Counters, Timers and Real-Time Clock

Introduction

In the previous lab, you learned how the Clocking Wizard can be used to generate a desired clock frequency and how the IP Catalog can be used to generate various IP including counters. These two functional circuits are fundamental circuits used in creating timers and real-time clocks. In this lab, you will generate several kinds of counters, timers, and a real-time clock. *Please refer to the Vivado tutorial on how to use the Vivado tool for creating projects and verifying digital circuits*.

Objectives

After completing this lab, you will be able to:

- Define a parameterized model
- Model counters using behavioral modeling
- Design counters using the IP Catalog tool
- Compare and contrast the counters developed using the behavioral modeling and the IP Catalog
- Design timer circuits using the cores and using additional circuits modeled in HDL
- Create a real-time clock

Generic Declarations Part 1

The VHDL language supports model parameterization, i.e. write a model in HDL and reuse the same model number of times by passing different constant values (e.g. bus widths and delays). The `generic` construct can be used to define a parameterize-able model. Here is an example of defining a width as a parameter with a default value of 7.

```
generic (WIDTH : integer := 7);
```

The parameter (WIDTH) must be defined before it can be used in the code. In the example above, it is declared as a `generic` parameter that can be passed between modules. Here is an example, in VHDL, where the constant is declared in an entity statement and is used in a process.

```
entity generic_example is
      generic (WIDTH : integer := 7);
...
architecture behavior of generic_example is
begin
      for I in WIDTH loop
            byte_data(I) <= ‘1’;
            data_dest(WIDTH - I) <= data_source(I);
      end loop;
end process;
```

Note that the parameter WIDTH is defined before it is used. In the example above, assuming *`I`* was initialized to 0, the `for` loop will create eight parallel bits all set to 1 in the byte_data register. Similarly, a byte swapped bus will be created between `data_dest` and `data_source`.

Generic statements can also be used to define time delays. In VHDL, the delay needs to be declared as a **time** type with the units defined (for example: *ns*). The parameter can then be used in place of the numerical value of the delay where specified. In the example below data_output is assigned the value of ‘1’ after a 5 ns delay.

```
entity generic_delay_example is
```

```
      generic (delay_a : time := 2 ns);
...
architecture behavior of generic_delay_example is
begin
      data_output <= '1' after delay_a;
end process;
```

Parameters declared as generics can also be passed from a top level module to an instantiated module, overwriting the parameters declared value in the lower level module. This allows instantiated code to be re-used without needing re-writes. We start with the lower level module:

```
entity instance_A is
      generic (example_A : time := 5 ns);
      port (
            in_A : in std_logic;
            out_A : out std_logic
            );
architecture behavior of instance_A is
begin
...
      out_A <= in_A + '1' after instance_A;
end process;
```

Now we can overwrite the generic in instance_A by passing a parameter value from a higher level module that instantiates it.

```
entity top_level is
...
architecture behavior of top_level is
component instance_A is
      generic (example_A : time);
      port (
            in_A : in std_logic;
            out_A : out std_logic;
            );
end component;
...

begin
      A1 : instance_A
      generic map (example_A => 20 ns)
      port map (
                  in_A => in_A,
                  out_A => out_A
            );
...
end process;
```

Notice that we are setting the example_A parameter value to be 20 ns in the code above. 20 ns will override the 5 ns value in the instance_A module. This way, a parameter from a higher level module can propagate to a lower level module via passing without needing to change the instantiated code. If you want to use the default value then you do not need to define the `generic map` statement in the instantiation.

1-1. Design a carry-look-ahead adder similar to that you designed in Part 4-1 of Lab 2 but using gate-level modeling. Define 2 units delay for each kind of gate that you use in the full-adder circuit using the generic statements. When creating hierarchical models, use 1 ns delay for inverters, 3 ns delay for *and* and *or* gates, and 4 ns delay for *xor* gates. Develop a testbench to verify the functionality and to see the delays propagated through the hierarchy. Output to the simulator console the expected versus actual values for both the sum and the cout. Do not implement the design in hardware.

Counters and Used Resources Part 2

Counters are fundamental circuits used in measuring events. They may be classified as simple freerunning binary up counting, up/down counting, modulo-k counting, Johnson counting, gray-code counting, and special sequence counting. In a given design or set of designs, designer use counters of different widths a number of times. The parameter and defparam statements covered in Part 1 are used to obtain different widths.

Counters normally use adder and subtractor circuits which can be implemented in the Xilinx FPGA either using LUTs and/or FFs, or DSP48 slices. You can control the type of resources to be used by using the synthesis attribute, called USE_DSP48 with a value of "yes" or "no", in the VHDL code. USE_DSP48 instructs the synthesis tool how to deal with synthesis arithmetic structures. By default, mults, mult-add, mult-sub, mult-accumulate type structures go into DSP48 blocks. Adders, subtractors, and accumulators can also go into these blocks but by default, are implemented with the fabric instead of with DSP48 blocks. The USE_DSP48 attribute overrides the default behavior and force these structures into DSP48 blocks. The attribute can be placed in the RTL on signals and modules.

Here is a simple example of using the attribute in a VHDL code:

```
entity use_dsp48_example is
    port (
        clk : in std_logic;
        A : in std_logic_vector(7 downto 0);
        B : in std_logic_vector(3 downto 0);
        Q : out std_logic_vector(11 downto 0)
    );
attribute use_dsp48 : string;
attribute use_dsp48 of use_dsp48_example : entity is "yes";
end use_dsp48_example;
...
```

2-1. Design an 8-Bit up/down counter using behavioral modeling. Your model should define COUNT_SIZE as a generic and use it in the model. The counter will use the on-board 100 MHz clock source. Use the clocking wizard to generate a 5 MHz clock, dividing it further by a clock divider to generate a periodic one second signal. <u>Set the synthesis attribute to not to use the DSP48 slices.</u> Use the BTNU button as reset to the circuit, SW0 as enable, SW1 as the Up/Dn (1=Up, 0=Dn), and LED7 to LED0 to output the counter output. Go through the design flow, generate the bitstream, and download it into the Nexys4 board. Verify the functionality. Fill out the following information after reviewing the Project Summary tab.

1. Number of BUFG/BUFGCTRL _____________
   Number of slices used: _____________

Number of registers used:	_____________
Number of DSP48A1 slices used:	_____________
Number of IOs used:	_____________

2-2. Use the 8-Bit up/down counter design from 2-1. <u>Set the synthesis attribute to force the use of the DSP48 slices.</u> Use the BTNU button as reset to the circuit, SW0 as enable, SW1 as the Up/Dn (1=Up, 0=Dn), and LED7 to LED0 to output the counter output. Go through the design flow, generate the bitstream, and download it into the Nexys4 board. Verify the functionality. Fill out the following information after reviewing the Project Summary tab.

2.	Number of BUFG/BUFGCTRL	_____________
	Number of slices used:	_____________
	Number of registers used:	_____________
	Number of DSP48A1 slices used:	_____________
	Number of IOs used:	_____________

2-3. Design an 8-Bit up/down counter using the 8-Bit core generated using the IP Catalog system. When generating the core, set the setting to use the fabric resource. Use the clocking wizard to generate a 5 MHz clock from the on-board 100 MHz clock source, dividing it further by a clock divider to generate a periodic one second signal. <u>Set the synthesis attribute to not to use the DSP48 slices.</u> Use the BTNU button as reset to the circuit, SW0 as enable, SW1 as the Up/Dn (1=Up, 0=Dn), and LED7 to LED0 to output the counter output. Go through the design flow, generate the bitstream, and download it into the Nexys4 board. Verify the functionality. Fill out the following information after reviewing the Project Summary tab.

3.	Number of BUFG/BUFGCTRL	_____________
	Number of slices used:	_____________
	Number of registers used:	_____________
	Number of DSP48A1 slices used:	_____________
	Number of IOs used:	_____________

2-4. Use the 8-Bit up/down counter design from 2-3 but with the counter regenerated to use the DSP48 slices. <u>Set the synthesis attribute to force the use of the DSP48 slices.</u> Use the BTNU button as reset to the circuit, SW0 as enable, SW1 as the Up/Dn (1=Up, 0=Dn), and LED7 to LED0 to output the counter output. Go through the design flow, generate the bitstream, and download it into the Nexys4 board. Verify the functionality. Fill out the following information after reviewing the Project Summary tab.

4.	Number of BUFG/BUFGCTRL	_____________
	Number of slices used:	_____________
	Number of registers used:	_____________
	Number of DSP48A1 slices used:	_____________
	Number of IOs used:	_____________

Timers and Real-Time Clock Part 3

Timers and real-time clock are natural applications of counters. The timers include a stop-watch timer and a lapse timer The real-time clocks are used in a day to day life for keeping track of the time of the day.

3-1. Design a stop-watch timer using the IP Catalog system to generate an appropriate sized (precision) counter core with the desired input control signals to measure a time up to five minutes at a resolution of one-tenth of a second. Instantiate the core a number of required times and add the required additional circuitry to display the time in M.SS.f format on the four 7-segment displays. The design input will be a 100 MHz clock source, a reset signal using the BTNU button, and an enable signal using SW0. When the enable signal is asserted (ON) the clock counts, when it is de-asserted (OFF) the clock pauses. At any time if BTNU is pressed the clock resets to the 0.00.0 value. Verify the design functionality in hardware using the Nexys4 board.

3-2. Design a countdown timer using the behavioral modeling to model a parameterized counter down counter with the desired input control signals to show the count down time from a desired initial value set by the two slide switches of the board at a second resolution. Display the time in MM.SS format on the three 7-segment displays. The design input will be a 100 MHz clock source, a re-load signal using the BTNU button, an enable signal using SW0, and SW7-SW6 as the starting value in number of whole minutes. When the enable signal is asserted (ON) the clock counts, when it is de-asserted (OFF) the clock pauses. When the BTNU is pressed the timer loads to MM.00, where the value of MM is determined by the SW7-SW6 settings (MM=00 will be ignored). Verify the design functionality in hardware using the Nexys4 board.

3-3. Design a real-time clock using the IP Catalog system to generate an appropriate sized (precision) counter core with the desired input control signals. Instantiate it two times and add the required circuit to display the time in MM.SS format on the four 7-segment displays. The design input will be a 100 MHz clock source and a reset signal using the BTNU button. At any time if BTNU is pressed the clock resets to 00.00. Verify the design functionality in hardware using the Nexys4 board.

Conclusion

In this lab, you learned how to parameterize models using generic statements so they can be used in subsequent designs. You also designed and compared the resources usage of counters modeled behaviorally and using the IP Catalog tool. You also designed clocking applications using the counters.

Answers

1.	Number of BUFG/BUFGCTRL	______2_______
	Number of slices used:	______13______
	Number of registers used:	______32______
	Number of DSP48A1 slices used:	______0_______
	Number of IOs used:	______12______
2.	Number of BUFG/BUFGCTRL	______2_______
	Number of slices used:	______5_______
	Number of registers used:	______1_______
	Number of DSP48A1 slices used:	______2_______
	Number of IOs used:	______12______
3.	Number of BUFG/BUFGCTRL	______2_______
	Number of slices used:	______12______
	Number of registers used:	______32______
	Number of DSP48A1 slices used:	______0_______
	Number of IOs used:	______12______
4.	Number of BUFG/BUFGCTRL	______2_______
	Number of slices used:	______6_______
	Number of registers used:	______9_______
	Number of DSP48A1 slices used:	______1_______
	Number of IOs used:	______12______