**COMP 2659 Course Project – Stage 7: Music and Sound Effects**

Released: Thursday, March 7, 2019
Target Completion Date: Friday, March 15, 2019 (at the latest)

Overview

In this stage you will add music and sounds effects to your game. First, you will implement a low-level PSG library. Then, you will use the PSG library to implement a "music" module. Finally, you will build an "effects" module on top of the PSG library.

An overview of the Atari ST's PSG is given in the lecture notes, and introductory exposure to programming the chip was given in a recent lab. It is your responsibility to review this material, and to consult the YM2149 documentation and an ST memory map for the important technical details.

Requirement: Provision of Low-Level PSG Routines

With the exception of the `Super` system call, you may not use any operating system support for the implementation of these routines, nor for the "music" and "effects" modules that are layered on top of it. All sound must be generated by programming the PSG chip directly.

You must first create a file module named `psg`. It must provide the following generic (game-independent) PSG control functions:

- `void write_psg(int reg, UINT8 val);`

  Writes the given byte value (0-255) to the given PSG register (0-15). This is a helper routine to be used by the other functions in this module.

- `UINT8 read_psg(int reg);`

  **Optional** – useful for testing purposes.

- `void set_tone(int channel, int tuning);`

  Loads the tone registers (coarse and fine) for the given channel (0=A, 1=B, 2=C) with the given **12-bit** tuning.

- `void set_volume(int channel, int volume);`

  Loads the volume register for the given channel.

- `void enable_channel(int channel, int tone_on, int noise_on);`

  Turns the given channel's tone/noise signals on/off (0=off, 1=on).

- `void stop_sound();`

  Silences all PSG sound production.

Each function must check its input parameters. If an input value is out-of-range, the function must ignore the request.

Requirement: Music

During game play a simple song must loop continuously[1]. Use PSG channel A for playing the melody[2].

One simple way to implement a melody is as a circular array of notes (each with pitch data and maybe duration data). Use the 70 Hz vertical blank clock to time the playback of notes.

You must create a file module named `music` which provides the necessary functionality. The following design is suggested:

- `void start_music();`

  Begins the playing of the song by loading the data for the first note into the PSG.

- `void update_music(UINT32 time_elapsed);`

  Advances to the next note of the song **if necessary**, as determined by the amount of time elapsed since the previous call. The time elapsed is determined by the caller. It is intended that this value is equal to the current value of the vertical blank clock, minus its value when the function was last called.

If the above design approach is taken, the main game loop could be updated as illustrated below:

```
initialize model
render model (first frame)
start music
set quit = false

repeat until quit
     if input is pending
          process async event                ← update model
     if clock has ticked
          process sync events                ← update model based on time elapsed
          render model (next frame)
          update music                       ← based on time elapsed
```

Requirement: Sound Effects

At least two significantly different sounds effects must be generated during game play. At least one sound effect must be due to a synchronous event (or due to a condition-based event triggered by a synchronous event). At least one must be due to an asynchronous event (or due to an event triggered by an asynchronous event). Use PSG channel C for playing sound effects[3].

Sound effects must make use of the PSG's envelope generator in some way.

[1] You can have different songs for different game stages if you want to get fancy. This is not required.
[2] If you wish to include a second musical line (e.g. a harmony), use channel B. This will limit sound effects to channel C.
[3] If you wish the playback of two sound effects to overlap in time, also use channel B. This will limit music to channel A.

The following two functions must be added to the `psg` module:

- `void set_noise(int tuning);`

  Loads the noise register with the given tuning.

- `void set_envelope(int shape, unsigned int sustain);`

  Loads the PSG envelope control registers with the given envelope shape and 16-bit sustain.

You must create an `effects` file module. It must have one `play_`*`effect`* function for each type of sound effect in your game.

The event handling functions in the `events` module must be modified to play the relevant effect whenever its triggering event occurs.

Requirement: Test Drivers

Your PSG module must be accompanied by a test driver program which invokes and thoroughly exercises its routines. Similarly, the "music" and "effects" modules must each be accompanied by a test driver.

Add tests for each function you write as you go. If you avoid prompt testing, it will be difficult to build robust code.

Other Requirements

Your code must be highly readable and self-documenting, including proper indentation and spacing, descriptive variable and function names, etc. No one function should be longer than approximately 25-30 lines of code – decompose as necessary. Code which is unnecessarily complex or hard to read will be severely penalized.

For each function you develop, write a short header block comment which specifies:

1. its purpose, from the caller's perspective (if not perfectly clear from the name);
2. the purpose of each input parameter (if not perfectly clear from the name);
3. the purpose of each output parameter and return value (if not perfectly clear from the name);
4. any assumptions, limitations or known bugs.

At the top of each source file, write a short block comment which summarizes the common purpose of the data structures and functions in the file.

You should add inline comments if you need to explain an algorithm or clarify a particularly tricky block of code, but keep this to a minimum.