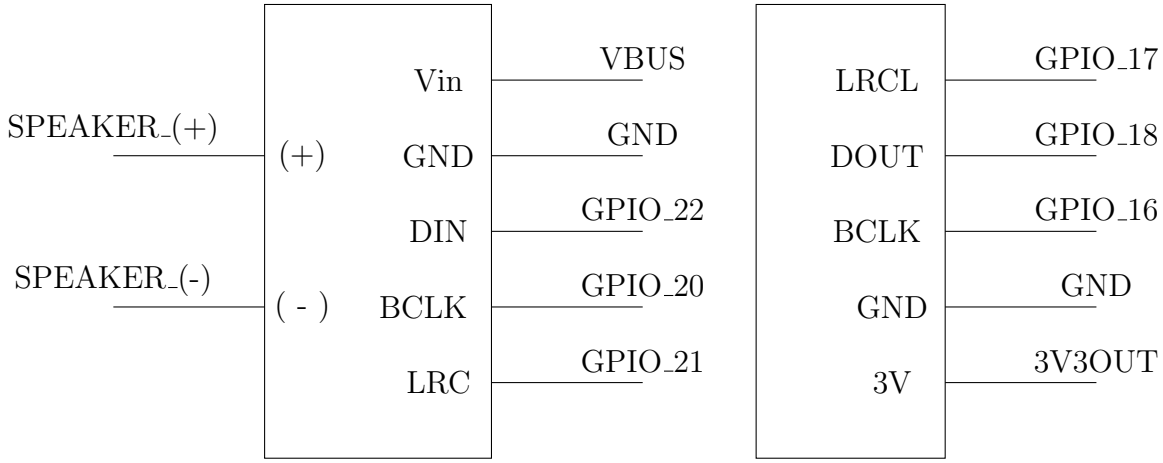
BIOENGR 121: Homework 7

Allen Kuo

March 10, 2025

1. The original code provided was modified so that sampling and output rates were set at 8000Hz and configured to support storage of 8 seconds of data. Here the code stores the first recorded 8 seconds of audio and continuously plays it. Whenever a user wants to overdub more audio, they can hold the BOOTSEL button to add more audio where they want and it overdubs it at that location. This is done by tracking the location of the playback and adding the audio signals from the new recordings at the specific location that the BOOTSEL was pressed.

Circuit Diagram



Code:

```
#include <I2S.h>

// initialize I2S objects for input and output
I2S i2sMic(INPUT);
I2S i2sSpeaker(OUTPUT);

const int sampleRate = 8000;         // set sample rate to 8000Hz
const int duration = 8;              // set record duration to 8 seconds
const int bufferLength = sampleRate * duration; // calculate total buffer size
for 8 seconds

int16_t audioData[bufferLength];
int playbackPos = 0;    // index for playback
int recordPos = 0;      // index for storing new samples

const int overdubButton = 15; // BOOTSEL button for overdubbing

void setup() {
    // start serial communication
    Serial.begin(1000000);

    // configure I2S input (for microphone)
    i2sMic.setDATA(18);               // set data pin for I2S input
    i2sMic.setBCLK(16);               // set bit clock pin for I2S input (LRCLK
is BCLK + 1)
    i2sMic.setBitsPerSample(32);      // set sample size to 32-bit
    i2sMic.setFrequency(sampleRate);  // set input sample rate
    i2sMic.begin();                   // initialize I2S input for audio
recording
```

```
    // configure I2S output (for speaker)
    i2sSpeaker.setBCLK(20);           // set bit clock pin for I2S output (LRCLK
 is BCLK + 1)
    i2sSpeaker.setDATA(22);           // set data pin for I2S output
    i2sSpeaker.setBitsPerSample(32);  // set sample size to 32-bit

    // check if I2S output initialization is successful
    if (!i2sSpeaker.begin(sampleRate)) {
        Serial.println("failed to initialize I2S output");
        while (1);  // stop if output fails
    }

    // prompt user to press BOOTSEL to start recording
    Serial.println("press BOOTSEL to start recording");

    // wait for BOOTSEL button press
    while (!BOOTSEL) {}

    // notify that recording has started
    Serial.println("recording started");
    recordAudio();  // initiate audio recording
    Serial.println("recording complete, overlaying audio in loop");
}

void recordAudio() {
    long startTime = millis();  // get the current time to track the start of
recording
    int idx = 0;                // variable to keep track of the buffer index

    // record audio for 8 seconds
    while (millis() - startTime < (duration * 1000)) {
        if (idx < bufferLength) {
            int32_t leftChannel, rightChannel;

            // read 32-bit stereo audio sample from the microphone
            i2sMic.read32(&leftChannel, &rightChannel);

            // store the left channel 16-bit sample in the buffer
            audioData[idx] = leftChannel >> 16;
            idx++;  // move to the next buffer position
        }
    }
}

void loop() {
    // loop through the audio buffer for playback
    for (int i = 0; i < bufferLength; i++) {
        int16_t currentSample = audioData[playbackPos];  // get the current
audio sample for playback

        // check if overdub button is pressed
        if (digitalRead(overdubButton) == LOW) {
            int32_t leftChannel, rightChannel;

            // read a new sample from the microphone
            i2sMic.read32(&leftChannel, &rightChannel);
            int16_t newSample = leftChannel >> 16;  // extract the left channel
sample

            // mix the current sample with the new sample
            currentSample = combineSamples(currentSample, newSample);

            // update the buffer with the mixed sample
            audioData[playbackPos] = currentSample;
        }

        // output the mixed sample to the speaker
        i2sSpeaker.write32(currentSample << 16, currentSample << 16);

        // increment the playback position and loop back if end of buffer is
reached
        playbackPos++;
```

```
            if (playbackPos >= bufferLength) {
                playbackPos = 0;  // reset to the beginning for looping playback
            }
        }
    }

    // function to combine two audio samples\
    int16_t combineSamples(int16_t oldSample, int16_t newSample) {
        int32_t combined = (int32_t)oldSample + (int32_t)newSample;  // sum the old
    and new samples

        // clamp the combined sample to avoid overflow/underflow
        if (combined > 32767) {
            combined = 32767;  // limit to maximum 16-bit value
        }
        if (combined < -32768) {
            combined = -32768;  // limit to minimum 16-bit value
        }

        return (int16_t)combined;  // return the combined sample as 16-bit
    }
```

Picture: