University of California at Berkeley
College of Engineering
Department of Electrical Engineering and Computer Sciences

EECS150 J. Wawrzynek
Spring 2002 E. Caspi

Project Checkpoint 7: Velocity and Loudness

We will use velocity information from a MIDI note-on command to control the *perceived loudness* of a sound. In practice, we must scale the corresponding physical quantity, the waveform's *amplitude*. To scale the amplitude of a waveform we simply scale each individual sample before sending it to the DAC.

The relationship between signal amplitude and the perceived loudness of a sound is complex. Fortunately, the ear is far less sensitive to amplitude changes than to other musical qualities of a sound such as pitch. Thanks to this insensitivity we can use any of several approximations for controlling the loudness of a sound without detrimental effects. Perceived loudness is proportional to signal amplitude, but it is not directly proportional. Specifically, to increase the loudness of a sound by a factor of 2 we need to increase its amplitude by a factor of 2.8. Likewise, to decrease the loudness of a sound by a factor of 2 we must decrease its amplitude by a factor of 2.8. This relationship is expressed by the formula:

$$AMP = (2.8)^{\log_2 LOUDNESS}$$

Using this formula, our synthesizer can convert a desired loudness value (taken directly from velocity) into an amplitude scaling factor. The synthesizer must multiply the output samples by the scaling factor AMP to obtain the desired amplitude. The samples stored in the EPROM templates are all at maximum amplitude. These sample values will thus be scaled on their way to the DAC by a factor: $0 \le AMP \le 1$.

Described below are two schemes for scaling the amplitude of the waveform that will give satisfactory results. Consider these schemes and implement one of them as part of your synthesizer.

1. **Table of Multipliers.**
   Approximate the amplitude scaling function using a table-lookup operation. To be perfectly accurate, the table would be indexed by velocity (a 7-bit number) and would contain 128 entries for all possible velocities/amplitudes. A table of such size is probably overkill for this application. Ignoring some low-order bits of velocity allows the table to be smaller, at the cost of lower accuracy. A table of 16 entries (indexed by the 4 most-significant bits of velocity) will give satisfactory results. The *width* of the table determines the accuracy of the approximation of the above function. Again, 4-bits is sufficient. In this scheme you will need a 16x4 ROM to store the table, as well as a multiplier circuit that takes a 4-bit multiplier (positive

only) and a 16-bit multiplicand (2's complement). This multiplier circuit is slightly different from the one we discussed in lecture, because all the values stored in the table are less than 1. The binary point is assumed to be just to the left of the MSB. This modification is simple. Try a few cases by hand, and you will see.

2. **Table of Shift Amounts.**
   We can make a further approximation to the amplitude scaling function by using a shifter instead of a multiplier circuit. A multiplication by scaling factor 0=AMP=1 is approximated by a right-shift.

Below are listed some table values (4-bits each) that you can use for the above schemes. If you do not like the results obtained with the following numbers, you may change them as you like. The project spec requires that velocity scaling be implemented with a lookup table, but the table contents are up to you.

ROM Address	Scheme 1: Multiplier	Scheme 2: Shift Amount
0	1	6
1	1	4
2	1	3
3	2	3
4	3	2
5	3	2
6	4	1
7	5	1
8	6	1
9	7	1
10	9	0
11	10	0
12	11	0
13	12	0
14	14	0
15	15	0