Princess Sumaya University for Technology

King Abdullah II Faculty of Engineering

Electrical Engineering Department

DIGITAL ELECTRONICS LAB
2-BIT SUBTRACTOR
FINAL PROJECT

Authors:

Mayas Al-Hamdan	20220408	Computer Engineering
Saif Saeed	20220306	Computer Engineering

Supervisor:

Eng Eyad Al-Kouz

December 15, 2025

Abstract

This report presents the design and implementation of a 2-bit subtractor using Verilog HDL. Adopting a bottom-up structural methodology, the project progressed from verifying fundamental logic gates to constructing a 1-bit full subtractor, which was subsequently cascaded to form the final multi-bit arithmetic unit. The design was simulated using Icarus Verilog, with build automation facilitated by Makefile to streamline the verification process. Waveform analysis confirmed functional correctness across all test vectors, specifically validating accurate borrow propagation and underflow handling. The results demonstrate the efficacy of modular design in creating scalable digital logic circuits.

TABLE OF CONTENTS

1 INTRODUCTION

Digital circuits have become a consistent factor that we increasingly depend on in our lives, and what now is a simple circuit once was the pinnacle of glorious automated calculation. With proper scaling, the 2-bit subtractor - a combinational digital logic circuit capable of computing the arithmetic difference between two 2-bit binary numbers - can complete subtraction operations in microseconds. In its essence it accepts a minuend and a subtrahend, producing a 2-bit difference and a borrow-out bit.

1.1 OBJECTIVES

The objective of this project is to create this simple circuit in Verilog Hardware Description Language (HDL). This includes the various test benches and waveform analysis processes that have to take place in order to validate the design. For the purposes of this project, the IcarusVerilog compiler, the VVP runtime engine and the WaveTrace extension on Visual Studio Code are used.

1.2 THEORY

While its theoretical basis will be explained in stages below, there is no finite-state-machine (FSM) involved in this logic:

The least significant bit (LSB) is calculated by subtracting the least significant bits. Since there is no previous stage, the initial borrow-in is typically grounded, but the implemented design will accept a variable borrow-in value. It produces the LSB difference and an intermediate borrow.

The most significant bit (MSB) is calculated by subtracting the most significant bits and the borrow from the previous stage. It produces the MSB difference and the final borrow-out.

2 PROCEDURE AND METHODS

A bottom-up structural design methodology was employed, ensuring logic correctness at the component level before scaling to multi-bit arithmetic units.

1. Foundation (src/basic): Development began with the implementation and verification of the fundamental logic in basic.v.
2. 1-Bit Integration (src/one-bit-sub): The logic was encapsulated into a onebit.v module to function as a standard full subtractor unit.
3. Multi-Bit Scaling (src/two-bit-sub): Two instances of the 1-bit subtractor were cascaded to construct the twobit.v module, demonstrating modular reuse.

Each development stage underwent isolated verification to ensure signal integrity:

- Testbenches: Dedicated testbench files (tb_basic.v, tb_onebit.v, tb_twobit.v) were written for every module to inject test vectors.

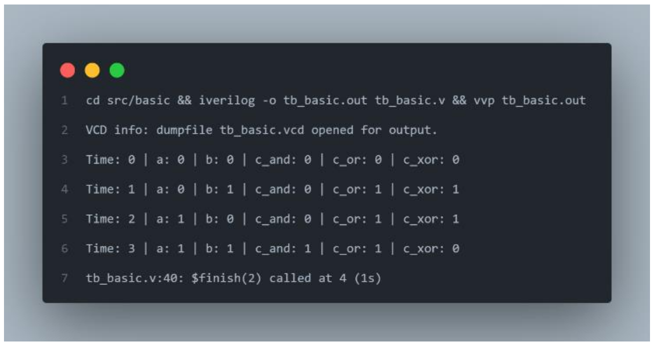
- Waveform Analysis: Logic states were validated via simulation, with results captured as waveforms (e.g., tb_onebit.png) to confirm adherence to truth tables prior to integration.

A Makefile was implemented to automate the compilation and simulation workflow using Icarus Verilog. Specific targets (basic, one, two) automatically navigate to source directories, compile the design using iverilog, and execute the simulation with the vvp runtime. A clean target utilizes recursive search commands to locate and remove generated binary (.out) and waveform (.vcd) artifacts, ensuring a clean target environment for subsequent builds.

3 Results and Discussions

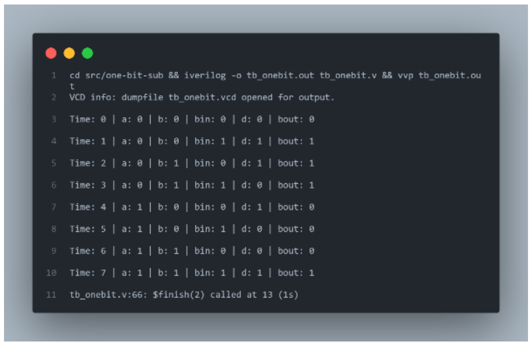
The simulation results confirm the functional correctness of the design at three hierarchical levels: basic logic gates, the 1-bit full subtractor, and the 2-bit cascaded subtractor.

3.1 Figures



```
cd src/basic && iverilog -o tb_basic.out tb_basic.v && vvp tb_basic.out
VCD info: dumpfile tb_basic.vcd opened for output.
Time: 0 | a: 0 | b: 0 | c_and: 0 | c_or: 0 | c_xor: 0
Time: 1 | a: 0 | b: 1 | c_and: 0 | c_or: 1 | c_xor: 1
Time: 2 | a: 1 | b: 0 | c_and: 0 | c_or: 1 | c_xor: 1
Time: 3 | a: 1 | b: 1 | c_and: 1 | c_or: 1 | c_xor: 0
tb_basic.v:40: $finish(2) called at 4 (1s)
```

Figure 1: Basic Gates Testbench Output.



```
cd src/one-bit-sub && iverilog -o tb_onebit.out tb_onebit.v && vvp tb_onebit.ou
t
VCD info: dumpfile tb_onebit.vcd opened for output.
Time: 0 | a: 0 | b: 0 | bin: 0 | d: 0 | bout: 0
Time: 1 | a: 0 | b: 0 | bin: 1 | d: 1 | bout: 1
Time: 2 | a: 0 | b: 1 | bin: 0 | d: 1 | bout: 1
Time: 3 | a: 0 | b: 1 | bin: 1 | d: 0 | bout: 1
Time: 4 | a: 1 | b: 0 | bin: 0 | d: 1 | bout: 0
Time: 5 | a: 1 | b: 0 | bin: 1 | d: 0 | bout: 0
Time: 6 | a: 1 | b: 1 | bin: 0 | d: 0 | bout: 0
Time: 7 | a: 1 | b: 1 | bin: 1 | d: 1 | bout: 1
tb_onebit.v:66: $finish(2) called at 13 (1s)
```

Figure 2: One-bit Subtractor Testbench Output.

The foundational logic gates (AND, OR, XOR) were tested against their standard truth tables. The simulation iterated through all 2-bit input combinations (4 cases in this instance). The c_xor output correctly asserted high only when inputs a and b differed (Time 1 and 2). The c_and output asserted high only when both inputs were high (Time 3), while c_or asserted high if at least one input was high. This shows validated results for all three required basic gates for this design.

The 1-bit subtractor was verified by driving inputs a (minuend), b (subtrahend), and bin (borrow-in) through all 8 possible states. The difference output d correctly reflects the XOR sum of the inputs. For example, at Time 1 (0 - 0 - 1), d is high. The bout signal correctly indicates underflow. All three major cases are elucidated in the following:

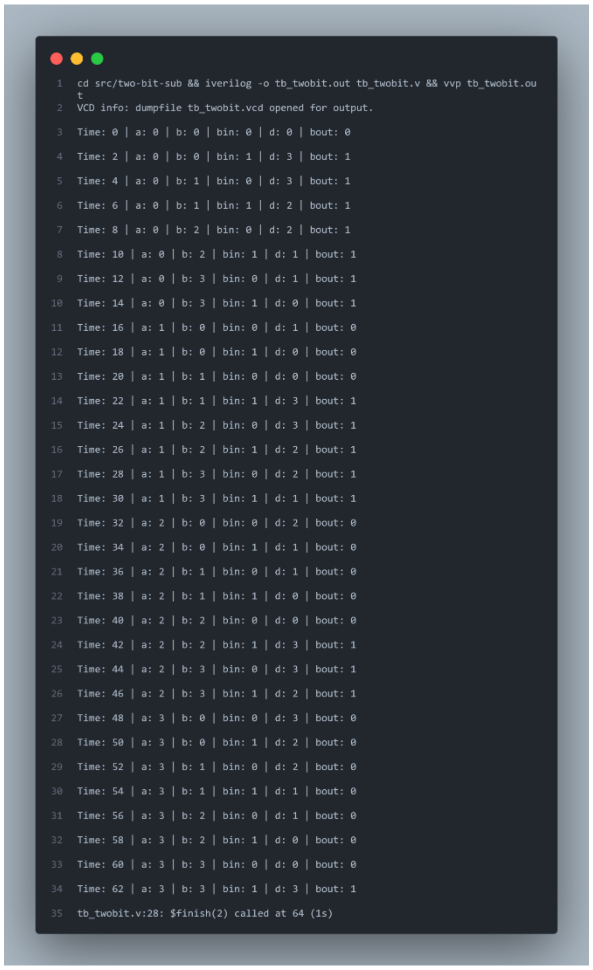
- Case: 0 - 1 (Time 2) resulted in d=1 and bout=1.
- Case: 1 - 1 (Time 6) resulted in d=0 and bout=0.
- Case: 1 - 1 - 1 (Time 7) asserted both outputs (d=1, bout=1)

The 2-bit subtractor, constructed by cascading two 1-bit modules, was tested with 2-bit vectors a and b alongside the initial borrow bin. The results validate correct arithmetic wrapping.

- At Time 2, the operation was 0 - 0 - 1. The result correctly wrapped to 3 with bout asserted, representing -1 in 2-bit logic.
- At Time 22, the operation 1 - 1 - 1 resulted in d=3 and bout=1.
- At Time 38, 2 - 1 - 1 resulted in d=0 with no borrow.
- At Time 52, 3 - 1 correctly yielded d=2.

The simulation confirms that the borrow-out from the LSB (Least Significant Bit) correctly propagates to the MSB (Most Significant Bit). For instance, at Time 19 (0 - 1 - 1), the LSB generated a borrow that reduced the MSB, resulting in d=2 (10) and bout=1.

The basic testbench waveform in Figure 4 verifies the fundamental building blocks (AND, OR, XOR) used in the later stages.



```
cd src/two-bit-sub && iverilog -o tb_twobit.out tb_twobit.v && vvp tb_twobit.ou
t
VCD info: dumpfile tb_twobit.vcd opened for output.
Time: 0 | a: 0 | b: 0 | bin: 0 | d: 0 | bout: 0
Time: 2 | a: 0 | b: 0 | bin: 1 | d: 3 | bout: 1
Time: 4 | a: 0 | b: 1 | bin: 0 | d: 3 | bout: 1
Time: 6 | a: 0 | b: 1 | bin: 1 | d: 2 | bout: 1
Time: 8 | a: 0 | b: 2 | bin: 0 | d: 2 | bout: 1
Time: 10 | a: 0 | b: 2 | bin: 1 | d: 1 | bout: 1
Time: 12 | a: 0 | b: 3 | bin: 0 | d: 1 | bout: 1
Time: 14 | a: 0 | b: 3 | bin: 1 | d: 0 | bout: 1
Time: 16 | a: 1 | b: 0 | bin: 0 | d: 1 | bout: 0
Time: 18 | a: 1 | b: 0 | bin: 1 | d: 0 | bout: 0
Time: 20 | a: 1 | b: 1 | bin: 0 | d: 0 | bout: 0
Time: 22 | a: 1 | b: 1 | bin: 1 | d: 3 | bout: 1
Time: 24 | a: 1 | b: 2 | bin: 0 | d: 3 | bout: 1
Time: 26 | a: 1 | b: 2 | bin: 1 | d: 2 | bout: 1
Time: 28 | a: 1 | b: 3 | bin: 0 | d: 2 | bout: 1
Time: 30 | a: 1 | b: 3 | bin: 1 | d: 1 | bout: 1
Time: 32 | a: 2 | b: 0 | bin: 0 | d: 2 | bout: 0
Time: 34 | a: 2 | b: 0 | bin: 1 | d: 1 | bout: 0
Time: 36 | a: 2 | b: 1 | bin: 0 | d: 1 | bout: 0
Time: 38 | a: 2 | b: 1 | bin: 1 | d: 0 | bout: 0
Time: 40 | a: 2 | b: 2 | bin: 0 | d: 0 | bout: 0
Time: 42 | a: 2 | b: 2 | bin: 1 | d: 3 | bout: 1
Time: 44 | a: 2 | b: 3 | bin: 0 | d: 3 | bout: 1
Time: 46 | a: 2 | b: 3 | bin: 1 | d: 2 | bout: 1
Time: 48 | a: 3 | b: 0 | bin: 0 | d: 3 | bout: 0
Time: 50 | a: 3 | b: 0 | bin: 1 | d: 2 | bout: 0
Time: 52 | a: 3 | b: 1 | bin: 0 | d: 2 | bout: 0
Time: 54 | a: 3 | b: 1 | bin: 1 | d: 1 | bout: 0
Time: 56 | a: 3 | b: 2 | bin: 0 | d: 1 | bout: 0
Time: 58 | a: 3 | b: 2 | bin: 1 | d: 0 | bout: 0
Time: 60 | a: 3 | b: 3 | bin: 0 | d: 0 | bout: 0
Time: 62 | a: 3 | b: 3 | bin: 1 | d: 3 | bout: 1
tb_twobit.v:28: $finish(2) called at 64 (1s)
```

Figure 3: Two-bit subtractor simulation output.

- The simulation ends with inputs a=1 and b=1.
- c_and is High, c_or is High, c_xor is Low.

The gates behave exactly according to standard boolean truth tables.

The waveforms in Figure 5 demonstrate the logic of a single-bit subtraction unit with borrow-in.

- The signals toggle through all 8 binary combinations of a, b, and bin.
- When a=0 and b=1, the difference d goes High and bout goes High.
- At the very end, with all inputs High, both d and bout remain High.

The final set of waveforms in Figure 6 validates the cascaded integration of two 1-bit subtractors.

- The input a (minuend) is held constant for long durations while b (subtrahend) cycles rapidly from 0 to 3.
- When a=0 and b increases (0, 1, 2, 3), the output d correctly wraps around in reverse order (0, 3, 2, 1).

- The bout signal is High whenever a < b. This confirms that the borrow bit is correctly propagating from the LSB to the MSB and out of the system.

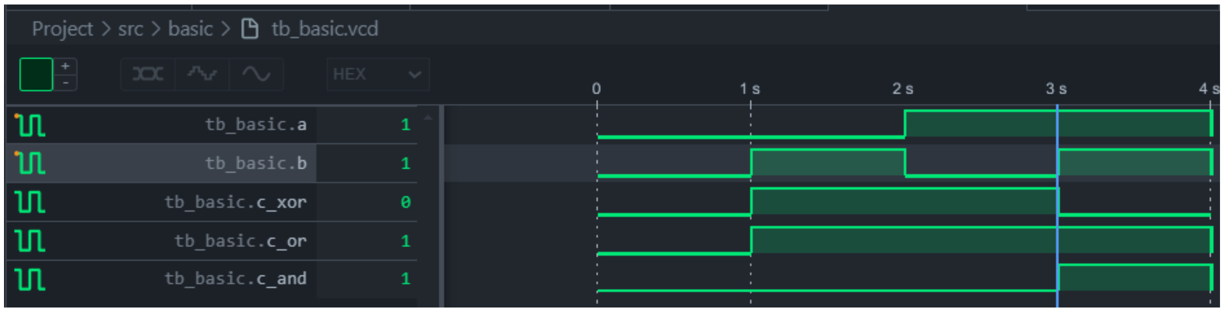


Figure 4: Basic testbench waveforms.

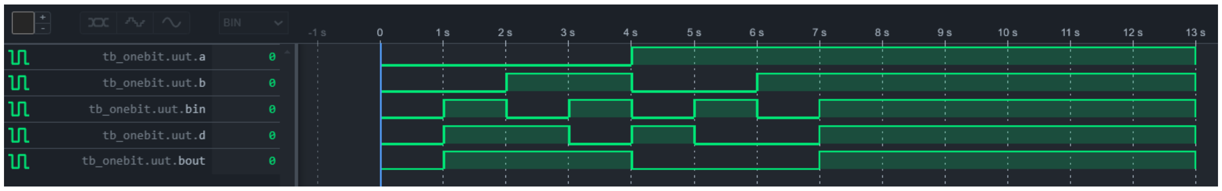


Figure 5: One-bit subtrctor testbench waveforms.

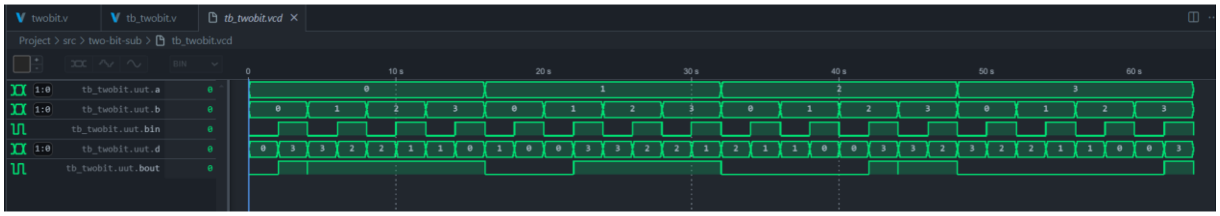


Figure 6: Two-bit subtractor testbench waveforms.

4 CONCLUSIONS

This project successfully validates the hierarchical implementation of a 2-bit subtractor using a bottom-up design methodology. By verifying fundamental logic gates before encapsulating them into 1-bit full subtractors, the design achieved modular scalability and robust fault isolation.

Simulation results via Icarus Verilog confirmed arithmetic correctness across all test vectors, including successful borrow propagation and underflow handling in the cascaded architecture. The integration of build automation through Make further streamlined the verification process, ensuring reproducible and efficient testing cycles.

5 REFERENCES

[1] S. Williams, "Icarus Verilog," *https://www.google.com/search?q=icarus.com*, 2025. [Online]. Available: http://iverilog.icarus.com. [Accessed: Dec. 15, 2025].

[2] M. M. Mano and M. D. Ciletti, *Digital Design: With an Introduction to the Verilog HDL*, 6th ed. Upper Saddle River, NJ, USA: Pearson, 2018.

[3] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2003.

[4] T. L. Floyd, *Digital Fundamentals*, 11th ed. Upper Saddle River, NJ, USA: Pearson, 2015.