**Abstract**

This is the final project report for EE4144, Introduction to Embedded Systems with Professor Dr. Prashanth Krishnamurthy. Please contact Jim Dong if before using any parts of this code.

The goal of this project was to create a sound synthesizer using a STM32F4 Discovery Board. The STM32F4 board would take an input from a terminal emulator, in this case, we used Tera Term, and output a specific frequency. The concept is to create a MIDI piano keyboard where a key on a standard computer keyboard would map to a specific frequency of a key on a piano. Outputting sound needs to be done in real time as this would be expected of an instrument. The amount of time the sound outputted should also be faithful to the amount of time a key on a piano outputs its sound. Of course, simply outputting a single frequency would not sound like like a frequency played on a piano because a musical pitch consists of several different frequencies besides the fundamental frequency. To simulate musical pitch digitally would require complex transformations, which is beyond what the STM32F4 board is capable of and beyond the scope of the project, and as a result, we will only consider outputting the fundamental frequency. After extensive testing of the STM32F4 project, it has been concluded that the project was successful in meeting its stated goals.

**Introduction**

This project serves as a way to apply the concepts of an embedded system using things learned from lectures and laboratory experiments in EE4144, Introduction to Embedded Systems.

It consists of using a standard US computer keyboard where the user can input characters from the keyboard into a terminal emulator. The terminal emulator connects to a serial port which allows a byte of data from the user to be sent to that port. The STM32F4 board will read a value from the serial port and calculate a digital value to be sent to the on-board CS43L22 Digital-to-Analog converter (DAC). This chip converts the digital value into the corresponding analog value to be output from the speaker. The STM32F4 board provides a 3.5mm audio jack to connect a speaker or headphone.

**Description**

**Hardware Components**

Three interfaces were used on the STM32F4 Discovery Board, the mini-USB, the micro-USB and the audio jack. The mini-USB provides power to the board and allows writing of a program to the board’s flash memory using KEIL uVision software. The micro-USB allows a serial port connection between the board and a computer. The audio jack allows a speaker or headphone to be connected to the board. The onboard Digital-to-Audio chip, Cirrus Logic CS43L22, controls this audio interface.

**Software Libraries**

There were two software libraries utilized by this project to create a serial port connection between the STM32F board and a computer and to output a sound by controlling the Cirrus Logic CS43L22 chip. The USB VCP for STM32F4xx was written by Tilen Majerle, and provides a way to read and write a byte to a serial port. It also provides is a way to check if the serial port is connected. The second library utilized was written by Andreas Finkelmeyer and allows us to send a single 16-bit audio sample to the speaker.

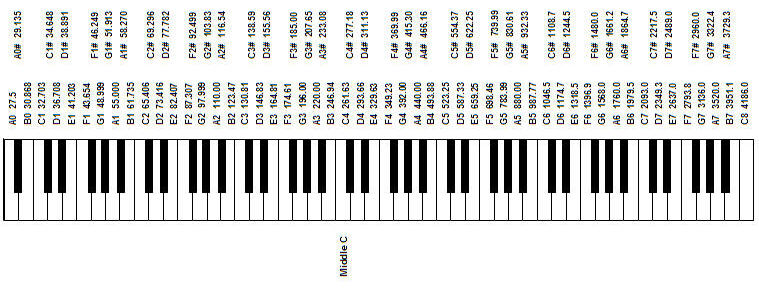
For the computer to interface with the serial port, the STM32 virtual COM port driver would need to be installed which provides virtualization of a communication port as if it were a real port. This allows a program like PuTTY or Tera Term to connect to the serial COM port, which will allow the user to send a byte from a keyboard to the serial COM port using the terminal.

**Project Software**

There are three main functions that provide most of the functionality to the program. The function, read\_serial\_usb\_byte(&k), allows the program to read a byte from the serial port and store this byte into the memory that is pointed to by &k. The value of k would be generated by the keyboard input from the user. Each character on a computer keyboard corresponds to a different byte value. Since there are eight bits in a byte, a maximum of 256 different characters is possible. A single audio sample can be calculated by: audio\_sample = (int16\_t) (volume \* arm\_sin\_f32(audio\_freq\*ts)), which calculates a single 16 bit sample that is able to be outputted through the speaker. The volume variable corresponds the amplitude of the generated sine wave. The audio\_freq variable would be the frequency at which the sound would be outputted by the speaker. This value is multiplied by, ts, time in seconds and the sin of the product would be calculated. This value would then be converted into a 16-bit number and set as the audio\_sample. The function, send\_to\_speaker(audio\_sample), allows sending a 16-bit value to the speaker.

There are five file scope variables that the program uses outside of the main program loop. The audio\_freq variable will be the frequency at which the speaker will be outputting. The value of this will change depending another value, k, which will be described below. The volume variable is arbitrarily set to 700 as this was found to be a good volume for headphone use. The variable k is 8-bits and holds the current key value that was read from the serial port. The variable, hold\_freq\_time, is the amount of time a frequency should be outputted. The rationale behind this is that lower frequencies on a piano typically are longer in duration than higher frequencies with respect to time. The value of this will be set depending on the value for k. The variable, time\_last\_freq\_change, records the last time in seconds that the frequency was changed.. This will be useful in finding the total duration current frequency has outputted so far. Since the program should run continuously, all the code after the global variables is encapsulated with an unconditional while loop, while(1). The variable, ts, contains the time in seconds. mmsTicks is incremented every 0.1ms so by dividing by 10,000, we obtain time in seconds. A byte is read from the serial port using the function, read\_serial\_usb\_byte(&k), and will return what was read by writing it to the variable, k. We then switch on k depending on forty different cases. Theses cases are ASCII values corresponding to the user's input from the keyboard. The grouping of these cases correspond to the harmonic series on a piano. It begins with the fifth harmonic, 440Hz, because it was found early on that the harmonics lower than the fifth would not produce any sound through the speaker. It ends in the eighth harmonic with the frequency 4186Hz.

In each case, the byte will be written to the serial port using write\_serial\_usb\_byte(k). This is unnecessary in producing sound but is helpful for to the user since this byte will appear in the terminal so he/she will know what key was pressed. It also sets a predefined frequency to the variable audio\_freq which corresponds the piano key’s frequency. The frequencies of each piano key were found online. The full range of the frequencies is listed below courtesy of www.sciencebuddies.org :



In addition, the time for each key to output its time is also predefined and the variable, hold\_freq\_time, will be set accordingly. When a key is pressed, the time at which the key was pressed is recorded. This time information is set in the variable, time\_last\_freq\_change which will set it to the current time.

After the switch statement, there is a if statement to ensure that the time that each frequency is outputted via the speaker does not exceed its defined output frequency time as specified by the hold\_freq\_time variable. The if loop is defined below:

if (ts > ( hold\_freq\_time + time\_last\_freq\_change)) {

audio\_freq =0;

time\_last\_freq\_change = ts;

}

It compares the current time, ts, with the key’s hold\_freq\_time added with what time the key was set. If it returns false, then the frequency will continue to be outputted through the speaker. If it returns true, that means that enough time has elapsed and the frequency should stop being outputted through the speaker. The speaker is turned off by setting the audio\_freq to zero. The variable, time\_last\_freq\_change, is updated to the current time since the frequency has been changed to zero. Finally, the audio\_sample is calculated using: audio\_sample = (int16\_t) (volume \* arm\_sin\_f32(audio\_freq\*ts)), and is outputted to the speaker using, send\_to\_speaker(audio\_sample). These two are library functions and were briefly described above in the section, Software Libraries.

**Conclusion**

The project was successful in creating a sound synthesizer that resembled the functionality of a piano. The project incorporated two software libraries that provided methods for using the speaker and the serial port connection. It provides the user with the ability of pressing a key on the keyboard and having a predefined frequency play through a speaker. Communication from the computer to the board was made possible through the use of a terminal emulator, which allows the user to send an ASCII 8-bit key using the keyboard to the serial port. The program loaded onto the flash memory of the STM32F4 board was successfully able to read this byte and chose the correct defined frequency and output it via a speaker. The project accurately maps four octaves of a piano to standard computer keyboard and allows the user to use the keyboard to generate a sound, as you would use the instrument.

**Appendix**

**volatile** **uint32\_t** mmsTicks = 0; */\* counts 0.1ms timeTicks \*/*  
  
*// SysTick Handler (Interrupt Service Routine for the System Tick interrupt)*  
**void** SysTick\_Handler(**void**){  
 mmsTicks++;  
}  
  
*// initialize the system tick*   
**void** init\_systick(**void**){  
 SystemCoreClockUpdate(); */\* Get Core Clock Frequency \*/*  
 **if** (SysTick\_Config(SystemCoreClock / 10000)) { */\* SysTick 0.1 msec interrupts \*/*  
 **while** (1); */\* Capture error \*/*  
 }  
}  
  
  
**int32\_t** main(){  
 SystemInit();  
 init\_systick(); *//initializes the system tick handler*  
 init\_serial\_port\_usb(); *//initializes the usb serial port*  
 init\_speaker(); *//initializes the speaker*  
   
 **int16\_t** audio\_sample; *//one audio sample is 16 bits which is what the send\_to\_speaker() function uses as an argument*  
 **float** audio\_freq = 0; *//the frequency of the audio sample*  
 **int** volume = 700; *//volume is how loud the output volume will be, it is the amplitude of the sin wave*   
 **uint8\_t** k; *//ascii key character value, used to hold the current key press*  
 **float** hold\_freq\_time = 0; *//amount of time to output a frequency for a single key, k*  
 **float** time\_last\_freq\_change = 0; *//holds the most recent time when the key, k was changed*  
   
 **while** (1) {

*//time in seconds, mmsTicks increments every 0.1ms so dividing by 10,000 will give you time in seconds*  
 **float** ts = mmsTicks / 10000.0;

*// this function reads a key from the serial port and will return the value of*   
read\_serial\_usb\_byte(&k);

*//the key by writing it in the memory pointed to by &k*   
 *//depending on which ascii key, k, was pressed, it writes the k value back to the serial usb port, sets the corresponding*   
*//frequency of that key, sets how long the frequency lasts, and sets the current time at which the key was pressed*  
**switch** (k) {  
   
*/5th harmonic first regular row on a standard US keyboard*   
 **case** **'1'**: write\_serial\_usb\_byte(k); audio\_freq = 440; hold\_freq\_time = 0.360 ;time\_last\_freq\_change = ts; **break**; *//A4*  
 **case** **'2'**: write\_serial\_usb\_byte(k); audio\_freq = 466; hold\_freq\_time = 0.357 ;time\_last\_freq\_change = ts; **break**; *//A4#*   
 **case** **'3'**: write\_serial\_usb\_byte(k); audio\_freq = 494; hold\_freq\_time = 0.353 ;time\_last\_freq\_change = ts; **break**; *//B4*  
 **case** **'4'**: write\_serial\_usb\_byte(k); audio\_freq = 523; hold\_freq\_time = 0.350 ;time\_last\_freq\_change = ts; **break**; *//C4*  
 **case** **'5'**: write\_serial\_usb\_byte(k); audio\_freq = 554; hold\_freq\_time = 0.347 ;time\_last\_freq\_change = ts; **break**; *//C5#*  
 **case** **'6'**: write\_serial\_usb\_byte(k); audio\_freq = 587; hold\_freq\_time = 0.344 ;time\_last\_freq\_change = ts; **break**; *//D5*  
 **case** **'7'**: write\_serial\_usb\_byte(k); audio\_freq = 622; hold\_freq\_time = 0.341 ;time\_last\_freq\_change = ts; **break**; *//D5#*  
 **case** **'8'**: write\_serial\_usb\_byte(k); audio\_freq = 659; hold\_freq\_time = 0.338 ;time\_last\_freq\_change = ts; **break**; *//E5*  
 **case** **'9'**: write\_serial\_usb\_byte(k); audio\_freq = 698; hold\_freq\_time = 0.334 ;time\_last\_freq\_change = ts; **break**; *//F5*  
 **case** **'0'**: write\_serial\_usb\_byte(k); audio\_freq = 740; hold\_freq\_time = 0.330 ;time\_last\_freq\_change = ts; **break**; *//F5#*  
 **case** **'-'**: write\_serial\_usb\_byte(k); audio\_freq = 784; hold\_freq\_time = 0.327 ;time\_last\_freq\_change = ts; **break**; *//G5*  
 **case** **'='**: write\_serial\_usb\_byte(k); audio\_freq = 831; hold\_freq\_time = 0.323 ;time\_last\_freq\_change = ts; **break**; *//G5#*  
   
*//6th harmonic second regular row on a standard US keyboard*   
 **case** **'q'**: write\_serial\_usb\_byte(k); audio\_freq = 880; hold\_freq\_time = 0.320 ;time\_last\_freq\_change = ts; **break**; *//A5*  
 **case** **'w'**: write\_serial\_usb\_byte(k); audio\_freq = 932; hold\_freq\_time = 0.318 ;time\_last\_freq\_change = ts; **break**; *//A5#*  
 **case** **'e'**: write\_serial\_usb\_byte(k); audio\_freq = 988; hold\_freq\_time = 0.316 ;time\_last\_freq\_change = ts; **break**; *//B5*  
 **case** **'r'**: write\_serial\_usb\_byte(k); audio\_freq = 1047; hold\_freq\_time = 0.314 ;time\_last\_freq\_change = ts; **break**; *//C5*   
 **case** **'t'**: write\_serial\_usb\_byte(k); audio\_freq = 1109; hold\_freq\_time = 0.312 ;time\_last\_freq\_change = ts; **break**; *//C6#*  
 **case** **'y'**: write\_serial\_usb\_byte(k); audio\_freq = 1175; hold\_freq\_time = 0.310 ;time\_last\_freq\_change = ts; **break**; *//D6*  
 **ase** **'u'**: write\_serial\_usb\_byte(k); audio\_freq = 1245; hold\_freq\_time = 0.308 ;time\_last\_freq\_change = ts; **break**; *//D6#*  
 **case** **'i'**: write\_serial\_usb\_byte(k); audio\_freq = 1319; hold\_freq\_time = 0.306 ;time\_last\_freq\_change = ts; **break**; *//E6*  
 **case** **'o'**: write\_serial\_usb\_byte(k); audio\_freq = 1397; hold\_freq\_time = 0.304 ;time\_last\_freq\_change = ts; **break**; *//F6*  
 **case** **'p'**: write\_serial\_usb\_byte(k); audio\_freq = 1480; hold\_freq\_time = 0.302 ;time\_last\_freq\_change = ts; **break**; *//F6#*  
 **case** **'['**: write\_serial\_usb\_byte(k); audio\_freq = 1568; hold\_freq\_time = 0.300 ;time\_last\_freq\_change = ts; **break**; *//G6*  
 **case** **']'**: write\_serial\_usb\_byte(k); audio\_freq = 1661; hold\_freq\_time = 0.298 ;time\_last\_freq\_change = ts; **break**; *//G6#*  
   
*//7th harmonic shift + first regular row on a standard US keyboard*   
 **case** **'!'**: write\_serial\_usb\_byte(k); audio\_freq = 1760; hold\_freq\_time = 0.296 ;time\_last\_freq\_change = ts; **break**; *//A6*  
 **case** **'@'**: write\_serial\_usb\_byte(k); audio\_freq = 1865; hold\_freq\_time = 0.294 ;time\_last\_freq\_change = ts; **break**; *//A6#*  
 **case** **'#'**: write\_serial\_usb\_byte(k); audio\_freq = 1980; hold\_freq\_time = 0.292 ;time\_last\_freq\_change = ts; **break**; *//B6*  
 **case** **'$'**: write\_serial\_usb\_byte(k); audio\_freq = 2093; hold\_freq\_time = 0.290 ;time\_last\_freq\_change = ts; **break**; *//C7*  
 **case** **'%'**: write\_serial\_usb\_byte(k); audio\_freq = 2218; hold\_freq\_time = 0.288 ;time\_last\_freq\_change = ts; **break**; *//C7#*  
 **case** **'^'**: write\_serial\_usb\_byte(k); audio\_freq = 2349; hold\_freq\_time = 0.286 ;time\_last\_freq\_change = ts; **break**; *//D7*  
 **case** **'&'**: write\_serial\_usb\_byte(k); audio\_freq = 2489; hold\_freq\_time = 0.284 ;time\_last\_freq\_change = ts; **break**; *//D7#*  
 **case** **'\*'**: write\_serial\_usb\_byte(k); audio\_freq = 2637; hold\_freq\_time = 0.282 ;time\_last\_freq\_change = ts; **break**; *//E7*  
 **case** **'('**: write\_serial\_usb\_byte(k); audio\_freq = 2794; hold\_freq\_time = 0.280 ;time\_last\_freq\_change = ts; **break**; *//F7*  
 **case** **')'**: write\_serial\_usb\_byte(k); audio\_freq = 2960; hold\_freq\_time = 0.278 ;time\_last\_freq\_change = ts; **break**; *//F7#*  
 **case** **'\_'**: write\_serial\_usb\_byte(k); audio\_freq = 3136; hold\_freq\_time = 0.276 ;time\_last\_freq\_change = ts; **break**; *//G7*  
 **case** **'+'**: write\_serial\_usb\_byte(k); audio\_freq = 3322; hold\_freq\_time = 0.274 ;time\_last\_freq\_change = ts; **break**; *//G7#*  
   
*//8th harmonic shift + second regular row on a standard US keyboard*   
 **case** **'Q'**: write\_serial\_usb\_byte(k); audio\_freq = 3520; hold\_freq\_time = 0.272 ;time\_last\_freq\_change = ts; **break**; *//A7*  
 **case** **'W'**: write\_serial\_usb\_byte(k); audio\_freq = 3729; hold\_freq\_time = 0.270 ;time\_last\_freq\_change = ts; **break**; *//A7#*  
 **case** **'E'**: write\_serial\_usb\_byte(k); audio\_freq = 3951; hold\_freq\_time = 0.268 ;time\_last\_freq\_change = ts; **break**; *//B7*  
 **case** **'R'**: write\_serial\_usb\_byte(k); audio\_freq = 4186; hold\_freq\_time = 0.266 ;time\_last\_freq\_change = ts; **break**; *//C8*  
}   
   
*//this checks to see if the amount of time each key should output its frequency has elapsed*   
*//it compares the current time, ts, with how long each key should output its frequency + when the key was pressed*  
*//it 'turns off' the speaker by setting its frequency to 0*  
**if** (ts > ( hold\_freq\_time + time\_last\_freq\_change)) {   
 audio\_freq =0;   
 time\_last\_freq\_change = ts;   
}  
   
*//a sin function is used so that it outputs a signal in uniform intervals*  
audio\_sample = (**int16\_t**) (volume \* arm\_sin\_f32(audio\_freq\*ts)); *// calculate one sample for the speaker*  
*//the CMSIS arm\_sin\_f32 function from arm\_math.h is typically faster than sin() from math.h*  
send\_to\_speaker(audio\_sample); *//send one audio sample to the audio output*

}

}