Design of a PC Stereo Microphone Interface

J. Harrison, A. Lam, N. Young, and J. Zacharko

Abstract – This report outlines the design of a portable digital stereo audio recorder. The device records two channels of 16-bit audio samples at frequency of 16 kHz. The audio data is stored in non-volatile flash memory so that data is retained without power. The data can be transferred to a computer via a Universal Serial Bus (USB) connection. A serial terminal program is used to capture a binary data stream, which can be directly stored as a .wav file.

I. INTRODUCTION

The motivation for this design project comes from the classic ‘cocktail party problem’ in which multiple conversations are occurring simultaneously making it difficult to discern between distinct audio sources [1, 2]. The project is intended to provide an aid to audio source separation by recording stereo audio in a format suitable to the algorithms. This digital device is a hardware interface to a personal computer via a Universal Serial Bus (USB) connection that allows stereo audio, recorded with two microphones, to be downloaded. With this device, over a minute and a half of FM quality stereo audio can be recorded, and later transferred to a computer.

II. HARDWARE DESIGN

A generalized block diagram for the system is shown in fig. 1. The main components of the system are an analog subsystem consisting of microphones and filters. This connects to an analog to digital converter. A microcontroller oversees the operation of the recorder by receiving user input and by moving data between subsystems. Non-volatile flash memory is used to store the recorded audio. A USB connection allows data to be sent to a PC. The entire system is powered from a battery, connected through a switch, to regulated supplies. Feedback is provided to the user via LEDs.

A. Analog Design

The portable stereo recording device was designed specifically to record the human voice which ranges from 300 Hz to 3.4 kHz [3]. The analog portion of the hardware design was very important for it ensured that the correct data was collected, amplified and sampled for the device’s intended use. The architecture consists of two EMKAY MD9765A SZ omni-directional electret condenser microphones and two eighth-order band-pass Bessel filters. Fig. 2 shows the analog design for audio input for one of the two microphones and filters and the table I summarizes the required components.

Electret microphones were chosen for their compact size, durability and low power consumption which are very important characteristics for a portable device. An electret microphone is a transducer that outputs voltages directly proportional to the air pressure. Pressure sensitive microphones respond much less to wind noise than directional (velocity sensitive) microphones. An omni-directional microphone was used because the magnitude response vs. frequency is comparatively flatter than other microphones. The frequency response of this microphone is essentially flat across the human voice band which makes it appropriate for the capturing of human voice [4]. This specific model has a high sound pressure level sensitivity of ~42 decibels (dB), where 0dB equals 1V/Pa. This allows the microphone to discern distant and faint audio sources.

Each audio input from the two microphones was filtered and amplified through an eighth order band-pass Bessel filter constructed using Texas Instruments’ OPA4342 Operational Amplifiers. These OP-Amps provide rail-to-rail inputs and outputs as well as a sufficient gain-bandwidth product and slew rate for this application.

---

Fig. 1. Generalized block diagram.

Fig. 2. Analog subsystem design.
The Bessel filter was primarily chosen for its linear phase response in the pass band as shown in fig. 3. The filter was designed for frequencies from 300 to 3.4 kHz with a gain of 40 dB. The theoretical transfer function of the filter in fig. 3 shows that it has a relatively flat pass band and sufficient roll off for recording the human voice band. A gain of 40 decibels allowed appropriate amplification of the microphone input for recording distances of within a meter without clipping of the signal. The measured magnitude response is also shown, but shows slight differences due to component tolerances.

B. Analog to Digital Conversion

The amplified and filtered audio data was sampled with Analog Device’s AD7654 analog to digital converter (ADC). The specific model was chosen for its 16-bit, 500 kSa/s, dual two-channel conversion [5]. This model has two sample and hold circuits that allow the two analog input channels from the amplifier stage to be sampled simultaneously. The two channels are sampled at 16kSa/s, which is sufficient to avoid aliasing for the voice band range according to Nyquist’s Theorem. This theorem states that for a perfect reconstruction to be possible, the sampling frequency must be at least twice the sampling frequency.

The ADC allows a very fast throughput of 512 kbits/s when sampling two channels at 16 kSa/s at 16 bits/Sa. This is calculated by equ (1).

\[
2 \text{ channels} \times 16 \text{kSa/s} \times 16 \text{bits/Sa} = 512 \text{ kbit/s.} \quad (1)
\]

The conversion process of the ADC was controlled in the impulse mode. In this sampling mode, conversion is controlled by the conversion start signal. By setting conversion start low, conversion is automatically initiated if the device is not busy. Upon completion of conversion, the ADC reads data via a parallel interface to the microcontroller. As the interface is only 8 bits, and a 16 bit result must be read, a method is required to read both bytes. This is achieved by using the BYTESWAP pin, controlled by the microcontroller, enabling both bytes in the sample to be continuously swapped to output data bus. The impulse mode is the lowest power dissipation mode since power is scaled linearly with the throughput of the device. Power is saved between conversions, making it ideal for a battery powered device.

C. Microcontroller

The PIC16F877A microcontroller was used for the hardware control of the ADC, memory, user interface, and computer interface. This specific PIC was chosen for its availability, and support. The PIC has 33 Input/Output pins [6], as well as various hardware peripherals are sufficient for the various devices it controls as shown on the schematic in fig. 4. The firmware for the project was written in assembly.

Through the user interface, the operation of the device is controlled through the PIC. In record mode, the PIC controls the ADC by signaling the conversion start pin to sample when the ADC is not busy. Upon completion of the analog to digital conversion, channel A and B will be toggled to send data to the output bus where a sample (2 bytes) of each channel will be read at a time by toggling BYTESWAP. Data is stored in the flash memory via control of the PIC through the Serial Peripheral Interface (SPI). In the SPI mode, the PIC allows eight bits of data to be synchronously transmitted and received simultaneously by controlling three pins: SDO, SDI, and SCK. The operating speed of the PIC was able to be clocked at 20 MHz to produce an SPI speed up to 5 Mbits/s to the memory. The high speed allows data to be successfully written to flash in a short time for practical use.

The PIC also allowed fast transferring of the recorded data to the PC through the Universal Asynchronous Receiver Transmitter (UART) at 1.25Mbaud. The fast transfer speed of data to the computer made the design more user friendly.

D. USB to TTL Converter

Using a USB to TTL converter, the recording device can be connected to a computer for transferring of recorded data. The FTDI Chips’ FT232BM USB UART (USB-Serial) IC was used for its simple and rapid connection of the PIC to the USB interface. Due to the high rate of data transfer, hardware flow control was required so that no data is lost. The FTDI chip
was able to transfer large amounts of data in a reasonable time period for uploading sampled data from the device to the PC as a .wav file. Due to the increasing complexity of the overall circuit, a USB to TTL Serial Adapter [7] module was chosen to simplify the design. This module contains the FT232BM IC, and all the required support circuitry. The transferring of one minute and forty seconds of data took approximately one minute to transfer.

E. Memory

Several factors were considered when choosing the memory. Serial options were weighed against parallel and volatile considerations over non-volatile. The former choice will impact such things as the number of I/O pins utilized on the MCU and the layout of the PCB. The latter choice will affect whether or not you will be able turn off the power of the device and still retain the data. To provide a useful solution that is both portable and energy efficient, non-volatile memory is necessary. This allows the user of the device to turn off the power and still retain the recorded audio material as one would expect the device to work. Serial memory was chosen due to the capability of easily increasing the memory size as opposed to a parallel solution.

For our memory solution, three AT45DB161B flash memory modules were used. The AT45DB161B’s main defining features are the Serial Peripheral Interface (SPI) and the 16 Mbit of bulk flash storage with 2 input buffers each sized 528 bytes [5]. This buffer size corresponds to exactly one page of data which becomes very useful when performing write operations. Flash memory technology is such that it takes much longer to write to the bulk memory than it does to write to the buffer. When the bulk flash memory is being written to, the maximum time to write a buffer to memory is 20 ms. With our desired sampling frequency of 16 kHz, 4 bytes are stored every 62.5 µs. This corresponds to filling up a buffer every 8.25 ms. Seeing that the audio data is continuous, it would be very inconvenient to have to wait 20 ms after every 8.25 ms of recorded audio. A possible solution to this situation is a technique called interleaving.

Interleaving is a way to arrange data in a non-contiguous way in order to increase performance [8]. For our purposes, this means that three separate memory modules are required. This decision stems from the fact that buffer are filled every 8.25 ms, and require the write to bulk flash to be completed by the next time the memory is written to. Equ. 2 can be used to find the required number of memories.

\[
(number \_ of \_ memories) \times 8.25ms \geq 20ms .
\]  

\[ (2) \]

Fig. 4. Digital subsystem schematic
### TABLE I
**COMPONENT VALUES FOR EACH ANALOG SUBSYSTEM**

<table>
<thead>
<tr>
<th>Reference Designator</th>
<th>Value</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>150 nF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C2, C4</td>
<td>68 nF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C3</td>
<td>100 nF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C5, C7</td>
<td>10 nF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C6</td>
<td>3.3 nF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C8</td>
<td>1.5 nF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>Microphone</td>
<td>n/a</td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>16 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R2</td>
<td>3.3 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R3, R7</td>
<td>10 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R4, R8</td>
<td>91 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R5</td>
<td>24 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R6</td>
<td>4.7 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R9</td>
<td>18 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R10</td>
<td>2.7 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R11</td>
<td>20 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R12</td>
<td>2.2 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>U1</td>
<td>OPA4342</td>
<td>n/a</td>
</tr>
<tr>
<td>Microphone</td>
<td>n/a</td>
<td></td>
</tr>
</tbody>
</table>

### TABLE II
**COMPONENT VALUES FOR THE DIGITAL SUBSYSTEM**

<table>
<thead>
<tr>
<th>Reference Designator</th>
<th>Value</th>
<th>Tolerance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>USB-TTL Converter</td>
<td>n/a</td>
</tr>
<tr>
<td>C1, C4, C16</td>
<td>1 µF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C2, C5, C9, C10, C14, C19, C20, C21</td>
<td>0.1 µF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C3, C6, C11, C12, C13</td>
<td>10 µF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C7, C8</td>
<td>22 pF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>C15</td>
<td>47 µF</td>
<td>10%, 25 V</td>
</tr>
<tr>
<td>D1</td>
<td>Standard T1 3/4 LED (Red)</td>
<td>n/a</td>
</tr>
<tr>
<td>D2</td>
<td>Standard T1 3/4 LED (Yellow)</td>
<td>n/a</td>
</tr>
<tr>
<td>D3</td>
<td>Standard T1 3/4 LED (Green)</td>
<td>n/a</td>
</tr>
<tr>
<td>D4</td>
<td>Standard T1 3/4 LED (Red)</td>
<td>n/a</td>
</tr>
<tr>
<td>R1, R2, R3, R4, R10, R11, R12</td>
<td>4.7 KΩ</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R5</td>
<td>130 Ω</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>R6, R7, R8</td>
<td>300 Ω</td>
<td>5%, 1/4 W</td>
</tr>
<tr>
<td>SW1, SW2, SW3</td>
<td>Standard 12mm Pushbutton</td>
<td>n/a</td>
</tr>
<tr>
<td>SW5</td>
<td>Standard Switch</td>
<td>n/a</td>
</tr>
<tr>
<td>U1</td>
<td>PIC16F877AP</td>
<td>n/a</td>
</tr>
<tr>
<td>U2</td>
<td>REF-192 2.5V Reference</td>
<td>n/a</td>
</tr>
<tr>
<td>U3, U4, U5</td>
<td>AT45DB161B</td>
<td>n/a</td>
</tr>
<tr>
<td>U6</td>
<td>AD7654</td>
<td>n/a</td>
</tr>
<tr>
<td>UR1</td>
<td>7805 +5 V Regulator</td>
<td>n/a</td>
</tr>
<tr>
<td>UR2</td>
<td>LD1117V33 +3.3 V Regulator</td>
<td>n/a</td>
</tr>
<tr>
<td>Y1</td>
<td>20MHZ Oscillator</td>
<td>n/a</td>
</tr>
</tbody>
</table>

According to eqn. 2, three memories are required to store the audio data. For explanatory purposes the three memory modules will be labeled mem_A, mem_B, and mem_C.

Each memory module contains two buffers labeled buf_1 and buf_2. The interleaving of the memory works as follows; Buf_1 of mem_A is written to first. As the buffer is writing to bulk flash of mem_A, buf_1 of mem_B is being written to. As buf_1 of mem_B is being written to flash, buf_1 of mem_C is being written to. Even though the buf_1 of mem_A may be busy by writing to bulk flash, buf_2 of mem_A can be filled. By the time this buffer is filled (8.25ms), the memory will be free to write buf_2 to bulk flash. This carries on for the last two memories as depicted in fig. 5. With the interleaving technique in place, audio can be continuously stored to memory. In addition to the functional benefits that interleaving provides, storage capacity is also increased by 200%. This means that approximately one minute and forty seconds of stereo audio data can be stored. Pull up resistors were required on the chip select lines of the memories to prevent a situation were more than one memory could be activated when the microcontroller’s pins are configured as inputs during system power up.

The three AT45DB161B flash memory modules are all communicated with via SPI. The memory works by sending the desired op-code and memory address over the SPI bus to the memory. From here, the memory executes the op-code (be it a write, read, or erase) on the desired memory address. This allows a logical looping structure to incorporate the previously mentioned data interleaving.

#### F. Power Supply

Due to the portable nature of the design, a battery pack was chosen. In order to determine a suitable battery, current measurements were taken of the entire system in the various states. To find average current consumption, values were calculated for an entire recording cycle of one minute and forty seconds of audio, with a twenty second pause and stop, and a complete transfer to the PC. A table of these values is shown in table II. A rechargeable RC battery pack was found to be a suitable fit for this application. Typical 9V RC NiCad

![Buffer Diagram](attachment:buffer_diagram.png)
battery packs have a 1000 mAh capacity, which would provide on the order of 17 record and transfer cycles on a single charge. This value, however, depends strongly on the battery technology, and its respective discharge curves.

A standard toggle switch was used to turn the device on and off. The unregulated source from the battery pack then connected to two linear regulators, providing +3.3V to the flash memories, and +5 V \( V_{DC} \) to the analog subsystem and remaining digital components.

### III. Firmware and User Interface

#### A. User Interface

The voice recorder uses four modes of operation. These are record, pause, stop and transfer shown in fig. 6. When the device is turned on, it begins in stop mode. The record/pause pushbutton begins the recording of voice and turns on the record LED (red). By again pressing the record/pause LED while recording, the device will pause recording and both the record (red) and pause LED (yellow) will light up. Pressing the record/pause LED again will resume recording. Pressing the stop button during recording or pausing will stop the recording and turn off the non-power LEDs. After connecting the device to a PC with the USB and opening a computer program that can accept a binary data stream, pressing the transfer pushbutton transfers a .wav file to the PC. The transfer LED (green) will light up during transferring and turn off when transfer is complete. The device is again in stop mode. Pressing stop during transferring will stop the transfer before completion and turn off the transfer LED. While in stop mode, the device can be turned on and off and maintain the ability to transfer the last recording.

To minimize false switching of modes in the device, firmware pushbutton de-bounce was implemented. When a given pushbutton is polled, the firmware checks if the pushbutton is pushed, then delays for 10 ms to wait until the pushbutton no longer sends a false signal that it has changed position. The firmware then continuously checks if the pushbutton is up and after the pushbutton is up waits another 10ms before changing the mode indicated. This causes the device to enter the correct mode only after the pushbutton is pressed then let go. Since the same pushbutton is used to place the device in the record and pause modes, this ensures that the device will enter the mode desired by the user.

#### TABLE III

<table>
<thead>
<tr>
<th>State</th>
<th>Current Draw</th>
<th>Average Consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stopped</td>
<td>41.00 mA</td>
<td>2.278 mAh</td>
</tr>
<tr>
<td>Recording</td>
<td>127.00 mA</td>
<td>42.333 mAh</td>
</tr>
<tr>
<td>Paused</td>
<td>102.00 mA</td>
<td>5.667 mAh</td>
</tr>
<tr>
<td>Transferring</td>
<td>49.00 mA</td>
<td>8.167 mAh</td>
</tr>
<tr>
<td>Total</td>
<td>58.444 mAh</td>
<td></td>
</tr>
</tbody>
</table>

| Capacity   | 1000.000 mAh |
| Cycles     | 17           |

#### B. Main Loop / Stop State

When the device starts up, it enters into the main loop. The device constantly polls the record and transfer buttons, which will cause it to enter the corresponding state. This is shown by fig. 7. When entering the stop mode from record or pause, the number of pages filled must be saved to allow proper assembly of the .wav file header. After the stop button is pushed from record or pause, the address of the last page accessed is stored to EEPROM from the page counter variable used for these two bytes. The page number is represented as the 14 most significant bits of two bytes as used in the memory op-codes. The EEPROM memory storing the page counter and the flash memory storing the voice data are non-volatile, meaning that this data will be retained when the device is switched on and off in stop mode. The page counter and the voice data contain all information needed to construct the .wav file.

#### C. Record State

During start up of the firmware when the device is first turned on, timer2 is setup to initiate an interrupt every 62.4 \( \mu s \) when activated. This allows us to sample voice using the analog to digital converter at over 16 kHz. Timer2 operates by incrementing a timer register at a chosen division (pre-scaler) of the frequency \( f_{osc}/4 \). This register is then compared to a chosen timer period 15 after every increment. If the timer equals the timer period, the timer is reset to zero for the next increment. Incrementing then continues as normal. An interrupt will occur when the timer matches the timer period a chosen number of times (post-scaler) [6]. The device uses a 20 MHz oscillator and the pre-scaler is set to 1, causing the timer to increment every 0.2 \( \mu s \). By setting the timer period to

![Fig. 6. Device state diagram.](image-url)
0x9C or decimal 156 and setting the post-scaler to 2, the device interrupts every 62.4 µs when timer2 is enabled. To turn on and off the timer, the global interrupt enable is switched on and off while the timer2 enable remains active. When the device enters record mode (fig. 8), the proper variables are initialized and the global interrupt enable is set. While in record mode, the sample and store subroutine (fig. 9) runs every 62.4 µs, each time timer2 interrupts. The firmware toggles the start conversion signal to the analog to digital converter to simultaneously sample both channels of audio data input from the analog circuitry. The firmware then prepares the memory to accept the sample data.

The device checks if a change in buffer is required by comparing a sample counter to the maximum number of samples per buffer of 132. After a buffer is filled, the chip select for that memory is toggled and a new op-code to write the buffer to flash is sent. The chip selects for the three memories are then toggled to activate the next memory in the sequence. A new page is erased on the new memory by sending the corresponding op-code via the SPI. The chip select is then toggled to start a new op-code. The next op-code indicates that a buffer-full of data will next be sent.

When the same page for all three memories are written to, the op-codes for erase page, write to buffer and send buffer to flash are all updated to indicate the next page of the memories. The write to buffer op-code is also changed from using buffer 1 to buffer 2. This allows us to write to a second buffer on each flash memory chip while the other buffer on the same chip is still writing its contents to flash.

Due to the chosen sampling rate of 16 kHz, the device requires a minimum of three flash memory chips in order to allow enough time to write a buffer to flash memory on a given chip before writing another buffer to that flash memory. After updating the op-codes for a new page, the next buffer is selected.

After the memory is ready to receive the data, the firmware checks if the conversion is complete using a control signal sent from the ADC. By preparing the memory before getting the sample from the ADC, the device minimizes waiting for analog to digital conversion to complete. Each sample generates four bytes of channel A low byte, channel A high byte, channel B low byte and channel B high byte. Two control signals are sent from the PIC to switch which byte is output to the data bus, one to switch between channels and one to switch between high byte and low byte.

The four bytes are taken from the ADC through the 8-bit data bus in the above order to match the Little-Endian format used in a .wav file. The ADC sends out its result as an unsigned output varying from 0x0000 to 0xFFFF centered at 0x8000. The most significant bit of the high byte for both channels is flipped before sending that byte to the memory through the SPI interface. This stores the data as a signed value centered at 0x0000 for the .wav format to understand the data. After each byte is moved to the data bus and operated on, the byte is sent to the memory via the SPI bus.

---

**Fig. 7. Main loop / stop state.**

- Start
  - Initialize UART for baud rate and flow
  - Initialize SPI port for bit rate and port
  - Initialize port directions for external devices and user interfaces
  - Check push button for user input
  - Was "Record" button pushed? YES
    - Call "Record" subroutine
  - "Transfer" button pushed? YES
    - Call "Transfer" subroutine
  - No
    - "Record" button pushed?
      - Yes
      - Call "Record" subroutine
    - No

**Fig. 8. Record state**

- Start 'Record' subroutine
  - "Record" LED on
  - Initialize pointers to flash memory address and buffer number
  - Set up timer to interrupt every 62.5us
  - Timer interrupt generated
    - Call "Sample & Store" subroutine
    - Return from interrupt
  - War: "Pause" button pushed? YES
    - Disable interrupts
      - Call "Pause" subroutine
      - Did routine finish with a "stop"?
        - YES
          - Save the number of samples taken
          - "Record" LED off
        - NO
          - Enable interrupts
  - Was "Stop" button pushed? YES
    - Disable interrupts
  - NO
    - Only this section is interruptible by the timer
      - Return from recording Process

After writing to a page of the memory for all three chips, the page counter is compared to the last page of memory. If the last page written is the last page of the memory, a flag is set to indicate that the memory is done and exit the interrupt subroutine. Outside of the interrupt subroutine, if the memory full flag is set, the timer is stopped from interrupting.

**D. Pause State**

While entering pause mode (fig. 10), the timer interrupt is disabled, stopping sampling of the Analog to Digital converter. None of the variables used in record mode are affected by pause mode. While in the pause subroutine, the two pushbuttons of record and stop are continuously checked. A stop flag is set if the stop button is pushed in the subroutine. The subroutine will end when either button is pushed. After exiting the subroutine, the device will restart the interrupt timer if the stop flag was cleared and enter the stop mode if the stop flag was set.

**C. Transfer State**

After the device enters transfer mode (fig. 11), the required counters are initialized to begin looping through each page of the main memory written to. The device reads the page counter stored in EEPROM when stop is pushed. This page counter is then stored as a two byte variable. This value is then decremented by four, to indicate the transfer of one less page. The value is decremented by four as the page counter is represented as the middle 12 bits of the two byte variable. The decremented value is then stored as the maximum page number to transfer to the PC.

---

**Fig. 9. Sample and store interrupt routine.**

---

**Fig. 10. Pause state.**

---

**Fig. 11. Transfer state.**
The device sends a .wav file to the PC by transferring a 44 byte .wav file header (fig. 12) to the PC, then the transferring the voice data as stored in the main memory (.wav file). This .wav file header, with the exception of the bytes indicating file length, is stored in data chunks contained in the code space of the PIC. To construct the header, the file size must be known. There are three memory chips and 132 samples per page. Multiplying the maximum page number to be transferred by 396 then gives the number of bytes of voice data contained in the file. The first four bytes of data send to the PIC are the ASCII characters ‘RIFF’. The next four bytes sent in Little-Endian format is the number of bytes contained in the file after these four bytes. By adding 36 to the number of bytes of audio data, this value is obtained. The next 36 bytes of data sent to the PC contain fields indicating that the file is a .wav file, that the data is linearly scaled, that the file contains two channels, the sample rate is 16 kHz, the byte rate is 64 kHz, and there are 16 bits per sample. The number of bytes of audio data is then sent, followed by the actual audio data.

When transferring the stored voice data to a PC, the last memory page accessed is not transferred to simplify the looping structure of the transfer subroutine and computation of quantities used in the .wav file header. Not transferring this data, 2.5 ms of voice data will be imperceptible to the user. Transferring the last page accessed, which our device does not implement, would require clearing any spurious data from older voice recordings stored within the same page address.

To audio data is read from the memory by incrementing a sample counter until the number of samples equals 132, the size of a page in memory. After the last data is received from a page, the memory chip selects toggle to deactivate the present memory and activate the next memory. An op-code to read a page of data is then sent to the memory and a page of data is again read from the memory. After all three memories corresponding to the present page counter are read, the page counter is incremented and the next memory in the sequence is read from. When the page counter is incremented the value is compared to the last page address that needs to be transferred. If the two are equal, transfer stops and the device enters stop mode.

The PIC sends the data to the PC by communicating with the USB to TTL module using the UART with hardware flow control. The USB to TTL module allows the PIC to treat the USB in the same way as it would a regular serial port. In order to achieve the fast data transfer, modified the driver to run at 1.25MBAud [8]. This allows for data to be transferred to the PC in half the time it takes to record the data. A program on the PC is used to capture the binary data stream sent from the device to a file, and saving it as a .wav file.

IV. TESTING

Current consumption was measured to establish an estimate of the battery life (Table 1). While the value is lower than expected, it is still sufficient for a functional device. The operation of the recorder was verified through actual use.

V. CONCLUSION

The design of a portable stereo audio recorder was successful as it accomplished the specifications set forth. Through the testing of the device, sound was recorded at a higher quality then originally expected. There are a few areas in the project which may benefit from some improvement. The first major area for improvement would be minimizing power consumption. The component count could be reduced by substituting the analog portion with a digital filter, but this would require a more powerful processor. The SPI flash memory could be replaced with a more convenient removable media, thus increasing the storage capacity. The audio storage would also benefit from compression algorithms.

Fig. 12. WAVE file format [10].

A pure tone was played, and recorded by the device. It was transfer to a PC, where it was played back. The resultant .wav file resembled an accurate reconstruction of the input audio with minimal noise and distortion. MATLAB was also used to verify the recorded data’s frequency characteristics. The frequency domain plot showed no noticeable aliasing, and the noise was not significant. Stereo voice data was then recorded at 16 kHz with 16 bit samples.
REFERENCES


