ESE 382 Digital System Design Using VHDL and PLDs

Spring 19, Ken Short
April 15, 2019 5:35 pm

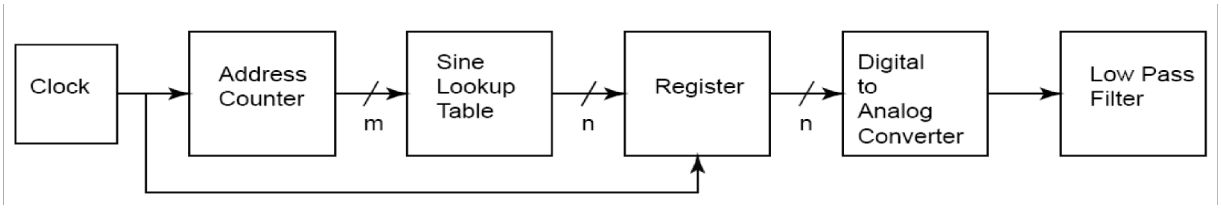
Laboratory 09: Simple Direct Digital Synthesis System
This laboratory is to be performed the week starting April 14th.

Prerequisite Reading
1. Chapter 9 of the text.
2. MT-085 Tutorial: Fundamentals of Direct Digital Synthesis (DDS).
3. Analog Devices AD9750 10-bit DA Converter Data Sheet.
4. Lattice LCMXO3L-6900C Data Sheet.

Purpose
Direct digital synthesis (DDs) is a digital technique for generating a periodic analog waveform. In its full form you can create periodic waveforms, such as a sine wave, with a specific frequency and phase. This is done using a system clock with a frequency higher than the synthesized output frequency. You can also precisely control the phase of the waveform. DDS ICs are available specifically for this purpose. However, we will design our own.

The design will be done in multiple stages over the remainder of the semester. We start with the simple DDS system of this laboratory and increase its capability and performance as the semester progresses.

A simple DDS system is shown conceptually in the following block diagram.



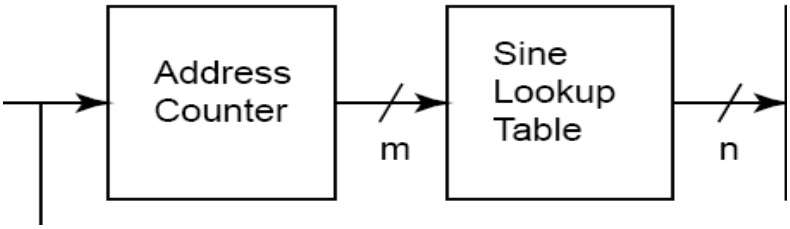
The clock produces a square wave output. On the rising edge of the clock the address counter is incremented. The address counter counts in binary from 0 to 2^m - 1 and then rolls over to 0.

The output of the address counter sequentially addresses locations in a sine lookup table. The lookup table has 2^m locations and each location is an n-bit word. If we want to generate a sine wave, then the values in the lookup table represent the amplitude of the sine wave, We could place the sampled amplitude values of a complete cycle of a sine wave in the table. In that case the first location would be the sine of 0 degrees. The second location would be the sine of $360/2^m$ degrees, and so on.

The output from the sine table is stored in a register on the rising edge of the next clock pulse. The output of this register drives a digital-to-analog converter (DAC) to produce the analog version of the sine wave.

A low-pass filter is then used to reduce any noise or artifacts in the analog output.

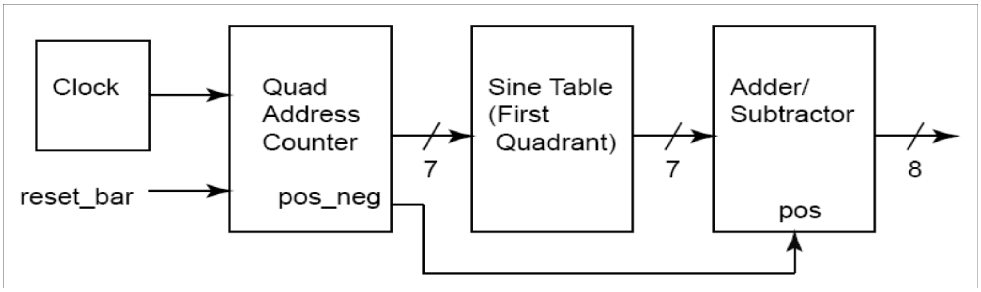
The entire DDS system is a mixed signal system. We are interested primarily in the digital part of the design. For this laboratory we will use an IC DAC that has an on-chip register. The clock is provided by a free running clock oscillator. So, we are left with only the address counter and sine lookup table to design and implement.



The sine wave output is more accurate with larger values of *m* and *n*. Parameter *m* determines how many samples of the sine wave we have in a single cycle. The more samples the more accurate the analog sine wave that is produced. If we use a value of $m = 7$ there would only be 128 samples in an entire cycle of the sine wave. However, if take advantage of the symmetry of a sine wave, we could store the amplitude values for only the first 90 degrees of the sine wave. We would then have an effective 512 samples of the complete sine wave cycle with *m* still equal to 7.

This approach requires that the counter count up for the first quarter of the sine wave cycle, then down for the second quarter, then up for the third quarter, and back down for the fourth quarter. We would also have to keep track of whether we are in the first half of a cycle or the second (negative) half of a cycle. Since, for the second half of the cycle we have to modify the output of the sine table before applying it to the DAC.

One approach that we can use to handle the negative half of the sine wave is to shift the entire sine wave up by a value equal to its peak value before applying the sine values to the DAC. This is the same a adding a DC component to the sine wave. A block diagram for the resulting system is as follows:



Design Tasks

Design Task 1: Quad Address Counter
The address counter entity is named `quad_address_counter` where quad means that it produces the appropriate count sequence to address a sine table that contains values for only the first quadrant of a sine wave cycle. The entity declaration is:

```
entity quad_address_counter is
    port(
        clk : in std_logic;       -- system clock, 50% duty cycle assumed
        reset_bar : in std_logic;-- active low asynchronous reset
        pos_neg : out std_logic;-- 1 for first half of cycle, 0 for second half
        address : out std_logic_vector(6 downto 0)-- address to sine table
        );
end quad_address_counter;
```

The counter counts on the rising edge of the clock. Write a behavioral description for the counter.

A non-selfchecking testbench is provided. This testbench generates the required reset and the system clock that runs for slightly more than a complete cycle of the sine wave.

Submit your design description and simulation output waveform as part of your prelab.

Design Task 2: Sine Lookup Table

The sine lookup table is a 128 x 7 table. The values in the table represent the amplitudes of the sine wave for the first quarter of its cycle. Table entries are integer values. Since table entries are 7-bits wide the table value when the sine is 1.0 is "7F". This means that the approximation for 1.0 in the table is 127/128 or 0.9921875. If the table was wider, that is, each entry contained more bits, the approximation would be more accurate. For example, for a 10-bit table the approximation for 1.0 in the table would be 1023/1024 or 0.999023.

The sine lookup table is a combinational circuit. To create the table entries you could write a program in C, or some other high-level language, to compute the 128 table values and write them to a file. You can then cut and paste these values into your VHDL description for the lookup table.

The entity declaration for the sine lookup table is:

```
entity sine_table is
    port(
        addr : in std_logic_vector(6 downto 0);-- table address
        sine_val : out std_logic_vector(6 downto 0)-- table entry value
        );
end sine_table;
```

Write a behavioral description for the sine table. One way to simulate your sine table description is to use it in a structural test design that includes the `quad_address_counter` and the testbench from Task 1.

Submit your design description and simulation output waveform as part of your prelab.

Design Task 3: Adder Subtracter

The adder subtracter uses the `pos_neg` output of the address counter to determine which half of the sine wave cycle is current and to modify the 7-bit value from the sine lookup table to produce

an 8-bit value to be input to the DAC. The output values from the `adder_subtracter` range from 128 at the start of a cycle to 255 at its positive peak and 0 at its negative peak.

The entity declaration for the sine lookup table is:

```
entity adder_subtracter is
   port(
      pos : in std_logic;-- indicates pos. or neg. half of cycle
      sine_value : in std_logic_vector(6 downto 0);-- from sine table
      dac_sine_val : out std_logic_vector(7 downto 0)-- output to DAC
      );
end adder_subtracter;
```

Write a behavioral description for `adder_subtracter`.

Submit your design description as part of your prelab.

Design Task 4: Simple DDS System

The entities from the first three tasks are interconnected to create a simple direct digital synthesis system. This top-level system has a structural architecture and the entity declaration:

```
entity simple_dds is
   port(
      clk : in std_logic;-- system clock
      reset_bar : in std_logic;-- asynchronous reset
      dac_sine_value : out std_logic_vector(7 downto 0)-- to DAC
      );
end simple_dds;
```

Submit your design description and simulation output waveform as part of your prelab.

Laboratory Tasks

Lab Task 1: Quad Address Counter

Using Aldec Active-HDL create a new workspace named `simple_dds`. In this workspace create a design named `quadrature_address_counter`. Import your VHDL source file for the `quad_address_counter` and the testbench provided.

Compile your source file. When your compilation is successful, functionally simulate your design using the testbench provided.

Have a TA verify that your design and testbench generate the appropriate output waveforms.

Lab Task 2: Sine Lookup Table
In the existing workspace create a second design. Name this design `sine_table`. Import your VHDL source file for `sine_table`.

Compile your source file. When your compilation is successful, functionally simulate your design.

Have a TA verify that your design and testbench generate the appropriate output waveforms.

Lab Task 3: Adder Subtracter
In the existing workspace create a third design. Name this design `adder_subtracter`. Import your VHDL source file for the `adder_subtracter`.

Compile your source file. When your compilation is successful, functionally simulate your design.

Have a TA verify that your design and testbench generate the appropriate output waveforms.

Design Task 4: Simple DDS System
In the existing workspace create a fourth design. Name this design `simple_dds`. Import your VHDL source file for the top level entity `simple_dds`.

Compile your source file. When your compilation is successful, functionally simulate your design.

Have a TA verify that your design and testbench generate the appropriate sine wave output.

Use Synplify to synthesize your design for a Lattice LCMXO3L-6900C target. Use the pin assignments provided to you in the laboratory. After synthesis is complete, from the **HDL Analyst** menu select **RTL,** then **Hierarchical View**. Print this diagram.

Use Lattice Diamond to fit the synthesized logic into a Lattice LCMXO3L-6900C FPGA. In Lattice Diamond, view and print the chip report.

Program an LCMXO3L-6900C FPGA and verify its functional performance in the hardware test system provided.

Have a TA sign off whether your programmed PLD meets the design specifications.

Questions

1. Compared to the testbenches that you have seen and written for combinational circuits, what additional stimulus signals do sequential UUT's always require? That is, what stimulus signals do you see in the testbench for Tasks 1 or 4 for this laboratory that you have not seen before?

2. In the testbench for Task 1, for how long is the reset signal `reset_bar` asserted? If you were to write a testbench for a timing simulation, as opposed to a functional simulation, what would determine the minimum amount of time that the system reset signal would need to be asserted?

3. Explain what the for loop does in the stimulus process of the testbench for Task 1.

4. If the system clock input `clk` was 1 MHz, what would be the frequency of the sine wave output of the system? Explain your answer and show your calculations.