**Verilog Lab 1 (ECS1005)**

**Objectives:**

The objectives of this lab are as follows

* To familiarize you with the Replit working environment
* To practice Working with basic gate based modelling in Verilog
* Understand and implement the concept of hierarchy in Verilog
* Understand the waveform signals when working with Verilog

**Task 1: Write Verilog code for a full adder**

A Full adder is a circuit that adds three bits. These three one-bit inputs to the full adder are named as **A**, **B** and **carry** **in (Cin)**. It generates two outputs, one bit each, called the **sum (S)** and the **carry out (Cout)**. The sum and carry generated for various input combinations of **A, B** and **Cin** can be seen as below. (Since there are three inputs in total to the full adder function, we will have a total of 8 combinations in its truth table).

Table

Description automatically generated with low confidence

Diagram

Description automatically generated

Using the truth table for **S** and **Cout**, lets carry out a K-map based optimization to get their minimal Boolean SOP expressions.

Diagram

Description automatically generated

For the **Cout** circuit, we can see that the optimized SOP expression for the three groups of 1’s is

**Cout = Cin.B + Cin.A+ A.B**

For the Sum circuit, the optimized SOP expression is simply derived from the 4 groups of 1 bit each (no larger grouping is possible).

**S = A’.B’.Cin + A’.B.Cin’ + A.B’.Cin’ + A.B.Cin**

Figure below shows the gate level implementations of full adder circuit for both **Cout** and **S** outputs. Let’s give some names to the 3 unnamed wires in **Cout** circuit (e.g., C\_wire0, C\_wire1, C\_wire2) and 4 unnamed wires in S circuit implementation (e.g., S\_wire0, S\_wire1, S\_wire2 and S\_wire3). We also need inverted versions of the three inputs **A**, **B** and **Cin**; we call these internal wires not\_A, not\_B and not\_Cin, respectively**.**

A diagram of a circuit

Description automatically generated

Let’s now write the Verilog code for a Full adder module. First log into your Replit account on the Google chrome browser and open the following Repl (it is a skeleton Repl for full adder we are working on).

<https://replit.com/@AyeshaKhalid5/ECS1005-Verilog-Lab1-Task1>

To make a local copy of this Repl under your account, click Fork button on the left of the following window.

A screenshot of a computer

Description automatically generated

You may change the project name if you wish. The Repl contains two files, *full\_adder.v* and *tb\_full\_adder.v.* Do not change the testbench file (*tb\_full\_adder.v*). Lets work on the *full\_adder.v* to complete our full adder circuit. The following port interface is already provided, the full\_adder is the module name.

A white background with black text

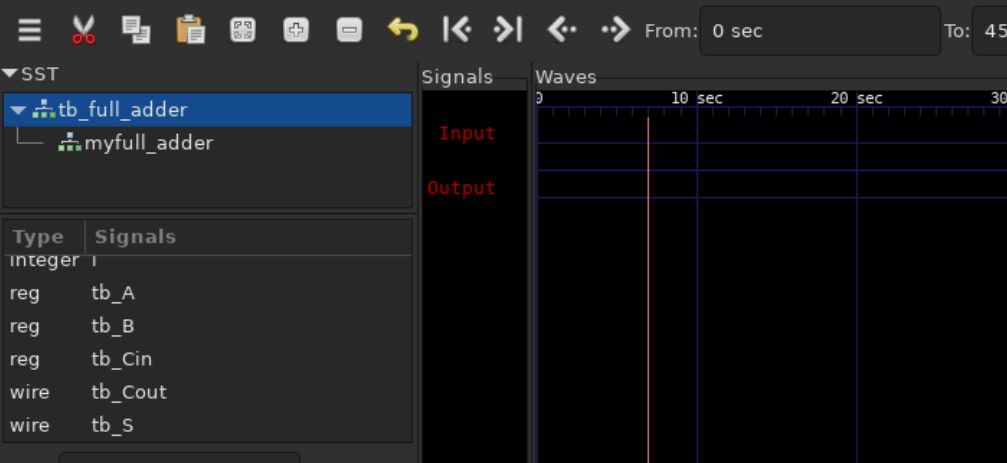
Description automatically generated

Let’s now carefully write the Verilog code for the SOP generating the outpus **S**. In the code listing below, the lines starting with // are comments. We need to declare some internal wires for Sum circuit, i.e., 3 wires for the input invertors and 4 other internal wires. We then use the 3 invertors instantiations, 4 and gates and 1 or gate instantiations to generate the complete listing of sum circuit. A screen shot of a computer program

Description automatically generated

Write down the code for the **Cout** circuit on your own. Remember, Verilog is case sensitive!

Once you are done, run the code by clicking the green **Run** button. We have named all the inputs and outputs to the adder are named with a suffix (tb\_ ) for distinction, though we could have used same names as the testbench is on a higher hierarchy of the modules. The wave form will open in a separate black window. Pick the inputs and output signals from the window on the bottom left and place them under Inputs or Outputs tags in the waveform on the right (drag and drop operation).



Once the wave form is arranged, you might have to Run the code again using the Run button. Analyse the waveform, does it match the expected outputs of a Full adder? Match it with the truth table we started with.

A green lines on a black background

Description automatically generated

**Task 2: Write Verilog code for a 4-bit adder**

Make a copy of your current Verilog project by opening the current repl and clicking fork. You update the name of the project to reflect that it is task2 of Verilog lab 1. Save the url of the new repl below.

Look at the example below in which we are adding two 4-bit numbers. We start adding from the least significant bit position on the right, here the **Cin**=0 and the 0th bits of both the numbers is 1. Using a full adder, we add these three bits and get (10)2 as the answer, i.e., **S**=0 and **Cout**=1. The **Cout** is forwarded to the next bit position (on the left) and becomes the **Cin** for that bit position. The next full adder at the 1st bit position has 0 and 1 as the two bits to be added, while the **Cin** is 1. Again, after addition using a Full adder a **S** and a **Cout** bit are generated. The **Cout** is forwarded to the bit position on the left as before. The **Cout** in binary addition propagates from a least significant bit position (at the right most) to the most significant bit position (at the left most). When we add 2 numbers each having N bits, we need N full adders, hence for adding the 4 bit numbers, we used 4 full adders. The final sum can be as large as (N+1) bits.

A picture containing table

Description automatically generated

We are now making a bigger adder that takes 2 numbers (4-bits each) and a carry in (**Cin**) to give a 1 bit carry out (**Cout**) and a 4 bits sum (**S**). We will use 4 instances of the full\_adder we used in task 1.

A diagram of a computer code

Description automatically generated

Make a new file for the 4 bit adder modules in the repl by clicking the new file button on the left pane of the repl. Name the file *four\_bit\_adder.v.*  Give the file the following interface. The interface or port signals for the *four\_bit\_adder.* is highlighted with a red boundary.

A screenshot of a computer

Description automatically generated

A diagram of a computer program

Description automatically generated

Make 4 instances of the **full\_adder** module and name them **full\_adder\_0**, **full\_adder\_1**, .., etc. Carefully (very carefully) connect the I/O ports of these full\_adder instantiations. The Carry in into the first full adder should be **Cin**, while its carry out (**Cout0**) should be the carry in of the next full adder and so on. The first two instantiations of the *full\_adder* module is given below, kindly continue for the next two instantiations.

A close-up of a number

Description automatically generated

Once the instantiations are complete, click Run. Remove any errors.

Lets now make the respective changes in the testbench module, the *tb\_full\_adder.v*. The DUT or the design under test is now the **four\_bit\_adder** module, whose interface signals are vectors (not bits).

A screenshot of a computer program

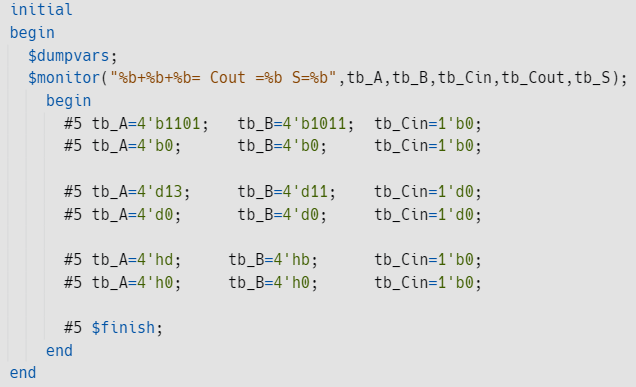
Description automatically generated

Change the instance name as well to four\_bit\_adder. Can you verify the following addition using the 4 bit full adder?

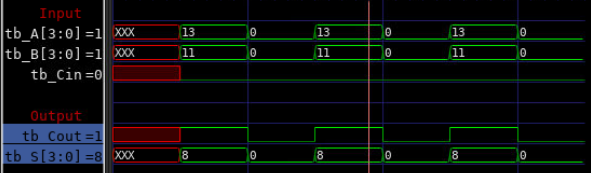
A picture containing table

Description automatically generated

The hint to change the testbench is as follows, where the same pair of numbers is specified as binary (tb\_a=4'b1101; tb\_b=4'b1011;), decimal (tb\_a=4'd13; tb\_b=4'd11;), and hexadecimal (tb\_a=4'hd; tb\_b=4'hb;). Make sure you are using the right tick ('). The tb\_cin is 0 for all the following cases.



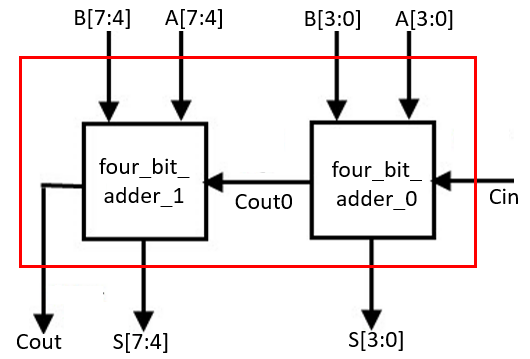
Verify the response using waveforms as shown below. Can you switch between hexadecimal and binary representation of signals in the waveform?



**Task 3: (DIY) Write Verilog code for an 8 bit adder**

Extend the *four\_bit\_adder* in task 2, into an *eight\_bit\_adder*. Start by making the copy (forking) of the repl in task 2. Save the URL below.

For an 8 bit adder, you will make a new module (in a new file called *eight\_bit\_adder.v*) and instantiate your 4 bit adder twice carefully according to following port binding.



You now will need 2 instances of the *four\_bit\_adder*. Make respective changes in testbench to add the last two digits of your ID to 10010. Run the code. Did you encounter any errors?

Verify that the correctness of the response of the adder by analysing the waveform.