VLSI Design Course Project

Monsoon 2025

Design and Implementation of a 5-bit Carry Look-Ahead (CLA) Adder

Submitted by:

Monishram Selvaraj
Roll Number: 2024102076

Instructor:

Prof. Abhishek Srivastava
CVEST, IIIT Hyderabad

Release Date: 1-Nov-25 | Due Date: 3-Dec-25

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

In digital arithmetic circuits, adders are fundamental building blocks used extensively in ALUs, microprocessors, and DSP systems. The simplest form of multi-bit addition is the **Ripple Carry Adder (RCA)**, where the carry output of each full adder propagates to the next stage. However, this creates a significant delay as the carry must "ripple" through all bit positions, resulting in a worst-case delay proportional to the number of bits.

The **Carry Look-Ahead Adder (CLA)** overcomes this limitation by computing all carry signals in parallel using generate and propagate logic, significantly reducing the critical path delay.

1.2 Project Objectives

This project aims to design and implement a 5-bit Carry Look-Ahead Adder with the following specifications:

- Technology: 180nm CMOS process
- Supply Voltage: $V_{DD} = 1.8V$
- Feature size: $\lambda = 0.09\mu m$
- Equal NMOS and PMOS lengths: $L_n = L_p$
- Output load: Inverter with $W_p/W_n = 20\lambda/10\lambda$
- Tools: NGSPICE for simulation, MAGIC for layout, Verilog/Vivado for FPGA implementation

1.3 CLA Adder Theory

1.3.1 Propagate and Generate Signals

For two n-bit numbers $A = a_n a_{n-1}...a_2 a_1$ and $B = b_n b_{n-1}...b_2 b_1$, the **propagate** ($p_i$) and **generate** ($g_i$) signals for each bit position are defined as:

$$p_i = a_i \oplus b_i \quad \text{(Propagate)} \tag{1}$$

$$g_i = a_i \cdot b_i \quad \text{(Generate)} \tag{2}$$

where $i = 1, 2, 3, 4, 5$ for a 5-bit adder.

Physical Interpretation:

- **Generate (g_i):** A carry is *generated* at position i when both a_i and b_i are 1.
- **Propagate (p_i):** A carry is *propagated* through position i when exactly one of a_i or b_i is 1.

1.3.2 Carry Equations

The carry out of the i^{th} bit position can be expressed as:

$$c_{i+1} = g_i + p_i \cdot c_i \tag{3}$$

Expanding this recursively for a 5-bit CLA (assuming $c_0 = 0$ or $c_0 = c_{in}$):

$$c_1 = g_1 + p_1 \cdot c_0 \tag{4}$$
$$c_2 = g_2 + p_2 \cdot g_1 + p_2 \cdot p_1 \cdot c_0 \tag{5}$$
$$c_3 = g_3 + p_3 \cdot g_2 + p_3 \cdot p_2 \cdot g_1 + p_3 \cdot p_2 \cdot p_1 \cdot c_0 \tag{6}$$
$$c_4 = g_4 + p_4 \cdot g_3 + p_4 \cdot p_3 \cdot g_2 + p_4 \cdot p_3 \cdot p_2 \cdot g_1 + p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot c_0 \tag{7}$$
$$c_5 = g_5 + p_5 \cdot g_4 + p_5 \cdot p_4 \cdot g_3 + p_5 \cdot p_4 \cdot p_3 \cdot g_2 + p_5 \cdot p_4 \cdot p_3 \cdot p_2 \cdot g_1 + p_5 \cdot p_4 \cdot p_3 \cdot p_2 \cdot p_1 \cdot c_0 \tag{8}$$

1.3.3 Sum Equations

The sum at each bit position is computed as:

$$sum_i = p_i \oplus c_{i-1} \tag{9}$$

For the 5-bit adder:

$$sum_1 = p_1 \oplus c_0 \tag{10}$$
$$sum_2 = p_2 \oplus c_1 \tag{11}$$
$$sum_3 = p_3 \oplus c_2 \tag{12}$$
$$sum_4 = p_4 \oplus c_3 \tag{13}$$
$$sum_5 = p_5 \oplus c_4 \tag{14}$$
$$C_{out} = c_5 \tag{15}$$

2 Structure of 5-bit CLA Adder

2.1 Top-Level Architecture

The complete 5-bit CLA adder system consists of:

1. **Input D Flip-Flops:** Two 5-bit registers for inputs A and B
2. **5-bit CLA Adder Core:** Combinational logic for addition
3. **Output D Flip-Flops:** 5-bit register for sum and 1-bit for carry out

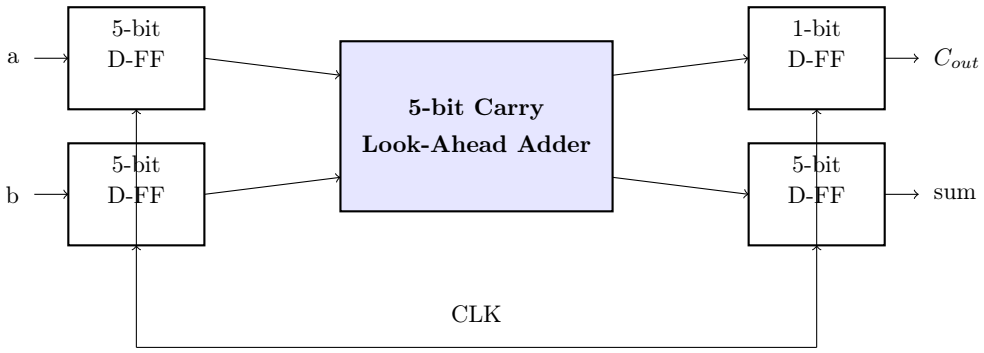


Figure 1: Block diagram of 5-bit CLA adder system

2.2 Internal Modules of CLA Adder

The CLA adder core is divided into three functional blocks:

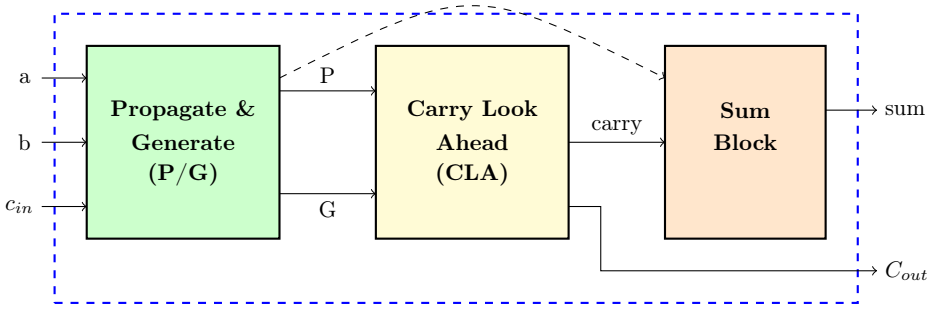


Figure 2: Internal modules of 5-bit CLA adder

2.2.1 Block 1: Propagate and Generate (P/G) Block

Function: Computes propagate and generate signals for all 5 bits in parallel.
Inputs: $a[5:1]$, $b[5:1]$
Outputs: $p[5:1]$, $g[5:1]$
Logic:

- Each p_i requires one XOR gate: $p_i = a_i \oplus b_i$
- Each g_i requires one AND gate: $g_i = a_i \cdot b_i$

Gate Count: 5 XOR gates + 5 AND gates = 10 gates

2.2.2 Block 2: Carry Look-Ahead (CLA) Block

Function: Computes all carry signals in parallel using P and G signals.
Inputs: $p[5:1]$, $g[5:1]$, c_0
Outputs: $c[5:1]$ (carry signals)

2.2.3 Block 3: Sum Block

Function: Computes final sum bits using propagate signals and carry signals.
Inputs: $p[5:1]$, $c[4:0]$
Outputs: $sum[5:1]$
Logic: Each sum_i requires one XOR gate: $sum_i = p_i \oplus c_{i-1}$
Gate Count: 5 XOR gates

2.3 Timing Requirements

As shown in the problem statement:

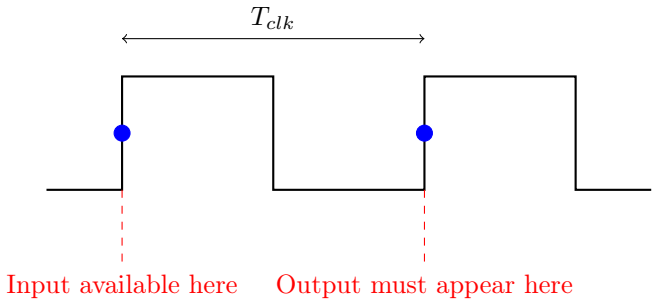


Figure 3: Timing requirement: Output must be ready before next rising edge

Requirement: Input bits are available before the rising edge of the clock, and the output must be computed and present at the next rising edge.

Timing Constraint:

$$T_{clk} \geq t_{cq} + t_{CLA} + t_{setup} \tag{16}$$

where:

- t_{cq}: Clock-to-Q delay of input flip-flops
- t_{CLA}: Propagation delay through CLA adder
- t_{setup}: Setup time of output flip-flops

3 Design Details: Topology and Sizing

3.1 Design Parameters

Table 1: Design Parameters

Parameter	**Value**
Technology	180nm
Feature Size (λ)	0.09 μm
Supply Voltage (V_{DD})	1.8 V
Minimum Length (L_{min})	$2\lambda = 0.18$ μm
NMOS Width (minimum)	$4\lambda = 0.36$ μm
PMOS Width (minimum)	$8\lambda = 0.72$ μm
Output Load Inverter	$W_p/W_n = 20\lambda/10\lambda$

3.2 Inverter Design

The basic inverter serves as the reference for sizing all other gates.

Sizing for symmetric switching:

$$W_n = 4\lambda = 0.36\ \mu m \tag{17}$$
$$W_p = 2 \times W_n = 8\lambda = 0.72\ \mu m \tag{18}$$
$$L_n = L_p = 2\lambda = 0.18\ \mu m \tag{19}$$

3.3 XOR Gate Design

The XOR gate is critical for both P/G generation and sum computation. We implement it using transmission gate logic for optimal performance.

XOR using Transmission Gates:

$$A \oplus B = A\bar{B} + \bar{A}B \tag{20}$$

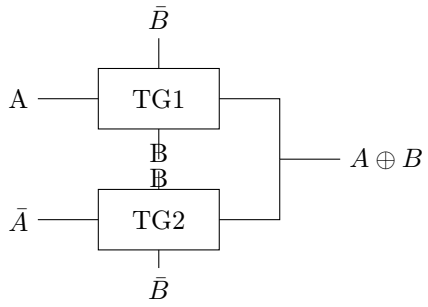


Figure 4: XOR gate using transmission gates

3.4 AND Gate Design

Implemented as NAND followed by inverter:

$$A \cdot B = \overline{\overline{A \cdot B}} \tag{21}$$

Sizing:

- NAND stage: As per Section 3.3
- Inverter stage: Sized for load driving capability

3.5 D Flip-Flop Design

The D flip-flop is implemented using the master-slave configuration with transmission gates.

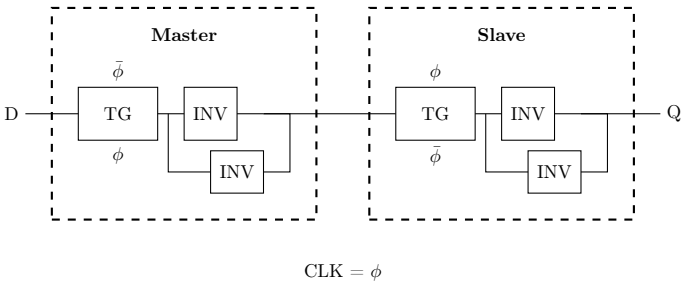


Figure 5: Master-slave D flip-flop using transmission gates

3.6 MAGIC Layout Implementation

3.6.1 Individual Gate Layouts

Inverter Layout

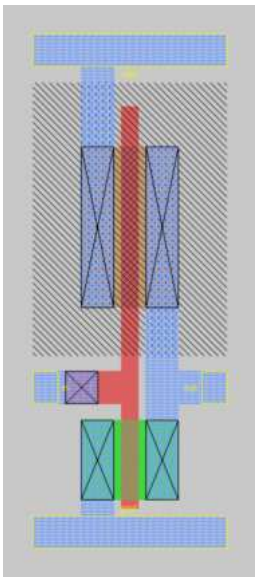


Figure 6: MAGIC layout of CMOS Inverter

2-Input NAND Gate Layout

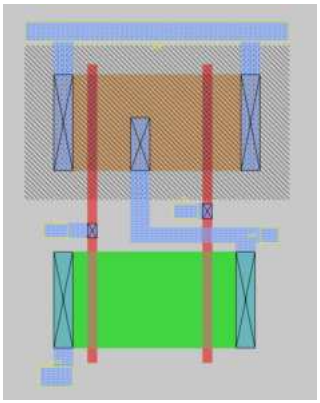


Figure 7: MAGIC layout of 2-input NAND gate

2-Input NOR Gate Layout

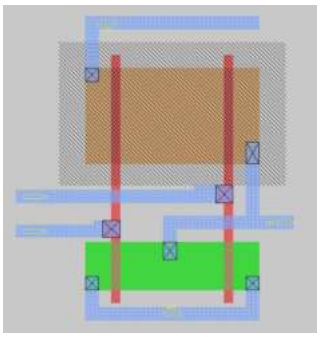


Figure 8: MAGIC layout of 2-input NOR gate

2-Input XOR Gate Layout

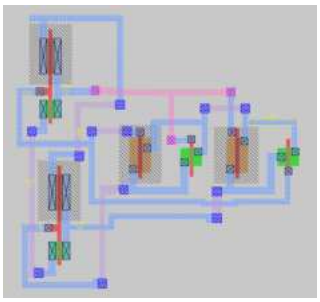


Figure 9: MAGIC layout of 2-input XOR gate using transmission gates

3.6.2 TSPC D Flip-Flop Layout

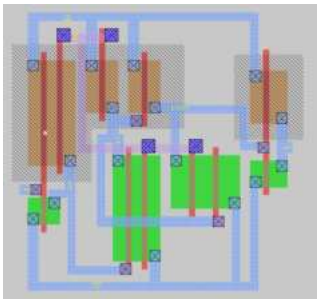


Figure 10: MAGIC layout of TSPC D Flip-Flop

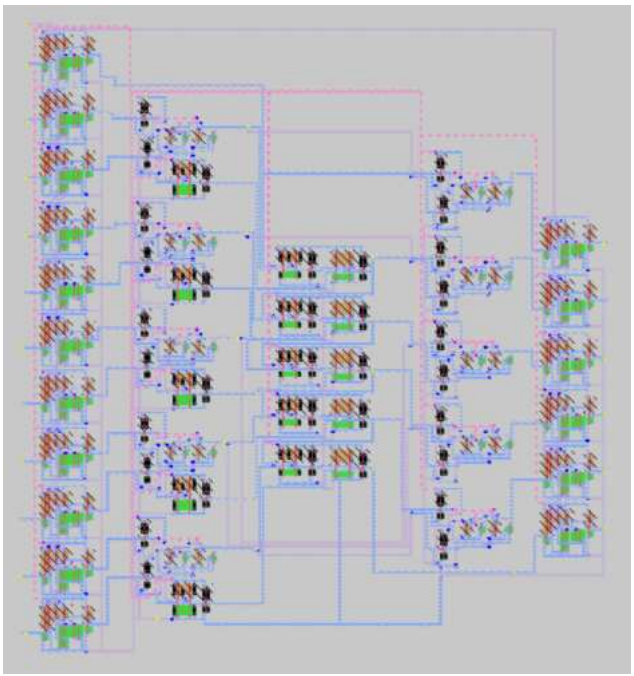


Figure 11: MAGIC layout of complete 5-bit CLA Adder with registers

Parameter	Value
Horizontal Pitch	103.77
Vertical Pitch	110.97

Table 2: Pitch Analysis for MAGIC Layout

4 D Flip-Flop: Structure and Timing Constraints

The positive-edge triggered D flip-flop is a fundamental sequential element that captures input data on the rising edge of the clock and holds it until the next rising edge.

4.0.1 TSPC (True Single-Phase Clock) Architecture

The TSPC D flip-flop consists of three cascaded stages using only a single clock phase:

1. **Stage 1 (Input Sampling):** Samples the input D when CLK = 0, holds when CLK = 1
2. **Stage 2 (Intermediate Hold):** Inverts and evaluates when CLK = 1, holds when CLK = 0
3. **Stage 3 (Output Driver):** Produces final output Q, precharged when CLK = 0, evaluates when CLK = 1

This configuration ensures that the output changes only on the rising edge of the clock, while requiring only the clock signal (no complementary clock needed).

4.1 Timing Parameters

4.1.1 Setup Time (t_{setup})

Setup Time is the minimum time before the active clock edge during which the input data must be stable for the flip-flop to reliably capture the correct value.

Mathematical Expression:

$$t_{setup} = t_{data\_stable} - t_{clock\_edge} \tag{22}$$

Physical Interpretation for TSPC:

- The data must propagate through Stage 1 and stabilize at node X before the rising clock edge
- When CLK transitions from 0→1, Stage 1 gets isolated, so D must be stable beforehand
- Violating setup time causes metastability or incorrect data capture at node X

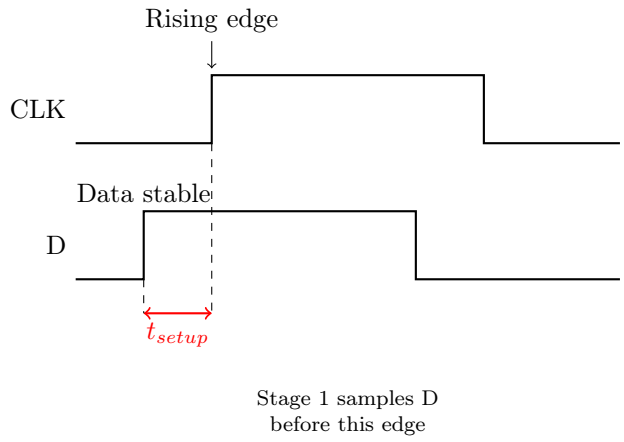


Figure 12: Setup time illustration for TSPC D Flip-Flop

4.1.2 Hold Time (t_{hold})

Hold Time is the minimum time after the active clock edge during which the input data must remain stable for the flip-flop to reliably capture the correct value.

Mathematical Expression:

$$t_{hold} = t_{data\_change} - t_{clock\_edge} \tag{23}$$

Physical Interpretation for TSPC:

- After the rising edge, Stage 1 must complete its isolation from input D
- The P1 transistor needs time to fully turn OFF and disconnect the input path
- Data changing too soon after clock edge can corrupt the captured value at node X

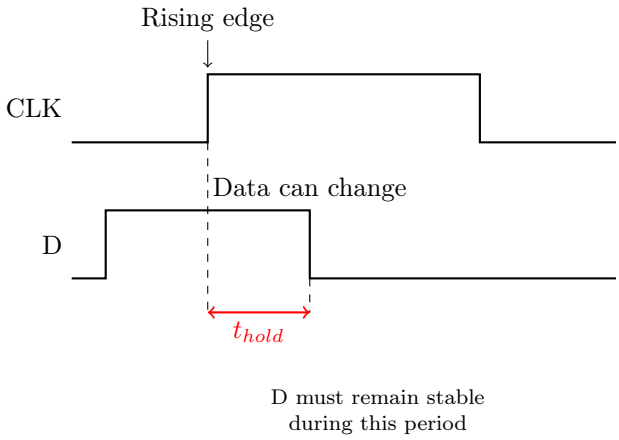


Figure 13: Hold time illustration for TSPC D Flip-Flop

4.1.3 Clock-to-Q Delay (t_{cq})

Clock-to-Q Delay is the time from the active clock edge to when the output Q becomes valid and stable.

Mathematical Expression:

$$t_{cq} = t_{Q\_valid} - t_{clock\_edge} \tag{24}$$

Physical Interpretation for TSPC:

- On rising edge, Stages 2 and 3 must evaluate sequentially
- Stage 2 inverts the value at X to produce Y
- Stage 3 then inverts Y to produce final output Q
- Total delay includes propagation through both stages

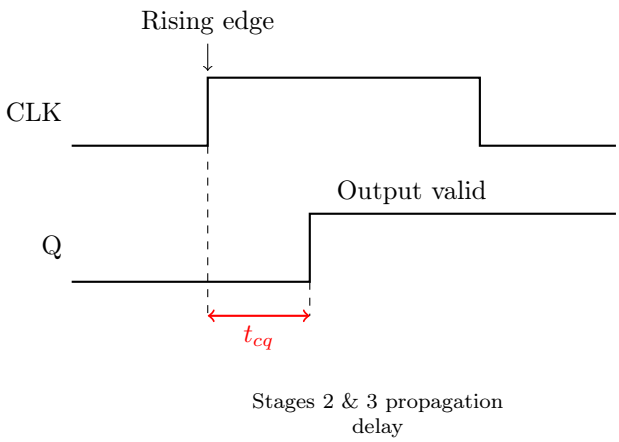


Figure 14: Clock-to-Q delay illustration for TSPC D Flip-Flop

4.1.4 Minimum Clock Period Calculation

For the flip-flop to operate correctly, the clock period must satisfy:

$$T_{CLK,min} = t_{cq} + t_{logic,max} + t_{setup} \tag{25}$$

where $t_{logic,max}$ is the maximum combinational logic delay between flip-flops.
For the CLA adder system:

$$T_{CLK,min} = t_{cq,DFF} + t_{CLA,max} + t_{setup,DFF} \tag{26}$$

Maximum Operating Frequency:

$$f_{max} = \frac{1}{T_{CLK,min}} = \frac{1}{t_{cq} + t_{CLA,max} + t_{setup}} \tag{27}$$

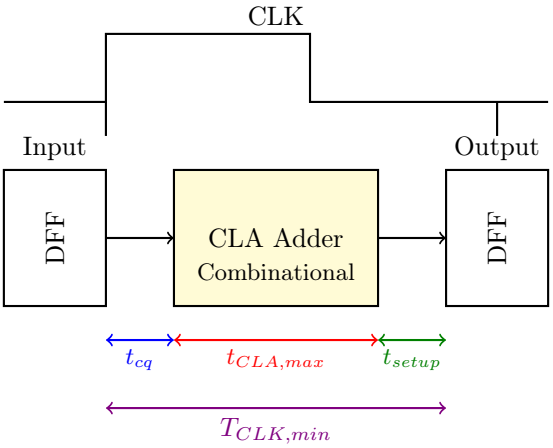


Figure 15: Timing path analysis for pipelined CLA adder system

4.2 Why Timing Constraints Matter

4.2.1 Setup Time Violation

When setup time is violated:

1. Data hasn't stabilized in Stage 1 before the clock rising edge
2. Node X may capture incorrect or metastable value
3. Stage 2 and 3 will propagate this incorrect value to output Q
4. **Solution:** Reduce clock frequency or optimize combinational logic delay

4.2.2 Hold Time Violation

When hold time is violated:

1. Data changes before Stage 1 fully isolates from input
2. P1 transistor hasn't completely turned OFF yet
3. New data may corrupt the intended captured value at node X
4. **Solution:** Add buffers/delays in the data path or use cells with negative hold time

4.3 NGSPICE Simulation Results

Note: Replace the placeholder images and values below with your actual NGSPICE simulation results.

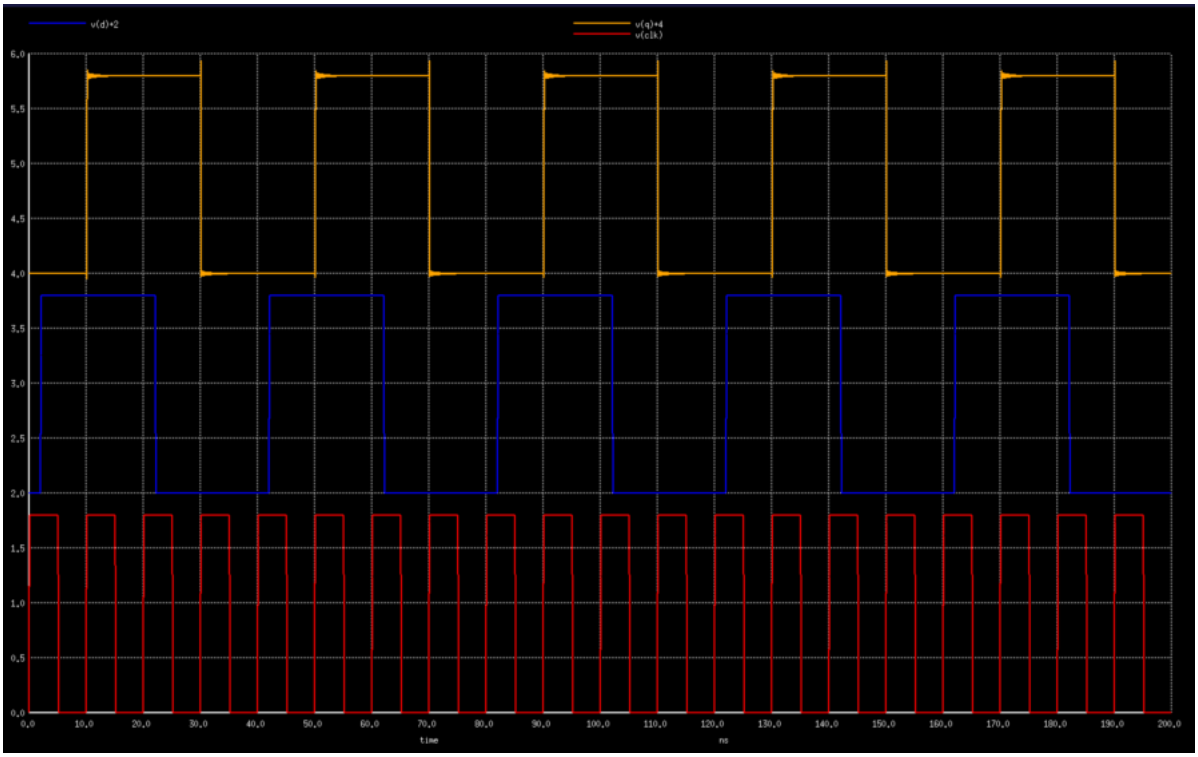


Figure 16: NGSPICE simulation waveforms for TSPC D Flip-Flop timing analysis

Table 3: TSPC D Flip-Flop Timing Parameters from NGSPICE Simulation

Parameter	**Symbol**	**Measured Value**	**Unit**
Setup Time	t_{setup}	62	ps
Hold Time	t_{hold}	54.5	ps
Clock-to-Q Delay (avg)	$t_{cq,avg}$	80.1	ps

```
  Measurements for Transient Analysis

t_d_rise_x_fall     =  6.635575e-11 targ=  2.091356e-09 trig=  2.025000e-09
t_d_fall_x_rise     =  5.813049e-11 targ=  2.213313e-08 trig=  2.207500e-08
```

Figure 17: Setup time measurement from NGSPICE

```
  Measurements for Transient Analysis

t_clk_rise_y_fall   =  3.430541e-11 targ=  5.930541e-11 trig=  2.500000e-11
t_clk_fall_y_rise   =  7.586534e-11 targ=  5.150865e-09 trig=  5.075000e-09
```

Figure 18: Hold time measurement from NGSPICE

```
  Measurements for Transient Analysis

t_clk_rise_q_fall   =  8.010024e-11 targ=  1.051002e-10 trig=  2.500000e-11
```

Figure 19: Clock-to-Q delay measurement from NGSPICE

4.4 Timing Constraints for CLA Adder System

For proper operation of the pipelined CLA adder with TSPC flip-flops:

Setup Time Constraint:

$$T_{CLK} \geq t_{cq,avg} + t_{CLA,max} + t_{setup} \tag{28}$$

Hold Time Constraint:

$$t_{cq,min} + t_{CLA,min} \geq t_{hold} \tag{29}$$

Maximum Clock Frequency:

$$f_{max,system} = \frac{1}{t_{cq,avg} + t_{CLA,max} + t_{setup}} \tag{30}$$

Calculation:

- $t_{cq,avg} = 80.1$ ps
- $t_{CLA,max} = 566$ ps
- $t_{setup} = 62$ ps

Then:

$$T_{CLK,min} = 80.1 + 566 + 62 = \ 708\text{ps} = 0.708 \text{ ns} \tag{31}$$

$$f_{max,system} = \frac{1}{0.708 \text{ ns}} \approx 1.412 \text{ GHz} \tag{32}$$

Table 4: Complete Timing Budget for CLA Adder System

Timing Component	**Measured Value**
Input DFF t_{cq}	80.1 ps
CLA Adder delay	566 ps
Output DFF t_{setup}	62 ps
Minimum Clock Period	**0.708 ns**
Maximum Frequency	**1.4 GHz**

4.5 Pre-Layout vs Post-Layout Simulation Comparison

Individual Gate Delays

Parameter	Post-layout	Pre-layout
Input DFF t_{cq}	80.1 ns	76.2 ns
Propagation Delay	566 ns	460 ns
Clock period	0.708 ns	0.574
Frequency	1.4 Ghz	1.714 Ghz

Table 5: Pre-Layout vs Post-Layout Delay Comparison

5 Stick Diagrams

Stick diagrams provide a simplified representation of the transistor layout, showing the relative positions of transistors, polysilicon gates, diffusion regions, and metal interconnects.

5.1 Stick Diagram Notation

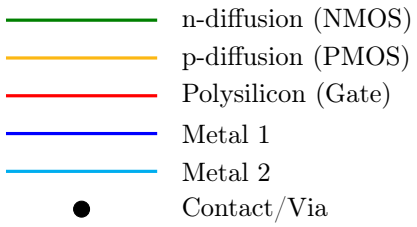


Figure 20: Stick diagram color convention

5.2 Inverter Stick Diagram

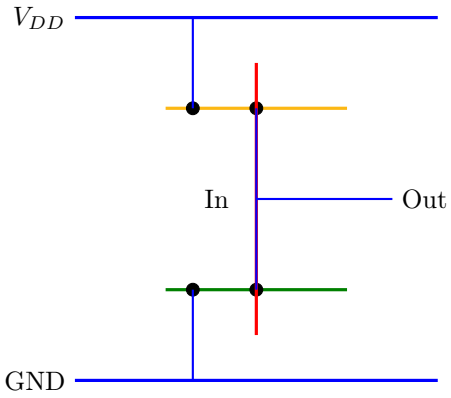


Figure 21: Stick diagram of CMOS inverter

5.3 2-Input NAND Gate Stick Diagram

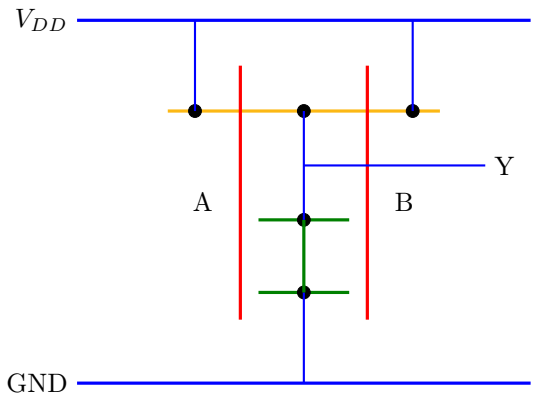
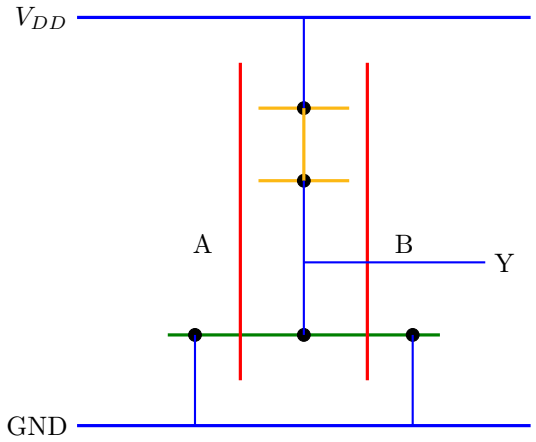


Figure 22: Stick diagram of 2-input NAND gate

5.4 2-Input NOR Gate Stick Diagram



5.5 2-Input XOR Gate Stick Diagram

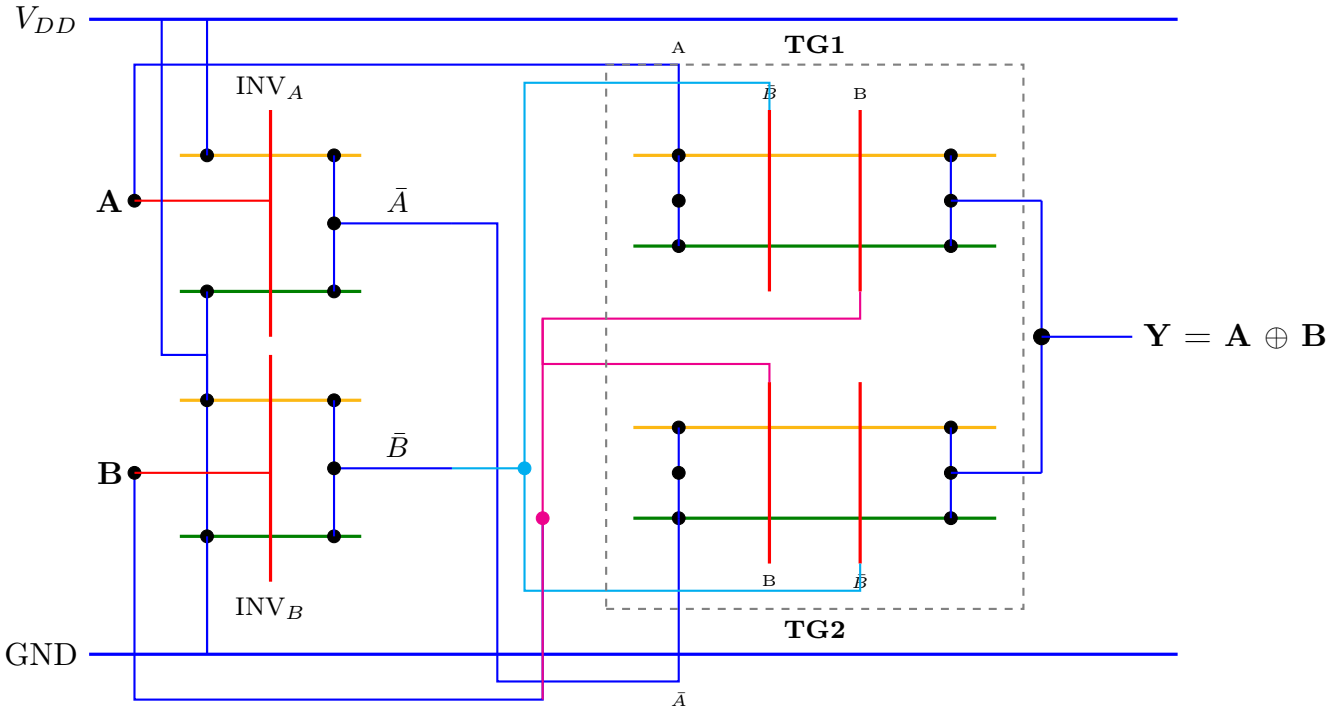


Figure 23: Stick diagram of 2-input XOR gate using transmission gates

5.6 TSPC D Flip-Flop Stick Diagram

The positive-edge triggered TSPC D flip-flop consists of three stages:

- **Stage 1:** Input sampling stage (transparent when CLK = 0)
- **Stage 2:** Intermediate hold stage
- **Stage 3:** Output driving stage (evaluates when CLK = 1)

5.6.1 Stick Diagram

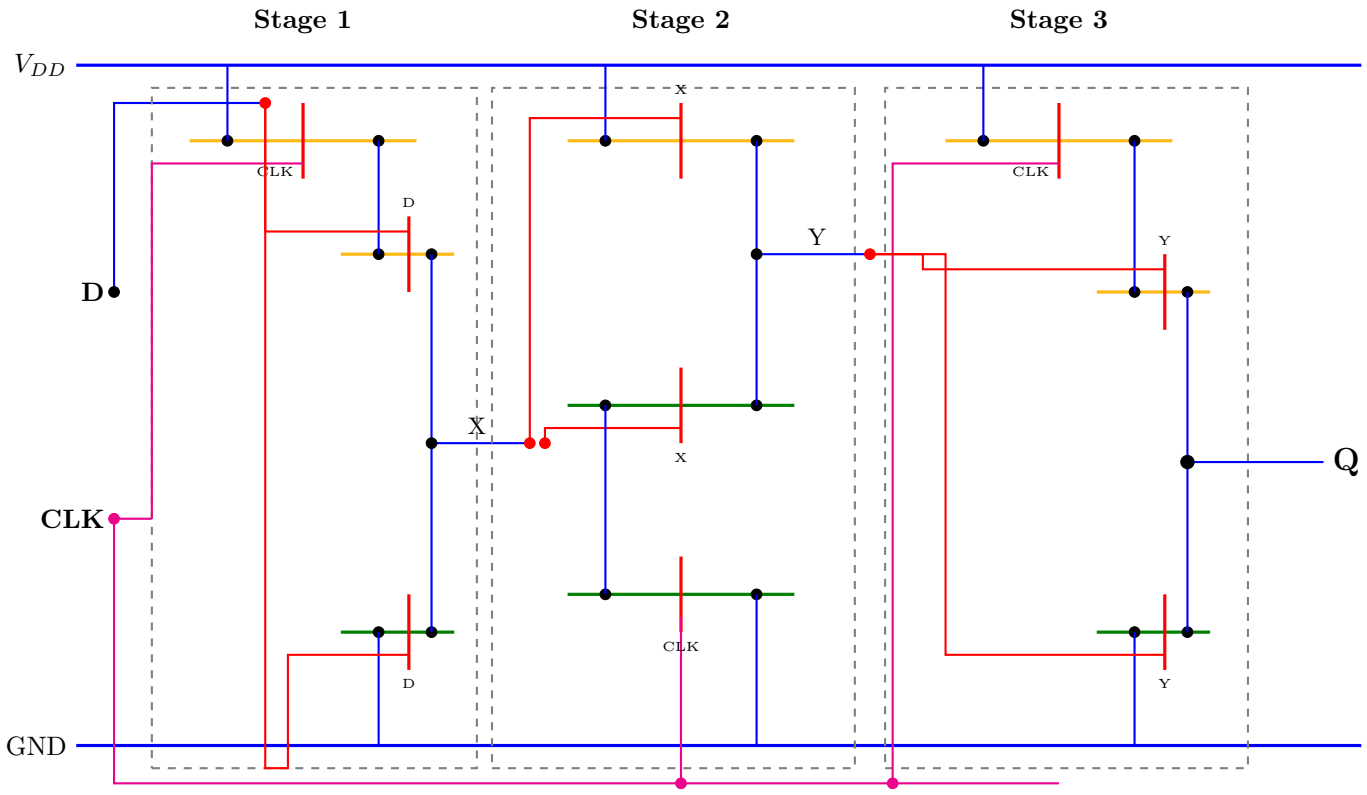


Figure 24: Stick diagram of TSPC D Flip-Flop

6 Post-layout output

6.0.1 Test Case Specification

Test Inputs:

- A = b10101 (decimal 21)
- B = b01011 (decimal 11)
- cin = 0

Expected Output:

$$\begin{aligned} \text{Sum} &= A + B + c_{in} \\ &= 21 + 11 + 0 \\ &= 32 \end{aligned}$$

Since the expected result is 32 (requires 6 bits), and we have a 5-bit adder:

- sum[4:0] = 00000 (decimal 0, which is 32 mod 32)
- cout = 1 (overflow/carry out)

6.0.2 Post-Layout Simulation Results

Input Waveforms

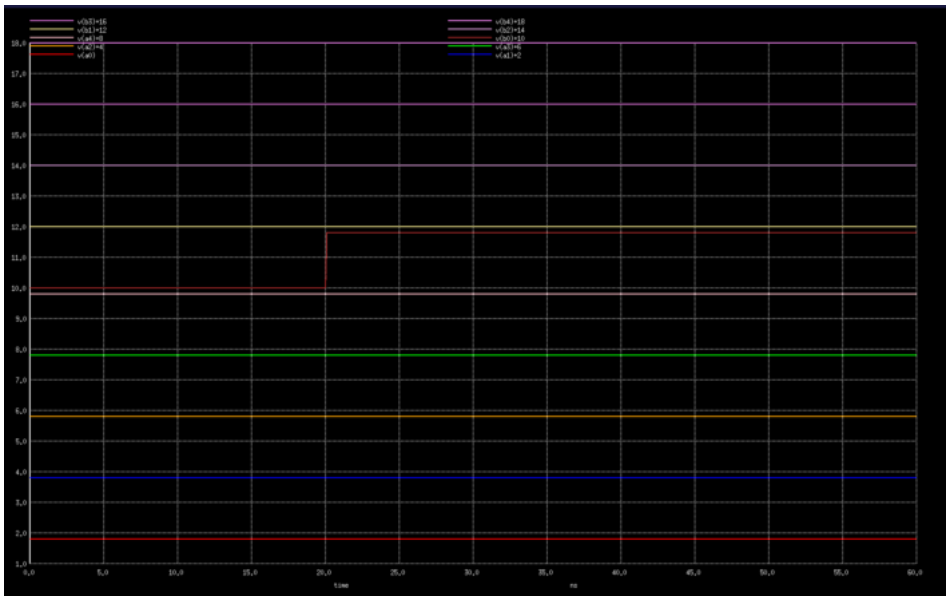


Figure 25: Post-layout simulation: Input waveforms for A=21, B=11, cin=0

Output Waveforms

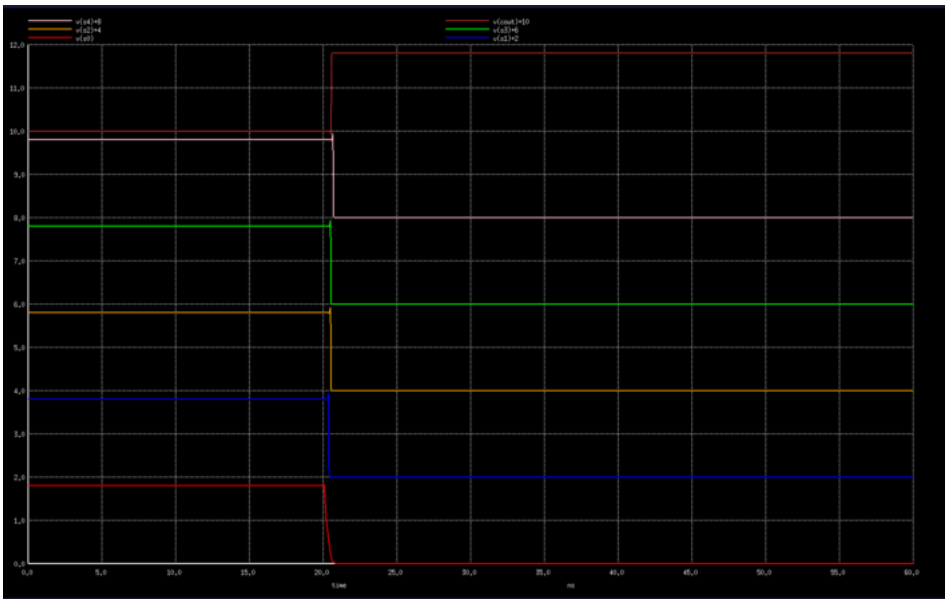


Figure 26: Post-layout simulation: Output waveforms showing sum=0, cout=1 (32 in 6-bit)

7 Verilog and FPGA Implementation

7.1 Introduction

This section presents the hardware description and FPGA implementation of the 5-bit Carry Look-Ahead (CLA) adder using Verilog. The design is synthesized using Xilinx Vivado and implemented on an FPGA board. Hardware verification is performed using oscilloscope measurements to validate the functionality.

7.2 Verilog HDL Description

7.2.1 Design Hierarchy

The 5-bit CLA adder is implemented using a hierarchical modular approach:

1. **Propagate-Generate (PG) Module**: Computes p_i and g_i for each bit
2. **Carry Look-Ahead (CLA) Module**: Computes all carry signals in parallel
3. **Sum Module**: Computes final sum bits using p_i and c_{i-1}
4. **Top Module**: Integrates all modules with input/output registers

7.3 GTKWave Waveform Visualization

7.3.1 Overview

GTKWave is an open-source waveform viewer used to visualize simulation results from Verilog testbenches. The CLA adder testbench generates a Value Change Dump (VCD) file which is then analyzed using GTKWave to verify functional correctness.

7.3.2 GTKWave Waveform Analysis

The GTKWave viewer displays all signals including inputs (A, B, cin), internal nodes (p, g, c), and outputs (sum, cout). The below waveform shows the output for inputs 10101 and 01011.

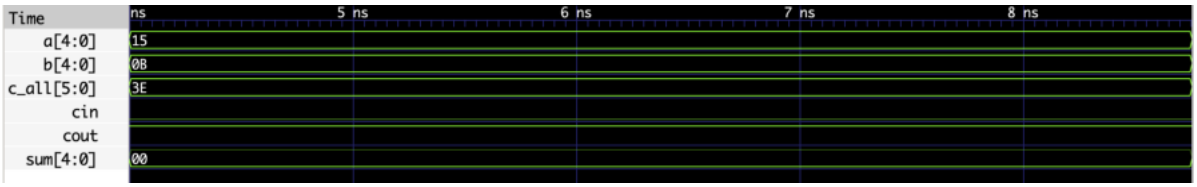


Figure 27: GTKWave waveform showing complete testbench simulation

7.4 FPGA Hardware Implementation

7.5 Oscilloscope Verification Results

The implemented design on FPGA is verified using an oscilloscope by probing the sum output bits and carry output. Multiple test cases are executed to validate correct operation.

7.5.1 Test Case 1:

Inputs:

- A = 10101 (decimal 21)
- B = 01011 (decimal 11)
- cin = 0

Expected Output:

- sum = 00000 (decimal 0)
- cout = 1

Figure 28: FPGA board output for Test Case 1: 21 + 11 = 32

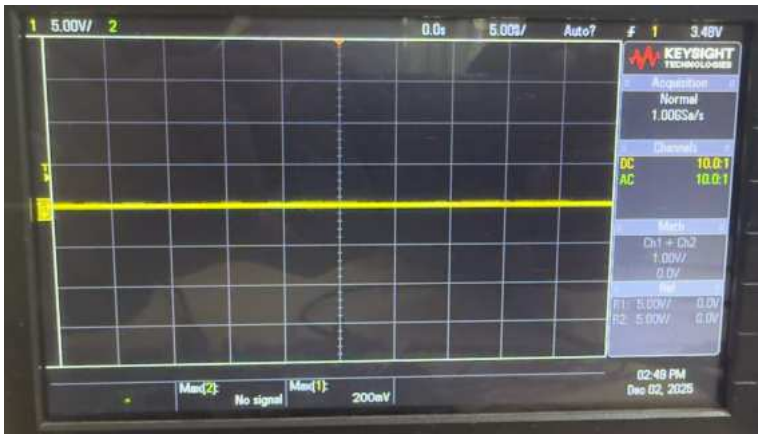


Figure 29: Bit 2 output (LOW

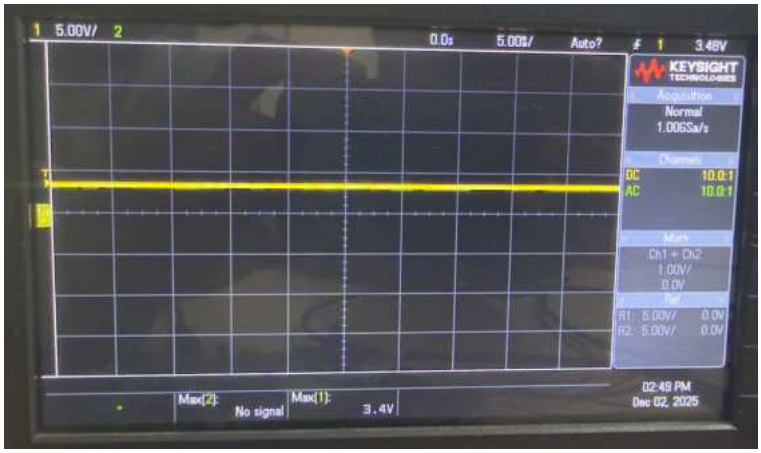


Figure 30: Bit 1 output (HIGH)