ESE 382 Digital System Design Using VHDL and PLDs

Spring 19, Ken Short
April 23, 2019 8:21 pm

Laboratory 10: Direct Digital Synthesis System with Frequency Selection
This laboratory is to be performed the week starting April 21st.

Prerequisite Reading
1. Chapter 10 of the text.
2. Section 15.9 of the text.
3. MT-085 Tutorial: Fundamentals of Direct Digital Synthesis (DDS).

Purpose
The direct digital synthesis (DDS) system that you designed for Laboratory 9 has a fixed output frequency that is a function of the system clock frequency. The output frequency f_o is given by the relationship:

$$f_o = \frac{f_c}{2^{m+2}}$$

where *m* is the width of the address to the sine table.

The output frequency in Laboratory 9 is a function of the number of bits, *m*, in the counter, because the size of the counter determines how long it takes to read out the sine table's values for a complete period of the sine wave. The table has 2^7 entries, so we used a 7-bit counter to address the table's entries and we read out the table's values 4 times in one period of the sine wave. Thus, each cycle of the sine wave took 2^9 clocks. If you wanted to change the output frequency, you would have to change the input clock frequency. Or, you could increase the width of the counter, but still only use the most significant 7 bits out of the counter to address the sine table.

In many applications it is desirable to have a DDS system where the output frequency can be dynamically changed without changing the input clock frequency. To dynamically change the output frequency, without changing the clock frequency, we have to change how many clocks it takes to readout the sine table's values for one period of the sine wave.

Since the size of the table is fixed, the only way to read the table entries at a different rate, without changing the clock frequency is to skip some of the table entries to get faster output frequencies or take for than one clock to read each table entry for slower clock frequencies.

Faster Output Frequency
Lets look first at how we could get higher output frequencies by skipping some table entries. For example, if we increment the address counter by 1 for each clock, we read every entry in the table four times during one period of the output sine wave. However, if we increment the counter by 2 for each clock, we read every other table entry four times in one period of the sine wave and the output frequency doubles, but the number of output samples in the output waveform is halved.

Note that when the address counter is incremented (and decremented) by 1 we have the lowest output frequency. For example, if the clock is 1 MHz the output frequency would be 1.953 kHz. If we incremented (and decremented) the address counter by 2, the output frequency becomes 3.906 kHz. In general, the output frequency becomes:

$$f_o = \frac{k \times f_c}{2^{m+2}}$$

where k is the amount that the address counter is incremented (or decremented) for each clock. For an m bit address counter, k ranges from 1 to 2^m - 1. Thus, with a 7-bit address counter and 1 MHz clock we can adjust the output frequency from 1.953 kHz to almost 250 kHz in steps of 1.953 kHz.

Nyquist's criteria says that if we have at least two samples for one period of a sine wave we can reconstruct the sine wave. In a practical system you would want to have at least 3 or 4 samples for one period and you would need a good low pass filter on the digital-to-analog converter's output to get a decent sine wave output.

One way to change the amount that the address counter is incremented (or decremented) is to add (or subtract) the contents of a register to (from) the counter. Then, the value loaded into this register determines the output frequency of the sine wave. Based on the desired output frequency, you would compute the required value of k:

$$k = \frac{f_o \times 2^{m+2}}{f_c}$$

This register is called the frequency register. **In addition, we will rename the address counter and call it the phase accumulator.** This is the name used for the address counter in commercial DDS ICs.

Slower Output Frequency

To get a slower frequency we can read everu value in the table, but take more than one clock pulse to read a value. We can take more clocks to read each value from the table by making the phase accumulator (address counter) wider. For example, we can make the frequency register and the phase accumulator a bits wide, where $a > m$. However, we only output the most significant m bits of the phase accumulator to the sine table. Thus, the size of the sine table doe not change. If we load the frequency register with 1, it will take $2^{(a-m)}$ clocks to cause the least significant bit of the most significant m bits to change by 1.

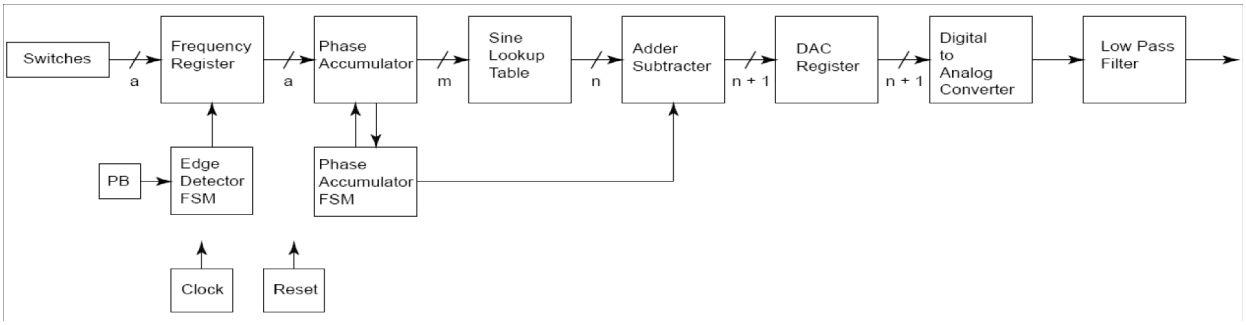
So, the relationship for the output frequency using the wider frequency register and phase accumulator would be:

$$f_o = \frac{k \times f_c}{2^{(m+2)} \times 2^{(a-m)}}$$

$$= \frac{k \times f_c}{2^{(a+2)}}$$

If this derivation is correct, the output frequency would not be a function of the table length. That seems a little counter-intuitive. Give it some thought.

Our modified DDS system is shown conceptually in the following block diagram.



The clock's output goes to all the sequential blocks of the system, as does the reset signal. Other than the clock and reset, the blocks are separated into two lines. The top line represents the data path. The second line represents the system's control.

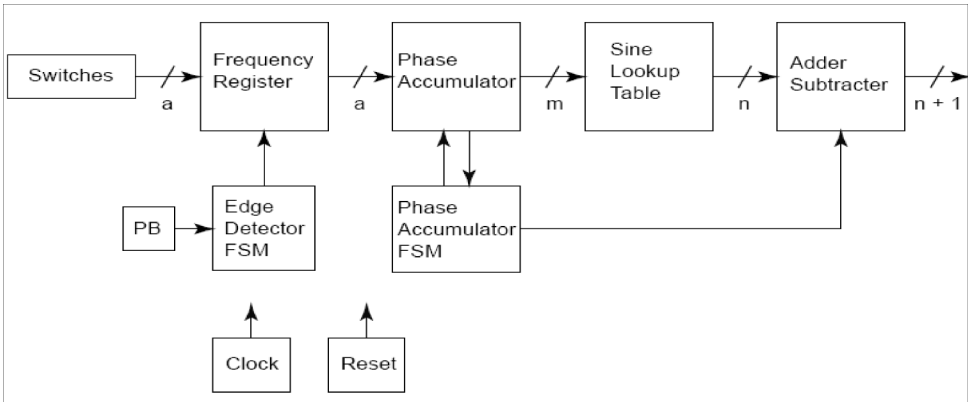
A data path is a pathway through which data flows and is modified. The modification is performed by the functional blocks in the data path. Here, the data starts with the value set on a bank of switches. This value is loaded into the frequency register and determines the output frequency of the sine wave.

The frequency register is loaded with the value from the bank of switches when pushbutton PB is pressed. The contents of the frequency register gets added to (or subtracted form) the contents of the phase accumulator on each positive edge of the system clock. The data from the phase accumulator provides the address to the sine table. The data output from the sine table is input to the Adder Subtracter. This subsystem either adds a constant to the sine table output or subtracts the sine table output from the constant. The effect of this operation is to create values between 0 and 255 which when input to the DAC will create a sine wave that is shifted by its peak value. The output of the DAC is filtered by the low-pass filter.

Thus, the data path in this DDS starts out as a digital data path and then becomes an analog data path.

In general, the control part of a system determines how and when data is modified as it flows through the data path. The control for this system consists of two finite state machines (FSMs). One is the edge detector FSM and the other is the phase accumulator FSM.

To consider this system in more detail, we first remove the analog blocks, which are not part of our VHDL design. This leaves us with the following block diagram.



The PB block represents a pushbutton and its associated debounce hardware. The pushbutton generates a single positive pulse when it is pressed and released. However, the duration of this pulse depends on how long the button is held down. But, we must have a pulse in response to the pushbutton press that is only one clock period in duration. The edge detector FSM generates this single one clock period wide pulse when it detects a positive edge transition at its input.

The address counter functionality from Laboratory 9 is separated into two blocks in this design. The phase accumulator is a counter with an input that controls whether it counts up or down. It has two status outputs that indicate when the count is greater than or equal to its maximum value and when it is less than or equal to its minimum value. The phase accumulator FSM looks at the status outputs from the phase accumulator and generates the direction of count control signal for the phase accumulator and the positive-negative control signal for the adder/subtracter.

In the future, we will want to be able to change the width of the frequency register, phase accumulator, lookup table, and adder subtracter and the length of the sine table to experiment with the performance of our design. To do this we describe an entity generically using parameters, so that for each use of the entity its structure and/or behavior is altered by the choice of parameter values. In VHDL, these parameters are called generics. Section 15.9 of the text discusses generics in detail.

Design Tasks

Design Task 1: Edge Detector

The entire system design must be fully synchronous. This means that each sequential entity in the system can only make a state transition in response to a positive edge of the system clock. No other clock sources are allowed in the system.

If a sequential entity needs to change state in response to the occurrence of an edge of a signal other than the system clock, that signal's edge must be detected and an enable signal that is one system clock wide in duration must be generated. This enable signal is then used to enable a sequential entity to change state on the next positive edge of the system clock.

Since you will later need to have some sequential entities that change state in response to the positive edge of signals other than the system clock and other sequential entities that change state in response to the negative edge of signals other than the system clock, it is best to have an edge detector that can be configured to detect either positive edges or negative edges.

The entity declaration for the desired edge detector is:

```
entity edge_det is
    port(
        rst_bar : in std_logic;     -- asynchronous system reset
        clk : in std_logic;         -- system clock
        sig : in std_logic;         -- input signal
        pos : in std_logic;         -- '1' for positive edge, '0' for negative
        sig_edge : out std_logic    -- high for one sys. clk after edge
        );
end edge_det;
```

Use the Moore FSM description of a positive edge detector in Listing 10.4.1 of the text as a basis for your design. Draw a state diagram and write code for a three process FSM. Write a non self-checking testbench to provide a verification of your edge detector. Make the clock period 1 us. See the previous laboratory's posted testbench example for code for a system clock and reset.

You can write a single concurrent signal assignment statement for a projected waveform to create the pushbutton press stimulus in your testbench.

Simulate your design using your own testbench.

Submit your state diagram and listing of your source file and non self-checking testbench as part of your prelab.

Design Task 2: Frequency Register

The frequency register holds the value that is added to the phase accumulator on each clock pulse. It uses generic `a` as shown in its entity declaration to specify its width. We will start with `a` being 14. With this value, the smallest change in the output frequency is f_c/65,536. With f_c = 1 MHz, the smallest change in output frequency is 15.2587 Hz.

In general, you would determine the value of `a` based on how small a change in the output frequency you want to be able to control.

The entity declaration for the frequency register is:

```
entity frequency_reg is
   generic (a : positive := 14);
   port(
      load : in std_logic;   -- enable register to load data
      clk : in std_logic;    -- system clock
```

```
        reset_bar : in std_logic; -- active low asynchronous reset
        d : in std_logic_vector(a-1 downto 0); -- data input
        q : out std_logic_vector(a-1 downto 0) -- register output
        );
end frequency_reg;
```

Write a behavioral description for the frequency register. Write a non self-checking testbench to provide a verification of your frequency register. See the testbench provided for Task 6 for ideas on how to generate the appropriate stimulus.

Submit your design description and non self-checking testbench as part of your prelab.

Design Task 3: Phase Accumulator
The entity declaration for the phase accumulator is:

```
entity phase_accumulator is
    generic (
        a : positive := 14;-- width of phase accumulator
        m : positive := 7   -- width of phase accum output
        );
    port(
        clk : in std_logic;        -- system clock
        reset_bar : in std_logic; -- asynchronous reset
        up : in std_logic;  -- count direction control, 1 => up, 0 => dn
        d : in std_logic_vector(a -1 downto 0);    -- count delta
        max : out std_logic;   -- count has reached max value
        min : out std_logic;   -- count has reached min value
        q : out std_logic_vector(m - 1 downto 0)   -- phase acc. output
        );
end phase_accumulator;
```

The phase accumulator increments or decrements on each positive edge of the clock, based on its `up` control input. If `up` is 1, the phase accumulator's value is increased by the value at its `d` input. However, the phase accumulator's value must never exceed all 1s. If `up` is 0, the phase accumulator's value is decreased by the value at its `d` input. However, the phase accumulator's value must never be less than all 0s.

The phase accumulator's `max` output is 1 only when the phase accumulator's value is all 1s. The phase accumulator's `min` output is 1 only when its value is all 0s.

Write a behavioral description for the phase accumulator.

Submit your design description and non self-checking testbench as part of your prelab.

Design Task 4: Phase Accumulator FSM

The phase accumulator FSM controls the direction of count of the phase accumulator based on the values of the status signals `max` and `min` from the phase accumulator. It generates an output `up` that controls the direction of count of the phase accumulator. It also generates an output named `pos_neg` that tells the adder/subtracter whether it is currently the positive half cycle of the output sine wave or the negative half cycle that is being generated.

The entity declaration for the phase accumulator FSM is:

```
entity phase_accumulator_fsm is
   port(
      clk : in std_logic; -- system clock
      reset_bar : in std_logic;  -- asynchronous reset
      max : in std_logic; -- max count
      min : in std_logic; -- min count
      up : out std_logic; -- count direction
      pos : out std_logic -- positive half of sine cycle
      );
end phase_accumulator_fsm;
```

You must first create a state diagram for a Moore FSM that implements the necessary operations. Write a three process Moore FSM to implement this entity.

Write a non self-checking testbench to provide a verification of your edge detector. Make the clock period 1 us. Simulate your design using your own testbench.

Submit your state diagram, design description and non self-checking testbench as part of your prelab.

Design Task 5: Sine Table and Adder/Subtracter

The sine table and adder/subtracter will be kept at the same size for now as they were in Laboratory 9, so they will not require generics. You can use the same code for these entities as you used in Laboratory 9. For completeness their entity declarations are repeated in this section. The entity declaration for the sine 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;
```

Note that the output of the phase accumulator is 14 bits. So we have 14 bits to address a sine table. However, our sine table only uses a 7 bit address, so only 7 bits from the phase accumulator are connected to the sine table.

The adder/subtracter uses the `pos_neg` output of the phase accumulator FSM to determine which half of the sine wave cycle is currently being generated and to modify the *n*-bit value from the sine lookup table to produce an (*n* + 1)-bit value to be input to the DAC. The output values from the adder/subtracter range from $2^{(n)}$ at the start of a cycle up to 2^{n+1} - 1 at its positive peak and down to 0 at its negative peak.

The entity declaration for the adder/subtracter 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;
```

Design Task 6: DDS System with Frequency Selection

The entities from the previous tasks are interconnected to create a direct digital synthesis system with frequency selection. This top-level system has a structural architecture and the entity declaration:

```
entity dds_w_freq_select is
    generic (a : positive := 14; m : positive := 7);
     port(
         clk : in std_logic;-- system clock
         reset_bar : in std_logic;-- asynchronous reset
         freq_val : in std_logic_vector(a - 1 downto 0);-- selects frequency
         load_freq : in std_logic;-- pulse to load a new frequency selection
         dac_sine_value : out std_logic_vector(7 downto 0);-- output to DAC
         pos_sine : out std_logic-- positive half of sine wave cycle
         );
end dds_w_freq_select;
```

Label the component entities as follows in your structural description:

U1: edge detector FSM
U2: frequency register
U3: phase accumulator
U4: phase accumulator FSM
U5: sine lookup table
U6: adder subtracter

The output `pos_sine` indicates whether the sine wave is currently in its positive half cycle or its negative half cycle. It is simply a copy of an existing internal signal brought to an output pin. It will be handy for synchronizing the oscilloscope and for trouble shooting.

A non-self-checking testbench is provided for this task. Simulate the complete system with a clock frequency of 1 MHz.

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

Laboratory Tasks

The verification for this design will be done differently than in past laboratories. The assumption is that you have written a testbench and tested each entity outside of the laboratory and know that they each work correctly stand alone. It is also assumed that you have successfully verified the top level of your design outside of the laboratory.

You will load you complete design at the beginning. You will then verify to a TA that it works in stages.

Using Aldec Active-HDL create a new workspace named `dds_w_freq_select`. In this workspace create a design named `dds_w_freq_select`. Import all your VHDL source file for the `dds_w_freq_select` and your top level testbench. Compile and simulate your design.

Lab Task1: Edge Detector and Frequency Register Verification

One of the laboratory partners, lets call that person partner A, must demonstrate to the TA that the edge detector and frequency register perform as required. This must be done by explaining to the TA what the correct operation of these two entities is and showing the TA the relevant portions of the output waveforms and the relevant signal values to prove that these entities perform properly.

Demonstrate to a TA that the edge detector and frequency register portions of your design operate properly.

Lab Task2: Phase Accumulator and Phase Accumulator FSM

The other laboratory partner, partner B, must demonstrate to the TA that the phase accumulator and phase accumulator FSM perform as required in the context of the entire system. This must be done by explaining to the TA what the correct operation of these two entities is and showing the TA the relevant portions of the output waveforms and the relevant signal values to prove that these entities perform properly.

Demonstrate to a TA that the phase accumulator and phase accumulator portions of your design operate properly.

Lab Task3: Sine Table and Adder/Subtracter

Laboratory partner A, must demonstrate to the TA that the sine table and adder/subtracter perform as required in the context of the entire system. This must be done by explaining to the TA what the correct operation of these two entities is and showing the TA the relevant portions of the output waveforms and the relevant signal values to prove that these entities perform properly.

Demonstrate to a TA that the sine table and adder/subtracter portions of your design operate properly.

Lab Task 4: DDS System with Frequency Selection
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 target FPGA. In Lattice Diamond, view and print the chip report.

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

Laboratory partner B, must explain and demonstrate to the TA that the entire design programmed into an FPGA performs as required in the test system provided. This must be done by explaining to the TA what the correct operation of the system is and showing the TA the output sine wave being generated at different specified frequencies. Be sure to explain to the TA what happens when you have very large values set on the frequency switches.

Measure your actual output frequency when you have set the frequency select register to produce 50 kHz.

Have a TA sign off whether your programmed FPGA met the design specifications.

Questions
1. Compute the values of *k* to get the following output frequencies:
a. 2 kHz
b. 10 kHz
c. 37.25 kHz
d. 50 kHz
e. 119 kHz

2. How wide would the phase accumulator have to be to be able to adjust the output frequency to within 1 Hz using a 10 MHz system clock?

3. How accurate was your output frequency when it was suppose to be 50 kHZ. What was the percent error? Show your calculations.