Report: 3-Bit Adder Design and Verification

1. Introduction

The goal of this assignment was to design and verify a 3-bit binary adder using different representations commonly used in digital design. The adder takes two 3-bit numbers along with a carry-in input and produces a 3-bit sum and a carry-out output. To ensure correctness, the design was expressed and analyzed in multiple forms: a truth table, Verilog HDL, CNF equations, and benchmark format.

This exercise also demonstrated the workflow in Google Colab, where each step can be reproduced from the provided notebook and input files. By completing the assignment, I gained experience in moving between high-level functional specifications and lower-level representations that are used for synthesis and verification.

2. Truth Table

The first step was to generate the full truth table for the adder. The file adder_3-bit_tab.csv lists all 128 possible input combinations of the six operand bits (A2:A0 and B2:B0) and the carry-in (Cin). For each case, the outputs (S2:S0 and Cout) were computed according to binary addition rules.

For example:

A2	A1	A0	B2	B1	B0	Cin	S2	S1	S0	Cout
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	1	1	0	1	1	0
1	1	1	1	1	1	1	1	1	1	1

3. Verilog Implementation

Next, the adder was expressed in Verilog (adder_3-bit.v). The module adder_3_bit defines the inputs (A0, A1, A2, B0, B1, B2, Cin) and outputs (S0, S1, S2, Cout).

The generated Verilog is in structural netlist form, with intermediate wires corresponding to logic gates. Each output is defined as an OR of minterms that match the truth table. This ensures synthesizability and allows the design to be simulated with standard Verilog tools.

4. CNF Representation

For formal verification, the logic was also written in Conjunctive Normal Form (adder_3-bit.cnf). For example:

- The least significant sum bit, S0, is described as:
  $S0 = (A0 \lor B0 \lor \neg Cin) \land (\neg A0 \lor \neg B0 \lor \neg Cin) \land (A0 \lor \neg B0 \lor Cin) \land (\neg A0 \lor B0 \lor Cin)$

- The carry out from the least significant bit is expressed as:
  $Cout0 = (\neg A0 \lor \neg B0) \land (\neg A0 \lor Cin) \land (\neg B0 \lor Cin)$

These CNF clauses are useful for SAT solvers, enabling formal checks of correctness beyond simulation.

5. Bench Format

The ISCAS benchmark file (adder_3-bit.bench) provides another compact representation of the adder. It lists inputs and outputs, then specifies internal logic using NOT, AND, and OR operators. This format is widely used in logic synthesis and academic benchmarks, making the design easier to integrate into automated flows.

6. Reproducibility in Google Colab

The entire workflow was carried out in Google Colab using the notebook Veritas_tutorial_Adder_3_bit.ipynb. To reproduce the results:
1. Upload the notebook and input files (.csv, .v, .cnf, .bench).
2. Run all notebook cells sequentially.
3. Observe how each representation is generated and validated against the truth table.

This ensures that anyone can replicate the assignment results using the provided materials.

7. Conclusion

In this assignment, a 3-bit adder was implemented and verified across multiple design representations:
- The truth table captured functional behavior.
- The Verilog HDL ensured synthesizability and simulation.
- The CNF form supported formal reasoning.
- The bench format made the design compatible with synthesis benchmarks.

By moving between these formats, I gained a better understanding of the design verification flow, from specification to hardware description to formal proof. This exercise highlighted how different representations serve complementary roles in digital design.