ECSE 324 Lab 5 Report

Group 21

December 7, 2017

Dong Han 260658048

Yutian Jing 260680087

In this lab, low-level ARM programming techniques are combined to implement a musical synthesizer.

1. Make waves

This part basically includes the generation of the audio signal. The sampling rate provided by the DE1-SOC board is 48000Hz, and the sample values for a complete period of a 1Hz sine wave is given for further use.

The method for the signal value generation is defined by the figure below:

$$\text{index} = (f * t) \bmod 48000,$$
$$\text{signal}[t] = \text{amplitude} * \text{table}[\text{index}].$$

Figure 1 Signal calculation

This method (named "signal") takes f (frequency) and t (sample point) as inputs and returns the signal value at that specific sample on the wave of frequency f. A temp integer is set to hold the value of f *t at first, then the value of temp % 48000 is passed to a float named "index". The decimal part and integer part "index" is separated and passed into a float and an integer respectively for the last calculation:

float interpolated = (1 - decimals) *sine [indexLeftOfDecimal] + (decimals) *sine [indexLeftOfDecimal + 1].

This final result is returned at the end of this method.

In the next sectors of this lab, this method will be used for any calculations for the signal value of the wave with any one of the 8 frequencies.

2. Control waves

This part requires a control functionality on the music synth using a PS/2 keyboard.

Two timers are used in this part, one for keyboard (timer 1) and one for audio (timer 0). The keyboard timer reads input data periodically to avoid resource waste of CPU capacity and other hardware since human type speed is slow comparing with the CPU clock speed. The audio timer feeds the generated samples to the audio codec periodically (its value is fixed at 20 microseconds, calculated by 1sec / 48000). The frame of the whole chunk of code inside int main () is one infinity while loop containing two if statements with two timer flag checks. After each flag check, hps_tim1_int_flag and hps_tim0_int_flag was reset to 0. The two timer flags are defined and globaled in ISRs.s and published in the ISRs.h header file.

8 signals with different frequencies are to be written here. There are multiple ways of code writing to write signals to the audio codec using audio_write_data_ASM,

however since the limitation of the hardware performance and for efficiency concerns, the waveform of these 8 signals must be stored in 8 arrays all has a length of 48000, by taking 48k samples on these signals. If these signal values are calculated live in each iteration of the while (1) loop, too much CPU task calculations are required, and it would be harmful if the function of signal superposition is required (we performed live calculations once, the result comes out for any number greater or equal to 5 key superpositions, the sound started to break into pieces).

To read keyboard inputs, a char* called “data” is set, and its value is passed into an int called “in”. for each read_ps2_data_ASM(data), “in” will hold the value of the make code or break code currently in the char address. 8 parallel if statements were set to check which value “in” is holding, corresponding to which key is pressed. If “in” equals to 0xF0, another read_ps2_data_ASM(data) and assigning for “in” is needed to refresh the code in “in” to the break code after 0xF0, and 8 if statements were set to check which code “in” equals to, corresponding to which key is released.

8 flags (Booleans) for each note are set outside the while loop. If any press of the 8 keys is detected, the matching flags are set to 1, if any release of the 8 keys is detected, the matching flags are set to 0. Keys M and N are set to increase and decrease the volume, if M or N press is detected, an integer “amp” increase or decrease by 2, where “amp” is set outside the while loop. It is a coefficient for signal with a relationship of product (amp *signal).

In each iteration of the audio timer, each one of the 8 flags set before are checked, if any of those flags is 1, signal s += amp *t^{th} element of the corresponding note array (which means the note signal value at sample t). For superposition, it’s important to use += here. Rather than one t for all 8 arrays, 8 t are used (t0 - t7), they are all 0 at first and will increment by 1 after audio_write_data_ASM (s, s), and are reset to 0 at different values. This is to make sure there is no breakpoints or discontinuities in the audio played. Since 8 signals are not all integer frequencies, at the end of 48k samples on these frequencies there exist “tails” that are values for the last incomplete wave period. If these “tails” are played, for each loop a little “bump” will be heard in the audio output. The reset values for t is defined by (int)48000/f, if the 8 counters t0-7 are greater or equal to the respective values, the corresponding t are reset to 0.

With all the above-mentioned builds, the keyboard control for the music synth works successfully.

3. Display waves

In this part, the waveform is required to be displayed on the computer monitor, also with the control from keyboard.

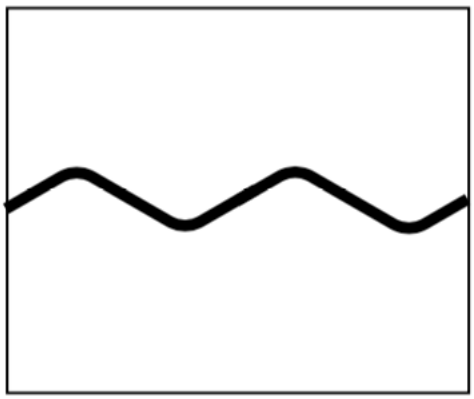

Figure 2 waveform for a single note

In this last part, apart from the two timers used above, one additional timer for VGA display will be used as well. While the total signal is writing to the audio codec in the audio timer, an array of length 320 (called “arrr”) is also saving the modified signal. The signal is divided by 800000 to reduce its size and fit the 240 range (y range) of the VGA output. For each t counter ranging from 0 to 48k, the array is taking point every audio timer cycle, and every 320 timer cycles the array is full and retaking the values to its first element. In the VGA timer, the 320 values in the array is written to the VGA output inside a for loop. Since the VGA timer is much slower than audio timer (to create a viewable wave speed for human), within 1 VGA timer cycle the later array value will be refreshed to the beginning wave values, and thus create a discontinuity in the displayed wave. A flag called “flag” must be added to determine if a continuous set of 320 points in the array is ready, if ready then perform VGA_draw_point_ASM, and set to 0 after the VGA_draw_point_ASM for 320 points is completed.