The basic goal of this lab is to design logic to create a DTMF tone dialer. The project specifications are as follows:

Connect a PS/2 keyboard to the PS/2 port on your board. Using the numeric keypad as your dialer, you should be able to generate tones through the Digilent peripheral speaker board.

Each row of the keypad and column of the keypad will create a different tone. Thus, when you press a key, your logic should generate two tones at once. For example, pressing 5 will generate a sound consisting of two tones at 1336Hz and 770Hz. The tone should be at least 40ms long.

Design tips:

**Keyboard**

Use the keyboard module from the last lab. It should generate an ascii_data_available interrupt signal whenever ascii data is available from the keyboard.

**State Machine**

Your main state machine should be fairly straightforward. Wait for an interrupt from the keyboard module, and when you get it store the ascii data in a register and then go to another state waiting for the interrupt to go away.
**Signal Generation**

You can use a clock divider to generate the 7 required frequencies. Note, that this will generate a square wave which will have harmonics at higher frequencies than what you desire. You can ignore these higher frequencies. However, if you want to generate a clean monotone signal, you will need to create a sine wave. The simplest way to do this, is to create a lookup table from which you can extract the sine values for different time points. Another way to do it is to use a Taylor series approximation of the sine wave to three or four terms. This will require the use of multipliers however.

When you generate the sound, you will be summing the waves from two different frequencies. The ascii data that was stored in the register by the state machine can control some combinational logic that determines which column frequency and which row frequency you are going to use in your summation. You can use an adder to simply add the two values together. Using the square wave approach, you will just be adding two bits together, to create a two-bit value. This two-bit value can be extended to the required number of bits for your PWM module. If you are using a sine wave approach, your lookup table should probably match the resolution of your PWM module.

Each tone should only be 40-50ms long. One way to do this is to create a process that starts counting and sets a sound_valid flag once you get a new ascii code from the keyboard. When you have reached 40ms, you can clear the sound_valid flag. This sound_valid flag can control a mux that determines if the summed wave or zeros gets passed to the PWM module.

**PWM Module**

Since you don’t have a D/A converter on your board, the simplest way to generate the analog signal is to use PWM signals. PWM signals change the width of the pulse depending on the desired amplitude of the analog signals as shown in the figure below. \( fl \) represents the frequency of the pulse, i.e. how often the pulse is generated. The resolution of the pulse determines how small you can make a pulse. For example, if you have an 8-bit input value, and you use your 50Mhz system clock to generate the pulses, the resolution is 20ns and the period of the PWM waveform is 5120ns. Thus \( fl = 195 \text{Khz} \).
To see how this PWM signal can create an analog wave, consider this Fourier series expansion of the PWM signal. $V$ is the maximum possible output voltage, and $d$ is the duty cycle of the PWM waveform, i.e. the fraction of the PWM period that the signal is high.

$$f(t) = \frac{a_0}{2} + \sum_{n=1}^{\infty} a_n \cos\left(\frac{2n\pi t}{T}\right) + b_n \sin\left(\frac{2n\pi t}{T}\right)$$

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \cos\left(\frac{2n\pi t}{T}\right) dt$$

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(t) \sin\left(\frac{2n\pi t}{T}\right) dt$$

$$= \frac{1}{T} \int_{-Td/2}^{Td/2} V dt + \frac{2}{T} \sum_{n=1}^{\infty} \int_{-Td/2}^{Td/2} V \cos\left(\frac{2n\pi t}{T}\right) dt \cos\left(\frac{2n\pi d}{T}\right) + \sum_{n=1}^{\infty} \int_{-Td/2}^{Td/2} V \sin\left(\frac{2n\pi t}{T}\right) dt \sin\left(\frac{2n\pi d}{T}\right)$$

$$= Vd + 4Vd \sum_{n=1}^{\infty} \sin(n\pi d) \cos\left(\frac{2n\pi}{T}\right)$$

If you pass this signal through a low-pass filter, the time-varying portions of the signal are removed, and all your left with is the DC portion $Vd$. Thus, the duty cycle, $d$, determines the ratio of the output signal to the maximum output voltage. So, if you want to generate an analog signal that is 2.2V and your maximum voltage is 5V, you want to set your duty cycle to 44%. If you have 8-bits of resolution, this translates to an input digital value of $0.44 \times 256$ or ~113 or x71.

Thus, your PWM module will have to take in a value with number of bits equal to your desired resolution, and output a PWM waveform with pulse widths corresponding to the amplitude of the input signal. The input signal can not change too fast, or the PWM module will not be able to keep up. For example, with the numbers given above, the input signal can not have a frequency greater than 195KHz. That is not an issue for this lab, since the signals are very low frequency.

**Speaker Module**

Speaker module documentation is available on the website. The speaker will be connected to one of the connectors on the peripheral interface module (PIM) which can then be connected to one of the expansion headers on your Spartan-3 board. Your system.ucf file has pins defined for the b1 connector for the 8 expansion connectors on the PIM. Each connector has 4 usable data pins. On the speaker module, only data pin 1 is used, so make sure that your code uses data pin 1 on one of the 8 expansion connectors. This pin is connected to a low pass filter before being sent to the speaker, so this will handle the PWM filtering for you.