Lab 5 - Audio

Goals

- Interact with audio on the Microbit
- Play arbitrary tones using PWM
- Play analog waveforms using PWM

Equipment

- Computer with build environment
- Micro:bit and USB cable

Documentation

- nRF52833 datasheet: https://infocenter.nordicsemi.com/pdf/nRF52833_PS_v1.5.pdf
- Microbit schematic: https://github.com/microbit-foundation/microbit-v2-hardware/blob/main/V2/MicroBit_V2.0.0_S_schematic.PDF
- Lecture slides are posted to the Canvas homepage

Github classroom link: https://classroom.github.com/a/4cxn_9dK

Lab 5 Checkoffs

You must be checked off by course staff to receive credit for this lab. This can be the instructor, TA, or PM during a Friday lab session or during office hours.

- **Part 2: Audio**
    a. Demonstrate the pwm_square_tone code and application
    b. Demonstrate the pwm_sine_tone code and application
    c. Demonstrate the record_and_play code and application

Also, don't forget to answer the lab questions assignment on Canvas.

Lab Steps

Part 1: Setup

1. Find a partner

- Rule: you can pick any partner you want, but you can't pick the same partner twice
- You MUST work with a partner
    - We don't have enough computers otherwise

2. Set up your computer

- You can also use your personal computer if it's set up and you prefer
- Log into Windows
    - Password: 327-19s
- Open VMWare Workstation Player
- Open the virtual machine: CE346
    - It'll load for a minute and then ask you to log in
    - Password: microbit

3. Create your Github assignment repo

- There is a github classroom link on the first page of this document. Click it!
- Pick a team name
- Pick your partner
- Generally, do what it says
- At the end, it should create a new private repo that you have access to for your code
    - Be sure to commit your code to this repo often during class!
    - If your computer crashes, all your files WILL BE LOST unless committed and pushed to Github.
- That link might 404. If it does, you first have to go to https://github.com/nu-ce346-student and join the organization

4. Create a personal access token

- If you still have your personal access token from prior weeks, you can skip this part
- Go to your github profile -> Settings
    - Then Developer Settings on the left
    - Then Personal Access Tokens on the left
    - The click the Generate New Token button on the top right
- Add a note that is the name of this token (not important, type anything)
- **IMPORTANT:** check the repo checkbox below the name of the token
- Then scroll to the bottom of the screen and click the Generate Token button
- This will create a password that allows you to clone repos
    - It will only show this once, so copy-pasting it into a google doc in your personal drive would probably be useful
    - It will be in gray at the top of the screen

5. Clone your lab repo locally

- Open a terminal
- You can clone the repo right to the home directory of the computer
    - Remember, everything in this VM will disappear when it powers down
- At the top right of your shiny new private repo, there is a green button that says code. Copy the HTTPS link to your git repo from there.
- **IMPORTANT:** run this command and don't forget the flags

  ```
  git clone <YOUR-REPO-HTTPS-LINK-HERE> --recursive --shallow-submodules
  ```

    - Remember to include both of those flags!
    - Recursive is necessary to clone submodules
    - Shallow submodules makes it like five minutes faster to run
    - If you forgot the flags, the command to fix this is:

      ```
      git submodule update --init --recursive --depth 1
      ```

Part 2: Audio

1. Play square wave tones over speaker with PWM

- cd into `software/apps/pwm_square_tone/` You'll be editing `main.c` here

- Initialize the PWM peripheral

  Use the `nrfx_pwm_init()` within the `pwm_init()` function

  - [Documentation for the nrfx_pwm driver]()
  - You will need to make an `nrfx_pwm_config_t` local variable to pass into the initialize function
    - `output_pins` is an array of four GPIO pin numbers, use SPEAKER_OUT for the speaker pin and NRFX_PWM_PIN_NOT_USED for unused pins
    - `base_clock` should be 500 kHz
    - `count_mode` should be Up
    - `top_value` is the default value for `COUNTERTOP` which we will overwrite later (pick anything for now)
    - `load_mode` should be Common
    - `step_mode` should be Auto
  - We don't care about callback events for the PWM, so passing in `NULL` for the callback is fine

- Play a tone using the PWM peripheral

  To do so, you'll need to fill in the code for `play_tone()` and call it from `main()`

  - To stop the PWM, use `nrfx_pwm_stop()`
  - Use `NRF_PWM0->COUNTERTOP` to access the `COUNTERTOP` register
    - `COUNTERTOP` is the number of ticks in a PWM square wave cycle (ticks/cycle) so you'll pick a value based on the PWM input clock (ticks/second) and the desired output frequency (cycles/second)
  - The PWM sequence data array has already been created for you globally. You will need to edit the value in order to set a 25% duty cycle
    - Take a look at `sequence_data` at the top of the file
    - Note: you don't care if it's left or right alignment here, just set a value so that the toggle occurs at 25% of the way counting up to `COUNTERTOP`

        - 25% will make it audible but quiet. Going to 50% will make it louder.

    - To start the PWM, use `nrfx_pwm_simple_playback()`
        - The instance is defined for you at the top of the file
        - You'll want to give it a flag so it plays the note indefinitely.

    - Here is a list of [frequencies for the tones of a piano]()

- Play multiple tones to form music

    - You can play whatever you want as long as it consists of at least four distinct notes, so feel free to play a short jingle or song

    - If you don't have any particular music in mind, the comments in main right now guide you through the A major arpeggio scale (A_4, C#$_5$, E_5, A_5), which is sufficient

    - You almost certainly want to stop the PWM after the last note so it doesn't continue playing it forever

- **Checkoff**: demonstrate your code and app to course staff

    - If it is too quiet to hear, you might have to bump duty cycle back up to 50% for the demonstration
    - *Question:* how did you determine COUNTERTOP, and what are the units of each part of that equation?

2. Play sine wave tones over speaker with PWM

Differently in this part (and part 3), we will now play samples from a buffer at a constant data rate. This more closely simulates how a DAC would work: there is some rate that analog samples are being produced at, with the analog value changing each time. We had to pick some sampling rate, and choose 16 kHz which is fast enough to handle *most* audible frequencies.

- cd into `software/apps/pwm_sine_tone/` You'll be editing `main.c` here

- The sine wave computation has been provided for you

  Look through the code already in the file. You shouldn't have to change any of this, but you will need to use the `sine_buffer` for creating your own output samples.

  It takes about 50 μs to calculate a single sine wave value. So for large buffer values, this can take a noticeable amount of a second to finish. We calculate the sine wave once at boot to make this more efficient, and then reuse the sine data to create the samples for our tones.

- Initialize the PWM peripheral

  Edit the function `pwm_init()` to do so

  - This will be almost the same as the initialization from the previous part of the lab except that the PWM clock should be set to 16 MHz

  - Also you will need to actually set a COUNTERTOP value here
    - In this example we have a constant frequency that we're playing PMW periods at, which is defined as the `SAMPLING_FREQUENCY`
    - For each data sample period, two PWM periods should play
    - We will later set the value of repeats so each PWM period is played twice

- Enable playing sine wave data over the speaker using PWM

  Edit `play_note()` to do so

  - To create each sample, you'll step cumulatively through the sine wave buffer and pull out the current sample value
    - Imagine a buffer of 10 samples of a sine wave for the entire sine period of 2π radians. If you want to go twice as fast as this, you step by twos and use every other sample.

        - Here, we have 500 samples for a single sine wave period, and we need to step through them to create our own output samples based on the desired note frequency. The step value has been provided for you
        - Iterate through the `samples` buffer copying over values from `sine_buffer`
        - Make sure to wrap your cumulative steps back around if it would go past the SINE_BUFFER_SIZE.
            - Don't set it to zero though! It should be a type of modulus operation (except that you can't actually modulus floats)
            - Setting it to zero would be a discontinuity in the sine wave output, which will sound sort of like a click or wobble in the sound.

    - Next, create the PWM sequence. You can copy some of this from the previous part of the lab.
        - You should set repeats to 1, for a total of two PWM periods per sample
        - Note that the value is the number of repetitions after the first play (so the total number of times a sample is played will be `1+repeats`)

    - Finally, start the playback. You should loop the playback forever with the proper flag

- Pass in a maximum scale value to `compute_sine_wave()` in `main()`
    - The sine wave values range from 0 to `max_value`
    - Use COUNTERTOP as the maximum value to make your life simpler, as it means the sample data will already be portions of COUNTERTOP

- Play multiple tones to form music

    - The easiest choice here is to do whatever you did for the previous part of lab

    - You again almost certainly want to stop the PWM after the last note so it doesn't continue playing it forever

Checkoff: demonstrate your code and app to course staff

3. Record audio and play it back over the speaker

- cd into `software/apps/record_and_play/` You'll be editing `main.c` here

- The ADC side of the application has been provided for you

  Look through the code already in the file. You shouldn't have to change any of this, but you'll definitely have to interface with it, so understanding what it is doing will be helpful. Particularly note the sampling rate, as that will determine PWM timing.

- Initialize the PWM peripheral

  Edit the function `pwm_init()` to do so

  - This will be almost the same as the initialization from the previous part of the lab except that the PWM clock should be set to 16 MHz

  - Also you will need to actually set a COUNTERTOP value here
    - For each data sample period, one or more PWM periods should play
    - This depends on the value of repeats you choose when setting up the PWM sequence

- Play audio samples over the speaker using PWM

  Edit play_audio_samples_looped() to do so

  - First, modify each sample in place so it represents a PWM duty cycle rather than ADC counts. There is a `#define` for the maximum ADC value at the top of the file
    - Again, you don't have to worry about alignment here. Just set each value so that the toggle occurs at a duty cycle proportional to how large of an analog value you read

  - Next, create the PWM sequence. You can copy some of this from the previous part of the lab.
    - You almost certainly want a non-zero repeat count. If repeat is zero, each analog sample is represented by exactly one PWM period. Instead, we want each sample to be represented by a few periods so that there is more than a single waveform to average energy across
    - Increasing the repeat count will require modifying `COUNTERTOP` however. Otherwise your playback will be at an incorrect frequency. All repeated periods should happen within a single sample period.

    - Note that the value is the number of repetitions after the first play (so the total number of times a sample is played will be `1+repeats`)

  - Finally, start the playback. You should loop the playback forever with the proper flag

- Test the record and play application

  That should be everything needed to make the application work. You'll need to speak relatively loudly and pretty close to the Microbit in order for the output to be audible.

  - If the output is loud and the correct speed, but distorted, you may be clipping the signal (it may have reached max). Try speaking a little softer

  - If the output is slow, you likely miscalculated the `COUNTERTOP` value, leading to the output playing at a slower frequency than it was recorded

  - You might want to make it so that pressing a button stops the PWM from playing so that it doesn't keep repeating you forever

- **Checkoff**: demonstrate your code and app to course staff