ESE 382 Digital System Design Using VHDL and PLDs

Spring 20, Ken Short
revised April 6, 2020 10:55 am

Laboratory 7: Simple Sequential Circuits

To be performed the week starting April 3rd. There is no prelab to submit. However, you must submit the material requested for each Task by 5:00 pm on Friday, April 8th. This same deadline applies to all laboratory sections. A document will be posted on Blackboard describing the electronic submission procedure.

Prerequisite Reading

1. Descriptive material on stimulators from Aldec Active-HDL help menu.
2. Using Stimulators - Tutorial (on Blackboard in Aldec Tutorials folder).
3. Waveform Editor Features - Tutorial (on Blackboard in Aldec Tutorials folder).

Purpose

The purpose of this laboratory is for you to write design descriptions for four simple sequential circuits. The last sequential circuit is combined with a combinational circuit to create a simple structural system.

You will also gain experience in using Aldec Active-HDL's stimulators to drive the inputs of simple sequential circuits to verify their operation. Stimulators allow you to interactively change the stimulus values to a UUT. Note that these stimulators are not part of the VHDL language. Although the preferred way to verify a sequential circuit is to write an appropriate testbench, it will be awhile (Chapter 13) before we cover that topic in detail. In the interim, you will need to be able to quickly functionally verify simple sequential circuit descriptions that you have written.

The Active-HDL simulator provides a stimulators feature that is used to provide simple stimulus (inputs) to a design entity for purposes of verification. A stimulator is a user-defined virtual stimulus that can be attached to a signal. For example, one kind of stimulator is a clock stimulator. You can specify the clock's frequency or period and its initial offset time, duty cycle, and initial value.

Another type of stimulator is a hotkey stimulator. This stimulator allows you to assign a key on your keyboard to a signal. Normally, you assign the values of 0 and 1 to this stimulator. Each time you press the key, the value of the associated signal changes state.

For a simple sequential circuit you can use a clock stimulator and several hotkey stimulators to provide the stimulus to verify the circuit's operation.

Once you have completed a simulation, you can save the resulting waveforms. You can then automatically create a testbench from the waveforms of the previous simulation by using the Test

Bench Generator Wizard. **However, you cannot automatically generate a testbench on the student version of Active-HDL, only the version in the laboratory. So, you will not be able to automatically generate a testbench from a waveform at home.** The Test Bench Generator Wizard allows you to define test vectors by selecting “Test vectors from file” and browsing to select the waveform file. The test vector file selected is the saved waveform file from the stimulator based simulation. This allows you to apply the same stimuli to a revised version of your design description or to another UUT.

To learn how stimulators work and how to use them, read the Using Stimulators tutorial on Blackboard. Also, use the index in Active-HDL’s help window to search on stimulators and read the help information on this topic.

Verifying sequential circuits requires much more thought than verifying combinational circuits. When verifying a sequential circuit, it is important to first determine all of the circuit’s features that must be verified. Next you must determine an optimal sequence of stimulus values to apply to verify these features. **These steps constitute a verification plan.**

Design Tasks

You must design three sequential circuits. The first two are variations of circuits described in the text. The third is a structural system that combines a sequential circuit and a combinational circuit.

Design Task 1: Latch versus Flip-flop

It is important to understand the differences between a latch and a flip-flop and the implications of those differences. In this task, you write a design description for each of these memory types. You then create a structural entity that connects the latch and flip-flop to the same data and clock inputs. You are then able to observe and verify the differences in operation of these two devices by applying the same stimulators to their data and clock inputs.

The entity declaration for the latch is:

```
entity d_latch is
      port(
         d : in std_logic;       -- data input
         le_bar : in std_logic; -- latch enable input (active low)
         ql : out std_logic      -- latch output
      );
end d_latch;
```

The architecture for `d_latch` must be written in behavioral style. Use stimulators to verify its operation.

The entity declaration for the flip-flop is:

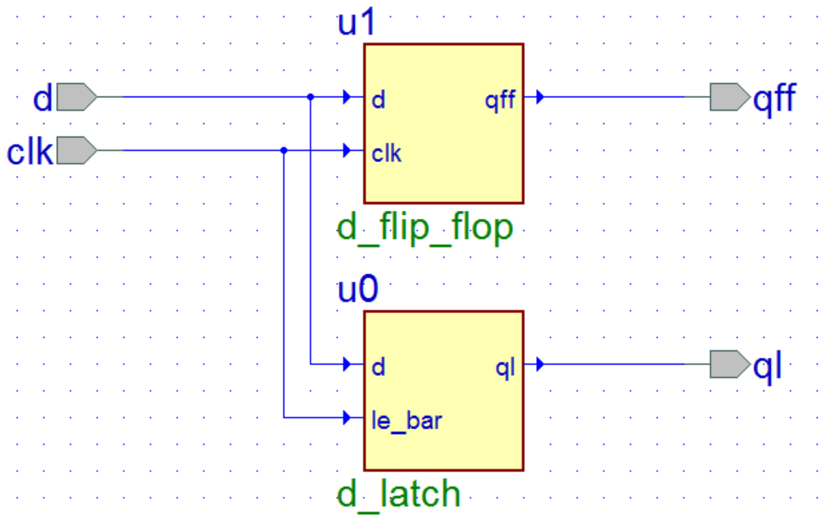
```
entity d_flip_flop is
    port(
        d : in std_logic;       -- data input
        clk : in std_logic;     -- clock input
        qff : out std_logic     -- flip_flop output
         );
end d_flip_flop;
```

The architecture for `d_flip_flop` must also be written in behavioral style. Use stimulators to verify its operation.

The top-level entity declaration is:

```
entity latch_vs_flip_flop is
   port(
      d : in std_logic;         -- data input
      clk : in std_logic;       -- clock input
      ql : out std_logic;       -- latch output
      qff : out std_logic       -- flip-flop output
      );
end latch_vs_flip_flop;
```

The architecture for `latch_vs_flip_flop` must be written in structural style. The following block diagram, generated using Code to Graphics, represents this structure:



Use stimulators to verify the operation of `latch_vs_flip_flop`. You can verify the operation of the latch and flip-flop by applying clock stimulators to both the data and clock inputs. To demonstrate transparency, the frequency of the data input must be higher than that of the clock input.

Submit your properly commented design description source files, list of circuit features to be verified, description of required stimulus waveforms, and an annotated waveform editor output showing what is being verified during each portion of the waveforms.

Design Task 2: D flip-flop with Enable Input

The entity declaration for a D flip-flop with an enable input is:

```
entity dff_en is
    port(
        d : in std_logic;        -- data input
        clk : in std_logic;      -- clock input
        en : in std_logic;       -- enable input
        rst_bar : in std_logic; -- asynchronous reset
        q : out std_logic       -- output
        );
end dff_en;
```

The `rst_bar` input is the only asynchronous input. When `rst_bar` is asserted, the flip-flop is cleared. When `rst_bar` is unasserted and `en` is asserted, the flip-flop stores its input data on a rising clock edge. When `rst_bar` is unasserted and `en` is also unasserted, the flip-flop does not store its input data on a rising clock edge.Write a behavioral design description. Use stimulators to verify your D flip-flop's functional operation.

Submit your properly commented design description source files, list of circuit features to be verified, description of required stimulus waveforms, and an annotated waveform editor output showing what is being verified during each portion of the waveforms.

Design Task 3: Mesmerizer

A sequential system is to be designed that drives a single horizontal array of seven LEDs. This array is driven by a vector named `led_array`. An array element that is 1 turns ON its associated LED. A 0 turns OFF its associated LED. You need to create a continually repeating sequence of patterns on the LED array. The sequence of patterns, represented by the `led_array` values needed to control the LEDs, is:

```
1000001
0100010
0010100
0001000
0010100
0100010
1000001
```

When this sequence of patterns is output to the LED array at a constant rate, the ON LEDs will move across the array in opposite directions.

To implement this system, you must use a sequential entity and a combinational entity connected together. The sequential entity is a binary counter named `counter`. It has an active-high enable input named `en` and an active-low synchronous reset named `rst_bar`. And, of course, it has a clock input, `clk`. The output of the counter is a vector named `count`. The counter resets to 0.

The combinational entity is named `pattern_gen`. Its input is named `pat_num` and its output is `led_array`. Entity `pattern_gen` takes the counter's output as its input and generates the values for `led_array`.

Draw a block diagram of the interconnection of the two entities to create the system. On your block diagram label entity `counter` u0 and label entity `pattern_gen` u1.

Write a complete design description for `counter`. Name its architecture `int_count`. Use an integer to store the count.

Write a complete design description for `pattern_gen`. Name its architecture `table_lookup`. Use a table lookup to implement `pattern_gen`.

Write a complete design description for the top-level entity, named `mesmerize`, that connects `counter` and `pattern_gen` to create the system. In the top-level entity, the counter is always enabled. The the inputs and outputs to `mesmerize` are: `clk`, `rst_bar`, and `led_array`.

Submit your properly commented design description source files, list of circuit features to be verified, description of required stimulus waveforms, and an annotated waveform editor output showing what is being verified during each portion of the waveforms.

Questions

1. What are the initial states of a latch and flip-flop at the very beginning of a simulation? What would be the corresponding initial states of a latch or flip-flop in an actual PLD?

2. Why is it important in structural designs to check the Code-to-Graphics diagram after compilation or check the HDL Analyst hierarchical view after synthesis?