CPE 322 Digital Hardware Design Fundamentals

Simulation Assignment #2

Simulation of Finite State Machine Designs using a Verilog Testbench

The goal of this assignment is to give you an opportunity to model in Verilog in a structural manner and verify through simulation your structural implementation of the sequential adder that you created for Part 2 of Homework 2. This assignment should also give you some practical experience in creating a basic Verilog testbench that supplies the input stimulus to your gate-level implementation.

Reference

In this laboratory you should use the *Questa*® simulator in its stand-alone mode. A short video that describes how to use the *Questa* Simulator by itself, independent of *Quartus*®, can be referenced under the *Panopto*® tab on the CPE 322 Canvas® site and is entitled "Using Verilog Testbenches within Questa®".

Background

The previous Homework 2 assignment asked you to develop multiple implementations of a general sequential adder which would add two arbitrary length binary numbers **A** and **B** to produce a sum **S** of the same length as **A** and **B**. These two inputs were assumed to be sent bit-serially in time, with the least significant bit being sent first. The output, **S**, is also assumed to produce the sum in a bit-serial manner starting with the least significant bit first (see Figure 1). The serial adder design also contained a **Reset** and an **Enable** input. The **Reset** input is the means by which an external controller puts this design back into its initial state so it can begin the summing process for the next number. The Enable input allows an external controller to pause your design (in effect skipping clock cycles) so that it ignores inputs that are not yet valid. It is the job of the controller to make sure the Boolean network has been reset when a new number is sent and that the network will only be enabled when each input is valid.

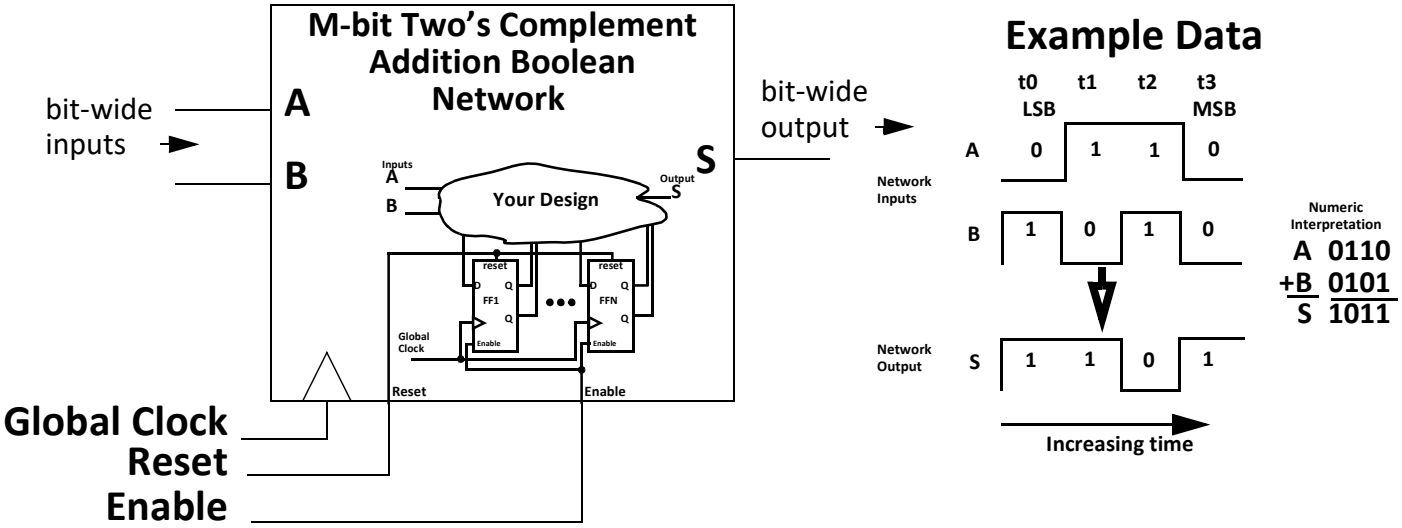


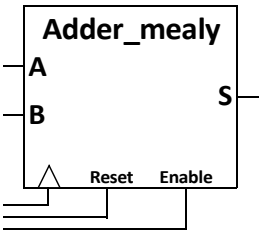
Figure 1: Functional Overview of Sequential Adder

Assignment Overview

In this simulation assignment you are being asked to create a structural model of the Mealy sequential adder implementation that you created in Part 2 of Homework 2. This implementation utilized 2-level AND, OR, and NOT gate logic and a D Flip-flop. You will then be asked to verify the correct operation of your implementation through simulation. To accomplish this you will first develop a structural simulation model in Verilog. Then you will develop a test sequence that drives your simulation model in a manner that all possible state transitions in your Mealy State diagram for the sequential adder are activated. Then you will convert your sequence over to a Verilog testbench that will drive your design. Finally, you will use the *Questa*® simulator to verify your that your design works as expected.

Part 1: Development of a Structural Verilog Model

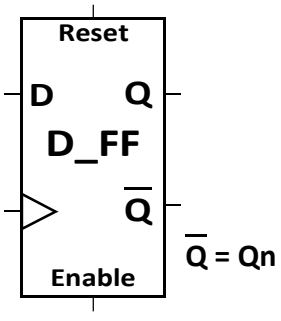
In this part of the assignment you are to develop a structural model in Verilog and place it in a file that is named ***Adder_mealy.v***. The model should be fully structural in nature, only containing Verilog logic primitives (such as ***and***, ***or***, and ***not***) and a D flip-flop, named **D_FF**, which should also be treated in your structural representation as a base component. Your module itself should be given the name **Adder_mealy** and contain the exact port-list variable names that are shown below in Figure 2.




```
// Structural Mealy adder model from Homework 2 Part 2
module Adder_mealy (input Clock, A, B, Enable, Reset, output S);
  // Enter your fully structural model for your sequential Mealy
  // implementation of the arbitrary bit length adder here. This
  // implementation should be created using the Verilog primitives of
  // and, or, and not, as well as an instantiation of the D Flip-flop
  // module as defined by the D_FF module shown below.
endmodule
```

Figure 2: Structural Verilog Model, Adder_mealy, Template

Flip-flops are not included in the basic Verilog primitive set so you will need to instantiate an external component that itself is modeled in Verilog in some way. You can utilize the D flip-flop model for such a component which is shown in Figure 3 below:




```
// Rising Edge-triggered D flip-flop with synchronous Reset
module D_FF (input Clock, D, Reset, Enable, output reg Q, Qn);
  always @ (posedge Clock)
    begin
      if (Reset==1) begin Q=0; Qn=1; end
      else if (Enable==1) begin Q=D; Qn=~D; end;
    end
endmodule
```

Figure 3: D Flip-Flop Component Model

This flip-flop model is a behavioral one, but at the intended level of abstraction you are treating the D flip-flop as a component to your design and are using it as a basic building block. As far as you are concerned the D flip-flop could itself be build using combinational logic with feedback or with some lower-level representation. You only desire for it to perform its function correctly. Place this D flip-flop module for the D flip-flop directly in your ***Adder_mealy.v*** file along with the your **Adder_mealy** module.

Part 2: Input Pattern Creation

In this part of the assignment you are to create a stimulus input pattern sequence that will fully activate all state transition arcs of your finite state machine graph for the Mealy adder. This means that your input sequence should drive your finite state machine through all of its possible state transition that are in your finite state machine representation (including cases where you transition back to the same state you are in). When multiple combinations of inputs values of **A** and **B** cause the same state transition to occur, your pattern should drive the state graph through all of these combinations that cause this state transition even if they share the same state transition arc on the state graph. When you have developed this pattern clearly state the corresponding numeric values of input variables **A** and **B** that this represents, assuming that your implementation is first reset and then the individual bits from **A** and **B** are sent in the order of lessor significant bits first as specified in the design requirements. You should also determine what is the value of **S** that you are expecting if your design is working correctly based upon your knowledge of what **A** + **B** should equal to given the binary interpretation of these numbers. You should also record the number of elements in each input pattern for **A** and **B**. This is in effect the size of the numbers being added together. Place this information in a pdf file which is named ***Adder_mealy_input_seq.pdf***.

Part 3: Verilog Testbench Generation

In this part of the assignment you are to create a Verilog testbench file that encodes the input sequence that you created in Part 2. Name this file ***Adder_mealy_tb.v***. The file should be created in the following manner:

- A module that is named **Adder_mealy_tb** should be created which has a null port list.
- Inside the **Adder_mealy_tb** module you should declare the internal **reg** variables that will be used to drive your **A**, **B**, **Clock**, **Reset**, and **Enable** inputs to your **Adder_mealy** module.
- Also inside the **Adder_mealy_tb** module declare an internal **wire** variable to receive the value of the **S** output of your design.
- Instantiate your **Adder_mealy** module structurally as the unit under test. Give it the reference name **UUT**. You can use either port-order or port-name association for this instantiation.
- Provide your stimulus using procedural constructs that are contained within either or both an ***initial*** or ***always*** region within the **Adder_mealy_tb** module.
  -Your design should be clocked with a 50% duty cycle clock that has a period of 100 ns. Your clock can be free-running or designed to stop at the end of the simulation.
  -All input signals should be driven on the non-active edge of the clock (falling edge) of the clock.
  -Drive the **Enable** signal to a logic high for the duration of the reset operation and test pattern sequence. (You are not required to fully test that the **Enable** input works correctly in this assignment.)
  -At the beginning of the simulation use the first clock cycle to reset your simulation so it can add a new set of numbers.

Part 4: Design Verification of Mealy Adder using the Questa® Simulator

In this part of this assignment you are to use the *Questa®* simulator in its stand-alone mode to verify the correct functionality of your design.

-Create a new *Questa®* project. In this project include both the ***Adder_mealy_tb.v*** and the ***Adder_mealy_tb.v*** files that you created. Compile all files.

-Select the **Adder_mealy_tb** module as the top-level module and enter the simulation mode.

-Select the signals **Reset**, **Clock**, **A**, **B**, **S**, and **Enable** for viewing (to do this use the ***add list*** and **add wave** simulation probe commands.)

-run the simulation for a time period that is just long enough for the reset cycle and the number of clock cycles required for your input sequence to complete.

-analyze the output **S** produced by your sequence, verify that it is correct. If not make changes to your model and/or to your original design and then repeat this process.

-Create a screen shot of the waveform, and save the screen shot into a PNG type image file. Then edit this file using a graphics editor such as paint so it is clear where you are reading the output relative to the active edge of your clock when you are determining the values of **S**. Then save this edited file to the file ***Adder_mealy_wave.png***.

Note: Before creating the screen shot, adjust the waveform output in *Questa®* so that the names of the signals can be easily seen. It is may be useful to *undock* the wave window so that it can be viewed independently of the rest of the Questa windows and so that a better screen shot can be taken. To undock this window just click on the middle button on the upper right-hand side of the Wave window as shown in Figure 4.

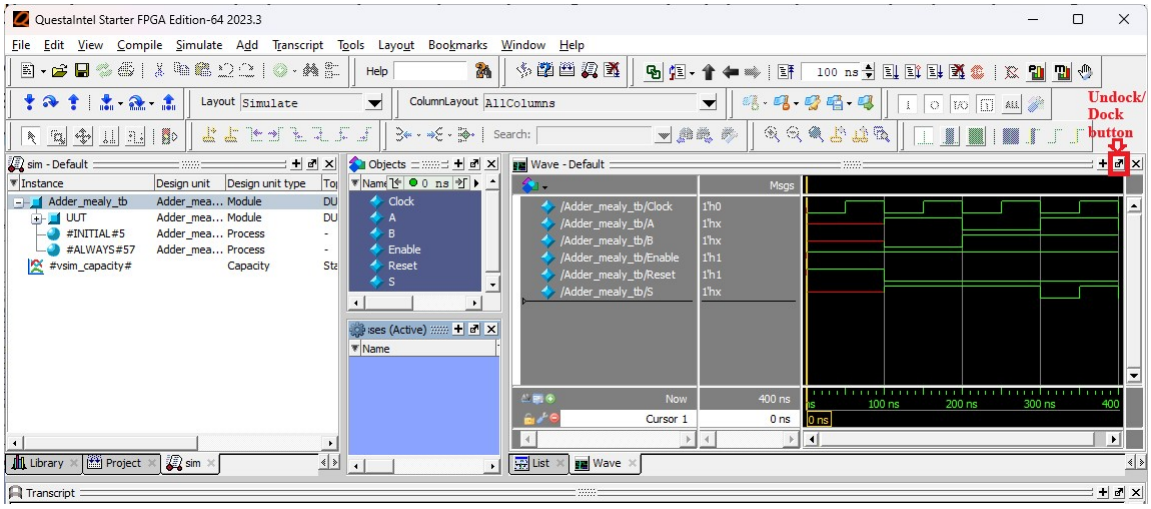


Figure 4: Undocking the Questa® Waveform window

Hint: If you want to expand the waveform so it fills the entire window use the zoom tools or press the "f" on the keyboard after you have selected this window.

-Create a screen shot of the textual List output, and save the screen shot into a PNG type image file. Before doing this set the List preferences so that it displays on delta delays. After creating the PNG file, then edit it using a graphics editor such as paint so it is clear where you are reading the output relative to the active edge of your clock when you are reading the values of **S**. Then save this edited file to the file ***Adder_mealy_list.png***.

Deliverables

Place all of the files in a single folder and use standard zip compression to place all of the files into a single zip file. Give this file the name [charger id]_Sim2.zip, where the [charger id] represents the first part of your UAH email address. The files that should be present in this zip file include:

1. ***Adder_mealy.v*** (from Part 1),
2. ***Adder_mealy_input_seq.pdf*** (from Part 2),
3. ***Adder_mealy_tb.v*** (from Part 3),
4. **Adder_mealy_wave.png** and **Adder_mealy_list.png** (from Part 4).

Submission

This assignment is to be submitted on Canvas before its due date. Acceptable file format is zip. This simulation assignment will be graded based on correctness, completeness and clarity of presentation. A rubric is provided.