Lab Assignment 1

Due Thursday, April 7

1 Objectives

This lab provides a review of VHDL design methods you learned in Engs 31. You will construct four modules, which together make a variable-frequency audio oscillator. The combination of a **phase accumulator** and **sine wave lookup table** (LUT) makes a **direct digital synthesis** (DDS) frequency generator. A **rotary encoder** varies the frequency, and a **pulse width modulator** (PWM) converts the sine wave samples to audio. You can listen to the sound, look at the signal on an oscilloscope, and study its frequency content with a spectrum analyzer.

2 Direct digital synthesis

Modern high precision signal generators use direct digital synthesis (DDS) rather than analog oscillators. The theory of DDS is nicely described in the IEEE article [1], which you should read for background. The main result is that a sinusoidal signal is generated at discrete instants of time by the equation

$$X[n] = \cos(\Phi[n]) = \cos(2\pi f n T_s) = \cos\left[(2\pi f T_s)n\right]$$
$$Y[n] = \sin(\Phi[n]) = \sin(2\pi f n T_s) = \sin\left[(2\pi f T_s)n\right]$$

where T_s is the clock period,

$$T_s = \frac{1}{f_s}.$$

and the quantity $2\pi f T_s$ is a phase increment, $\Delta\Phi$. This phase increment is related to the signal's frequency by

$$f = \frac{\Delta\Phi}{2\pi T_s}$$

By setting a phase increment we can obtain a sine wave at any desired frequency. By computational necessity the phase increment is quantized to be a rational (sub)multiple of 2π,

$$\Delta\Phi = \frac{\Delta_{\mathrm{ACC}}}{2^N} \times 2\pi$$

where Δ_{ACC} ranges from 0 to $2^N - 1$, *i.e.*, Δ_{ACC} is an N-bit number. Then the output frequency is related to the clock frequency by $f = f_s \times \Delta_{\mathrm{ACC}}/2^N$, *i.e.*, the output frequency is also quantized to one of 2^N values. We can achieve finer frequency steps by increasing N.

Inserting the quantized frequency into the expression for the sine wave gives

$$Y[n] = \sin\left[(2\pi f T_s)n\right] = \sin\left[2\pi\, f_s \Delta_{\mathrm{ACC}}/2^N\, T_s n\right] = \sin\left(\frac{2\pi}{2^N}\, n\Delta_{\mathrm{ACC}}\right) = \sin\left(\frac{2\pi}{2^N}\, \Delta[n]\right)$$

where

$$\Delta[n] = \Delta[n-1] + \Delta_{\mathrm{ACC}}$$

is an N-bit number accumulated in a register by adding Δ_{ACC} at each clock time.

Since there are only 2^N values of phase, and hence of Y, we can precompute and store these values in a 2^N-element memory (lookup table). (If we want the cosine as well, it just takes another 2^N memory to store those values.) Then at each clock time we add Δ_{ACC} to the register holding Δ and use Δ as the index into the table to look up the value of Y rather than computing it. The DDS sine wave generator is simply an accumulator (register and adder) and a lookup table (*e.g.*, a read-only memory).

The spacing between adjacent possible frequencies is called the resolution of the synthesizer, and is

$$\Delta f = \frac{f_s}{2^N} \ .$$

More bits in the accumulator gives higher resolution. Suppose we want a frequency of 6.5122 MHz accurate to 1 Hz and are given clock frequency of 115.2 MHz. The size of our phase accumulator register is found by

$$\Delta f = 1 \text{ Hz} = \frac{115.2 \times 10^6}{2^N}.$$

Solving for N we find the number of bits required is $\lceil 26.78 \rceil = 27$. The number we need to add to the accumulator every clock cycle to obtain 6.5122 MHz is

$$\Delta_{\mathrm{ACC}} = 6.1522 \times 10^6 \times \frac{2^{27}}{115.2 \times 10^6} = 7167832.519, \text{ which rounds to } 7167833.$$

The frequency we will obtain is $f = \frac{7167833 \times 115.2 \times 10^6}{2^{27}} = 6.1521996$ MHz, which is within 1 Hz of 6.5122 MHz, as desired.

If the desired frequency is an integer multiple of Δf, the DDS output frequency will be exact. Otherwise, there will be short term variations in the frequency. Although the long-run average is correct and the main component of the spectrum will be at the desired frequency, the short-run variations cause spurious frequency components to also appear. The analysis and removal of these so-called spurs is beyond the scope of this course. If you are interested, you can learn more about DDS in the articles by Cordesses [1, 2].

Several integrated circuit manufacturers offer sophisticated DDS systems on a chip. See, for example, the family of chips produced by Analog Devices, at `http://www.analog.com/en/rfif-components/direct-digital-synthesis-dds/products/index.html`.

3 Pulse width modulator audio output

In order to listen to the synthesized waveform, you need a digital-to-analog converter. You won't get the DAC on the board working until Lab 3, so for now you will implement a pulse width modulator (PWM). Pulse width modulation is a simple but often effective substitute for a digital-to-analog converter. [3]

The basic idea behind PWM is that the amplitude of a signal sample is mapped into the width of a square pulse. For example, suppose you have 8-bit signal samples (256 values). In 256 clock periods you can make a pulse whose duration (width) varies form 0 to 255. The PWM takes a sample of value M (between 0 and 255) and outputs a pulse of width M clock cycles. With a sufficiently high clock rate, when this binary

PWM waveform is output through a speaker, the speaker's electromechanical dynamics act as a lowpass filter to approximately reconstruct the original 8-bit signal from the PWM waveform.

A simple way to make a PWM is to run a counter (sticking with the 8-bit example) that goes from 0 to 255 and repeats. The count is compared with the 8-bit signal value you want to convert. As long as the count is less than the signal value, the output of the comparator is a '1', otherwise it is '0'. This is called single-ended PWM. The rising edges of the pulses are regularly spaced, and the irregular spacing of the trailing edges causes noticeable distortion when you listen to the output. A better way, called double-ended PWM, makes the centers of the pulses to be regularly spaced rather than their leading edges. It is only a little harder to implement than single-ended PWM and I recommend you try both and make them selectable by a slide switch so you can compare them.

Because each M-bit signal sample requires 2^M clock times for encoding the pulse, the PWM's clock must run 2^M times (or a multiple thereof) faster than the sampling rate of the signal. Alternatively, you can update the phase accumulator at the system clock rate and take every 2^Mth sample for the PWM, throwing the rest away. This is called decimation, or downsampling, and in Labs 5 and 6 you will learn to do this carefully and efficiently for a communications application. You also want the repetition rate of the PWM signal (called the carrier frequency) to be beyond the limit of human hearing. Otherwise, a constant half-scale input will make an audible square wave at the carrier frequency that interferes with the desired signal. In our application, we're making an audio signal so the sampling rate for updating the phase accumulator can be as low as 44 kHz, which is considerably smaller than the 133 MHz system clock. However, you must make sure that $133 \times 10^6/2^M > 22$ kHz.

4 Design and testing

4.1 RTL design process

- Your task is to design and instantiate a DDS sine wave oscillator. In this lab we will use the 133 MHz clock to the FPGA and design for frequencies in the audio range up to 10 kHz, with at least 1 Hz resolution. Later in the course we will want to reuse the design to create signals in the MHz range, so use generics to pass in the phase accumulator register size, system clock frequency, and a default starting frequency. The increment Δ_{ACC} is an input port in the entity. For example,

```
entity DDS is
  generic (
        ASIZE      : natural := ??;                              -- accumulator size in bits
        LUTSIZE     : natural := ??;                              -- LUT resolution, in bits
        CLKFREQ    : natural := 133000000;                       -- main clock
        STARTFREQ  : natural := 440 );                           -- default synthesizer output frequency
  port (clk        : in  std_logic;                              -- system clock
        stepSize   : in  std_logic_vector(ASIZE-1 downto 0);     -- Delta_ACC (connect to up/down counter
        synth_out  : out std_logic_vector(ASIZE-1 downto 0); )   -- LUT output
end DDS;
```

- Recall that in Engs 31 you designed an up-down counter controlled by a rotary encoder. Because you've already designed the quadrature decoder once, it is supplied to you in a separate .vhd file on Blackboard. You just have to incorporate that code into your design for an up-down counter, and the use that up-down counter to provide the value of Δ_{ACC} for the synthesizer. Use one of the slide switches on the board to select between a 10 Hz and 100 Hz frequency increment (per click of the rotary encoder).

- The Xilinx Core Generator supplies a core called Sine/Cosine Look-Up Table. The data sheet for this core is posted at Blackboard and is also available inside the Core Generator tool. With this core, you can set the number of address (phase) bits and data (sine) bits. I suggest you start with a 10-bit phase and a 12-bit output (but make these sizes generic in your phase accumulator architecture so you can change them later). Note, though, that having a 10-bit phase does not mean you have a 10-bit phase accumulator. You need the high number of bits in the accumulator to get good frequency resolution, even if the phase truncation leads to noise in the output.

  If you have time at the end, it is an interesting experiment to study phase noise (number of address bits) and quantization noise (number of data bits) in the LUT.

  The output of the sine wave LUT is a signed (twos-complement) integer. For conversion, either by a DAC or a PWM, the value must be converted to an unsigned *offset binary* format. This is done by adding a bias to the signed integer so that the minimum value is zero rather than -2^{M-1}, where M is the number of LUT output bits. This is actually very easy to do and doesn't require addition.

- Finally, design and implement the PWM.

4.2 Design verification

Before you testbench the design, run it through synthesis and make sure your VHDL is generating hardware that makes sense (at least at the RTL level).

Then verify that your DDS works by writing a testbench and doing functional simulation with ModelSim. Your testbench should simulate a change in the phase increment by simulating a turn of the rotary encoder knob. Be reasonable with this verification so that it doesn't take too long, *i.e.*, use a smallish accumulator (easy if you made the accumulator size generic). Your ModelSim waveform should show all important signals from each module of your design: phase accumulator, LUT input and output, and PWM output. Note that ModelSim can display the LUT output as a waveform, so you can display a sinusoid and also measure its period and compare with theory.

4.3 System Tests

Following successful simulation, write a UCF file (use the basic file on Blackboard), implement the digital design and verify that the complete system works using a speaker, an oscilloscope, and a spectrum analyzer.

- Send the output of the PWM to the audio jack and also to a pin you can probe.

- Listen to the sound as you adjust the frequency up and down. Watch the PWM output on an oscilloscope. Put the PWM output through an RC filter and observe the output on an oscilloscope.
- Look at the PWM output on a spectrum analyzer and adjust the frequency. Note that some of the spectral peaks change in frequency, and others do not. Hypothesize where these are coming from, and do an analysis to verify your idea.
- Reimplement your design, using switches to select more or less coarsely quantized phase and more or less coarsely quantized data from the LUT, and compare the qualities of phase noise vs quantization noise.

5 Project idea—improved spur suppression

There are several methods for spur removal, some of which are well within the scope of Engs 128. This would make an interesting project.

6 Writeup

Write a concise narrative of all the design decisions you made, problems encountered, and how you solved them. Append to this narrative all your VHDL listings and simulation waveform printouts. Include graphs from the oscilloscope and spectrum analyzer that illustrate what you learned.

References

[1] L. Cordesses, "Direct digital synthesis: A tool for periodic wave generation (part 1)," *IEEE Signal Processing Magazine*, vol. 21, no. 4, pp. 50–54, July 2004.

[2] ——, "Direct digital synthesis: A tool for periodic wave generation (part 2)," *IEEE Signal Processing Magazine*, vol. 21, no. 5, pp. 110–112,117, September 2004.

[3] R. Camarota. How to control analog output from a CPLD using a pulse width modulator. http://v2.embedded.com/design/214502993