Arithmetic and Counting (Adder & Counter)

Submitted to: Prof. Dr.-Ing. Peter Schulz
Lab Group: Group 3 Team 5
Name: Minhazul Islam,
Sajjadur Rahman,
Mir Md Redwon Sagor

*january*2025

Contents

1 Introduction

This lab session involves designing, building, and testing a digital circuit that controls room features like lighting and blinds in a smart home. Using both Small-Scale Integration (SSI) chips and Field Programmable Gate Arrays (FPGAs), students get practical experience in digital electronics. Key tasks include creating a truth table, formulating Boolean logic, and using "Digital" software for simulation and verification. The design will then be translated into VHDL code for FPGA use. The circuit has four modes: Standard, Lunchtime, Nighttime, and Emergency, each dictating different behaviors. The lab also offers experience in configuring hardware, such as building circuits using only NAND gates and performing FPGA synthesis. This report will cover the methods used, the results obtained, and the key takeaways from the session.

2 Preparatory Tasks

2.1 Test Vectors

Below are the test vectors used in Task 1.1 and 2.1:

```
-- Definition of test-vector set
constant TEST_DATA: stimulus_response_array_type := (
--     a,          b,          cin,  y,          cout
    ("00000000", "00000001", '0', "00000001", '0'),
    ("00000001", "00000001", '0', "00000010", '0'),
    ("00000000", "00000001", '1', "00000010", '0'),

-- TODO: Add test vectors here

    ("11111111", "00000000", '1', "00000000", '1'),
    ("00000001", "00000001", '0', "00000010", '0'), --R1.1 Basic Functionality
    ("00000010", "00000001", '0', "00000011", '0'), --R1.2 Basic Functionality
    ("01111111", "00000001", '0', "00000001", '0'), --R2.1 Carry Propagation
    ("11111111", "00000001", '0', "00000000", '1'), --R2.2 Carry Propagation
    ("11111111", "00000010", '0', "00000001", '1'), --R2.3 Carry Propagation
    ("00000001", "00000001", '1', "00000011", '0'), --R3.1 Carry In Effect
    ("11111111", "00000001", '1', "10000001", '1'), --R3.2 Carry In Effect
    ("11111111", "00000000", '1', "00000000", '1'), --R3.3 Carry In Effect
    ("11111111", "00000001", '0', "00000000", '1'), --R4.1 Overflow Handling
    ("10000000", "10000000", '0', "00000000", '0'), --R5.1 Edge Cases
    ("11111111", "11111111", '0', "11111110", '1'), --R5.2 Edge Cases
    ("10101010", "01010101", '0', "11111111", '0'), --R6.1 Alternating Bits
    ("10101010", "01010101", '1', "01010101", '1'), --R6.2 Alternating Bits
    ("10000000", "10000000", '0', "00000000", '1'), --R7.1 High Order Bits
    ("10000001", "01111111", '0', "11111110", '0'), --R7.2 High Order Bits
    ("11110000", "00001111", '1', "00000000", '1'), --R8.2 Complementary Inputs
    ("00001111", "11110000", '0', "11111111", '0'), --R8.1 Complementary Inputs
    ("00110011", "11001100", '0', "11111111", '0'), --R9.2 Random Cases
    ("01010101", "01100110", '0', "00000000", '1')--R9.1 Random Cases


);
```

Figure 1: Test Vectors for Task 1.1

```
-- Test-vector set definition
constant TEST_DATA: stimulus_response_array_type := (

    -- counting up
    ("00000000","00000001", '0', "00000001", '0'),   -- 0 + 1 = 1
    ("00000001","00000001", '0', "00000010", '0'),   -- 1 + 1 = 2
    ("00000010","00000001", '0', "00000011", '0'),   -- 3 + 1 = 3
    ("00000011","00000001", '0', "00000100", '0'),   -- 3 + 1 = 4
    ("00000100","00000001", '0', "00000101", '0'),   -- 4 + 1 = 5
    ("00000101", "00000001", '0', "00000110", '0'),  -- 5 + 1 = 6
    ("00000110", "00000001", '0', "00000111", '0'),  -- 6 + 1 = 7
    ("00000111", "00000001", '0', "00001000", '0'),  -- 7 + 1 = 8
    ("00001000", "00000001", '0', "00001001", '0'),  -- 8 + 1 = 9
    ("00001001", "00000001", '0', "00001010", '0'),  -- 9 + 1 = 10



    -- counting down
    ("00001010", "11111111", '0', "00001001", '1'),  -- 10 - 1 = 9, cout
    ("00001001", "11111111", '0', "00001000", '1'),  -- 9 - 1 = 8, cout
    ("00001000", "11111111", '0', "00000111", '1'),  -- 8 - 1 = 7, cout
    ("00000111", "11111111", '0', "00000110", '1'),  -- 7 - 1 = 6, cout
    ("00000110", "11111111", '0', "00000101", '1'),  -- 6 - 1 = 5, cout
    ("00000101", "11111111", '0', "00000100", '1'),  -- 5 - 1 = 4, cout
    ("00000100", "11111111", '0', "00000011", '1'),  -- 4 - 1 = 3, cout
    ("00000011", "11111111", '0', "00000010", '1'),  -- 3 - 1 = 2, cout
    ("00000010", "11111111", '0', "00000001", '1'),  -- 2 - 1 = 1, cout
    ("00000001", "11111111", '0', "00000000", '1')   -- 1 - 1 = 0, cout
);
```

Figure 2: Additional Vectors for Task 2.1

2.2 Truth Table

Req_ID	Test description	stimuli operand A	stimuli operand B	stimuli carry in	expected response Y	expected response carry out	Testbench Simulation (check)	FPGA circuit (check)
R1.1	Basic Functionality	11111111	00000000	1	00000000	1	ok	ok
R1.2	Basic Functionality	00000001	00000001	0	00000010	0	ok	ok
R1.3	Basic Functionality	00000010	00000001	0	00000011	0	ok	ok
R2.1	Carry Propagation	01111111	00000001	0	00000001	0	ok	ok
R2.2	Carry Propagation	11111111	00000001	0	00000000	1	ok	ok
R2.3	Carry Propagation	11111111	00000010	0	00000001	1	ok	ok
R3.1	Carry In Effect	00000001	00000001	1	00000011	0	ok	ok
R3.2	Carry In Effect	11111111	00000001	1	10000001	1	ok	ok
R3.3	Carry In Effect	11111111	00000000	1	00000000	1	ok	ok
R4.1	Overflow Handling	11111111	00000001	0	00000000	1	ok	ok
R5.1	Edge Cases	10000000	10000000	0	00000000	0	ok	ok
R5.2	Edge Cases	11111111	11111111	0	11111110	1	ok	ok
R6.1	Alternating Bits	10101010	01010101	0	11111111	0	ok	ok
R6.2	Alternating Bits	10101010	01010101	1	01010101	1	ok	ok
R7.1	High Order Bits	10000000	10000000	0	00000000	1	ok	ok
R7.2	High Order Bits	10000001	01111111	0	11111110	0	ok	ok
R8.1	Complementary Inputs	11110000	00001111	1	00000000	1	ok	ok
R8.2	Complementary Inputs	00001111	11110000	0	11111111	0	ok	ok
R9.1	Random Cases	00110011	11001100	0	11111111	0	ok	ok
R9.2	Random Cases	01010101	01100110	0	00000000	1	ok	ok

Figure 3: Truth Table

2.3 Time Variations

The performance of digital adders can vary significantly based on their design:

- **Combinational Adder:** Outputs are computed directly in response to changes in inputs. This type of adder provides immediate results as soon as input data is altered.
- **Registered Adder:** Outputs are refreshed at the next clock edge, resulting in a delay as outputs depend on clock timing. This method ensures that the output remains stable during the clock cycle.

2.4 EPA Wave

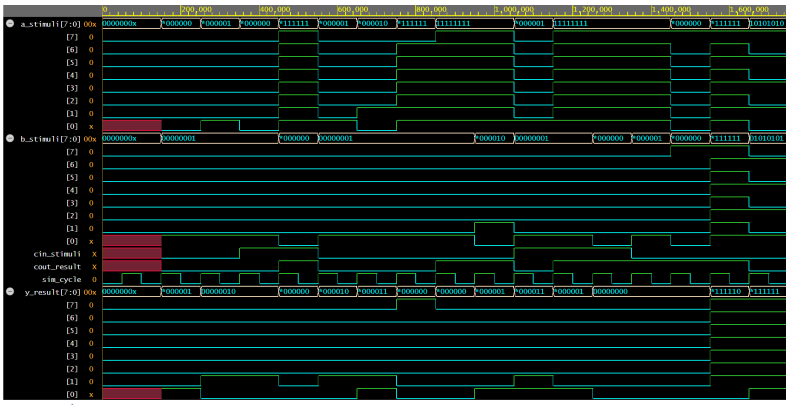


Figure 4: EP wave of Task 1.1

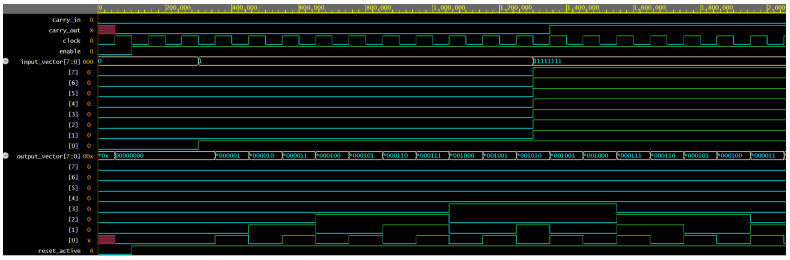
H 

Figure 7: EP wave of Task 2.2

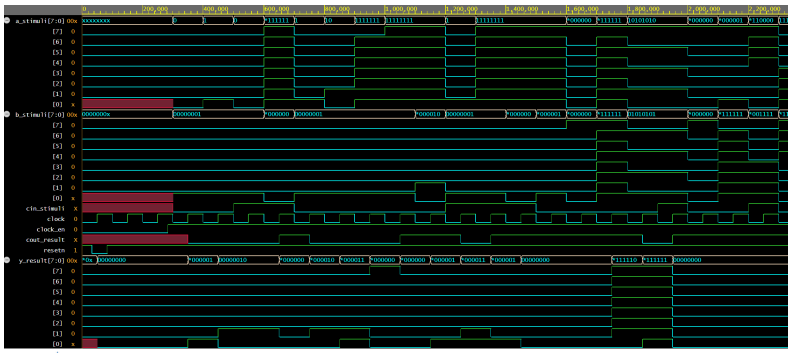


Figure 5: EP wave of Task 1.2

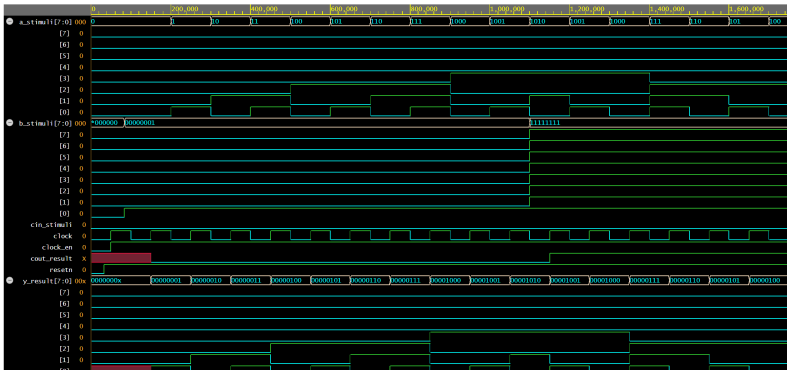


Figure 6: EP wave of Task 2.1

3 Laboratory Setup

3.1 Lab Task 1.1: Combinational Adder

The setup involved initializing a new RTL project in Vivado and integrating the VHDL and XDC files. The workflow was structured to efficiently manage design phases, beginning with the import of VHDL source files for design entry. Simulations were conducted using Vivado's "Run Simulation" feature to validate the functionality of the design. Synthesis processes were then executed to generate a synthesized design, confirming the simulation outcomes. This was followed by the implementation phase, where the design was programmed onto an FPGA. Lastly, a timing analysis was conducted to verify that the design met all specified timing constraints.

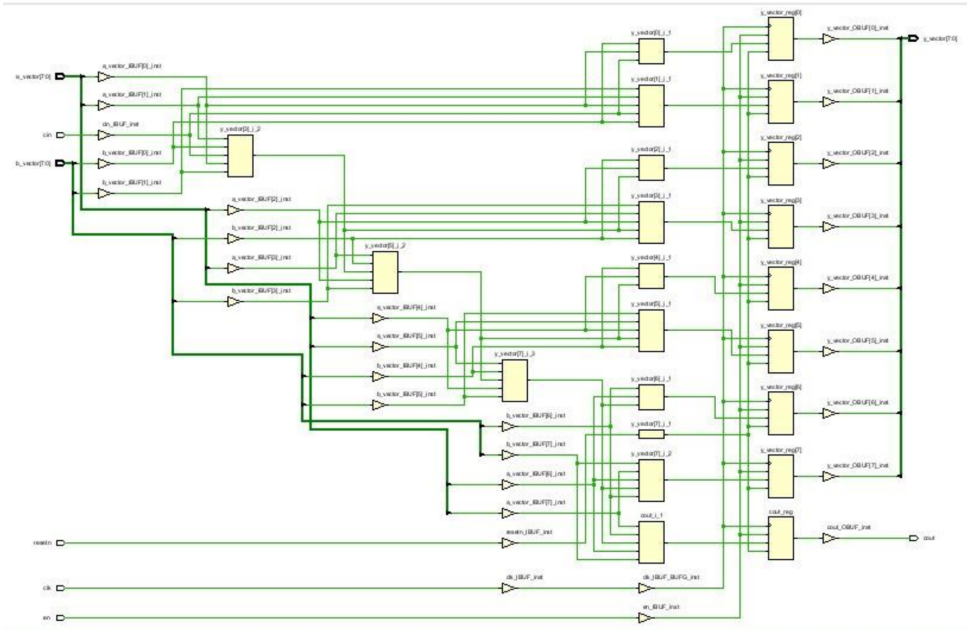


Figure 8: Schematic Diagram of the Combinational Adder

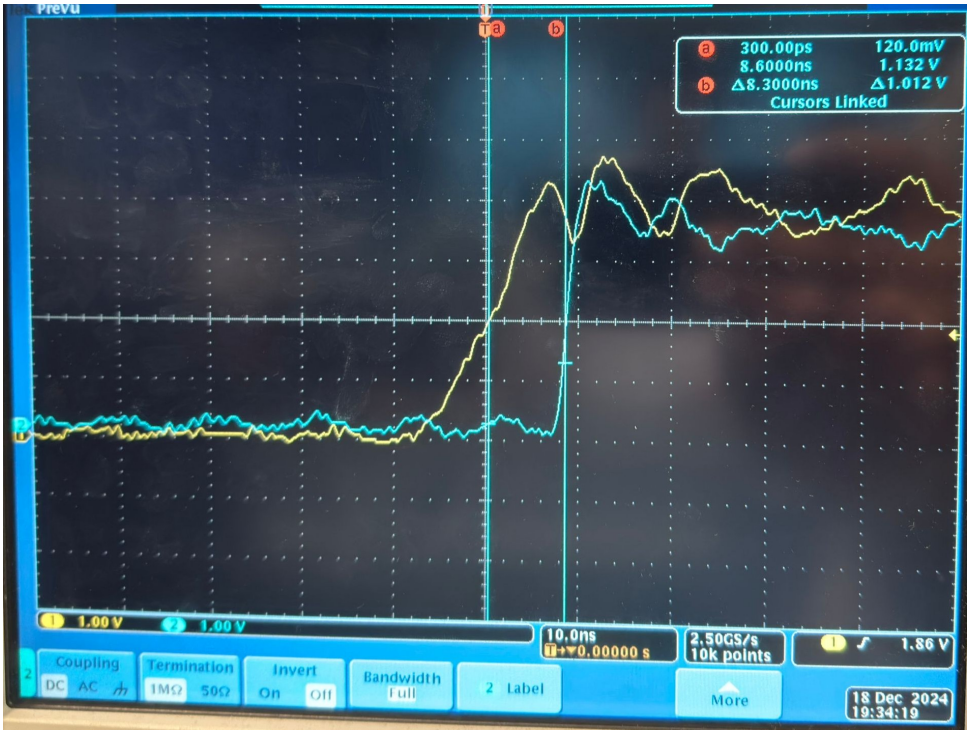


Figure 9: Propagation Delay for Task 1.1

Discussion:

- Observed outputs were consistent with expected results based on the truth table.
- A propagation delay of 8.3 ns was observed, which aligns with the design specifications and expected performance metrics.

3.2 Lab Task 1.2: Registered Adder

Setup Description:

- Added a result register to the combinational adder.
- Configured inputs for clock and reset signals.

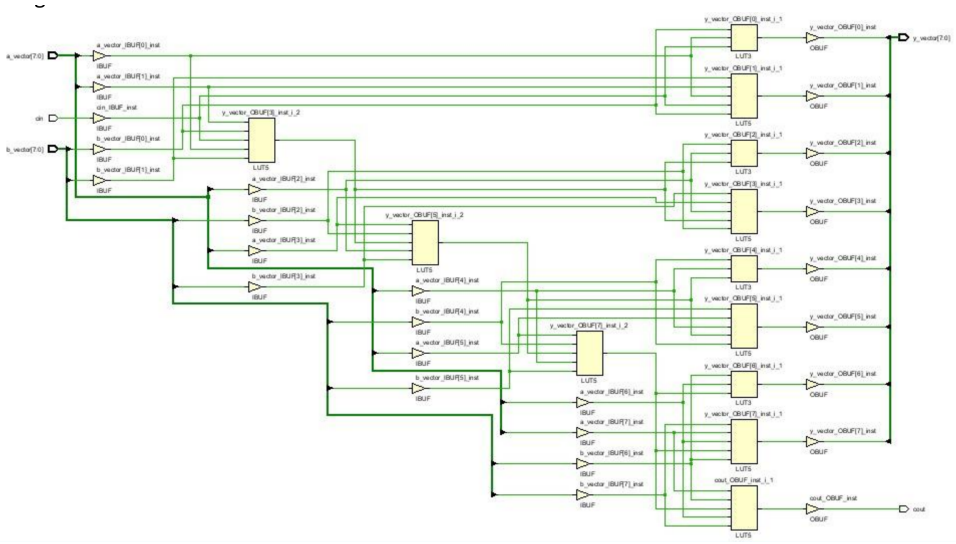


Figure 10: schematic 1.2

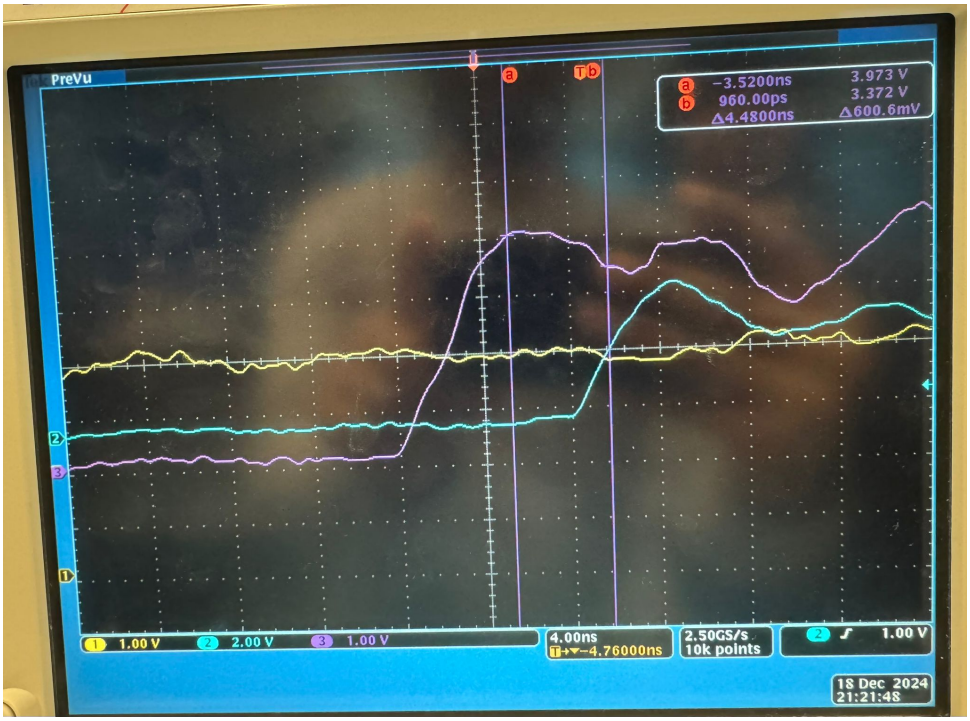


Figure 11: schematic 1.2

Discussion:

- Verified delayed output due to clock dependency.

3.3 Lab Task 1.3: Accumulator Setup

- We connected the "Y" output to the "a" input vector using cables, creating a feedback loop where the "Y" results are reused as the next "a" inputs. We tested this configuration with cases (3, 5, 9). For instance, if 3 is input, the output remains 3. Subsequently, if 5 is input, it is added to the previous output, yielding a result of 8 (5 + 3), and the process continues similarly. Accumulated values matched expected results.

4 Counter Implementation

4.1 Lab Task 2.1: Counter with External Feedback

Set up description:

- Configured inputs for up-counting and down-counting.
- Measured clock-to-output delay using an oscilloscope.

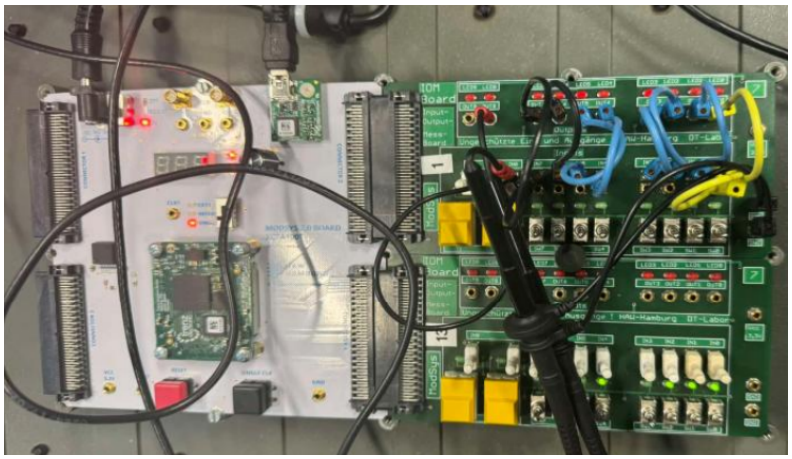


Figure 12: Circuit setup for task 2.1

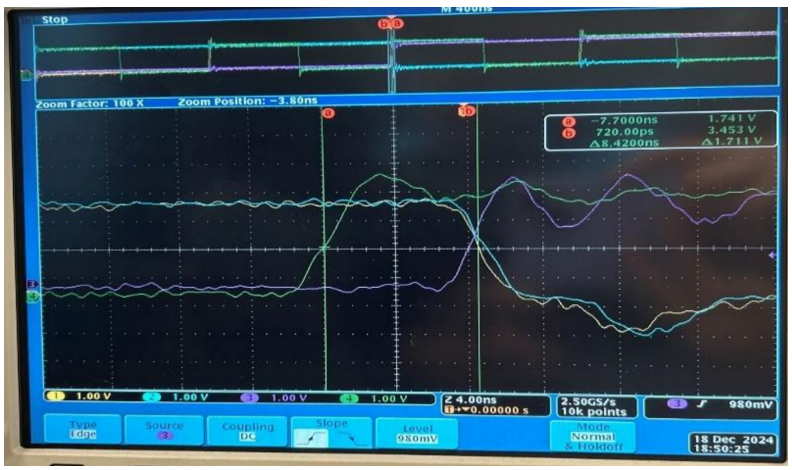


Figure 13: propagation delay

Figure 14: propagation delay

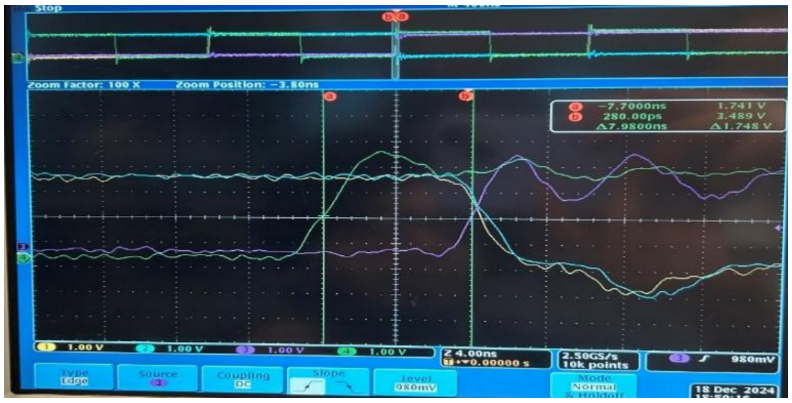


Figure 15: Propagation Delay

Discussion:

- Observed clock-dependent behavior with expected results for counting modes.
- Propagation Delay of 8.42 ns, 8.86ns, 7.98 ns was recorded

4.2 Lab Task 2.2: Counter with Internal Feedback

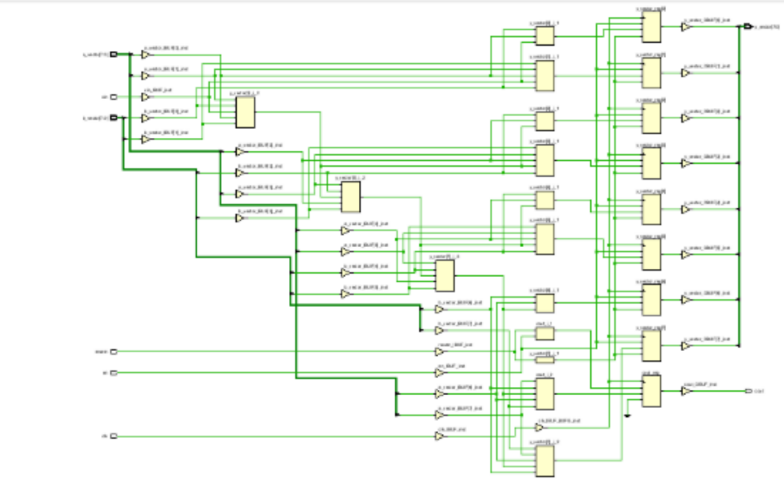


Figure 16: Schematic of Task 2.2

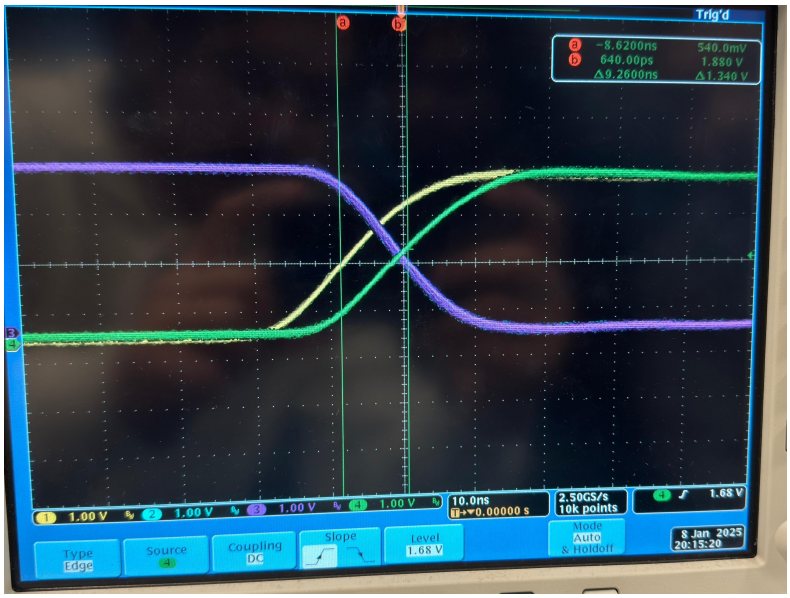


Figure 17: Propagation delay(for LSB) task 2.2

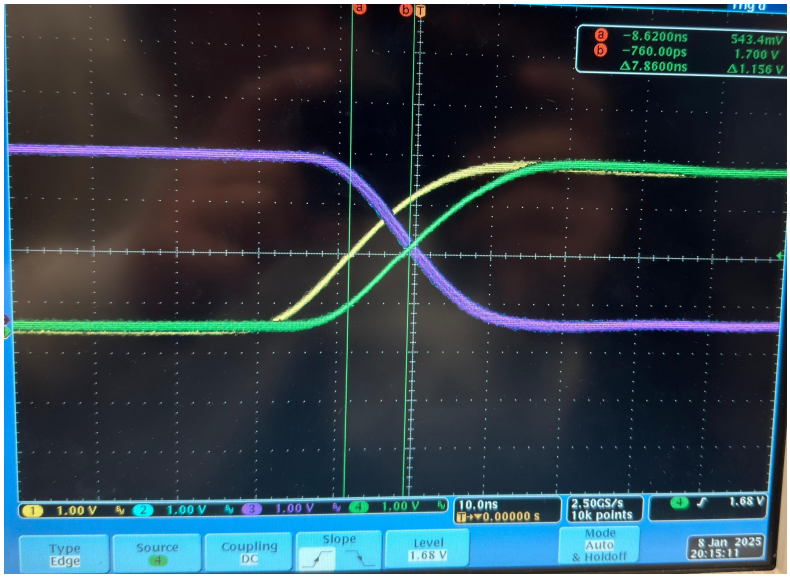


Figure 18: Propagation delay(for MSB) of Task 2.2

Discussion:

- Reduced propagation delay compared to external feedback
- Propagation Delay of 9.26 ns(LSB) and 7.86 ns(MSB)was observed.

5 Key Findings

5.1 Combinational Adder

- The adder responded instantaneously to changes in inputs, exhibiting a propagation delay of around 8.3 ns.
- Confirmed accurate functionality across all test scenarios, aligning with the expected results detailed in the truth table.

5.2 Registered Adder

- Exhibited dependency on the clock cycle, updating outputs solely at clock edges.
- Experienced longer propagation delays than the combinational adder, due to the need for clock synchronization.

5.3 Counter Implementation

- **Counter with External Feedback:**
  - Displayed counting actions reliant on clock timing with propagation delays noted as 7.98 ns, 8.42 ns, and 8.86 ns under various clock speeds and test conditions.
- **Counter with Internal Feedback:**

– Showed enhanced performance through reduced propagation delays, achieving 9.26 ns for the least significant bit and 7.86 ns for the most significant bit, thanks to internal feedback management within the FPGA.

6 References

- https://www.geeksforgeeks.org/counters-in-digital-logic/
- Lab Tasks- https://moodle.haw-hamburg.de/mod/folder/view.php?id=271332
- Moodle content- https://moodle.haw-hamburg.de/course/view.php?id=6984
- Lecture Slides from Moodle: https://moodle.haw-hamburg.de/pluginfile.php/336133/mod_resource/content/1/DT_W24_english_v_0.9.pdf

7 Appendix

7.1 Additional Task Resources

EDA Playground Link of the tasks

- Task 1.1: https://www.edaplayground.com/x/gJCV
- Task 1.2: https://www.edaplayground.com/x/Jh44
- Task 2.1: https://www.edaplayground.com/x/UxrP
- Task 2.2: https://www.edaplayground.com/x/7uUL