Author: Zesen Yao (zy4417), Alan Yilun Yuan(ayy17)

1.Questions

1. **Provide a trace table of Sinegen for several loops of the code. How many samples does it have to generate to complete a whole cycle?**

Answer: 8 samples per period, tested using variable watching and calculation by hand. Results are as follow:

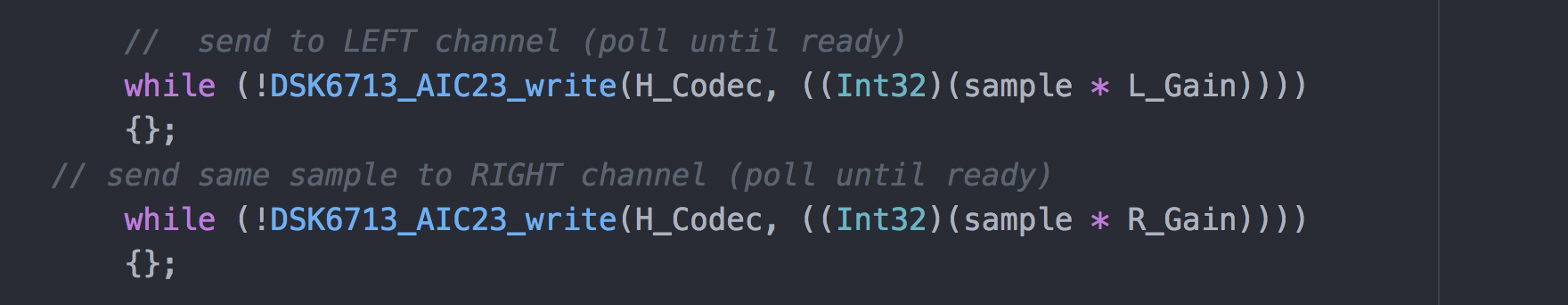
|  |  |
| --- | --- |
| Iterations | Sinegen Value |
| 1 | 0.7070 |
| 2 | 0.9999 |
| 3 | 0.7070 |
| 4 | 0.0000 |
| 5 | -0.7070 |
| 6 | -0.9999 |
| 7 | -0.7070 |
| 8 | 0.0000 |
| 9 | 0.7070 |

1. **Can you see why the output of the sinewave is currently fixed at 1 kHz? Why does the program not output samples as fast as it can? What hardware throttles it to 1 kHz? (If you are having problems working this out try changing the sampling frequency2 by changing sampling\_freq).**

Answer: The sampling frequency is set to 8khz and the number of samples per period is 8. 8k/8 = 1 k. As described in the lecture, the hardware DAC (periphal) constaints the clock rate at 1 kHz, regardless of how fast the processing units calculate.

1. **By reading through the code can you work out the number of bits used to encode each sample that is sent to the audio port?**

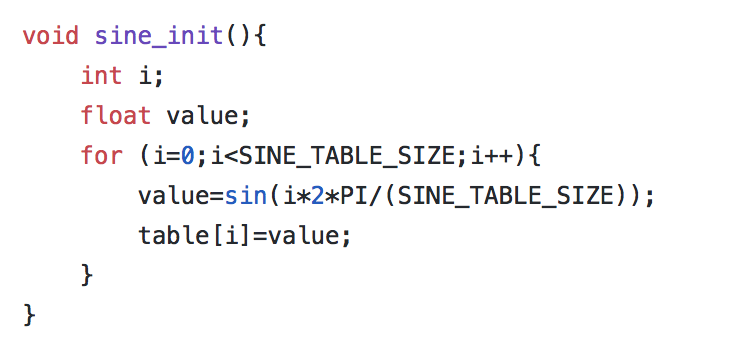
Answer: 32 bits as defined in the source code below.



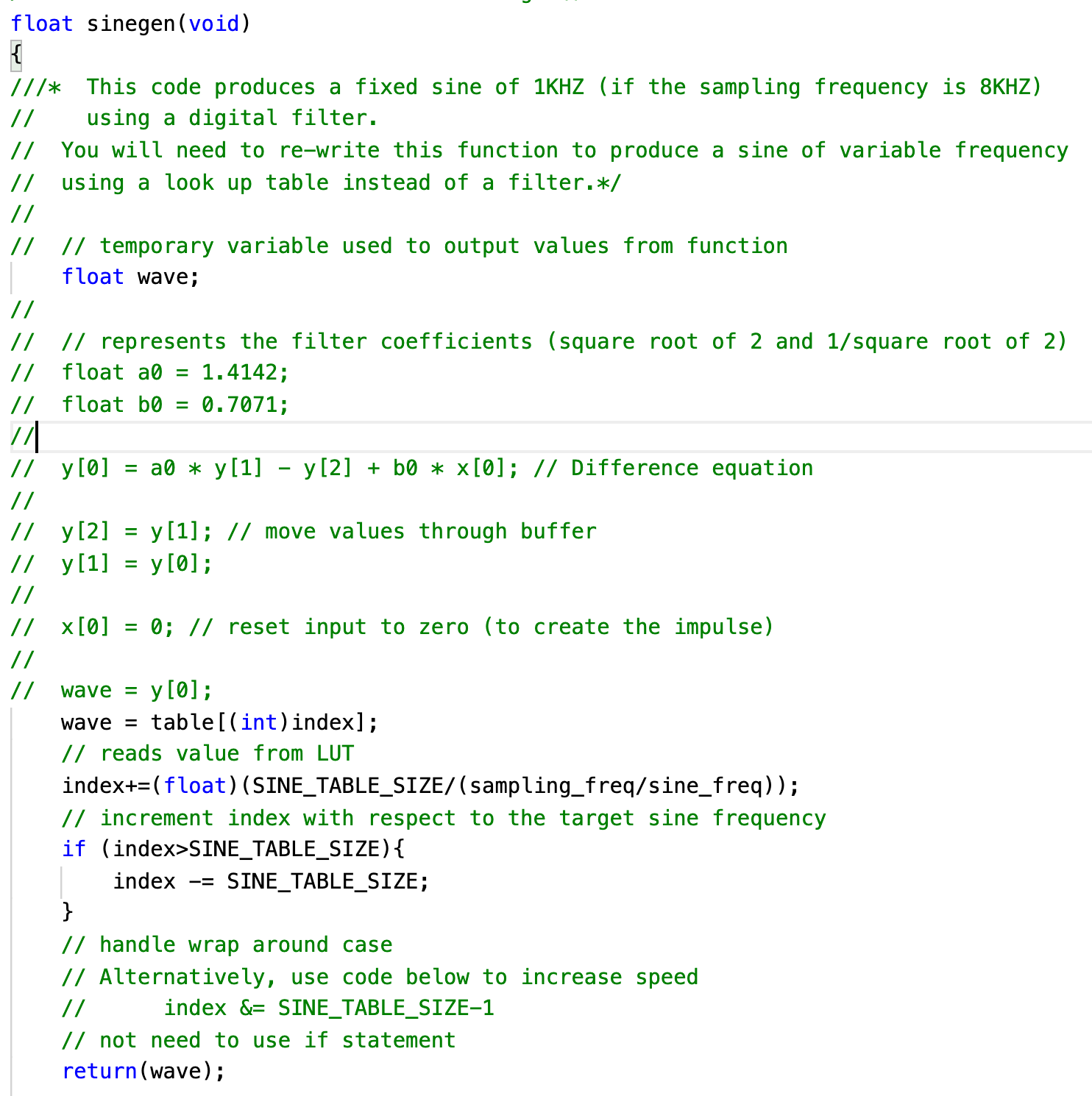
2.Operation of Code

To successfully generate a sine wave, values of sine functions at different points are needed. Therefore, an array called *table* is defined, size of which depends on variable*SINE\_TABLE\_SIZE.*

The first step of our code is to initialize hardware (as detailed by the original code) and array variable table. The library function sin() is able to return a double value of sine wave with increment of 2\*PI/SINE\_TABLE\_SIZE in radian. Next, the iteration of for loop will fill the array table with values of sine wave.



The second step is to generate an actual sine wave by placing the sinegen() function in the while loop of main function. While called each time, the sinegen () function returns the next value following the preceding one generated by the previous siniegen (). This memory capability is achieved with a global variable index and this variable is incremented by the value of (SINE\_TABLE\_SIZE/(sampling\_freq/sine\_freq)), which equals to the exact index increment needed to achieve a wave with specific sampling frequency and sine wave frequency. Therefore, this increasing variable allows sinegen () to access to correct element of a sine wave and return the respective value to the handle in main ().

****

Moreover, if we wish to increase the resolution (the smoothness of sine wave), the most obvious method would be to increase sampling rate, which is equivalent to reducing the step size on x-axis. Besides, a cleverer idea is to exploit the symmetricity of sine wave. It is well-known that the sine wave is symmetrical with respect to x-axis. Therefore, knowing the upper side of sine wave is equivalent to knowing the lower side of sine wave. Similarly, knowing the left side of upper sine wave is equivalent to knowing the right side of upper sine wave. With this idea, some manipulations on accessing the look-up table can achieve storing 256 values but effectively storing 256\*4 values. However, it should be noted that this mathematical trick increases the computational complexity significantly. It is worthwhile when the memory space is insufficient but higher resolution is demanded.

3.Bounds of frequency

**Lower Bound:** Due to the restrictions on sampling frequency, the lowest sampling frequency that can be set is 8000Hz. In this case, according to formula , the lower bound of wave frequency is 31.25Hz (8000Hz/256). Once the wave frequency is lower than 31.25Hz, the number of samples needed to maintain sampling frequency is higher. However, the program specifies that only 256 values are available, meaning that some values will be wrongly accessed more than 1 time. As a result, the sine wave generated is not as expected.

**Upper Bound:** According to the Nyquist theorem, the sampling frequency should be greater than twice the signal frequency. Therefore, given that the highest sampling frequency supported on this system is 96000Hz, the upper bound of our signal frequency is 48000Hz. If the wave frequency goes beyond this threshold, aliasing will occur and signal will therefore be corrupted.

4.Scope Trace

The following pictures show the scope trace between 10HZ to 4000HZ (the Nyquist frequency for 8000Hz sampling frequency).

In the case of 10 Hz, the frequency lower than lower bound indicates that the signal generated will be as expected. In fact, the sine wave was in such a bad shape that no pattern could be detected. On the other hand, in the case of 8000Hz, it can be observed that the signal generated was corrupted completely.

However, in the case of 200Hz and 2000Hz, clear sine wave could be observed. It should be noted that the smoothness of sine wave will decrease as the signal frequency increases, as a result of increasing step size.

|  |  |
| --- | --- |
| **A screen shot of a computer  Description automatically generated**10 Hz | **A picture containing electronics  Description automatically generated**200Hz |
| **A screenshot of a computer  Description automatically generated**2000 Hz | **A close up of a screen  Description automatically generated**4000 Hz (Nyquist Sampling frequency) |

The observations above are indeed what we expected.

5.Appendix



