**ECE 6213 – Design of VLSI Circuits**

**Fall 2022 – The George Washington University – Dr. Jerry Wu**

**Osama Yousuf – HW 4**

**Problem 1 – Chapter 7 P2 from textbook – UART Clock Generator**

Diagram

Description automatically generated

The following is a breakdown of my implementation of the different modules for the UART\_Clock\_generator, as well as corresponding verification tests.

**Module 1:** Divide\_by\_13

For verification, I simply recreated the waveforms from the book.

The source file is: *divideby13.v*, and the testbench is *divideby13\_tb.v*

Diagram

Description automatically generated with medium confidence

Graphical user interface

Description automatically generated

**Module 2:** Divide\_by\_8

For verification, I simply recreated the waveforms similar to Module 1. The output “clock\_by\_8” is high when the internal counter variable asserts a value of 7 (in line with clk\_1 from the previous code).

The source file is: *divideby8.v*, and the testbench is *divideby8\_tb.v*

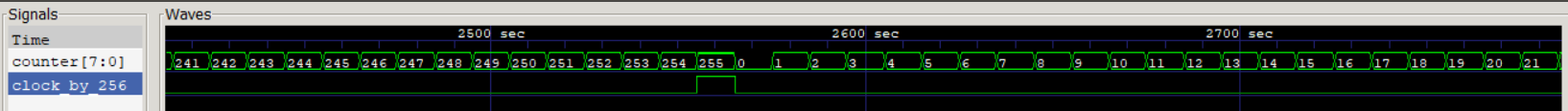
**Graphical user interface, application

Description automatically generated**

**Module 3:** Divide\_by\_256

For verification, I simply recreated the waveforms similar to Module 1. The output “clock\_by\_256” is high when the internal counter variable asserts a value of 255 (again, in line with clk\_1 from the previous code), as shown in the attached waveform.

The source file is: *divideby256.v*, and the testbench is *divideby256\_tb.v*

****

The 8-bit counter is exposed as an output since it has to be used as an input to the multiplexer for the UART\_Clock\_Generator module.

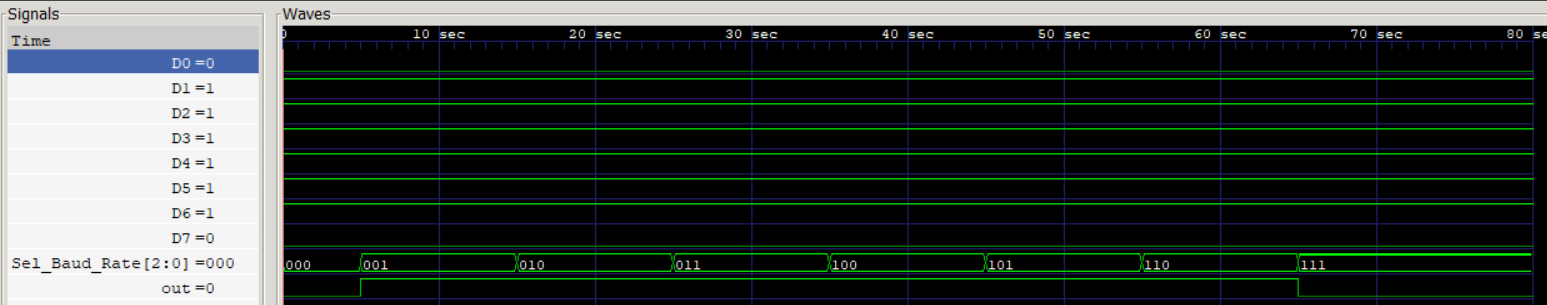
**Module 4:** 8 to 1 multiplexer

The source file is: *8to1mux.v*, and the testbench is *8to1mux \_tb.v*

For testing, I initialize each of the 8 inputs randomly, cycle through the select pins, and assert the correct output. I also implemented error monitoring, watchdog signals, as well as self-reporting in my testbench. The testbench executes with 0 errors.

Text

Description automatically generated



The waveform shows that the inputs are indeed correctly multiplexed on the out pin, based on the 3-bit Sel\_Baud\_Rate input pin.

**Overall Module:** UART\_Clock\_Generator

For verification, my testbench cycles through the 8 possible values of the “Sel\_Baud\_Rate” pin, and calculates the frequency of the two generated clock signals “Clock” and “Sample\_Clock” at a specified resolution parameter. This resolution parameter relates to how long the UART\_Clock\_Generator is allowed to run (in time cycles) at a given Sel\_Baud\_Rate value. For example, a resolution of 0.05 translates to a total time of 0.025 seconds per Sel\_Baud\_Rate, and a higher resolution of 2.00 translates to a total time of 1 seconds. The frequency of Clock and Sample\_Clock is simply extrapolated for a given resolution.

Verification results at different resolutions are presented below.

1. At very low resolution (0.025 second)

Table

Description automatically generated with medium confidence

1. At medium resolution (0.05 second)

Text

Description automatically generated

1. At high resolution (1.00 second):

Text

Description automatically generated

As can be seen, the frequencies of “Clock” and “Sample\_Clock” both converge to the true frequencies described in Table P7-2 in the book, shown below. I did not have the computing resource to compute the values at a higher resolution (for example, 10.00 seconds). The reported frequency values of the baud rate pairs from my testbench at a resolution of 1.00 seconds are within a range of +-5% from the true values from the book, and so are acceptable to justify the verification for our purposes.

Table

Description automatically generated

The source file is: *uart.v*, and the testbench is *uart \_tb.v*

**Note:** I have used error monitors, watchdog for finishing my simulations, self-checking, self-reporting, verilog events and tasks, and $random timings in my testbenches.