Standalone Wireless Humming Music Synthesizer
(HumSynth)
March 23rd, 2018

Chad Carlson
Anna Shabrova

Advisor: Wayne Pilkington
Table of Contents:

Abstract.......................................................................................................................................................... 1
I. Background.................................................................................................................................................. 2
II. Introduction................................................................................................................................................ 3
III. Product Design Engineering Requirements.......................................................................................... 6
IV. System Design - Functional Decomposition (Level 1).......................................................................... 8
V. Subsystem Descriptions............................................................................................................................ 10
   Wireless Microphone.................................................................................................................................. 11
   Audio Processor........................................................................................................................................ 12
   Audio Interface.......................................................................................................................................... 14
   User Interface........................................................................................................................................... 15
   Power System........................................................................................................................................... 16
VI. Physical Construction and Integration.................................................................................................. 20
VII. Integrated System Tests and Results..................................................................................................... 26
VIII. Conclusions.......................................................................................................................................... 31
IX. Bibliography.......................................................................................................................................... 32
Appendices..................................................................................................................................................... 32
   A. Analysis of Senior Project Design
   B. Schedule of Major Tasks and Milestones
   C. Budget
   D. Program Listing
Tables

Table 1: Industry Competitors ........................................................................................................5
Table 2: Function and Performance Requirements ........................................................................6
Table 3: Critical System Parameter Selection ..............................................................................7
Table 4: System Design Performance Budgets ............................................................................9
Table 5: Subsystem Performance and Functional Requirements ...............................................10
Table 6: Comparison of Interpolated Values with Expected Values ........................................29
Table 7: Time Benchmarks versus Specification ......................................................................30
Table 8: Parts List and Budget .....................................................................................................34
Table 9: Components Costs for Buck Converter .......................................................................35

Figures

Figure 1: Early Moog Synthesizers ..............................................................................................2
Figure 2: Minimoog Synthesizer .................................................................................................3
Figure 3: Level 0 Block Diagram ...............................................................................................7
Figure 4: Level 1 Functional Decomposition ............................................................................8
Figure 5: Wireless Microphone Subsystem Diagram ..............................................................11
Figure 6: Processor Subsystem Input/Output Diagram ............................................................14
Figure 7: Audio Interface Subsystem Input/Output Diagram ....................................................15
Figure 8: Schematic for Buck Converter with 1.25V Output ..................................................17
Figure 9: Steady State Output Voltage for 1.25V Output Buck Converter ..............................18
Figure 10: Linear Gain FFT of 1.25V Output Buck Converter ................................................18
Figure 11: Buck Converter Design for a 5V Output ................................................................19
Figure 12: Simulated Output Voltage for the 5V Buck Converter Design .............................19
Figure 13: Top View of Enclosure Prototype ..........................................................................20
Figure 14: Angle View of Enclosure Prototype ......................................................................21
Figure 15: Top View of Bottom Section of Enclosure Prototype ............................................22
Figure 16: Angle View of Bottom Section of Enclosure Prototype
Figure 17: Front Angle of Enclosure Model
Figure 18: Back Angle of Enclosure Model
Figure 19: Angle View of Enclosure Bottom Model
Figure 20: Angle View of Enclosure Top Model
Figure 21: Close up of Finger Joint
Figure 22: WSNAC with 100Hz Tone Input
Figure 23: WSNAC with Chorus G# Major Input
Figure 24: Gantt Chart for Spring Quarter
**Abstract**

The HumSynth is a voice controlled music synthesizer. Unlike the commonly found voice modulators on the market, this product uses a vocal input in place of the traditional keyboard interface. This allows the instrument to serve as an intuitive learning tool for those not proficient in music theory. By using their voice, the consumer is able to get a feel for different scales and keys additionally bypassing the physical learning curve of learning a traditional instrument.

By the date of submission this project performs basic functions in recreating the pitch input but does not meet the full specs. The enclosure is in its initial stages of design with several iterations of prototypes needed before the final configuration is selected.
I. Background

Musical synthesizers have become increasingly prevalent in the performance and production of music since their advent in the 1960’s with early Moog synthesizers. These early models provided new sounds and a flexibility to create sound that was previously unheard, but they were also incredibly bulky, costly, and difficult to use. Synthesis techniques developed over the years from amplitude modulation to frequency, intermodular, and granular synthesis. Each new technique introduced versality in the end product, however they also came with a steep learning curve that requires deep knowledge of electronics and wave theory. The physical instrument tends to retain the keyboard interface, additionally requiring consumers to have the knowledge to play a piano. Other synthesizers are purely digital, existing as plug-ins on a Digital Audio Workstation (DAW) which then requires, at minimum, either a laptop or desktop computer and also peripherals such as an audio interface. While technological advancements in digital processing have vastly improved the performance, size, and cost of musical synthesizers, they remain prohibitively expensive for current or aspiring musicians and they carry a steep learning curve both musically and technologically.
II. Introduction

The HumSynth consists of a standalone base station and microphone pair that can detect the pitch sung into the microphone and play the same pitch back with a selectable synthesizer in real time. It has the option to play to exact pitch sung or to correct the pitch to the nearest real musical note. The performer controls the volume of the synthesizer naturally, with the volume of the instrument changing relative to the loudness of the vocalist. The synthesizer provides a “Detune” setting, allowing the performer to modify all pitches produced by a fixed percentage. It also provides two options to control the attack and decay of the instrument. It may either be controlled by preset envelopes chosen appropriately for each instrument, or by the vocalist’s own vocal shaping. The system’s base station provides standard 3.5 mm mono and 1/4” standard audio inputs, as well as a MIDI output for external reproduction.

The HumSynth maintains the versatility in output of the common keyboard or software based synthesizers, but its voice control completely eliminates the challenges present with the use of most synthesizers on the market. The voice controlled interface eliminates the musical learning curve, as well as the cost of purchasing a physical instrument to practice with. It allows anyone to play their selected synthesizer sound with their voice without any musical knowledge, providing backing to their vocal line or as an intuitive way to learn music theory. This separates the HumSynth from other voice synthesizers on the market as the HumSynth does not simply modulate the user’s voice, but uses it for control like the pick of a guitar. The incredible ease of use in comparison to other synthesizers or

Figure 2 Minimoog Synthesizer
musical instruments make it an ideal tool for the education of young musical students who would otherwise have difficulty playing a physical instrument. Moreover, physical instruments are often too expensive for use in an educational setting or even for a casual learner, further stifling efforts to spread musical education and the arts. The HumSynth is a standalone device that requires no additional physical or software components for use, providing a cheaper alternative to most other commercial synthesizers or musical instruments.

The HumSynth is targeted towards two main types of customer. The first is someone who would like to learn or play music, but who may be intimidated by learning to play a physical instrument or may find it too difficult. The HumSynth is an instrument that can completely eliminate the learning curve normally associated with learning music as it removes the physical skills and techniques required to learn most instruments, and has a simple and easy user interface. Its straightforward singing operation and pitch correction feature make it accessible to anyone regardless of their age or experience. This can be especially beneficial for teaching young children music, who may otherwise have difficulty playing a physical instrument.

The second customer is any general music hobbyist, who can’t afford most current synthesizer options or may want to add a unique instrument to their solo performance. The standalone nature of the product and the relatively low cost of $80 give it a large advantage over competitors, while maintaining a versatility of sound with its programmability. Furthermore, its basic use doesn’t require any understanding of audio engineering, digital signal processing, or technical synthesis techniques such as additive, subtractive, modular, granular, etc. Instead, the HumSynth incorporates an organized and simple user interface.
Table 1: Industry Competitors

<table>
<thead>
<tr>
<th>Brand</th>
<th>Description</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>THE MOUTH by Tim Exile and Native Instruments</td>
<td>“THE MOUTH works by detecting the pitch of an incoming audio signal which is then auto-tuned to a selected musical scale or notes from a MIDI device... The auto-tuned signal is used to trigger the synthesizer, which in conjunction with the gate parameters adds additional melody and harmony” (&quot;THE MOUTH&quot;)</td>
<td>$69.00</td>
</tr>
<tr>
<td>Virta by Madrona Labs</td>
<td>A DAW based plug in, “Virta is a patchable toolbox for turning your voice or other instrument into wild new synth sounds” (Jones)</td>
<td>$89.00</td>
</tr>
<tr>
<td>the PIPE by SOMA laboratory</td>
<td>“PIPE is a voice / breath / mouth-controlled dynamic FX processor and synthesizer. It turns your voice into a powerful FX / beat / lead / soundscape synthesizer and it offers unprecedented levels of vocal processing, truly expanding the traditional boundaries of singing. Conventional singing with lyrics is also possible, and you can use the FX to add live modulations to your vocal performance.” (Kreimer)</td>
<td>$522.97</td>
</tr>
</tbody>
</table>

Although there are a few products that seem similar to the HumSynth, none match what the HumSynth seeks to provide. Three main competitors are listed in Table 1 above. THE MOUTH and Virta contain many of the same features as the HumSynth, however they require a DAW program which is not only an additional cost but has a learning curve of its own. Unlike these products, the HumSynth is a standalone system that won’t require the purchase of additional peripherals. The PIPE initially sounds exactly like the HumSynth, however it is more a voice-modulator than a voice-controlled synthesizer. Additionally, it has a high price point than the HumSynth aims for.
## III. Product Design Engineering Requirements

### Table 2: Functional and Performance Requirements

<table>
<thead>
<tr>
<th>Marketing Requirement</th>
<th>Engineering Requirement</th>
<th>Performance Requirement</th>
</tr>
</thead>
</table>
| User-friendly Interface | Must include an LCD display  
Must include keypad and button navigable menu for instrument and setting selection  
Must provide well organized interface for ease of use | Must utilize 128x64 pixel graphic LCD display  
Must have four directional buttons, one select button, and one back button for control  
Must have instrument output on/off button and keypad |
| Real Time Voice Control | Must detect pitch sung in real time  
Must synthesize the selected instrument at the correct pitch using the user specified ADSR and “live” or pitch correct mode settings  
Must modulate the instrument for every half tone change in frequency including vibrato changes | Must sample the input at a rate more than double the top of the vocal frequency range of approximately 500 Hz  
Must update the instrument pitch and synthesize the output within 25 ms before audible pitch change can be heard  
Must track the volume envelope and pitch sung within 1% of the actual value |
| Standalone System | Must not require any additional equipment for use excluding speakers or headphones for audio output  
Must provide wireless headset for instrument control | Must provide battery power for up to 8 hours  
Must provide all hardware required within a protected enclosure  
Must transmit input data with minimal signal loss |
| Standard Audio Output Interface | Must include standard ¼” and 3.5 mm audio outputs  
Must include a standard MIDI output | Must provide output drive hardware and connections  
Must provide UART connectivity to transmit MIDI data |
| Programmability | Must allow the user to upload custom instruments and ADSR waveforms  
Must allow user controls for a detune setting, “live” and pitch correct synthesis modes, preset or user controlled ADSR synthesis, and instrument selection | Must provide standard USB connectivity for upload capability  
Must provide menus for setting selection and instrument choice with control accessible via the base station of the device |
Table 3: Critical System Parameter Selection

<table>
<thead>
<tr>
<th>System</th>
<th>Parameter Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital Signal Processor</td>
<td>456 MHz clock speed, UART and I2C Data Busses, External Memory Interface</td>
</tr>
<tr>
<td>Analog-to-Digital Converter</td>
<td>12 bits, 119 kHz Conversion Rate, Accuracy +/- 1 LSB, 48 kHz Sampling Rate</td>
</tr>
<tr>
<td>Digital-to-Analog Converter</td>
<td>102 dB Dynamic Range, 101 dB SNR, 48 kHz Sampling Rate</td>
</tr>
<tr>
<td>Audio Interface</td>
<td>8 Ω Output Impedance for ¼” Speaker Jack with Class-D preamp, 16 Ω Output Impedance for 3.3 mm Headphone Jack Output, 220 Ω Pull-Up resistors for MIDI, UART transmission output with a baud rate of 31250</td>
</tr>
<tr>
<td>User Interface</td>
<td>128x64 pixel graphic LCD display</td>
</tr>
<tr>
<td>Wireless Microphone</td>
<td>Microphone Headset with Rechargeable Battery and USB Receiver for Audio Input</td>
</tr>
<tr>
<td>Power System</td>
<td>8000mA hour battery capacity for 6V source, 5V and 1.2 voltage regulators, supplies 5W at full load</td>
</tr>
</tbody>
</table>

Figure 3 HumSynth Level 0 Block Diagram
The user interface of the HumSynth will consist of an LCD graphic display and a four button directional pad with two buttons to select settings and revert to previous menus. The user will use the directional pad to move an indicator on the display over the various menu and setting selection choices. They will have the ability to choose from up to eight default instruments to synthesize, including any instruments that the user uploads after purchase. They can also select either the “live” mode, in which the instrument synthesizes the exact pitch sung, or the pitch-correct mode, where the pitch is synthesized to the next real, musical note frequency. The user may also choose to have the ADSR waveform for instrument synthesis follow the volume envelope of their voice, or for a preset waveform to be used for envelope formation. Finally, the user will have a detune setting option, where they may alter the frequency of the note synthesized by a fixed amount above or below the pitch sung.

IV. System Design - Functional Decomposition (Level 1)
Table 4: System Design Performance Budgets

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Performance Design Allocations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complete System</td>
<td>• Update of instrument pitch and envelope must be performed within 25 ms</td>
</tr>
</tbody>
</table>
| Wireless Microphone     | • Must transmit user input to audio interface for ADC with an SNR above 90 dB to prevent loss and distortion of the signal  
• Transmission time negligible with UHF RF wireless transmission |
| Audio Processor         | • Must update the pitch, envelope, and octave estimation of the input signal in 25 ms          
• Must continuously output the last pitch calculated  
• Must convert updated instrument synthesis to MIDI and output through UART in 3 ms |
| Audio Interface         | • Must convert the input sample and output the next DAC value once per sample (Fs = 48kHz)    
• Must drive ¼” and 3.5 mm jack output lines  
• Must transmit MIDI data through UART transmission |
| User Interface          | • Must provide external memory compatibility for instrument and setting storage, as well as user uploaded data |
| Power System            | • Must use a minimum of 90% of the source power  
• Must not exceed operating temperature of 45°C |
## V. Subsystem Descriptions:

Table 5: Subsystem Functional and Performance Requirements

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Engineering Requirement</th>
<th>Performance Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wireless Microphone</td>
<td>• Must provide wireless analog audio transmission with USB receiver</td>
<td>• Must provide 90 dB SNR to help preserve the input signal</td>
</tr>
<tr>
<td></td>
<td>• Must provide negligible transmission time</td>
<td>• Must provide negligible transmission time</td>
</tr>
<tr>
<td>Audio Processor</td>
<td>• Must calculate the pitch sung including the correct octave</td>
<td>• Must perform instrument update within 20 ms, fast enough to maintain “real time”</td>
</tr>
<tr>
<td></td>
<td>• Must track the volume envelope for amplitude and ADSR control</td>
<td>operation in relation to human perception of pitch change and allowing time processing</td>
</tr>
<tr>
<td></td>
<td>• Must synthesize the chosen instrument with the correct frequency and envelope settings</td>
<td>time</td>
</tr>
<tr>
<td></td>
<td>• Must perform instrument update within 20 ms, fast enough to maintain “real time”</td>
<td>• Must detect pitch within 3% of pitch sung</td>
</tr>
<tr>
<td></td>
<td>• Must track the volume envelope of the input signal within 1%</td>
<td>• Must track the volume envelope of the input signal within 1%</td>
</tr>
<tr>
<td></td>
<td>• Must provide UART transmission with a 31250 baud rate</td>
<td>• Must provide UART transmission with a 31250 baud rate</td>
</tr>
<tr>
<td>Audio Interface</td>
<td>• Must perform accurate and fast ADC of the input signal</td>
<td>• Must match the output impedance to both a 8 Ω speaker load and a 16 Ω headphone load</td>
</tr>
<tr>
<td></td>
<td>• Must perform accurate DAC of the output instrument data</td>
<td>• Must perform ADC and DAC conversion with SNR &gt; 90 dB, with 12 bit ADC resolution, and</td>
</tr>
<tr>
<td></td>
<td>• Must provide the output circuitry necessary to drive the ¼” speaker and 3.5 mm</td>
<td>sampling rates of 48 kHz</td>
</tr>
<tr>
<td></td>
<td>headphone jack outputs</td>
<td></td>
</tr>
<tr>
<td>User Interface</td>
<td>• Must provide display and controls for setting and instrument selection</td>
<td>• Must provide USB connectivity for programming capability</td>
</tr>
<tr>
<td></td>
<td>• Must provide interface for user programming of the device</td>
<td>• Must utilize 128x64 pixel graphic LCD display</td>
</tr>
<tr>
<td></td>
<td>• Must provide control over the instrument output</td>
<td>• Must have four directional buttons, a keypad, one select button, and one back button</td>
</tr>
<tr>
<td></td>
<td></td>
<td>for control</td>
</tr>
<tr>
<td>Power System</td>
<td>• Must provide 8 hours of use</td>
<td>• Must have a minimum efficiency of 90%</td>
</tr>
<tr>
<td></td>
<td>• Must run on AA batteries</td>
<td>• Must have a line regulation at full load with input changed from 5.5V to 6.5V &lt; 1%</td>
</tr>
<tr>
<td></td>
<td>• Must not let the device reach temperatures above 45℃</td>
<td>• Must have an output voltage ripple &lt; .5% of output voltage</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Must minimize harmonics for DSP supply rail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Deliver 5W at full load</td>
</tr>
</tbody>
</table>
**Wireless Microphone:**

**Technology Choices and Alternative Approach Considerations:**

The wireless microphone subsystem is responsible for the accurate transmission of the audio input data to the base station for conversion and processing. Utilizing UHF wireless transmission method allows for the transmission time of the input signal to be negligible in comparison to the overall system time requirements. Bluetooth was considered as a transmission method, but any discrete device option were not viable solutions, as they would require a custom built microphone to allow for connection. Instead, a commercial wireless headset was chosen, utilizing a compressor microphone, with good dynamic range and a frequency response spanning the vocal audio range, and a USB receiver so that the audio data can be transmitted to the base station. The headset and receiver connection utilizes UHF wireless transmission and transmit an analog signal to the base station.

**Verification Test Plan:**

The testing of the wireless microphone only requires audible confirmation, verifying its output after connecting it directly to a speaker. If necessary for greater pitch accuracy, the audio signal will be recorded onto a computer for spectral analysis. This may be used to design a filter to compensate for the any deficiencies in the microphone’s frequency response that may distort the audio.

*Figure 5 Wireless Microphone Subsystem Diagram*
**Audio Processor:**

**Technology Choices and Alternative Approach Considerations:**

The audio processor subsystem is responsible for all of the pitch detection and instrument synthesis calculations. These computationally intensive operations require a processor with great speed or parallel computing capability to perform them in real time applications. Two possible options for the system’s processor were a microprocessor, such as the ARM core series, or an FPGA processor with parallel DSP cores. The microprocessor provided a low cost solution, while the FPGA had the necessary computational speed required, but a better solution, combining the benefits of each solution, was found using a Digital Signal Processor optimized for real time operation. The TI C6748 series 456 MHz, floating point DSP chip was chosen for this device. It provides the high computational speed necessary for the pitch detection algorithms used, for the synthesis of the instrument, and for the filtering used to improve the audio out, and the floating point architecture provides easy program development. The DSP also includes the necessary UART capability to transmit the synthesized instrument after it has been converted into MIDI data, as well as the capability to connect to external memory if additional program storage is required.

To perform real time pitch detection, common approaches include statistical methods like autocorrelation, the square difference function (SDF), or the maximum likelihood algorithm, which are used to find the fundamental frequency of a frame of input data. However, many of these methods either require long algorithms that take too much time, or they do not perform accurately with a complex waveform such as a voice signal. Another statistical method, called the windowed special normalization of the autocorrelation function (WSNAC) overcomes these flaws, by combining a version of the autocorrelation and SDF functions in a way that provides much greater accuracy in fundamental frequency detection. While the implementation of this function may be computationally intensive, the chosen DSP processor includes a hardware architecture designed to maximize the efficiency of the sort of multiply-accumulate (MAC) computations required for these algorithms. The WSNAC function can also provide both vibrato and octave estimates given the correct window sizes.
For instrument synthesis, various methods of pitch synthesis were considered, including subtractive, additive, frequency modulation, and wavetable synthesis. Due to the system’s real time requirements, wavetable synthesis was chosen for its simplicity, speed, and ability to replicate real instruments with correct waveform input upon initialization. After the wavetable has been initialized, the processor can quickly service the audio codec with the correct sample to output from the table. Given enough system memory, many tables can be synthesized, and the user may select which instrument to output at any time. It is also easy to mix a predefined or user controlled ADSR envelope with the output sample, requiring only a multiplication between the sample and value of the current phase of the ADSR envelope.

**Verification Testing Plan:**

In order to completely test the functionality of the processor subsystem, we will test each major functional requirement and performance specification against the system requirements. The speed of the pitch detection algorithm will be verified to meet its time requirements by using the Real-time Analysis Tools (RTA) in TI’s Code Composer Studio (CCS) to benchmark the time required by each function. The frequency detected and output by the processor will also be logged using the RTA tools, comparing the accuracy against the input frequency and using a modulated input to verify the correct tracking of fast pitch variations such as vibrato. The actual sound output from the synthesizer will be audibly tested for its accuracy in reproduction. Finally, the MIDI output will be played using a common DAW program to verify that the instrument date was encoded correctly.
Audio Interface:

Technology Choices and Alternative Approach Considerations:

The audio interface subsystem is responsible for the analog to digital conversion of the input signal, as well as the digit to analog conversion of the synthesized instrument. It is also required to provide the output circuitry necessary to drive a ¼” speaker jack and a 3.5 mm headphone jack, including the correct output impedance and any preamplifiers necessary. While discrete modules could be used for each of these functions, many audio codec modules combine these functions and provide additional signal processing capabilities.

The module chosen for this device was the TI AIC3263 audio codec. It provides ADC and DAC sampling rates of 48 kHz to cover the complete audio range, with 12 bit ADC resolution for high quality audio sampling. The ADC has a conversion rate of 118 kHz to allow for additional processing time, and the DAC has a 102 dB SNR to provide high quality reproduction of the signal. In addition, the codec contains built-in DSP modules for effects processing and signal filtering. The codec also provides analog stereo inputs, multiple digital audio busses for processor connections, and direct DAC outputs to a Class D amplifier to drive a speaker output load and a headphone output with matched output impedance.
Verification Testing Plan:

To verify the functionality of the audio interface, the ADC and DAC conversion will be tested for speed performance and accuracy against the system performance specifications utilizing TI’s software interface for the audio codec to measure the test parameters. Any additional signal processing performed by the internal DSP of the codec will be tested in a similar manner to ensure the total output transmission time meets the system specifications. The operation of the speaker and headphone output will also be tested by sending a known signal through the DAC and verifying the output audibly. The output impedance of the circuitry will also be verified for matched output impedance using a simple test circuit and multimeter.

User Interface:

Technology Choices and Alternative Approach Considerations:

The user interface subsystem is responsible for providing the user with the ability to select which synthesizer instrument they want, as well as the various setting options for synthesis modes and envelope selections. The device must provide a program that uses a display and buttons to navigate through an interface of menus organizing every device option. This interface must also a interface to facility the programming of the device with new instruments and ADSR waveforms. A standard USB connector will be available to use for device programming.
Most portable device screens utilize some form of LCD display. Touch screen and TFT displays offer exceptional quality and programming capability, but at much too large of a cost, and simple character displays do not offer great resolution or aesthetic quality. To minimize cost, while maintaining an acceptable degree of quality, a graphical LCD display from Newhaven Display. This display offers 128x64 pixel display, providing the capability for a more detailed menu interface. To control the interface, directional button, a keypad, a select button, and a back button will be used, with a program written to ensure access to all device features and menus. An additional button will be added to turn the output of the synthesizer on or off, so that it does not continuously output. An concept of the user interface can be seen in the Packaging Design section below.

**Verification Testing Plan:**

A functional test of the user interface will be performed to ensure that all mechanical buttons operate with the correct function including the keypad. The program will be evaluated with the controls to ensure that there are no software bugs that prevent access to any device feature or crash the system for a given input. The interface will also be used to program the device with a customer ADSR waveform the confirm programming capability.

**Power System:**

**Technology Choices and Alternative Approach Considerations:**

The power system is responsible for the turn on operation of the device and ensuring safe operation for up to 8 hours. The entire device will have a max operating voltage of 5 V with the core logic supply input of the TI C6748 DSP requiring 1-1.3V for operation. The 5V voltage rail is supplied through a buck converter design. The buck is sourced through a replaceable four series AA battery pack for a total source voltage of 6V. The buck uses an LTC1624 IC to regulate the output voltage. The LTC1624 is chosen specifically for its high efficiency and its external power switch capability. With the ability to choose the power switch, a discrete MOSFET can be used to increase the efficiency of the overall buck converter design.
To avoid the harmonics and design challenges associated with a output voltage buck converter, a linear regulator will step down the 5V output from the buck converter to source the TI C6748 DSP’s 1-1.3 V core supply input. Choosing a linear regulator over a switching regulator introduces a loss of efficiency. However, a steady output voltage is a major benefit when dealing with a noise sensitive system. The LT3022-1.2 is chosen specifically for its high efficiency capabilities compared to other linear regulators. It’s fixed 1.2V output falls within the required 1-1.3 V supply input for the DSP.

To see if a more efficiency buck converter could be used to directly power the TI C6748, the LTC1624 was simulated to operate with an output of 1.25 V and the results can be seen in Figures X1-X2. However, the best simulation resulted in an output voltage ripple of 22mV. With a 1.25V output, this voltage ripple is 1.76% of the output voltage. This does not meet the necessary <.5% of output voltage engineering requirements. The addition of harmonics can also negatively affect the performance of the DSP.

![Figure 8 Schematic for Buck with 1.25V Output](image-url)
The output of the buck converter is set based on the values of \( R_1 \) and \( R_2 \) seen on Figure X. The equation used to calculate the \( R_1 \) and \( R_2 \) values can be found in the LTC1624 datasheet and is as follows:

\[
V_{\text{out1.19}} - 1 = \frac{R_2}{R_1}
\]

For an output voltage of 5V and .1% tolerance resistors, \( R_1 \) is chosen to be 25.5k\( \Omega \) and \( R_2 \) is calculated to be 81.6k\( \Omega \). The sensing resistor, \( R_4 \), is chosen based on the maximum load current. From the datasheet, \( R_4 \) is calculated with the equation \( R_4 = 100\text{mV}/I_{\text{max}} \) where \( I_{\text{max}} = I_{\text{full}} - I_{\text{load}} = 1\text{A} \). The minimum inductor value is found through the inductor current equation \( V_i = L\frac{dI_L}{dt} \). However, at high switching frequency, the slope of the inductor is approximately linear so \( V_i = L\frac{\Delta I}{\Delta t} \) where \( \Delta I \) is the output current ripple of the buck converter. When the switch is turned off, \( V_i = V_{\text{out}} = 5\text{V} \). \( \Delta t = (1-\)
D)T where D is the duty cycle of the switch signal and T is the switching period. For a buck converter, D=Vout/Vin and T= 1/(200kHz) can be found on the datasheet of the LTC1624. We can choose the value of \(\Delta I\) to be a low number such as 90mA. We find \(L = Vout/(\Delta I/\Delta t)\) is approximately 50\(\mu\)H. C1 and C2 are increased in simulation until the output voltage ripple is adequately minimized. Component values for C3, R3, C5, and C4 are fixed parameters specified on the LTC1624 datasheet.

![Figure 11 Buck Converter Design for a 5V Output](image1)

![Figure 12 Simulated Output Voltage for the 5V Buck Converter Design](image2)

**Verification Testing Plan:**

The Power supply can be tested by attaching the output to a resistance that will draw the estimated device load current. The Power Electronics laboratory gives access to an electronic load. An electronic load allows us to more easily vary the load current to test the proper operation under all load conditions. The 5 V buck converter was designed to operate on a maximum of 1 A. With the electronic load, we will the load current from 10% to 100% load (.1A to 1A) and take efficiency
calculations ($\eta = \frac{P_{\text{out}}}{P_{\text{in}}}$) for every 10% increment. Proper operation would result from having an efficiency above 90% for all percent loads. Load Regulation is tested by measuring the voltage output of the voltage regulator at 10% and full load. These values are then plugged into the load regulation equation to ensure load regulation is below 2%.

$$\% \text{ Load Regulation} = \frac{V_{\text{min}} - V_{\text{load}}}{V_{\text{max}} - V_{\text{load}}} \times 100\%$$

Finally, line regulation is tested by finding the output voltage under possible input voltage variants. This input voltage range is expected to be a minimum of 5.5V and maximum of 6.5V. The output voltage is measured at these two points and line regulation is calculated with the following equation:

$$\% \text{ Line Regulation} = \frac{V_{\text{min}} - V_{\text{output}}}{V_{\text{nom}} - V_{\text{output}}} \times 100\%$$

where $V_{\text{nom}}$ is the desired 5V. This percentage must be below 1% to meet engineering requirements.

### VI. Physical Construction and Integration

The enclosure contains all the components of the system while leaving the necessary ports accessible. An initial prototype was constructed on a plastic 3-D printer and shown in Figures 13 - 16 below.
The top of the enclosure shown in Figure 13 above showcases the importance of keeping track of whether a value is a radius or a diameter. The d-pad controller is, in fact, half the size shown and allows the next iteration of the enclosure to be closer to the size of the board itself. The LCD screen and the d-pad controller are inset on the top of the enclosure so as to provide a streamlined look.

![Angle View of Enclosure Prototype](image)

Figure 14 Angle View of Enclosure Prototype

Ports that are not in use, as seen in Figure 14 above, are nevertheless accessible due to the requirement of accessing the power port on the far right. A curved side would allow all unused ports to be covered and only leave accessible the ports that are listed in the features.
There is space around the board due to the supposed requirements of the interface components. However, since the error in d-pad dimensions was discovered, the enclosure can now be fitted to the board size resulting in a less bulky end product.
The enclosure features passive ventilation, for the purpose of managing heat, in the form of slots, as seen in Figure 16. Further testing would reveal which parts of the system produce the most heat and slots would be placed accordingly.

In the next iteration, the d-pad is now a comparable size to the LCD and the enclosure is closer to the 4” X 5” dimension of the board. Ventilation slots are not shown, as the placement of those depends on the results of thermal testing.

Figure 17 Front Angle of Enclosure Model
This next iteration of the enclosure covers the VGA port as that is not a feature of the system. The audio jacks, USB, and power ports are accessible as well as other ports that cannot be easily covered without also covering the required ports.
The back of the top piece has an opening for the ports on the audio-interface side of the board. Since all the ports are grouped together, there is a restriction in the ability to cover some while leaving the relevant ports accessible.
To ensure that the top and bottom pieces are aligned when the enclosure is closed, both pieces feature a finger joint, with one side sliding into the other. The plastic 3-D printer is unable to create sturdy features that small, so the final product would be 3-D printed out of metal. The metal enclosure would be inherently durable and have the additional benefit of shielding the board from EM interference.

VII. Integrated System Tests and Results:

The final system was not able to utilize the custom board design initially planned, instead using the C6748 LCDK development board to test and run the synthesizer program. It includes the same processor as originally planned, and has a similar audio codec (AIC3106) as the custom design. The board has a simple line in and out audio interface, but could not provide ¼” or MIDI outputs
required by the design. However, the main functionality and requirements of the system, including pitch detection, instrument synthesis, and real-time operation could still be verified using the LCDK.

Before testing the full program written for the HummSynth with the development board, the pitch detection algorithm was modeled and tested in Matlab to more easily verify its behavior and accuracy. The resulting WSNAC function, calculated with a 100Hz, sine tone input, is shown in Figure 14. The input signal was combined with a Hanning Window to provide a sharper and more identifiable peak near the fundamental frequency of the input signal, and a window size of 1024 was chosen to allow for low frequency detection down to at least 80Hz, the bottom of the male vocal range. The fundamental frequency is chosen by finding the index of the first primary peak that occurs after the zero index. This is can be made more difficult if the input has strong harmonics, producing sub and superharmonic peaks, which will be seen in the next test signal input. The first primary peak found as a result of the WSNAC calculation with the pure sine tone exactly matched the expected fundamental frequency. The Matlab function that processes the audio data samples at 44.1kHz, which equals a frequency of exactly 100Hz when divided by the fundamental frequency peak index found at 441. Next, the detection algorithm was tested with an input similar to that expected for the synth, using an audio recording of a chorus singing G# Major (415.3Hz). The algorithm performed

![WSNAC Function](image)

*Figure 22 WSNAC with 100Hz Tone Input*
well, with the WSNAC function shown in Figure 15. The pitch found did vary between input frames, but the calculation remained within 3% of the known input frequency, such a value of 412.1Hz calculated from the frame shown in Figure 15. Here it was necessary to excluded the subperiodic peak (mentioned previously) found before the fundamental to avoid octave errors in the result. These results were improved slightly using parabolic interpolation with the two points around the chosen peak, but the accuracy of the pitch mainly depended on the function itself.

![WSNAC Function](image)

*Figure 23 WSNAC with Chorus G# Major Input*

After confirming the operation of the pitch detection algorithm, the full program, including the instrument synthesis, was run using the development board to determine if the design would function and do so while meeting each of our performance specifications. The audio codec was verified alongside the processor, as their operation and success were deeply integrated and dependent, allowing us to utilize the CCS analysis tools to analyze their behaviors simultaneously. First, a song was passed into the system and passed through the audio codec, transmitting the received audio samples back to the audio codec and listening to the output for distortion. This confirmed the operation of the codec, as well as communication between the audio codec and the processor. Next, the wavetable synthesizer was initialized to output various frequencies to test its ability to reproduce them across the vocal range

28
without distortion or pitch errors. It was necessary to interpolate between the samples in the wavetable to output frequencies with non-integer increments landing between stored values. The linear interpolation method utilized produced many frequencies without distortion, but it could not calculate the interpolated values accurately enough to reproduce every frequency required by the device. Table 6 shows a series of five values output for a synthesizer frequency of 415Hz that was scaled by a factor 1500. Although the error was minimal, it grew as the phase moved toward $\pi/2$, and it was enough to cause audible distortion that clouded the tone and created a harsh sound. The issue was exacerbated by a loss of precision from the data type conversion necessary for transmitting the data between the processor and audio codec. We have yet to solve this problem, but the device’s ability to demonstrate successful pitch detection in real-time was not impeded by the issue.

<table>
<thead>
<tr>
<th>Interpolated Value</th>
<th>Expected Value</th>
<th>Percent Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1497.29</td>
<td>1497.83</td>
<td>0.04</td>
</tr>
<tr>
<td>1490.25</td>
<td>1491.35</td>
<td>0.07</td>
</tr>
<tr>
<td>1478.88</td>
<td>1480.58</td>
<td>0.11</td>
</tr>
<tr>
<td>1463.20</td>
<td>1465.52</td>
<td>0.16</td>
</tr>
<tr>
<td>1443.27</td>
<td>1446.25</td>
<td>0.21</td>
</tr>
</tbody>
</table>

Finally, the synthesizer was tested using the pitch detection algorithm to update the frequency for every 1024 samples input. This gave the processor 21.3ms to complete the pitch calculation before the processor would fail to meet real-time requirements for the system. Table 7 shows the time required to complete each subtask within the entire pitch calculation, recorded using the development software’s RTA tools. The fully integrated system was able to meet real-time requirements and update the pitch at least once per frame, but some of the more time intensive functions caused issues with requirements of the processor and audio communication. The time required to save the task and update the transmission data was longer than the maximum setup time allowed for the transmitter, causing the
output to intermittently stop. This required reinitialization of the transmitter, and resulted in an output that would come in and out of hearing at a slow enough rate to hear audibly. This was another issue that reduced the quality and appeal of the output sound, but did not prevent the verification of system on a functional level.

### Table 7: Time Benchmarks versus Specification

<table>
<thead>
<tr>
<th>Subtask Calculations</th>
<th>Time Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Autocorrelation Function</td>
<td>3.53 ms</td>
</tr>
<tr>
<td>Modified Square Difference Function</td>
<td>4.51 ms</td>
</tr>
<tr>
<td>WSNAC Function</td>
<td>0.34 ms</td>
</tr>
<tr>
<td>Fundamental Peak Detection</td>
<td>53.2 µs</td>
</tr>
<tr>
<td>Interpolation and Pitch Frequency</td>
<td>14.3 µs</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>8.4475 ms</strong></td>
</tr>
<tr>
<td><strong>System Requirement</strong></td>
<td><strong>21.3 ms</strong></td>
</tr>
</tbody>
</table>

While the accuracy of the pitch detection algorithm was verified in Matlab, we used the development platform’s RTA tools to view the frequencies calculated and output to the synthesizer. There was no loss in accuracy for the algorithm when implemented on the processor, and frequencies that could be successfully reproduced were audibly confirmed using a tone generator as the input to the wireless microphone. The correct output was also achieved by humming into the microphone, although it was hard to maintain a clear frequency that could be reproduced by the synthesizer.
**IX. Conclusions:**

Chad Carlson:

The final result of the HumSynth did not meet all of the feature requirements, and it was not able to produce a quality sound across the frequency range required, but it did achieve the fundamental basis for the system. It was able to reliably detect the pitch through the microphone input, and update it fast enough to meet real-time requirements. One of the major issues that caused the on and off toggling of the output could be eliminated by simply adapting the processor and the audio codec to a different transmission protocol that does not mute the output when internal errors occur. The other main issue keeping the final system from a basic, but full functionality, was the inaccuracy of the interpolation method used. This problem could be addressed by including more internal memory to the system, allowing for greater table length and synthesizer data, and a high order interpolation, such as cubic interpolation. We attempted to implement a higher order method, but we did not have better success and could not confirm the accuracy of our implementation. Overall, the original concept has been successfully achieved, and with minor improvements to the program and the option to utilize a better development platform in the future, there is no doubt that the full realization of the HumSynth can be achieved.

Anna Shabrova:

The enclosure for the HumSynth only went through one iteration of prototyping. Further iterations would take into consideration ergonomics (adding in curved features) and visual aesthetic in addition to functionality. Additionally the final design would be constructed out of metal for structural integrity, durability, and EM shielding. A larger LCD screen would also be desirable, however with the requirement of a graphical LCD a larger LCD would be difficult to achieve without a sharp increase in price point. Apart from integration, the user interface met the specifications and is a convenient and intuitive method of interaction with the system.
X. Bibliography


“The PIPE.” *SOMA Laboratory*, somasynths.com/pipe/.

Appendices:

A. Analysis of Senior Project Design
B. Schedule of Major Tasks and Milestones
C. Budget
D. Program Listing

A. Analysis of Senior Project Design

1. **Summary of Functional System Requirements:** The product will be able to synthesis the voice of the user in real-time to 8 different musical instrument sounds. In addition, it will give the user the option to play of using “Pitch-Corrected” mode or “Live” mode as well as the option to adjust the ADSR envelope of the synthesized waveform.

2. **Primary Constraints:** The first challenge we will face in this project will understanding the principles of digital signal processing. Following this, we will deal with the difficulties of applying digital signal processing techniques to a real-time operating system.

3. **Economic:** A target budget of $125 dollars will be made for the manufacturing of this system. This will include purchase of the microcontroller, digital signal processor, audio and power circuit components, mechanical casing, and fabrication of design for the base station PCB. Any testing equipment for debugging the microcontroller or analog circuits will be obtained from the one of the several engineering labs in the EE building.

4. **Commercial Manufacture:** There is currently no commercial equivalent to our product that would give a strong idea of what a reasonable competitive price would be. However, the closest competitors are software plugins that sell at price ranges of $40-100. To compete here, the final design of the system will have a production cost of $80.

5. **Environmental:** Components in the design will be carefully chose to limit the use of toxic materials such as lead, mercury, and other common elements found in electronic devices. Attention will also be paid to where the PCB will be manufactured and which materials will
be used for the products casing. As researched by the Greenpeace Organization of East Asia, brominated flame retardants are used in circuit boards and plastic casing. These retardants do not break down easily and are building up in the environment.

6. **Manufacturability:** The manufacturing process for the product will be chosen when we choose a commercial PCB manufacturer. Ideally, we will choose one of high product reliability. Since this product is not expected to be put in high stress situations, there will only be the worry of making a PCB that will resist a moderate fall. However, the enclosures on the casing must allow for protection from a spill of water and resistant to dust build up.

7. **Sustainability:** The biggest challenge with maintaining the completed device operating will be ensuring that the casing is durable and ensures enclosure to liquids or dust. In addition, research must be made on the lifespan of all individual components of the device to give an accurate estimation of the device lifespan. Software packages could upgrade the Raspberry Pi to decrease power consumptions of the device and increase overall lifespan. This, however, will require users to stay connected to the manufacturer through methods such as email as the device will not have a function to alert the user when new updates are available.

8. **Ethics:** Should the device be marketed towards educators, it must be designed in a way to aid education. If the device becomes more of a distraction for classroom learning than it is a useful teaching tool, then the purpose of the device is lost. Also, the device must give the user the assurance that it will function for the length of its lifetime. If the device fails in the middle of a performance three or four years after purchase, it could potentially be the fault of the manufacturer.

9. **Health and Safety:** The power circuit of the device must ensure safe operation during the product’s lifetime. However, failure of any other subsystem of the product does not pose any danger to the user. Any worries for the health and safety of the user comes with the material options for the manufacturing of the product. Since this device could potentially be used by children, the materials used should not cause any harm when put inside the mouth or ingested. Additionally, the shape of the casing must have no sharp edges.

10. **Social and Political:** This products aims to create a social impact on education. By providing a device that introduces STEM topics using music, more students will gain interest in further STEM education. This was shown to work in Mr. Howe’s master thesis for Georgia Tech. Additionally, if this device proves to work in this environment, the politics behind classroom education could be reevaluated. More attention would be given to STEAM (Science, Technology, Engineering, Arts, Mathematics) education to help aid the argument that the arts are fundamental for a well rounded education.

11. **Development:** This project will require a development in knowledge of digital signal processing and real-time operating systems. It will also require an evaluation of electronic and casing materials to ensure the health and safety younger users.
B. Schedule of Major Tasks and Milestones:

<table>
<thead>
<tr>
<th>Task</th>
<th>April</th>
<th>May</th>
<th>June</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microphone Selected</td>
<td>Anna</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microphone Purchased</td>
<td>Anna</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pitch Detection Design</td>
<td>Chad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pitch Detection Implemented</td>
<td>Chad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pitch Detection Debug</td>
<td>Chad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Source Selected</td>
<td>Jair</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Configuration</td>
<td>Jair</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hardware Implemented</td>
<td>Jair</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Casing Prototype</td>
<td>Anna</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Casing Finalized</td>
<td>Anna</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface Design</td>
<td>Anna/Chad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interface Implemented</td>
<td>Anna/Chad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional Test</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transmission/Receipt Test</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Usability Test</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>First System Demo</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Product Verification</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Power Test</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report First Draft</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Senior Project Expo</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Demo</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report Final Draft</td>
<td>ALL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 24 Gantt Chart for Spring Quarter

Budget:

Table 7: Parts List and Budget

<table>
<thead>
<tr>
<th>Count</th>
<th>Description</th>
<th>Size</th>
<th>Part Number</th>
<th>Manufacturer</th>
<th>Per Unit Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Digital Signal Processor</td>
<td>13.106mmx13.106mm</td>
<td>TMS320C6748</td>
<td>Texas Instruments</td>
<td>$25</td>
</tr>
<tr>
<td>1</td>
<td>Audio Codec</td>
<td>4.81 mm × 4.81 mm × 0.625 mm</td>
<td>TLV320AIC3263</td>
<td>Texas Instruments</td>
<td>$5.25</td>
</tr>
<tr>
<td>1</td>
<td>Stereo ¼” audio jack (TS) w/NC switch</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>$1.20</td>
</tr>
<tr>
<td>1</td>
<td>High Impedance 3.5 mm Headphone Jack (TRS)</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>$3.30</td>
</tr>
<tr>
<td>Count</td>
<td>Description</td>
<td>RefDe</td>
<td>Value</td>
<td>Description</td>
<td>Size</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
<td>-------</td>
<td>-------</td>
<td>-------------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>LCD display</td>
<td>N/A</td>
<td>128x64 pixel graphic</td>
<td>113mmx56mmx13.6mm</td>
<td>N/A</td>
</tr>
<tr>
<td>6</td>
<td>Buttons</td>
<td>SKHHAKA010</td>
<td>6mmx6mmx5mm</td>
<td>N/A</td>
<td>ALPS</td>
</tr>
<tr>
<td>1</td>
<td>Switching Voltage Regulators Hi Eff SO-8, N-Chennel Reg</td>
<td>LTC1624CS8#PBF</td>
<td>0.244 in x 0.197 in</td>
<td>Analog Devices</td>
<td>$9.11</td>
</tr>
<tr>
<td>1</td>
<td>Linear Regulator</td>
<td>LT3022</td>
<td>5mm x 3mm</td>
<td>N/A</td>
<td>Analog Devices</td>
</tr>
<tr>
<td>1</td>
<td>Headset</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Pyle</td>
</tr>
</tbody>
</table>

Table 9: Component Costs for Buck Converter
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C2</td>
<td>6uF</td>
<td>Aluminum Electrolytic Capacitors - Leaded 6.0uF 25volts - 10% +75% 6.3x13mm</td>
<td>6.3 mm x 13 mm</td>
<td>99285-4-7320</td>
</tr>
<tr>
<td>1</td>
<td>S1</td>
<td>N/A</td>
<td>MOSFET 40V 3Ohm</td>
<td>0.205 in x 0.82 in</td>
<td>VN0104N3-G</td>
</tr>
<tr>
<td>1</td>
<td>D1</td>
<td>N/A</td>
<td>Diode</td>
<td>.027 in x .107 in</td>
<td>STPSC2H12D</td>
</tr>
<tr>
<td>1</td>
<td>R3</td>
<td>6.8kΩ</td>
<td>RES 6.8K OHM 1/4W 5% AXIAL</td>
<td>(2.30mm x 6.00mm)</td>
<td>CF14JT6K80CT-N</td>
</tr>
<tr>
<td>1</td>
<td>C3</td>
<td>470pF</td>
<td>CAP CER 470PF 2KV Y5P RADIAL</td>
<td>(7.50mm diameter)</td>
<td>1286PH-ND</td>
</tr>
<tr>
<td>1</td>
<td>C5</td>
<td>1000pF</td>
<td>CAP CER 1000PF 760VAC Y5U RADIAL</td>
<td>(9.00mm diameter)</td>
<td>BC2374-ND</td>
</tr>
<tr>
<td>1</td>
<td>C4</td>
<td>0.1uF</td>
<td>CAP CER 0.1UF 50V X7R RADIAL</td>
<td>(4.00mm x 2.50mm)</td>
<td>BC3324-ND</td>
</tr>
<tr>
<td>1</td>
<td>R4</td>
<td>0.01Ω</td>
<td>RES 0.03 OHM 1/2W 3% AXIAL</td>
<td>8.38mm</td>
<td>605HR030E-ND</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>R2</td>
<td>68kΩ</td>
<td>RES 68K OHM</td>
<td>1/4W 5% AXIAL</td>
<td>(2.30mm x 6.00mm)</td>
</tr>
<tr>
<td></td>
<td>R1</td>
<td>20kΩ</td>
<td>RES 20K OHM</td>
<td>1/4W 5% AXIAL</td>
<td>(2.30mm x 6.00mm)</td>
</tr>
<tr>
<td></td>
<td>Rload</td>
<td>2.5Ω</td>
<td>RES 2.5 OHM</td>
<td>3.75W 5% AXIAL</td>
<td>4.78mm x 14.27mm</td>
</tr>
</tbody>
</table>

**Program Listing:**

```
// ******************************************************
// Title: HumSynth
// File: main.b
// Author: Chad Carlson
//
// Brief:   Header file for the HumSynth Program
#
#include <xdc/std.h>          // mandatory - have to include first, for BIOS types
#include <ti/sysbios/BIOS.h>  // mandatory - if you call APIs like BIOS_start()
#include <xdc/fpg/global.h>   // header file for statically defined objects

#include <ti/sysbios/knl/Clock.h> // when using Clock module
#include <ti/sysbios/knl/Task.h>   // when using Tasks
#include <ti/sysbios/knl/Semaphore.h> // when using Semaphores
#include <xdc/runtime/Timestamp.h> // when using Timestamp APIs (TSCL/H), 32bit, 64bit
#include <xdc/runtime/System.h>   // for runtime system calls (e.g. system_printf)
#include <xdc/runtime/Log.h>

#ifndef MAIN_H_
#define MAIN_H_
```

```
// Includes
// ******************************************************
#include "evmc6748.h"
#include "evmc6748_gpio.h"
#include "evmc6748_i2c.h"
#include "evmc6748_mcasp.h"
```
#include "evmc6748_aic3106.h"
#include "math.h"
#include "mathf.h"

#define PI (3.14159265358979323846)
#define TWOPI (2*PI)
#define NHARMS (5)
#define SRATE (48000)
#define BUFSIZE (1024)
#define TABSIZE (4097)
#define PING (0)
#define PONG (1)
#define L2_LINESIZE (128)
#define HOLE_SIZE (127) // for guard point

// Buffer Structures - RCV_DATA_BUFFER (hist, data, hole), XMT_DATA_BUFFER

typedef struct rcv_data_buffer
{
    int16_t pingL[BUFSIZE];
    int16_t pongL[BUFSIZE];
} RCV_DATA_BUFFER;

typedef struct hanning_window
{
    float val[BUFSIZE];
} HANNING_WINDOW;

typedef struct snac_buffer
{
    float amp[BUFSIZE];
    float acf[BUFSIZE];
    float sdf[BUFSIZE];
    uint32_t calc;
    uint16_t peak_pos;
} SNAC_BUFFER;

typedef struct t_tabe_oscil
{
    double table[TABSIZE]; // pointer to array containing waveform
    double hole[9];
    double curfreq;
    double curphase;
double incr;
double dtablen;
double sizeovrsr;
double val;
}

// Prototypes
//----------------------------------------------------------------------------
void isrAudio(void);
void init_buffers(void);
void McASP_Init_TTO();
void AIC3106_Init_TTO();
void McASP_Start_TTO();
void USTIMER_delay(uint32_t time);
void gen_hanning_window();
void calc_acf(int16_t * x, float * w, float * restrict snac, int16_t length);
void calc_sdf(int16_t * x, float * w, float * restrict snac, int16_t length);
void calc_snac(float * restrict snac, float * restrict acf, float * restrict sdf);
uint16_t find_peak(float * restrict snac);
float parab_inter(float * restrict snac, uint16_t peak_pos);
double tabitick(double freq, double curphase, double incr, double table, double sizeovrsr, double dtablen);
void wave_gen(uint16_t length, uint16_t nharms);
void new_oscilt(double srate, uint16_t nharms);
void norm_gtable();

// Externs
//----------------------------------------------------------------------------
extern RCV_DATA_BUFFER rcv;
extern HANNING_WINDOW hann;
extern SNAC_BUFFER snac;
//extern GTABLE gtable;
extern OSCILT p_osc;
extern int IRAM;
extern uint16_t xmt;
extern cregister volatile unsigned int CSR; // control status register
extern cregister volatile unsigned int ICR; // interrupt clear register
extern cregister volatile unsigned int IER; // interrupt enable reg.
extern uint16_t pingPong;

//----------------------------------------------------------------------------
// Title: HummSynth
// File: main.c
// Author: Chad Carlson
//
// Brief: Initializes all memory buffers, the window function, and the synthesizer;
// Start BIOS scheduler
//------------------------------------------------------------------

// Includes
//-----------------------------------------------------------------------------
#include "main.h"
//-----------------------------------------------------------------------------

// GLOBALS (Buffers aligned on L2_cache_line_size boundaries
//
// Note: see main.h for details of the structure
//-----------------------------------------------------------------------------

#pragma DATA_ALIGN(rcv, L2_LINESIZE);
RCV_DATA_BUFFER rcv;

#pragma DATA_ALIGN(snac, L2_LINESIZE);
SNAC_BUF snac;

#pragma DATA_ALIGN(hann, L2_LINESIZE);
HANNING_WINDOW hann;

#pragma DATA_ALIGN(p_osc, L2_LINESIZE);
OSCILT p_osc;

uint16_t pingPong;

// main()---------------------------------------------

void main(void)
{
    init_buffers(); // zero buffers

    I2C_init(I2C0, I2C_CLK_400K); // init I2C channel

    McASP_Init_TTO(); // init McASP (modified from original BSL)
    AIC3106_Init_TTO(); // init AIC3106 (modified from original BSL)

    wave_gen(TABSIZE - 1, NHARMS);
    new_oscilt(SRATE, NHARMS);
    gen_hanning_window();

    ICR = (1 << 5); // clear INT5 (precaution)
    IER |= (1 << 5); // enable INT5 as CPU interrupt
}
McASP_Start_TTO();        // start McASP clocks
BIOS_start();               // return to BIOS scheduler

//----------------------------------------------------------------------------------
// init_buffers()
//----------------------------------------------------------------------------------

void init_buffers(void)
{
    int16_t i;

    // zero out data buffers explicitly (Rcv and Xmt)
    for (i = 0; i < BUFFSIZE; i++)
    {
        rsv.pingL[i] = 0;
        rsv.pongL[i] = 0;
        snac.amp[i] = 0;
        snac.acf[i] = 0;
        snac.sdf[i] = 0;
        pingPong = PING;
    }
    snac.calc = 0;
    snac.peak_pos = 0;
}

void new_oscilt(double sr, uint16_t nharm){
    // init osc
    p_osc.curfreq = 415.0;
    p_osc.curphase = 0.0;
    p_osc.incr = 0.0;
    p_osc.val = 0.0;

    uint16_t i, j;
    double harmonic = 1.0;
    double step;

    step = (double) TWOPI / (double) (TABSIZE-1);
    for(i=0; i < 1; i++)
    for(j=0; j < TABSIZE - 1; j++)
    {      
        p_osc.table[j] += (1500.0 * cos(step * harmonic * j));
    }  
    harmonic++;
}

p_osc.table[i] = p_osc.table[0];     // guard point
p_osc.dtable = (double) TABSIZE-1;
p_osc.sizeovrsr = p_osc.dtablen / (double) srate;

}

void gen_hanning_window(void)
{
    uint16_t i;
    for (i = 0; i < BUFFSIZE; i++) {
        hann.val[i] = 1.0 * (0.5 * (1 - cos(TWOPI*i/1023)));
    }
}

//------------------------------------------------------------------------------
// Title: HummSynth
// File: isr.c
// Author: Chad Carlson
//
// Brief: Interrupt Service Routine, Triggered when RDATA/XDATA ready
// Double buffered, calculates next output value before each transmit
//------------------------------------------------------------------------------
// Includes
#include "main.h"
//-------------------------------
// Defines
#define LEFT (0)
#define RIGHT (1)
#define RDATA 0x20 // R/XDATA interrupt mask
#define XDATA 0x20
//-------------------------------
// isrAudio()
//-------------------------------

void isrAudio(void)
{
    static int32_t dataIn32, dataOut32; //store McASP audio data
    static uint16_t blkCnt = 0;
    static int16_t *pInBuf_local;
    static uint16_t leftRight = LEFT;
    double val, freq = p_osc.curfreq, incr = p_osc.incr, sizeovrsr = p_osc.sizeovrsr,
    dtablen = p_osc.dtablen, curphase = p_osc.curphase;
    double *table = p_osc.table;

    //Init pointers for ping/pong at beginning of new BLK only

    if (blkCnt == 0)
{ 
    if (pingPong == PING) // PING Buffers init 
    { 
        pInBuf_local = rcv.pingL;
    }
    else // PONG Buffers init 
    { 
        pInBuf_local = rcv.pongL;
    } 
}

if (CHKBIT(MCASP->XSTAT, XUNDRUN)) {

    MCASP->GBLCTL = 0;
    // enable the audio clocks, verifying each bit is properly set.
    SETBIT(MCASP->XGBLCTL, XHCLKRST);
    while (!CHKBIT(MCASP->XGBLCTL, XHCLKRST)) {} 
    SETBIT(MCASP->RGBLCTL, RHCLKRST);
    while (!CHKBIT(MCASP->RGBLCTL, RHCLKRST)) {} 

    SETBIT(MCASP->XGBLCTL, XCLKRST);
    while (!CHKBIT(MCASP->XGBLCTL, XCLKRST)) {} 
    SETBIT(MCASP->RGBLCTL, RCLKRST);
    while (!CHKBIT(MCASP->RGBLCTL, RCLKRST)) {} 

    SETBIT(MCASP->RINTCTL, RDATA); // enable McASP XMT/RCV interrupts
    while (!CHKBIT(MCASP->RINTCTL, RDATA)) {} // see #defines at top of file
    SETBIT(MCASP->XINTCTL, XDATA);
    while (!CHKBIT(MCASP->XINTCTL, XDATA)) {} 

    MCASP->XSTAT = 0x0000FFFF; // Clear all (see procedure in UG)
    MCASP->RSTAT = 0x0000FFFF; // Clear all 

    SETBIT(MCASP->XGBLCTL, XSRCLR);
    while (!CHKBIT(MCASP->XGBLCTL, XSRCLR)) {} 
    SETBIT(MCASP->RGBLCTL, RSRCLR);
    while (!CHKBIT(MCASP->RGBLCTL, RSRCLR)) {} 

    /* Write a 0, so that no underrun occurs after releasing the state machine */

    MCASP->XBUF13 = 0;

    SETBIT(MCASP->XGBLCTL, XSMRST);
    while (!CHKBIT(MCASP->XGBLCTL, XSMRST)) {} 
    SETBIT(MCASP->RGBLCTL, RSMRST);
    while (!CHKBIT(MCASP->RGBLCTL, RSMRST)) {} 

    SETBIT(MCASP->XGBLCTL, XFRST);
while (!CHKBIT(MCASP->XGBLCTL, XFRST)) {}
SETBIT(MCASP->RGBLCTL, RFRST);
while (!CHKBIT(MCASP->RGBLCTL, RFRST)) {}

} // Receive code
if (CHKBIT(MCASP->SRCTL14, RRDY)) {

dataIn32 = MCASP->XBUF14;

if (leftRight == LEFT) {
    pInBuf_local[blkCnt] = (int16_t) dataIn32; // RCV LEFT
} else {
    blkCnt++;
}

leftRight ^= 1;
}

} // Transmit code
if (CHKBIT(MCASP->SRCTL13, XRDY)) {
    val = tabitick(p_osc.curfreq, curphase, incr, table, sizeovrsr, dtablen);
dataOut32 = (int16_t) val;
MCASP->XBUF13 = dataOut32; //
}

if (snac.calc == 1) {
    snac.calc = 0;
    Semaphore_post(acfReady);
}

if (snac.calc == 2) {
    snac.calc = 0;
    Semaphore_post(sdfReady);
}

if (snac.calc == 3) {
    snac.calc = 0;
    Semaphore_post(snacReady);
}

if (snac.calc == 4) {
    snac.calc = 0;
    Semaphore_post(posReady);
//IF end of buffer, copy rcv-to-xmt, zero blkCnt, swap PING-PONG boolean

if (blkCnt >= BUFSIZE) // if END OF CURRENT BLOCK
{
    pingPong ^= 1;       // swap pingPong indicator
    blkCnt = 0;          // zero blkCnt
    Semaphore_post(mcaspReady); // post SEM to copy RCV to XMT
}

// interpolating tick function

double tabitick(double freq, double curphase, double incr, double * table, double sizeovrsr, double dtablen)
{
    uint16_t base_index = (int) curphase;
    uint16_t next_index = base_index + 1;
    double frac, slope, val;
    incr = sizeovrsr * freq;

    frac = curphase - base_index;
    val = table[base_index];
    slope = table[next_index] - val;
    val += (frac * slope);
    curphase += incr;
    while (curphase >= dtablen)
        curphase -= dtablen;
    while (curphase < 0.0)
        curphase += dtablen;
    p_osccurphase = curphase;

    return val;
}

//-------------------------------------------------------------------------------
// Includes

#include "afc_calc.h"
#define RDATA 0x20
#define XDATA 0x20

// R/XDATA interrupt mask
void acf_calc()
{
    while(1)
    {
        Semaphore_pend(mcaspReady, BIOS_WAIT_FOREVER);

        if (pingPong == PONG) // if PONG, filter PING
        {
            calc_acf(rcv.pingL, hann.val, snac.amp, BUFSIZE);
            snac.calc = 1;
        }

        else // if PING, filter PONG
        {
            calc_acf(rcv.pongL, hann.val, snac.amp, BUFSIZE);
            snac.calc = 1;
        }
    }
}

void calc_acf(int16_t * x, float * w, float * restrict snac, int16_t length)
{
    uint16_t i, tau;
    float acf;

    #pragma MUST_ITERATE(BUFFSIZE,BUFFSIZE,BUFFSIZE);
    for (tau = 0; tau < BUFSIZE; tau++)
    {
        acf = 0.0;

        #pragma MUST_ITERATE(1,BUFFSIZE,1);
        for (i = 0; i < (BUFSIZE - tau); i++)
        {
            acf += x[i] * w[i] * x[i+tau] * w[i+tau];
        }
        snac[tau] = 2.0*acf;
    }
}

//----------------------------------------------------------------------------
// Title: HummSynth
// File: sdf_calc.c
void sdf_calc(void){

    while(1){

        Semaphore_pend(acfReady, BIOS_WAIT_FOREVER);

        if (pingPong == PONG){
            calc_sdf(rcv.pingL, hann.val, snac.sdf, BUFFSIZE);
            snac.calc = 2;
        }
        else{
            calc_sdf(rcv.pongL, hann.val, snac.sdf, BUFFSIZE);
            snac.calc = 2;
        }
    }
}

void calc_sdf(int16_t * x, float * w, float * restrict snac, int16_t length){
    uint16_t i, tau;
    float m_sdf;

    #pragma MUST_ITERATE(BUFFSIZE,BUFFSIZE,BUFFSIZE);
    for (tau = 0; tau < length; tau++)
    {
        m_sdf = 0.0;
    }

    #pragma MUST_ITERATE(1,BUFFSIZE,1);
    for (i = 0; i < (length - tau) ; i++)
    {
        m_sdf += w[i] * w[i+tau] * ((x[i] * x[i]) + (x[i+tau] * x[i+tau]));
    }
    snac[tau] = m_sdf;
}

//>Title: HummSynth
void snac_calc(void)
{
    while(1)
    {
        Semaphore_pend(sdfReady, BIOS_WAIT_FOREVER);
        calc_snac(snac.amp, snac.acf, snac.sdf);
        snac.calc = 3;
    }
}

void calc_snac(float * restrict snac, float * restrict acf, float * restrict sdf)
{
    uint16_t tau;
    #pragma MUST_ITERATE(BUFFSIZE,BUFFSIZE,BUFFSIZE);
    for (tau = 0; tau < BUFFSIZE; tau++)
    {
        snac[tau] = acf[tau]/sdf[tau];
    }
}

#include "main.h"

void pos_calc(void)
{
    while(1)
    {
        Semaphore_pend(snacReady, BIOS_WAIT_FOREVER);
    }
}
snac.peak_pos = find_peak(snac.amp);
snac.calc = 4;
}
}

uint16_t find_peak(float * restrict snac){

float max_cutoff = 0.9;
uint16_t pos = 0, max_pos = 0;

while (pos < 650 && snac[pos] > 0)
{
pos += 1;
}
max_pos = pos;
while (snac[max_pos] < max_cutoff){
    while (pos < 650 && snac[pos] <= 0.0)
    {
pos += 1;
    }
    while (pos < 650 && snac[pos] > 0.0)
    {
        if (snac[pos] > snac[pos-1] && snac[pos] >= snac[pos+1])
        {
            if (snac[pos] > snac[max_pos])
            {
                max_pos = pos;
            }
            pos += 1;
        }
    }
    if (pos >= 650)
        break;
}

return max_pos;
}

//---------------------------------------------------------------------------------
// Title: HummSynth
// File: freq_calc.h
// Author: Chad Carlson
//
// Brief: Calculates the WSNAC function for the last buffer filled
//
#include "main.h"

void freq_calc(void){
    float freq;
}
while(1){

    Semaphore_pend(posReady, BIOS_WAIT_FOREVER);

    freq = parab_inter(snac.amp, snac.peak_pos);
    if (freq >= 80 && freq < 500) {
        p_osc.curfreq = freq;
    } else {
        p_osc.curfreq = 0;
    }
}

float parab_inter(float * restrict snac, uint16_t peak_pos) {

    float x_val, delta;
    float prev = snac[peak_pos - 1];
    float peak = snac[peak_pos];
    float next = snac[peak_pos + 1];
    float bot = peak_pos;
    float div = next + prev - (2 * peak);
    if (div == 0.0) {
        x_val = bot;
    } else {
        delta = prev - next;
        x_val = bot + (delta / (2 * div));
    }

    x_val = 48000.0 / x_val;

    //LOG_printf(&trace, "%d", (long) x_val);

    return x_val;
}