for use with other serial MIDI devices. This feature makes our project a suitable keyboard interface compatible with Russell, Daniel, and Torin's synthesizer project.

Table 1. MIDI messages used in the keyboard controller

Type	Hex code	Description
Control	3F	Puts the ATSAM2195 chip in Parallel message mode.
Data	B0 7B 00	Clear all key events.
Data	B0 65 00 64 00 06 xx	Change the pitch bend sensitivity to xx.
Data	E0 00 xx	Change the pitch bend value to xx.
Data	90 xx yy	NOTE ON/NOTE OFF command. The value xx represents the key index, and yy represents the key velocity. Setting yy = 00 represents a NOTE OFF event.
Data	C0 xx	Change the instrument. The value xx represents the instrument number.

### Navigation menu

The navigation menu uses a tree-like data structure for navigating various settings. The data type used for the navigation menu is defined as follows:

```
typedef struct menu_t {
    unsigned char    menu_cnt;
    unsigned char*   setting_val_ptr;
    Boolean          inverse_order;
    setting_t        setting_idx;
    const prog_char** menu_items;
    const prog_char* label;
    struct menu_t*   parent;
    struct menu_t**  children;
} menu;
```

The purposes of the various struct elements are as follows:

Menu type element	Description
menu_cnt	Indicates how many menu items are assigned to the menu struct
setting_val_ptr	Stores the location where the setting value is stored in memory
inverse_order	If true, the function of the UP and DOWN buttons is swapped
setting_idx	Index describing the specific setting
menu_items	An array of strings stored in program memory
label	A string stored in program memory which is displayed before each menu item
parent	A pointer to the parent node for the menu struct
children	An array of pointers to children menu structs

During initialization of the Seeeduino module, the menu tree structure is set up. Two global menu pointers point to the root menu node (Main Menu) and the current menu node being navigated to. The menu tree is navigated using the keypad on the Crystalfontz CFA-533 LCD.

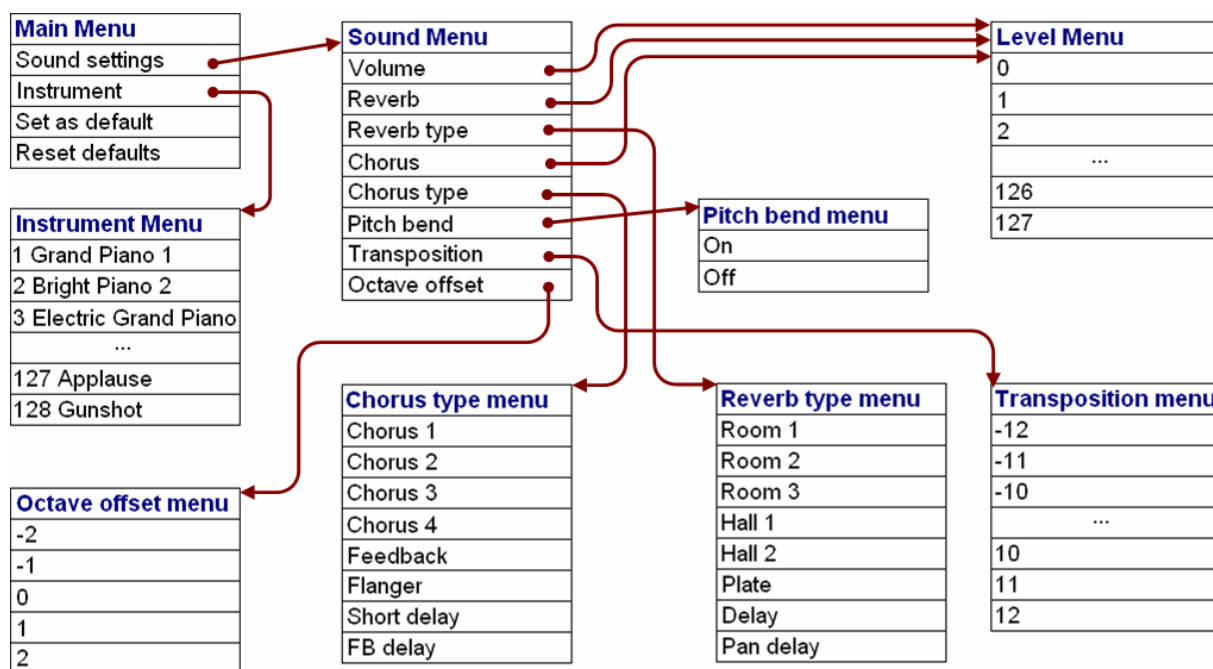


Figure 6. Menu tree structure

### LCD driver

The Crystalfontz CFA-533 LCD / keypad unit is used for displaying and navigating through the menu. The Seeeduino sends and receives information in “packets” from and to the LCD using serial 19200 baud RS-232. Each packet contains the following information:

- 1-byte Command
- 1-byte Data Length (0 – 22)
- Data
- CRC checksum of all previous bytes in the packet

The following LCD commands were used in our project:

Sent package (hex)	Returned package (hex)	Description
07 10 AA...	47 00	Print 16 characters (AA...) to first row of LCD
08 10 AA...	48 00	Print 16 characters (AA...) to second row of LCD
17 02 00 00	57 00	Disable automatic button reporting.
18 00	58 03 AA BB CC	Read keypad, polled mode. AA = Keys currently held down. BB = Keys pressed since last poll. CC = Keys released since last poll.

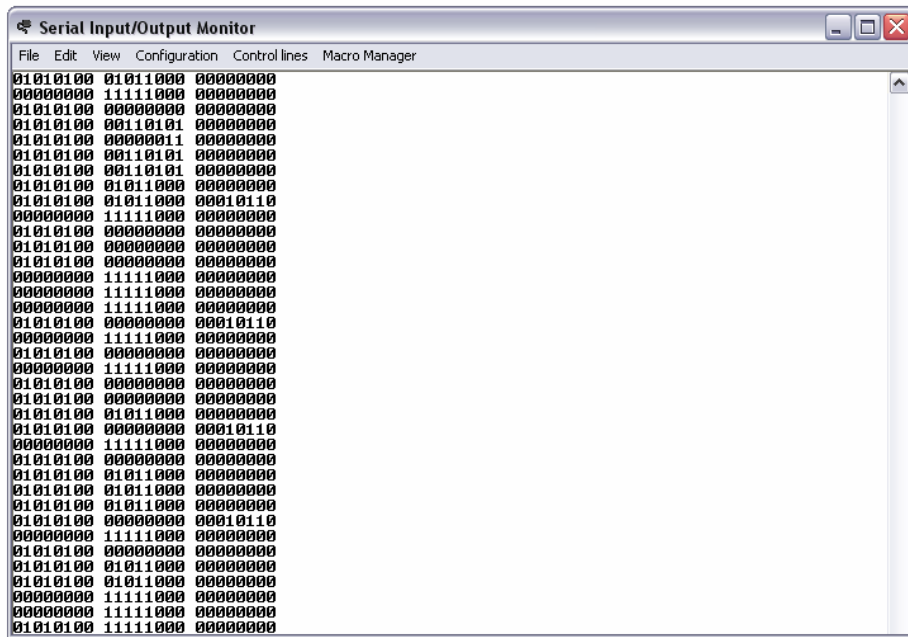
Commands sent to the LCD range from 0x00 to 0x3F. Return packets from LCD range from 0x40 to 0x7F for normal response, and from 0xC0 to 0xFF for error. Hex codes from 0x80 to 0xBF are response codes for reports which are not indirect responses to commands sent to the screen.

We found that even though an outgoing command is sent, the probability the command would succeed was around 30%. We wrote our LCD driver to continue attempting to send a packet until it receives the response code for the command sent (command | 0x40), which ensures that all commands are executed.

During initialization, the LCD displays a startup screen, and then displays the main menu. The user then navigates the menu using the buttons on the LCD keypad. The keypad buttons perform the following functions:

Keypad button	Function
Up	Navigate up or down the menu items for the current menu node, depending on the value of reverse_order
Down	Navigate down or up the menu items for the current menu node, depending on the value of reverse_order
Left	Navigate to the current menu node's parent
Right	Navigate to the child of the current menu node
Okay	Execute current command / set current setting / navigate to child of the current menu node
Cancel	Navigate to the root node (Main Menu)

The key data is polled once every millisecond. We found that the polled key data coming from the keypad was full of glitches, requiring some sort of debouncing technique to make the data usable. Shows an example of the key data returned from the LCD when no buttons are pressed down.



Buttons Buttons Buttons  
 pressed released held

Figure 7. Example of bad key data from LCD unit when no actual buttons are pressed down

The data for the buttons held was the cleanest and most usable. We implemented a debouncing technique which made sure that any errant 1's and 0's are filtered out from generating button events. To do this, we count all the read button states during a period of 15 keypad polls. We determined that if the count was greater than 3, a key press event would be generated, after which no new key events can be generated for 1 second. If the count was greater than 10, a key hold event would be generated, after which no new key events could be generated for 1 millisecond unless the button is released and pushed again. The resulting functionality allows you to push the button down and navigate through the menus slowly, or hold the key down and scroll faster through the menus.

When the user selects the Enter key, the LCD driver checks the setting\_val\_ptr field for the current menu node. If the pointer is anything other than null (0), it changes the setting for the current menu node to the current menu item index, displays a success message, transmits the setting over to the Keyboard Controller over the I<sup>2</sup>C bus, and exits. Menus that change settings include the Level Menu, Pitch Bend Menu, Transposition Menu, and others (see Figure 6). If the setting\_val\_ptr field for the current menu node is a null pointer, the current menu node does not modify a setting. In that case, the LCD driver then checks if there is a child node for the current menu item. If not, then the selection is processed as a command. The menu items "Set as default" and "Reset defaults" are two examples of commands in our navigational menu (see Figure 6). If there is a child node for the current menu item, the LCD driver simply navigates to the child node, same as the Right button on the keypad.

### I<sup>2</sup>C driver

When a setting is modified through the navigational menu, the setting index and value are transmitted over the I<sup>2</sup>C bus to the keyboard controller. After the setting and value are transmitted, the driver looks for a success message from the Keyboard Controller, signaling that the setting was successfully read.

The standard Wire.h library which is included in the Arduino environment was not generating necessary interrupts in the PIC controller. To fix this, we developed our own I<sup>2</sup>C drivers, which bit-banged the SCL and SDA outputs.

## Testing and results

### Test Specification

Overall testing of the system includes testing overall functionality, as well as testing several system restraints. The functionality testing includes smooth operation of the LCD navigational menu, verifying that settings made in the menu change the sound output, and verifying that all necessary key events are generated from the keyboard.

1. Maximum key latency: 0.8 milliseconds
2. Maximum I<sup>2</sup>C setting transfer time: 0.5 seconds
3. Maximum power consumption by Keyboard Controller and Synthesizer: 495 milliwatts
4. Maximum power consumption by LCD screen with backlight on, audio amplifier, and speakers: 750 milliwatts
5. Maximum power consumption for entire system: 2.5 watts

### Test plan

Overall functionality of our circuit is verified by operation of the keyboard and navigation menu. The key latency is measured by measuring the total time to poll all key positions, as well as sending MIDI events for each of the 49 keys to the synthesizer chip. This is measured by altering the keyboard controller code so it runs the keyboard driver in a continuous loop and outputs all 49 key events, then probing one of the column pins on the PIC processor on the oscilloscope. The period of the frequency measured is the worst-case key latency for the keyboard.

The minimum transfer time of settings from the Seeeduino to the PIC controller is measured in a similar fashion. The transfer time was measured from the transmission of the first setting from the Seeeduino until the time the PIC transmits back the success code.

Power consumption is measured for each individual unit on the keyboard using a digital multimeter. This is divided into four distinct measurements: 1) Amplifier and speaker, 2) Keyboard controller and synthesizer chip, 3) LCD display, and 4) Seeeduino.

### Test Cases

To measure key latency, we tested for the worst possible case of all 49 keys changing value at the same time, which was our theoretical (although perhaps physically impossible) worst-case delay for the keyboard controller.

The I<sup>2</sup>C transfer time includes sending two bytes of data, and receiving the success message back.

Power consumption by the amplifier is measured by pressing several keys at once at maximum volume on a fairly loud instrument. Power consumption by the LCD module is measured with the backlight on.

## Results

### Key latency

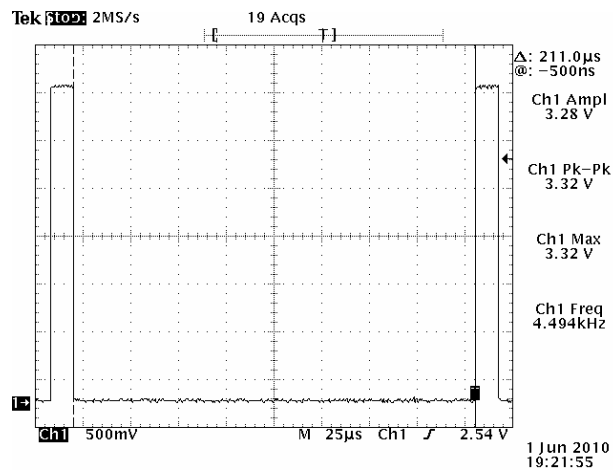


Figure 8. The total worst-case key latency was measured to be 211 μs, which far exceeds our project specification of 800 μs minimum.

### I<sup>2</sup>C setting transfer latency

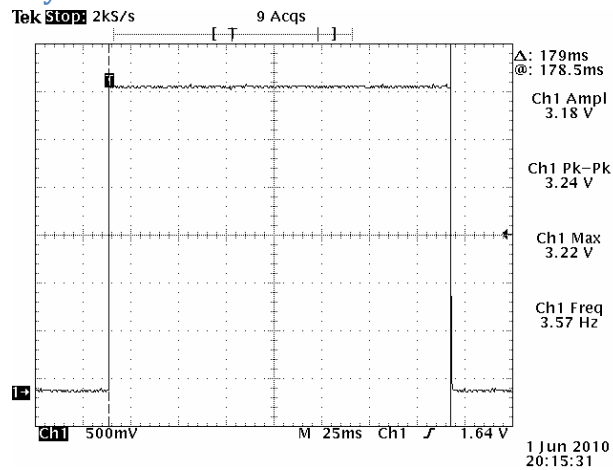


Figure 9. The total worst-case I<sup>2</sup>C setting transfer latency was measured to be 179 ms.

### Power consumption

Table 2. Measured power consumption of various circuit components

Circuit component	Peak power consumption
Keyboard Controller (PIC) / ATMEL synthesizer chip	0.216 watts
Speakers / audio amplifier	0.200 watts
LCD display with backlight	0.505 watts
Seeduino board	0.279 watts
<b>Total power consumption</b>	<b>1.20 watts</b>

As shown in Table 2, our circuit met the specification of a maximum power consumption of 2.5 watts.

## **Functionality**

After playing the keyboard, we were able to verify that the keyboard generated appropriate MIDI messages based off key events. Changed settings successfully were applied, and the navigation menu worked as specified.

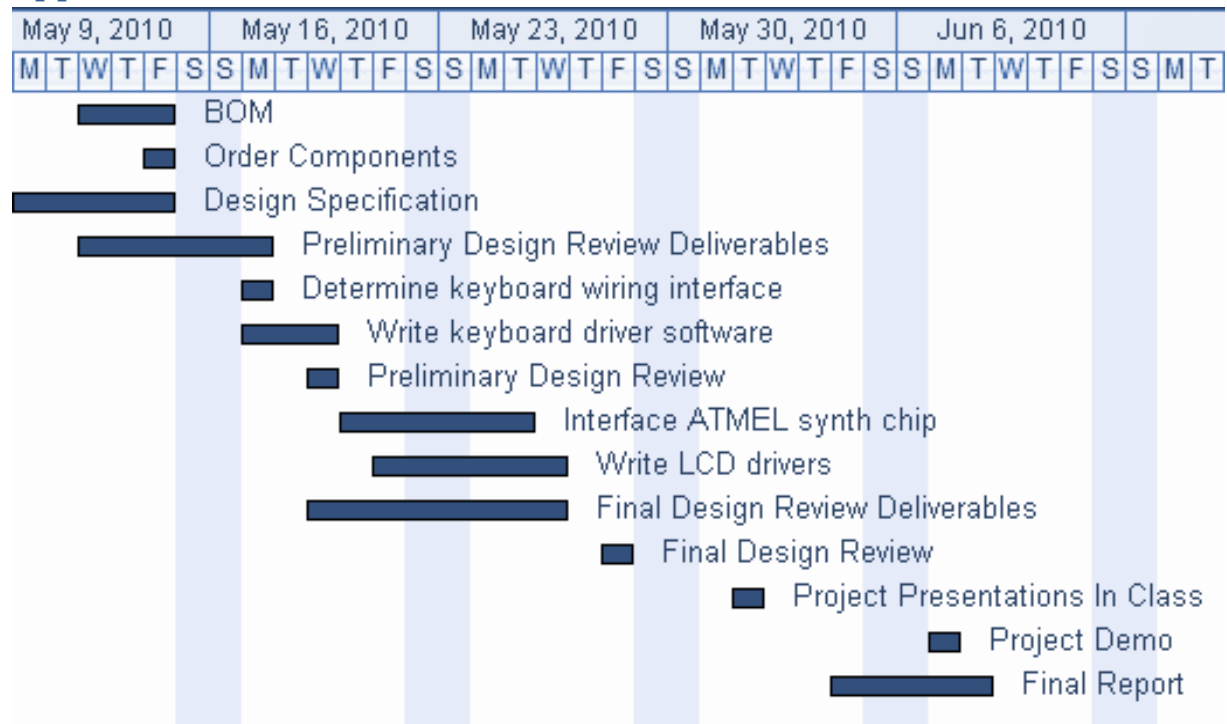
## **Conclusion**

Using an inexpensive used music keyboard as a basis, a more versatile keyboard was created. Using primarily common and accessible components allows for this project to be used as a foundation for other digitally synthesized instruments to be created. Keeping the design fundamentally simple and easy to replicate was achieved by utilizing serial communication busses as much as possible. An example of using serial communication where parallel would have been more common for this type of project was the use of the LCD/Buttons module. Furthermore, use of the powerful ATMEL synthesizer chip provides the user with extensive customization and creative potential.

## Appendix 1. Bill of materials

Count	Part Number	Description	Cost ea. (U.S. Dollars)
1	ATSAM2195	Atmel synthesizer chip	6.00
1	PIC18F4550	PIC microcontroller	6.00
1	ATMEGA328P	Atmel microcontroller	5.00
1	CFA533-YYH-KC	Crystalfontz 2×16 character RS232 LCD with 6-buttons	50.00
1	MD-500, Radio Shack	Dismantled keyboard	20.00
1		Audio jack	1.00
1		9.6 MHz crystal	2.40
1		20 MHz oscillator	2.40
1		470 pF ceramic capacitor	0.20
9		100 nF ceramic capacitor	0.20
4		1 μF electrolytic capacitor	0.30
3		2.2 μF electrolytic capacitor	0.30
1		4.7 μF electrolytic capacitor	0.30
2		10 μF electrolytic capacitor	0.30
2		10 Ω resistor, 5%, ¼ watt	0.10
1		100 kΩ resistor, 5%, ¼ watt	0.10
<b>Total:</b>			<b>98.20</b>

## Appendix 2. Schedule



## References

ATSAM2195 Low-power Single Chip Synthesizer with Effects, ATMEL, May 2007.

[http://www.atmel.com/dyn/products/product\\_card.asp?part\\_id=4142](http://www.atmel.com/dyn/products/product_card.asp?part_id=4142)

ATSAM2195 User Guide, ATMEL, June 2007.

[http://www.atmel.com/dyn/products/product\\_card.asp?part\\_id=4142](http://www.atmel.com/dyn/products/product_card.asp?part_id=4142)

Carter, Bruce, *A Single-Supply Op-Amp Circuit Collection*, Texas Instruments, November 2000.

<http://www.ti.com/sc/docs/psheets/abstract/apps/sloa058.htm>

*Intelligent Serial LCD Module Specifications*, Crystalfontz, Revision 1.1, June 2009.

<http://www.crystalfontz.com/product/CFA533-YYH-KS.html>