Counters, Timers and Real-Time Clock

Introduction

In the previous lab, you learned how the Architectural Wizard can be used to generate a desired clock frequency and how the CORE Generator system can be used to generate various cores 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 real-time clocks. *Please refer to the PlanAhead tutorial on how to use the PlanAhead 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 CORE Generator tool
- Compare and contrast the counters developed using the behavioral modeling and the CORE Generator
- Design timer circuits using the cores and using additional circuits modeled in HDL
- Create a real-time clock

Parameter and defparam Statements Part 1

The Verilog HDL language supports model parameterization, i.e. write a model in HDL and reuse the same model number of times by passing different parameter values (typically widths and delays). Use the `parameter` construct to define a parameterize-able model. There are two ways to change the values: (i) during the instantiation of the module (shown in the example below), and (ii) using a `defparam` statement. Here is an example of defining a width as a parameter with a default value of 4.

```
parameter WIDTH = 4;
```

The parameter (WIDTH) must be defined before it can be used in the model. Here is an example, in Verilog-2001 standard, of a dataflow model of reduction-`or` functionality.

```
module reduction_or #(parameter WIDTH = 4)(input ain[WIDTH-1:0], output
result);
assign result = |ain;
endmodule

module test_reduction_or_tb;
reg [3:0] ain1;
reg [6:0] ain2;
reg [9:0] ain3;

wire result1, result2, result3;

reduction_or U1 (ain1, result1);  // use default value of 4
reduction_or #(7) U2 (ain2, result2); // use value of 7
reduction_or #(10) U3 (in3, result3); // use value of 10

endmodule
```

Note that the parameter WIDTH is defined before it is used. In case of multiple parameter definitions, the values listed during instantiation maps to the order in which the parameters are defined. In Verilog-2001 standard, all parameters values must be defined, even when some of the values are default values, in case of multiple parameters.

For example,

```
module FA;
parameter AND_DELAY=2, XOR_DELAY=3;

… //gate-level model which uses AND_DELAY and XOR_DEALY in the gate instantiations
endmodule

module FA2BIT( ports listing);
FA #(4,5) f1(ports connections); // AND_DELAY will be 4 and XOR_DELAY will be 5
FA #(1,2) f2(ports conenctions); // AND_DELAY will be 1 and XOR_DELAY will be 2
endmodule
```

The `defparam` statement can also be used to change the parameters value from a higher level module by referencing the module instantiation. For example,

```
module TOP;
  defparam fa2.f1.AND_DELAY=4, fa2.f1.XOR_DELAY=5, fa2.f2.AND_DELAY=1,
fa2.f2.XOR_DELAY=2;

  FA2BIT fa2(ports connections);
endmodule
```

The parameter definition can be in the same file where it is used or in a separate file (like a header file in C programming language) and the header file could be included in the model using the `` `include `` directive. For example,

Let us define the following parameter in a file called header.v

```
parameter WIDTH = 4;
```

Then to use it in a file where reduction_or and test_reduction_or_tb modules are defined, use the following line at the beginning of the file.

```
`include header.v    // you can add a full path to the file if it is located
in some other directory path
```

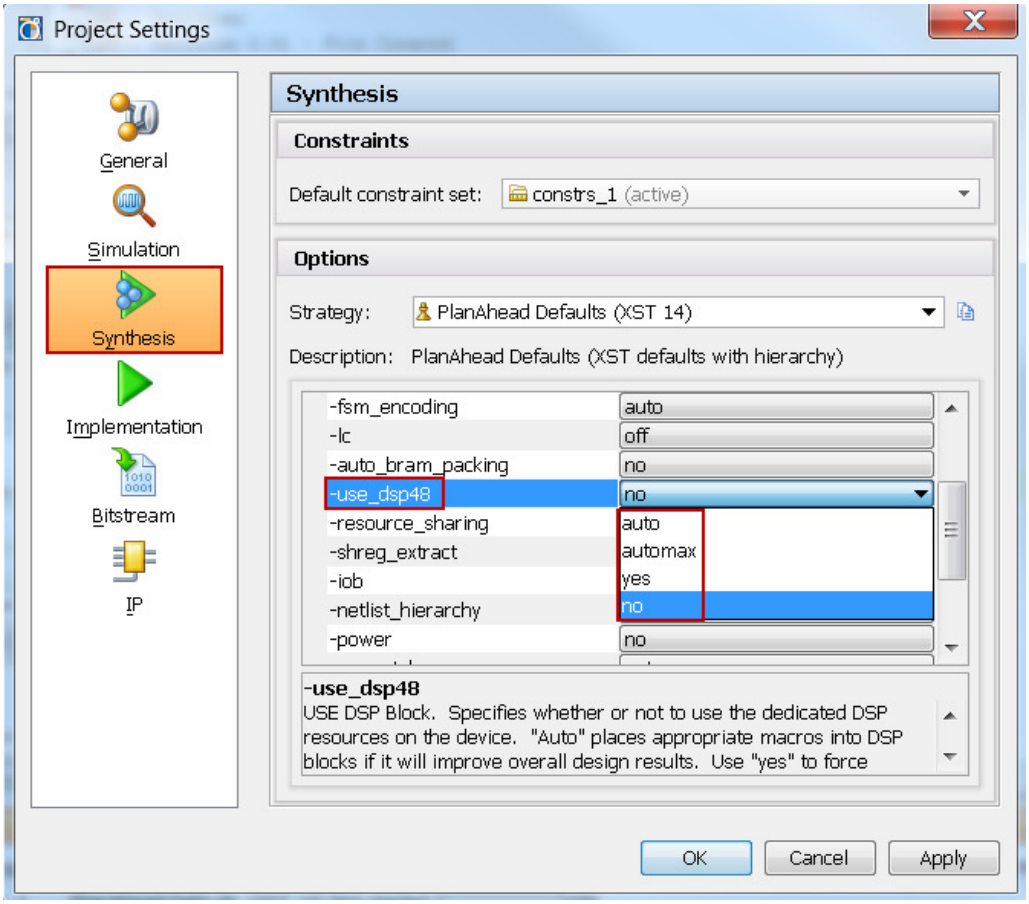
1-1. Design a carry-look-ahead adder similar to that you designed in Part 4-1 of Lab 2 but using gate-level modeling. Define 2 units delay for each kind of gate that you use in the full-adder circuit using the parameter statements. When creating hierarchical models, use 1 unit delay for inverters, 3 units delay for *and* and or gates, and 4 units delay for *xor* gates. Use the module instance parameter value assignment statement method. Develop a testbench to verify the functionality and to see the delays propagated through the hierarchy. Do not need to implement the design in hardware.

1-2. Modify the carry-look-ahead adder of 1-1 using the defparam statements, changing the values of the delays from the testbench. Perform the behavioral modeling simulation and observe the delays propagated through the hierarchy. Do not need to 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 free-running 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 or FFs, or DSP48 slices. You can control the type of resources to be used by setting an appropriate synthesis property. In PlanAhead, select **Project Settings > Synthesis > PlanAhead Defaults (XST 14) Strategy > -use_dsp48** and either select **no** to force the use of LUTs and FFs, **yes** to force the use of the DSP48 slices, or **automax** to let the tools decide depending on the width and type of the operations.



2-1. Design an 8-Bit up/down counter using behavioral modeling. Your model should define COUNT_SIZE as a parameter 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. Set the synthesis property to not to use the DSP48 slices. 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 Nexys3 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. Set the synthesis property to force the use of the DSP48 slices. 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 Nexys3 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 CORE Generator 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. Set the synthesis property to not to use the DSP48 slices. 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 Nexys3 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. Set the synthesis property to force the use of the DSP48 slices. 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 Nexys3 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 CORE Generator 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 Nexys3 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 Nexys3 board.

3-3. Design a real-time clock using the CORE Generator 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 Nexys3 board.

Conclusion

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

Answers

1. Number of BUFG/BUFGCTRL _____2_____
   Number of slices used: _____12_____
   Number of registers used: _____33_____
   Number of DSP48A1 slices used: _____0_____
   Number of IOs used: _____12_____

2. Number of BUFG/BUFGCTRL _____2_____
   Number of slices used: _____11_____
   Number of registers used: _____33_____
   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: _____33_____
   Number of DSP48A1 slices used: _____0_____
   Number of IOs used: _____12_____

4. Number of BUFG/BUFGCTRL _____2_____
   Number of slices used: _____11_____
   Number of registers used: _____25_____
   Number of DSP48A1 slices used: _____2_____
   Number of IOs used: _____12_____