Final Project Example

YM2149 Programmable Sound Generator Emulator

Overview

The YM2149 (a variant of the AY-3-8910) is a Programmable Sound Generator (PSG) that was produced in the 70s and 80s for generating sound effects and music. It was created in a time when most CPUs were too slow to generate audio by themselves, so dedicated chips were required. The AY-3-8910 was used in many microcomputers and game consoles. The YM2149 in particular was used in the Atari ST, so this project plays back Atari ST music.

The purpose of this example final project is to demonstrate a large project that a student could do after completing all of the labs. Additionally, this project demonstrates my understanding of the PSoC 5LP after creating all of the labs.

YM2149 Capabilities

The YM2149 has the following capabilities.

- Three voices. Each voice has:
    - Variable square wave tone generator. 12 bit resolution for choosing the tone.
    - Fixed volume. 4 bits of resolution. Can be substituted for the volume envelope.
    - Mixer. The inputs to the mixer are:
        - This voice's tone generator
        - The noise generator
        - The fixed volume or volume envelope.
    - The voice and noise mixer inputs can be enabled or disabled independently. If both are enabled, the mixer combines them using an OR gate. The volume is either the fixed volume or the volume envelope, never both. The volume is applied if the mixed square wave signal is high, otherwise the volume is 0.
- One noise generator. Generates a random square wave using a pseudo-random sequence. Can be mixed in to any of the voices.
- One volume envelope generator. There are 16 bits to represent the period of the envelope and 10 different envelope shapes. The resolution of the envelope's volume is 5 bits (the AY-3-8910 uses 4 bits). See Figure 1 for the shapes the envelope generator can take.
- Three Digital-to-Analog Converters (DACs). Each DAC generates the analog output of each voice ranging from 0 to 1 volt. The three voices are combined into one output pin. These DACs produces a logarithmic output from a 5-bit input (4-bits for the AY-3-8910), to match how human ears respond to volume. See Figure 2 for the volume curve.
- Two 8-bit bidirectional I/O ports. Not implemented or used in this project.
- There are a total of sixteen 8-bit registers that control the device.

See the AY-3-8910 manual [1] and YM2149 manual [2] for full details, including how each feature is controlled. The AY-3-8910 is nearly identical to the YM2149, but the AY-3-8910 manual has more details.

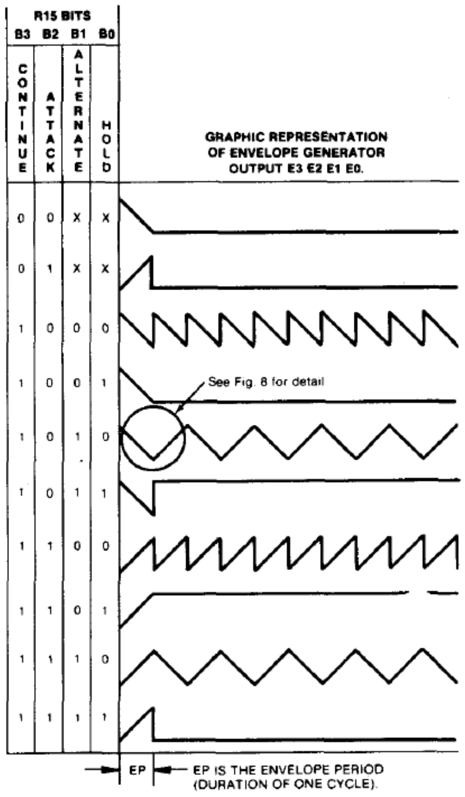


Figure 1: The shapes of the volume envelope [1]

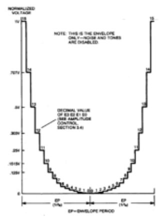


Figure 2: The DAC output voltage curve for the given 4-bit volumes in the AY-3-8919 [1]. Note that the YM2149 has the same curve, but with 32 steps instead of 16 (5-bit instead of 4-bit). See Table 1 below for the exact values given in this curve.

Table 1: The exact voltage responses shown in Figure 2.

Level	Output (V)
15	1.0
14	0.707
13	0.5
12	0.303
11	0.25
10	0.1515
9	0.125

Atari ST Specific Details

The Atari ST is a computer released for personal use in 1985 and was available through the early 1990s. Along with being used for various applications, it was a popular platform for games. Music production was also popular since it had MIDI ports.

Here are some details about how the Atari ST in particular used the YM2149.

- The input clock to the YM2149 is 2 MHz. Other devices used different input clocks.
- The Atari ST updated registers in the YM2149 fifty times per second.
- Some music producers took advantage of special methods to create special effects in the YM2149. For example, because each voice has a 4-bit fixed volume register, the YM2149 can produce 4-bit audio samples if the volume is changed rapidly. The Atari ST is capable of doing this, but the exact methods used for this and other special effects are not well documented. Therefore, these effects are not supported in this project. Making these effects possible should be possible by just changing the software, and maybe adding a new interrupt for timing the special effects.

YM Music Files

In order to reproduce music from the Atari ST on modern systems, YM music files were created to store music playback data that can be played back in a program that emulates the YM2149 and the way the Atari ST made use of the chip [3]. These music files include fifty YM2149 register dumps per second, which are used by this project. There is also some header information that isn't necessary for this project, although some of it seems to be data for the special effects.

These files are compressed with LHA, and the register data itself is interleaved (that is, all register 0 dumps are in a row, then all register 1s, etc.) to improve compression performance. Because a decompressed YM file is bigger than the SRAM available on the PSoC and solving this memory issue would be a challenge, these YM files were reformatted so it would be easier to parse and buffer them off of an SD card. Several of these reformatted YM files are provided as .txt files in the Converted folder in YMTools.zip.

The programs necessary to reformat YM files are provided with this project example in YMTools.zip. These two programs are necessary:

- lha.exe
- ymdump.exe

In order to convert a YM file to what is expected by this project:

1. Ensure the YM file is not in the same directory as lha.exe.
2. Open a Command Prompt and navigate to the directory lha is in.
3. Extract the YM file using the command `lha -x <YM filename>`
   Optionally, YM files can usually be opened and extracted with many file archiving programs, including 7zip.

4. The decompressed file is put into the lha directory. Note that the decompressed file name may be the same or different than the original filename depending on the name of the file before it was compressed, which is why the YM file needs to be in a different location.
5. Important: if the extracted file has an extension of .bin, this is an old YM format that is not supported by ymdump.
6. Convert the extracted YM file using the command `ymdump <extracted filename>`
7. The extracted YM file can now be deleted, and the resulting txt file is put on the SD card.

Implementation Summary

There are two forms of input and two forms of output in this program. The inputs are:

- The two pushbuttons on the development kit. The left button selects a file to play and the right button plays the file. There is no stop button, but the reset button can be used to stop playback.
- An SD card for holding the music files. SD cards can be used over SPI, which is a peripheral interface standard. The microSD breakout board from SparkFun (https://www.sparkfun.com/products/544) is used, but any SD or microSD breakout board with SPI can be used.

The outputs are:

- The character LCD. This display shows the currently selected music filename while waiting for the user to either request the next file or push the play button. While the music is playing, the display visualizes the music based on the tone and volume of each voice.
- An analog output pin for the music output. This is connected to a 3.5mm headphone jack. The best headphone jack to use is one that can be plugged into a breadboard, like this one: http://www.adafruit.com/products/1699

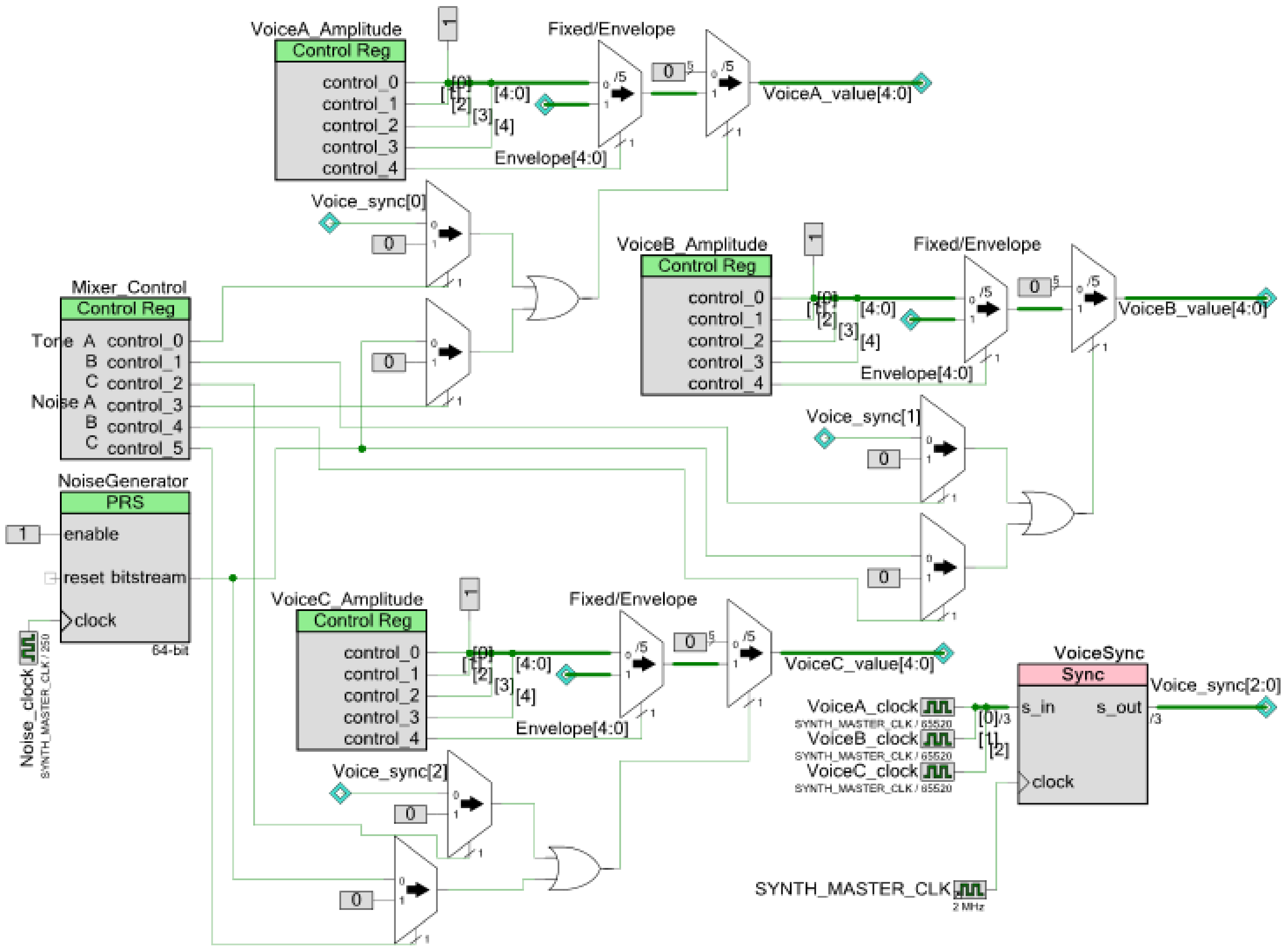
Hardware Implementation Details

All of the music generation is implemented in the hardware. Because there are many components necessary for generating the music, the design has been split over multiple schematic pages.

The green hollow diamond shapes found on the following schematics are called sheet connectors. These are used for connecting wires between or within schematics. Names given to wires are used to determine what sheet connectors are connected to each other. Names given to wires should be adjacent or near them.

There is one important design-wide component: SYNTH_MASTER_CLK. This is a design-wide clock that all music generation components use. It is set to 2 MHz, since that is the clock used for the YM2149 in the Atari ST.

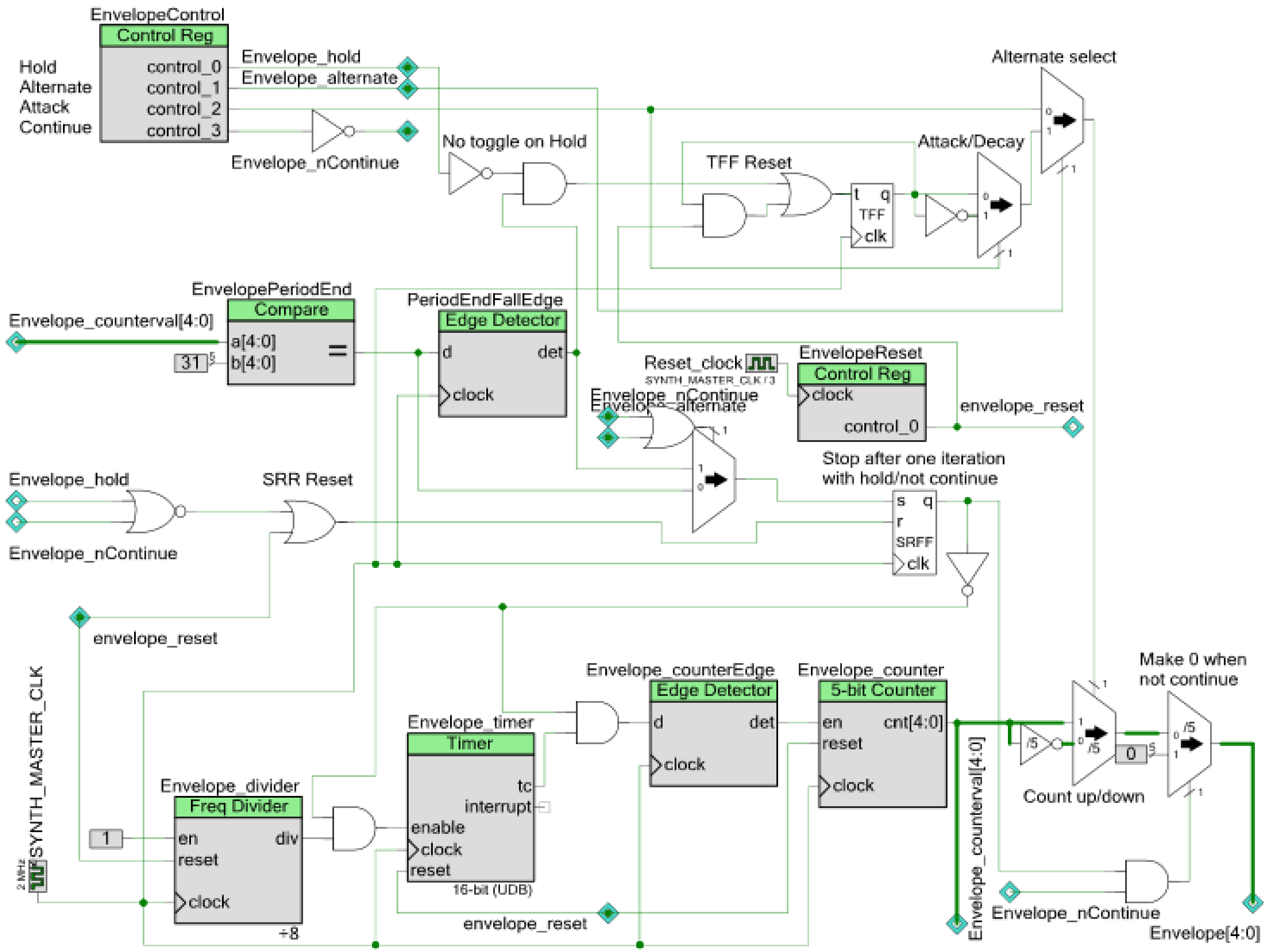
Voice Generation and Mixing



This schematic has the following parts:

- The noise generator, which is a pseudorandom sequence (PRS) generator configured to be 64-bit and using the time-division multiplex method (which basically means it takes four clock cycles to generate the next output). The input clock is used to vary the update rate. Note that the PRS does not have to be 64-bit; an 8-bit PRS should be sufficient. The difference is that the 64-bit PRS has a much longer sequence, so it takes a very long time before it repeats.
- The mixer control register. Setting a bit in this register to 0 enables the corresponding tone or noise for the corresponding voice. See the annotation on Mixer_Control to see what each bit controls.
- Three tone generating clocks, one for each voice (named VoiceX_clock, where X is the letter of the voice). The dividers on these clocks are changed to change the frequency of the tone.
    - These clocks are passed through a Sync component because clocks that are divided off a design-wide clock are not synchronized to that clock. On the other hand, clocks divided off of a system clock are synchronized to the master clock, so there are no issues. Basically, since these clocks are used together, they need to be synchronized together; otherwise PSoC Creator generates a warning. Having the clocks unsynchronized would have caused problems in the output schematic as well.
- Three groups of the same hardware to generate the output of the voice:
    - Voice amplitude register: the first four bits are the fixed volume of the voice. This 4-bit value is made 5-bit by appending a bit set to 1 in the 0 position of the value (note that a volume of 1 is silence). If the fifth bit in the register is 1, the current output of the envelope generator is used instead.
    - The multiplexer placed after the voice amplitude register chooses between the fixed volume and the current output of the envelope generator.
    - Below the register there are two multiplexers for choosing whether the tone and the noise are enabled for the voice. There is an OR gate to combine the outputs of these multiplexers, since both tone and noise can be enabled.
    - After the volume/envelope multiplexer is another multiplexer that sets the output of the voice to 0 or the chosen volume based on whether the tone/noise mix is 0 or 1, respectively.
    - The final voice output is connected to a sheet connector, which is referenced to the output schematic.

Volume Envelope Generator



This schematic has the following parts. As a reminder, all the shapes of the envelope are provided in Figure 3.

- Starting at the bottom of the schematic, there is a 16-bit timer for the envelope. The period of this timer is set to the 16-bit period for the envelope. The input to this timer is the master synth clock divided by 8, which is specified in the documentation (the divider is 16 in the AY-3-8910).
    - A regular clock set to SYNTH_MASTER_CLK/8 is not used in this case because there are only eight system timers and this project uses all of them. This is an instance where the clock divider can be constant, so the Freq Divider component was used instead. There are two other clocks in the project that could be substituted with the SYNTH_MASTER_CLK and a Freq Divider component if more variable clock dividers were needed, although doing so would use more UDB resources.
- The timer's compare output (which goes high when its period reaches zero) is connected to a basic 5-bit counter's enable input, with an edge detector in between so the enable is only high for one cycle, so the counter only counts once per timer tc. This counter is the current volume of the envelope before anything else modifies it. Unlike the more complicated counter component, this one counts up and has no period; it resets after it reaches its maximum value. Additionally, it outputs its value to hardware and has no software API.
    - This counter would be 4-bit if this project was emulating the AY-3-8910.
- Everything else in the design controls the above components based on the desired shape of the envelope.
    - There are four bits to control the shape of the envelope. See Figure 3 for the shapes made by a combination of these bits.
        - When the continue bit is set to 1, everything but attack is ignored. After the envelope finishes one complete period (that is, the counter increments from 0 to 31 or vice versa and holds the final value for the specified period), the output holds at 0. When the continue bit is set to 0, there is no effect.
        - When the attack bit is set to 1, the envelope initially counts up, otherwise it initially counts down.
        - When the alternate bit is set to 1, the envelope counter will switch directions when hitting the maximum or minimum value, creating a triangle pattern. When the alternate bit is set to 0, the envelope counter will reset when hitting the maximum or minimum value, creating a sawtooth pattern.
        - When the hold bit is set to 1, after the envelope finishes one complete period (that is, the counter increments from 0 to 31 or vice versa and holds the final value for the specified period), it holds the last value generated. If the hold bit is set to 0, there is no effect.
            - If both hold and alternate are set to 1, the counter is allowed to reset before holding.
    - EnvelopeControl is set to the four bits controlling the shape of the envelope. Whenever these bits change, the CPU also sets the EnvelopeReset register to 1 to reset the timer, counter, and state of the flip-flops.
    - The toggle flip-flop (TFF) is used to handle the alternating capability of the envelope.

        - This value toggles whenever the counter transitions from being 31 to being 0, except when hold is set to 1.
        - This value is inverted when attack is set to 1.
        - This value is ignored when alternate is set to 0, and the value of attack is passed through instead (via the “Alternate select” multiplexer).
            - The output of the “Alternate select” multiplexer determines whether the output of the counter should be inverted (inverting it essentially makes it count down).
        - TFF does not have a reset input, so the logic gates next to “TFF Reset” allow it to be reset if it is 1.
    - The SR flip-flop (SRFF) stops the timer and counter after one complete period (that is, the counter increments from 0 to holds the final value for the specified period) if the continue or hold bits are set.
        - If neither hold is 1 nor continue is 0, the reset (r) input is held high, and thus the SRFF remains 0.
        - In any case:
            - If either continue is 0 or alternate is 1, drive the set (s) signal on the SRFF to high after the counter transitions from 31 to 0.
            - Otherwise, drive the set (s) signal on the SRFF to high when the counter becomes 31.
        - If both set and reset inputs are held high, the SRFF remains 0. If only the set becomes high, the SRFF becomes 1 until reset goes high.
        - When the SRFF is 1, the timer and counter become disabled, so they hold their value without changing. Additionally, if continue is 0, the envelope output becomes 0.
    - The two multiplexers after the counter control whether it effectively counts up or down and setting the output to 0 when continue is 0 and a complete period has happened. The final output is sent to the voices schematic.

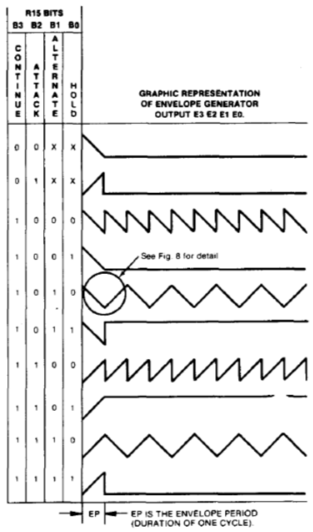
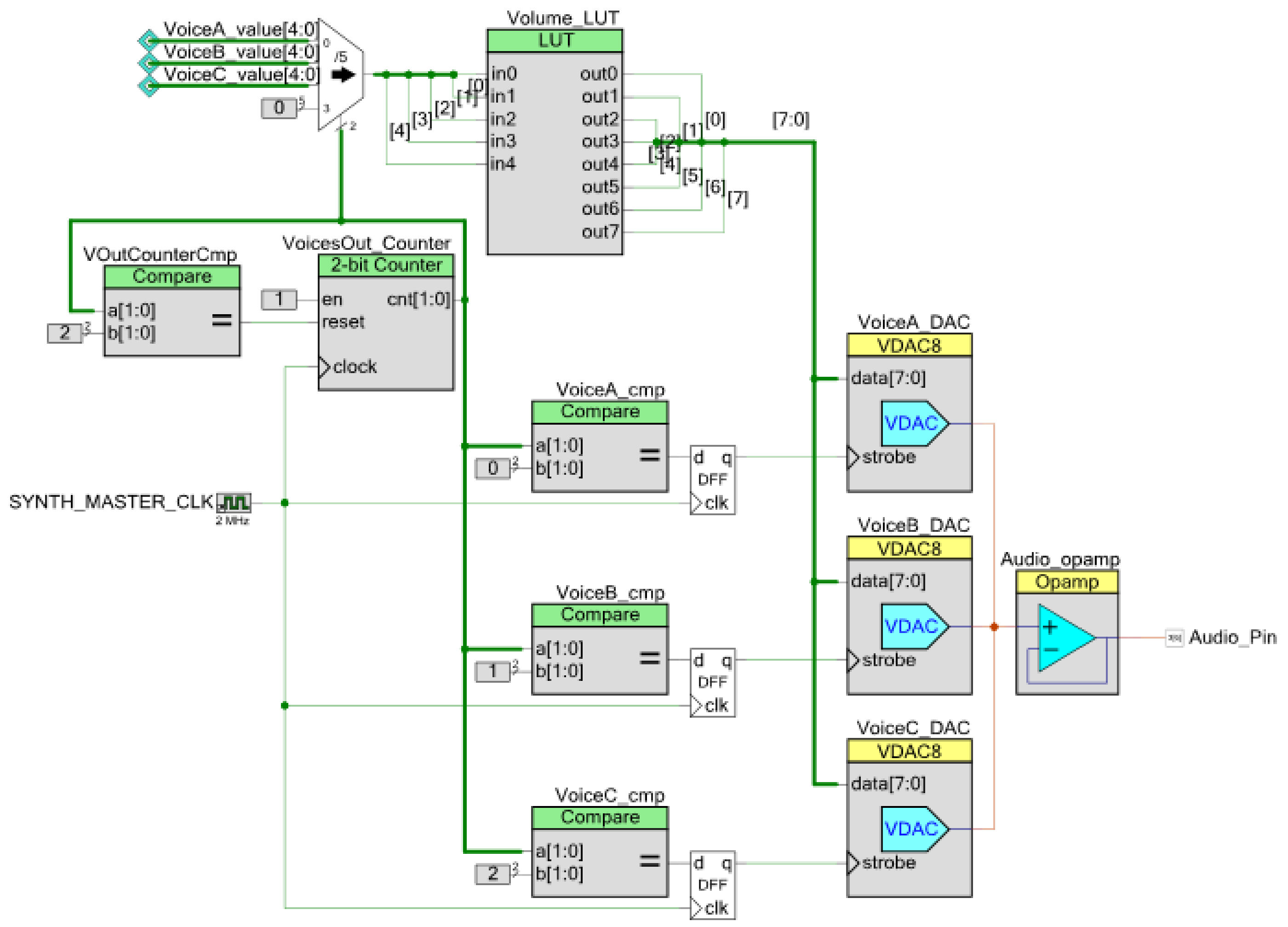


Figure 3: The shapes of the volume envelope [1]

Audio Output

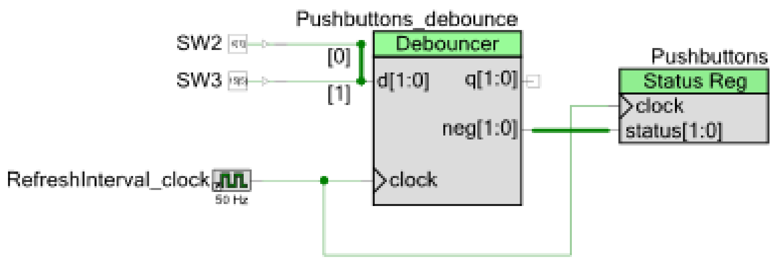


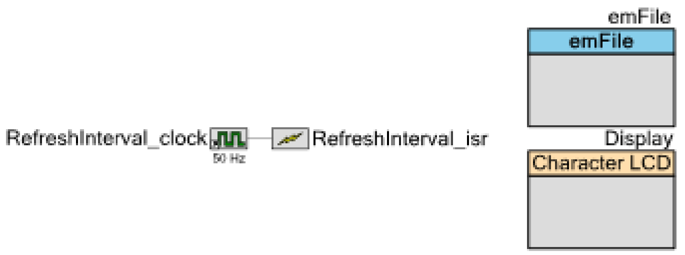
This schematic has the following parts:

- A 2-bit counter that counts from 0 to 2 and resets. VOutCounterCmp handles resetting the counter.
- A multiplexer is used with the counter to cycle through the three voices.
- A lookup table (LUT) is used to convert the volume of the chosen voice to an 8-bit logarithmically distributed output volume.
- Three DACs to convert each voice to an analog output. Each DAC is made to only change its output when its voice is selected by the multiplexer. The comparison components are used to allow for this. The D flip-flops are necessary to synchronize the changes in the DACs with the output of the LUT, which takes a short time to change.
- The output of the DACs are connected together to create one analog output.
- An opamp in follower mode to increase the maximum current of the analog output. This is not necessary when the output is connected to an amplified speaker, but it is necessary for driving small unamplified speakers or headphones.

There are two reasons why the voices cannot be directly sent to each DAC without being cycled through. The first reason is because there is only one path from the digital hardware to all four DACs. Attempting to connect individual DACs to different sources will generate an error. The second reason is because the LUT uses a large amount of UDB resources since it maps 5 input bits to 8 output bits, for a total of 32 8-bit values. This project as-is uses 20 of the 24 available UDB blocks. Adding two duplicates of this LUT makes the project require 25 blocks (the increase is an odd number since UDB blocks can hold multiple components, depending on their complexity). Reducing the size of the PRS from the voice generation schematic would free up some space, but duplicating the LUTs when it isn't necessary to do so is wasteful.

User Input and Miscellaneous





There are two pushbuttons for user input. Both are debounced using the same debouncer component to save resources. The pushbuttons are sent to a status register configured to be sticky so the software can read them. The interrupt method for responding to user input isn't used for simplicity, since the CPU is doing nothing else whenever it needs to respond to user input.

An interrupt is connected to a 50 Hz clock, which is used by the CPU to update the hardware fifty times a second. This interrupt emulates the interval the Atari ST updated the YM2149.

The emFile component is used to read files off the SD card. It includes the SPI interface for the SD card. This component requires an external library downloaded from Cypress, which is provided with the source code.

The Character LCD is used for displaying feedback to the user. Before playback, the Character LCD displays the currently selected file and "Next" and "Play" near the respective pushbuttons. During playback, a visualization of the music being played is displayed.

Software Implementation Details

The purpose of the software is to allow the user to choose a music file from the SD card to play and then update the hardware accordingly to produce the music. Additionally, while the music plays back, the music is visualized on the character LCD.

Waiting for Input

First off, the software uses the emFile library to look for files on the SD card. Note that files that do not have the proper format are not filtered out; playing back a non-converted YM file has not been tested. Then the name of the first file is printed to the display, along with the words "Next" and "Play" near their respective buttons. Then the software waits for input from the user.

If the user presses the next button, the next file from the folder is displayed. If there are no more files in the folder, the first file is displayed.

If the user presses the play button, the playback of the displayed file begins.

Music Playback

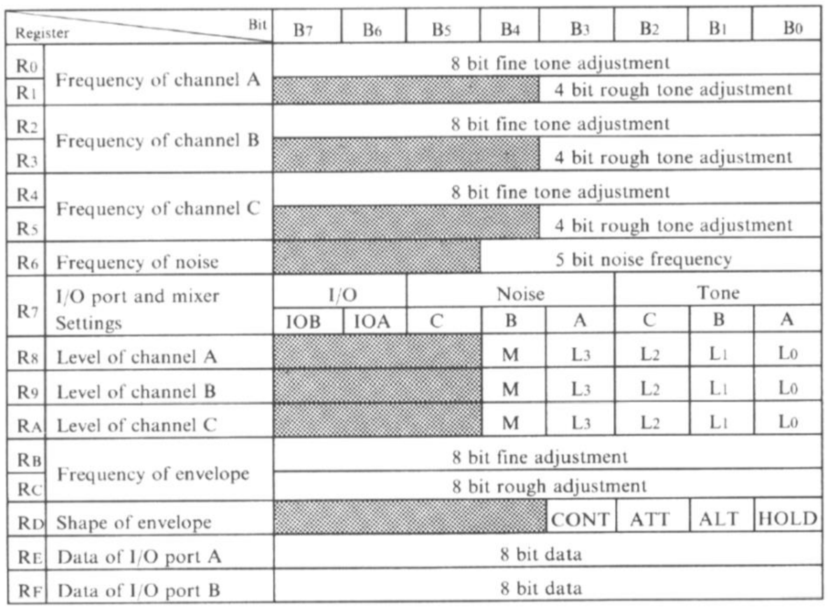
Music files are register dumps, formatted as one dump of all 16 registers per line. The purpose of each register is summarized in Figure 4. Exact details, such as exactly how the tone adjustment affects the tone generator, can be found in the manuals.

The music is buffered from the SD card by parsing lines and placing them into a circular queue. Multiple lines are read from the SD card at a time before parsing since doing so is more efficient; reading one byte at a time is slow. The first set of lines read from the SD card is parsed before starting playback to avoid the buffer from running out immediately.

After initial buffering, the interrupt for updating the hardware will start working. This interrupt takes the top register dump off the queue and updates the hardware accordingly. It also sets a flag saying that it has run.

The main loop continues to fill the queue with data read off the SD card. Once the queue is full, it will wait for an entry in the queue to fill. Also, if the flag saying that the interrupt has popped a dump off the queue has occurred is set, the display will be changed to visualize the music based on the volumes and tones of the channels. This visualization uses vertical bars to indicate the volume (0-15 pixels tall; 16 pixels for envelope volume control) and horizontal positioning of the bars to indicate the frequency.

When the end of the file is encountered, the visualization still happens, but otherwise the main loop waits. Once the interrupt encounters the end of the queue and sees that the end of the file was encountered, it changes the state of the main loop back to initialize waiting for input. Otherwise, the buffer ran out early and the system enters an error state.



Register	Bit	B7	B6	B5	B4	B3	B2	B1	B0
R0	Frequency of channel A	8 bit fine tone adjustment							
R1		▨				4 bit rough tone adjustment			
R2	Frequency of channel B	8 bit fine tone adjustment							
R3		▨				4 bit rough tone adjustment			
R4	Frequency of channel C	8 bit fine tone adjustment							
R5		▨				4 bit rough tone adjustment			
R6	Frequency of noise	▨			5 bit noise frequency				
R7	I/O port and mixer Settings	I/O		Noise			Tone		
		IOB	IOA	C	B	A	C	B	A
R8	Level of channel A	▨			M	L3	L2	L1	L0
R9	Level of channel B	▨			M	L3	L2	L1	L0
RA	Level of channel C	▨			M	L3	L2	L1	L0
RB	Frequency of envelope	8 bit fine adjustment							
RC		8 bit rough adjustment							
RD	Shape of envelope	▨				CONT	ATT	ALT	HOLD
RE	Data of I/O port A	8 bit data							
RF	Data of I/O port B	8 bit data							

Figure 4: Map of the sixteen YM2149 registers and their purpose

Setup

The entire project is provided with all dependencies. There should be no work required to build it, other than setting it as the default project.

Before programming the development kit, the character LCD should be plugged into the board. Additionally, the microSD breakout board (or equivalent) should be mounted on the breadboard and connected to the headers around it in this way:

- CS – P0_0
- DI – P0_1
- VCC – VDDD
- SCK – P0_3
- GND – VSSD
- D0 – P0_5
- CD – Leave disconnected (ignore if it doesn't exist)

The headphone jack should have both left and right pins connected to P3_6 and the ground pin connected to VSSA. On the suggested headphone jack breakout, the two outer pins are left and right and the center pin is ground; leave the second left and right pins disconnected.

See Figure 5 for an example hardware setup on the breadboard.

The SD card should be formatted with FAT32 and contain the converted music files. File names longer than eight characters (plus three character extension) will work, but their names will be cut off on the display. SDHC cards are compatible.

It is suggested that only devices with amplifiers are connected to the headphone jack. This project should be able to drive a small speaker by itself, but that has not been tested. Also, there is no master volume control in this project, so an amplified device is preferable since it can control its volume.

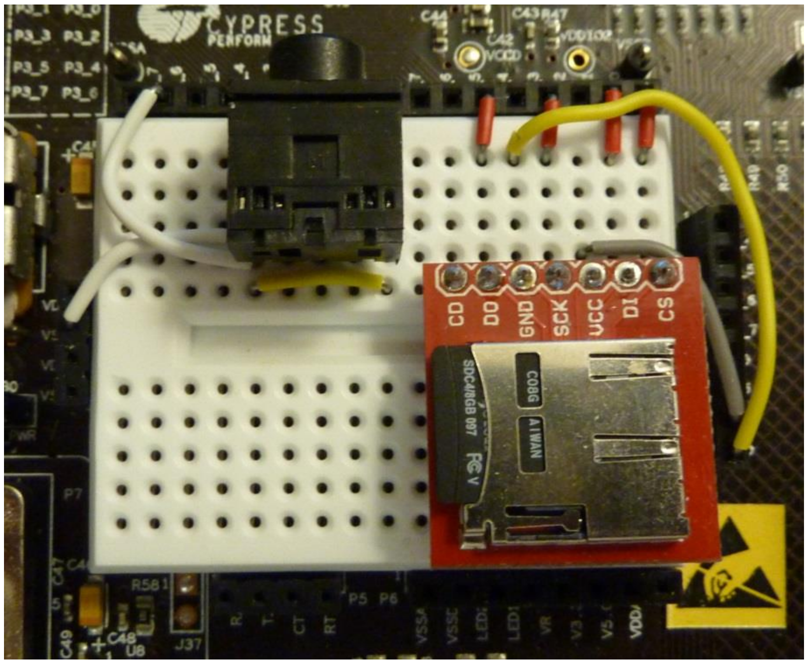


Figure 5: Example peripheral hardware setup

Results

This project reproduces music from the Atari ST fairly accurately.

Any music that does not make use of the volume envelope sounds almost exactly like what the emulator Ay_Emul produces. In music that uses the volume envelope, the voices that use the envelope sound a bit different from what the emulator produces. The reason why it sounds different is unknown, but it may be because the emulator is using a different curve for the volume levels. At one point the lookup table in this project was a bit off; once it was corrected, the voices in the music that use the envelope sounded different. So if the emulator uses a different curve than the datasheets specify, that would explain why it sounds different.

There is a video on YouTube that demonstrates this project here: https://www.youtube.com/watch?v=B2MvwRQ0q_I

Future Work

This project could be improved by figuring out why the envelope sounds a bit different from what is emulated. Additionally, the special effects that are possible on the Atari ST could be emulated by figuring out how they work and modifying the software accordingly. The YM files would also need to be converted such that they retain the additional data required for the special effects.

References

[1] General Instruments, *AY-3-8910/8912 Programmable Sound Generator Data Manual*. .

[2] Yamaha, “YM2149 Software-Controlled Sound Generator (SSG).” 1987.

[3] A. Carré, “YM File Format.” [Online]. Available: http://leonard.oxg.free.fr/ymformat.html. [Accessed: 11-May-2014].