LAB 11:

Timers & Digital Audio

Provided Files

- buttonpress.c, buttonpress.h, buttonpress.wav
- buttons.bmp, buttons.c, buttons.h
- digitalSound.c, digitalSound.h
- Example.gba
- examplesong.mp3
- gba.c, gba.h
- gbaplayer.bmp, gbaplayer.c, gbaplayer.h
- Makefile
- mode0.h
- print.c, print.h
- README.md
- screen.bmp, screen.c, screen.h
- sprites.c, sprites.h

Files to Edit/Add

- .vscode
  - tasks.json
- main.c
- play.c, play.h
- examplesong.c, examplesong.h, examplesong.wav
- 2+ extra songs of your choice! (.c, .h, .wav)

Instructions

In this lab, you will be completing several different TODOs, which will, piece by piece, implement digital audio using an mp3 player! Your code may not compile until you complete an entire TODO block, at which point the game should compile with a new component of the final outcome (unless otherwise specified).

TODO 1 - Setting up timer interrupts

In this TODO, you'll be setting up timer 2 and timer 3 to trigger interrupts at the correct intervals.

TODO 1.0

- In `setupInterrupts()`, turn on the bits corresponding to these interrupts:
  - VBlank
  - Timer 2
  - Timer 3

TODO 1.1

- Set up timer 2 and timer 3 in `playSong()`.
  - For timer 2, we want a timer that triggers an interrupt every second.
    - First, deactivate timer 2 using its control register.
    - Next, set the value of the data register to (65536 - 16384).
      - This is going to be the initial value of our timer (what it counts *up* from.
    - Button A is the button used for pausing/unpausing the song. If the MP3 player is not paused, reactivate the timer using these bits:
      - `TM_FREQ_1024` (sets frequency of timer to 16384 Hz)
      - `TM_IRQ` (enables interrupts)
      - `TIMER_ON` (turns timer on)
  - For timer 3, we'll make a cascading timer that triggers an interrupt once every 60 counts from timer 2 (one interrupt per minute).
    - Deactivate the timer.
    - Set the value of the data register to (65536 - 60).
    - If the MP3 player is not paused, reactivate the timer with the same bits as timer 2, but replace `TM_FREQ_1024` with `TM_CASCADE`.

TODO 1.2

- Go to `update()` in `play.c`.
  - Use the control registers for both timer 2 and timer 3 to disable the timers when the song is paused and re-enable them when the song is unpaused.
  - *Hint:* recall how you enabled and disabled timers in the previous TODO!

TODO 1.3

- In `interruptHandler()`, handle the timer 2 interrupt, which should occur once every second.
  - The `second` variable determines the number for seconds displayed on the MP3's timer. This variable should be incremented whenever timer 2 triggers an interrupt, with a maximum value of 59. If the value of `second` is greater than 59, it should be reset to 0.

TODO 1.4

- Handle the timer 3 interrupt, which should occur once every minute.
    - The `minute` variable determines the number for seconds displayed on the MP3's timer. This variable should be incremented whenever timer 2 triggers an interrupt, with a maximum value of 99. If the value of `minute` is greater than 99, it should be reset to 0.

TODO 2 – Sound setup + counting VBlanks

In this TODO, you will be updating how the VBlank interrupt is handled in the `interruptHandler()` function in `main.c`. The code we write here will help us to avoid sound cornerface (sounds generated by random data in memory) when digital audio is added to the program. If you're unsure as to what the variables in this section represent, refer to the `SOUND` struct in `digitalSound.h`.

TODO 2.0

- Handle `soundA` playing in the interrupt handler when a VBlank interrupt occurs.
    - This is where we want to determine if we need to stop playing `soundA` or not, so we'll only need to follow these steps if `soundA` is currently playing.
    - First, increment `vBlankCount` for the sound.
    - If the sound's `vBlankCount` is greater than or equal to its `durationInVBlanks`, we will know that we've reached the end of the song. You need to handle two cases here:
        - If the sound loops, restart the song using the `playSoundA()` function, using struct members from `soundA` as arguments.
        - If the sound does not loop, set `isPlaying` to false and turn off the timer and DMA channel used by `soundA`.

TODO 2.1

- Handle `soundB` playing in the interrupt handler when a VBlank interrupt occurs.
    - This logic should be nearly identical to that of handling `soundA`, now using `soundB` and its corresponding DMA channel and timer.

TODO 2.2

- In the `initialize()` function, call `setupSounds()`.

TODO 2.3

- Include `buttonpress.h` at the top of `main.c`. After building, take a look in `buttonpress.h` to see what variables can be used from this file.
    - `buttonpress.wav` is a sound effect that has been provided for you. The Makefile is configured to generate `buttonpress.c` and `buttonpress.h` using wav2c whenever you build the project.

TODO 2.4

- In the `main()` function of `main.c`, call the `playSoundB()` function using the appropriate variables from `buttonpress.h`. This sound effect should not loop.

Build and run. You'll know that everything works so far if you hear a clicking sound whenever you press one of the buttons on the music player!

TODO 3 – Converting + playing sound files

For this section, you'll need to make use of Audacity to convert `examplesong.mp3` into a .wav file with the correct format to run on GBA. After the file has been converted, you'll be able to play it on the music player!

TODO 3.0

Note: These steps may vary slightly between different versions of Audacity.

- Exporting the file
  - Open `examplesong.mp3` in Audacity.
  - Select the track.
  - Select Tracks -> Mix -> Mix Stereo Down to Mono.
  - Set the Project Rate (Hz) in the lower-left corner of the screen to 11025.
  - Go to Tracks -> Resample and choose 11025 as the new sample rate.
  - Select File -> Export -> Export Audio.
  - Be sure to save it in the lab folder with the rest of your .c and .h files!
  - Under the "Save as Type" dropdown, select "WAV (Microsoft)".
    - If there is an "Options" button, press it.
    - Name the file `examplesong.wav`.
    - You should see a dropdown menu for "Encoding". Set this option to "Unsigned 8-bit PCM".
    - When the "Edit Metadata" screen pops up, press the "Clear" button. Confirm that the metadata in the file has been removed and press "OK".
- Build your project. If `examplesong.c` and `examplesong.h` are generated in the build, then the export was successful.

TODO 3.1

- Include `examplesong.h` at the top of `play.c`.

TODO 3.2

- In the `init()` function of `play.c`, set the struct members for `songs[0]`.
  - `sampleRate`, `length`, and `data` should be assigned to their counterparts in `examplesong.h`.
    - You can cast `examplesong_data` to a `signed char*` to avoid compiler warnings!

- Optional: Choose a title for the example song! Feel free to write whatever you want here (just don't make it anything inappropriate, rude, or offensive!).
  - Note that not every character is available to use in titles. Check `screen.bmp` to see the full list of characters you can use.
- Optional: Choose a color to represent this song.
  - The color you choose will be placed into the palette and the sprite palette at runtime each time this song is loaded in the music player. You won't need to write this part, but it may be a good idea to find that code and understand how it works.

TODO 3.3

- In the `playSong()` function of `play.c`, call `playSoundA()` with the following arguments:
  - Use the `data` and `length` members from the `SONG` struct at index `s` of the `songs` array.
    - `songs[s]` is the current song that should be playing.
  - The `looping` attribute in `soundA` determines whether or not the song should loop.

Build and run. You should be able to turn on the music player and hear the example song now! However, you won't be able to pause or stop it by turning off the music player (you'll add those features in the next section). Also, the timer should start running -- if it isn't, revisit the steps from TODO 1.

TODO 4 – Pausing, unpausing, and stopping

We want to be able to control when our music plays and when it doesn't, so we have three functions to handle this in `sound.c`. In this TODO, we will use those functions to that end.

TODO 4.0

- Find the `update()` function in `play.c`.
  - Call `unpauseSounds()` or `pauseSounds()` when the A button is pressed, depending on whether or not sounds are paused.

TODO 4.1

- Find the `playSong()` function in `play.c`.
  - Call `pauseSounds()` if `paused` is true. `paused` is the variable in `play.c` that keeps track of whether or not the music is paused.

TODO 4.2

- Find the `goToOff()` function in `main.c`.
  - Call `stopSounds()` at the end of the function.

Build and run. After initially turning on the music player, the music should now be paused. Pressing the pause/play button should toggle pausing and playing, and turning off the music player should now stop the music as well.

TODO 5 – Your turn!

Now that everything is in place, it's time to add more songs to the music player! It's entirely up to you as to what songs you choose to upload (as long as it isn't anything inappropriate, rude, or offensive).

TODO 5.0

- Find two songs that you want to add to the music player and download them onto your device.
    - In general, the easiest way to obtain sounds or music is to find videos on YouTube and convert them to .mp3 files. You don't necessarily need .mp3 files as Audacity can work with many types of sound formats, but there's plenty of free YouTube to MP3 converters on the internet at your disposal. It will be more difficult to work with .wav files for this, though. If you save a .wav file that you haven't processed through Audacity yet in your project folder, wav2c will recognize it as a usable file and try to build using it.

TODO 5.1

- Convert these files to usable .wav files using the instructions from TODO 3.0.

TODO 5.2

- Include the header files generated by wav2c at the top of `play.c` after successfully importing the files into your project and building.

TODO 5.3

- Set the struct members of `songs[1]` and `songs[2]` in a similar manner to `songs[0]`, now using the variables from the songs you imported.

TODO 5.4

- If you want to add more songs, alter the value of the `SONGCOUNT` macro in `play.h`. This variable determines the length of the songs array, so making it larger will give you room to import more songs (at least as many as the GBA can store in memory)!

At this point, you should be done! ***Build and run your Project.gba and compare it against the example program.***

You will know your lab runs correctly if:

- Your program behaves the same as the provided Example.gba.

Submission Instructions:

Ensure that **cleaning** and building/running your project still gives the expected results. **Please reference the last page of previous assignments for instructions on how to perform a "clean" command.**

Zip up your entire project folder, including all source files, the Makefile, and everything produced during compilation **(including the .gba file)**. Submit this zip on Canvas. Name your submission Lab11_LastnameFirstname, for example:

> “Lab11_BerryGordy.zip”

It is your responsibility to ensure that all the appropriate files have been submitted, and that your submitted zip can be opened and everything cleans, builds, and runs as expected.