Project 20 : Sklansky Adder(4-bit)

A Comprehensive Study of Advanced Digital Circuits

By: Abhishek Sharma, Nikunj Agrawal, Ayush Jain, Gati Goyal

Contents

1 Project Overview

The *Sklansky Adder, also known as the **Divide and Conquer Adder, is a type of **parallel-prefix adder* that efficiently computes the sum of two binary numbers by generating carries in parallel. It uses a structured, hierarchical approach to compute carry signals quickly, reducing the delay associated with carry propagation.

2 Sklansky Adder(4-bit)

2.1 Description

A 4-bit Sklansky Adder performs binary addition on two 4-bit input numbers and is organized in a parallel prefix structure that optimizes carry computation. The addition is carried out in three stages: generate/propagate computation, carry computation, and sum generation.

2.2 Key Concepts of Sklansky Adder

1. *Parallel Prefix Adder*: Like other parallel-prefix adders (e.g., Brent-Kung, Kogge-Stone), the Sklansky adder generates the carry signals for each bit position in parallel rather than sequentially. This greatly speeds up the addition process.

2. *Divide and Conquer*: The Sklansky adder organizes the computation in stages by dividing the bits into groups and computing intermediate carry values across these groups. It recursively combines smaller groups of bits into larger groups until all carries are generated.

3. *Prefix Operators: The core idea of parallel-prefix adders, including Sklansky, is based on generating **generate* and *propagate* signals (G and P) for each bit: - *Generate (G)*: A carry is generated when both input bits at a certain position are 1. - *Propagate (P)*: A carry is propagated if at least one of the input bits is 1.

4. *Carry Generation Stages*: The adder consists of multiple stages, each progressively combining carry signals from smaller groups until all carry bits are computed.

5. *Structure: The Sklansky adder uses fewer logic levels than a **Kogge-Stone adder*, but has more fan-out at each stage, which may introduce wiring complexity.

2.3 RTL Code

Listing 1: Sklansky Adder(4-bit)

```
systemverilog
module sklansky_adder_4bit (
    input logic [3:0] A, // 4-bit input A
    input logic [3:0] B, // 4-bit input B
    output logic [3:0] Sum, // 4-bit output sum
    output logic CarryOut // Final carry output
);

    logic [3:0] G, P;  // Generate and propagate signals
    logic [3:0] C;     // Carry signals

    // Step 1: Generate and propagate signals
    assign G = A & B;  // Generate: G[i] = A[i] & B[i]
    assign P = A ^ B;  // Propagate: P[i] = A[i] ^ B[i]

    // Step 2: Compute carries using Sklansky prefix tree
    // First stage
    assign C[0] = G[0];
    assign C[1] = G[1]  (P[1] & G[0]);
    assign C[2] = G[2]  (P[2] & G[1])  (P[2] & P[1] & G[0]);
```

```
    assign C[3] = G[3]  (P[3] & G[2])  (P[3] & P[2] & G[1])  (P[3] &
        P[2] & P[1] & G[0]);

    // Step 3: Compute the sum and carry out
    assign Sum = P ^ {C[2:0], 1'b0}; // Sum = P ^ C
    assign CarryOut = C[3];  // Final carry out

endmodule
```

2.4 Testbench

Listing 2: Sklansky Adder(4-bit) Testbench

```
### Testbench for Sklansky Adder(4-bit) Adder

systemverilog
module tb_sklansky_adder_4bit;

    // Inputs
    logic [3:0] A;
    logic [3:0] B;

    // Outputs
    logic [3:0] Sum;
    logic CarryOut;

    // Instantiate the Sklansky Adder
    sklansky_adder_4bit uut (
        .A(A),
        .B(B),
        .Sum(Sum),
        .CarryOut(CarryOut)
    );

    // Test procedure
    initial begin
        $display("Running testbench for Sklansky Adder...");

        // Test case 1
        A = 4'b1010;
        B = 4'b1100;
        #10;
        $display("A = %b, B = %b, Sum = %b, CarryOut = %b", A, B, Sum,
            CarryOut);

        // Test case 2: Simple addition
        A = 4'b0110;
        B = 4'b0011;
        #10;
        $display("A = %b, B = %b, Sum = %b, CarryOut = %b", A, B, Sum,
            CarryOut);

        // Test case 3: Overflow case
        A = 4'b1111; // Maximum 4-bit value
        B = 4'b0001;
        #10;
        $display("A = %b, B = %b, Sum = %b, CarryOut = %b", A, B, Sum,
            CarryOut);
```

```
43
44          $stop;
45      end
46  endmodule
```

3 How it works ?

3.1 Steps in Sklansky Adder

1. *Generate and Propagate Signals*: Compute the generate (G) and propagate (P) signals for each bit of the inputs.

2. *Prefix Tree*: Use a tree-like structure to propagate the carry efficiently through the stages.

3. *Sum Calculation*: Once the carry signals are generated, compute the sum for each bit.

Explanation

The Sklansky Adder, also known as a Divide-and-Conquer Adder, is a type of parallel prefix adder that enhances the speed of binary addition by optimizing the carry propagation process. It is part of the family of parallel prefix adders, which aim to compute carry signals in a parallel manner to reduce the delay associated with traditional Ripple Carry Adders (RCA). The Sklansky Adder organizes the carry computation in a tree-like structure, making it faster and more efficient for larger bit-widths.

The Sklansky Adder computes the carries using a tree-like structure that divides the input bits into smaller groups, computes local carries within those groups, and then combines the results in the next level of the tree. This structure allows carry computation to be done in fewer stages than a ripple carry adder.

The Sklansky Adder works by computing these generate and propagate signals for each bit, then using a prefix tree structure to calculate the carry for each bit position.

3.2 Key Components

1. Carry Propagation in Addition: In binary addition, each bit can generate a carry that needs to be propagated to the higher bit positions. In a Ripple Carry Adder, this carry propagates from one bit to the next sequentially, which results in a delay that increases linearly with the number of bits.

2. Parallel Prefix Adders: Parallel prefix adders, such as the Sklansky Adder, speed up this process by computing the carry signals for multiple bit positions in parallel. They rely on generate (G) and propagate (P) signals to determine whether a bit will generate or propagate a carry.

3.3 Comparison with Other Adders:

Key Differences of Sklansky Adder Compared to Other Adders:

1. *Ripple Carry Adder*: - *Speed*: The Sklansky adder is significantly faster, as it uses parallel carry generation, whereas a ripple carry adder has a linear delay. - *Hardware*: Slightly more complex than a ripple carry adder due to the prefix tree.

2. *Kogge-Stone Adder*: - *Speed*: The Kogge-Stone adder is faster, with lower fan-out but more complex wiring. - *Hardware*: The Sklansky adder has simpler wiring than the Kogge-Stone adder, but higher fan-out per stage.

3. *Brent-Kung Adder*: - The Brent-Kung adder reduces fan-out but increases the number of logic levels compared to the Sklansky adder.

3.4 Advantages

- Low Propagation Delay:

The Sklansky Adder significantly reduces the delay associated with carry propagation. It achieves a logarithmic time complexity of O(logn) where n is the number of bits. This is because the carry signals are computed in parallel using a tree structure, making it faster than adders like the Ripple Carry Adder (RCA).

- Parallel Carry Computation:

The parallel prefix structure of the Sklansky Adder allows carry signals for all bit positions to be computed simultaneously. This parallelism helps speed up the addition process, especially for larger bit-widths.

- Scalability:

The Sklansky Adder can be easily scaled to accommodate larger bit-widths (e.g., 16-bit, 32-bit, or more). Its tree-like structure allows for efficient carry propagation across many bits, making it suitable for high-performance applications.

- Structured Design:

The divide-and-conquer methodology of the Sklansky Adder results in a highly structured design. This structure is beneficial when designing adders for hardware implementations, as it can be easily replicated for different bit-widths.

- Efficient for Wide Adders:

Compared to other adders like the Ripple Carry Adder (RCA), the Sklansky Adder performs better for wide adders due to its logarithmic delay, making it suitable for applications where performance is critical.

3.5 Disadvantages

- Uneven Fan-out:

One of the major drawbacks of the Sklansky Adder is its uneven fan-out. Certain nodes in the prefix tree structure have to drive multiple subsequent nodes, leading to unbalanced delays and potentially increasing the complexity of wiring and buffering in the circuit.

- Complex Circuitry:

The tree-like structure of the Sklansky Adder increases the circuit complexity compared to simpler adders like the Ripple Carry Adder. Although it reduces propagation delay, it requires more hardware and interconnections, making it more challenging to implement.

- High Wiring Overhead:

Due to the hierarchical combination of carry signals, the Sklansky Adder can experience high wiring complexity, especially in large designs. Managing interconnections can become a challenge, particularly when optimizing for area and power efficiency.

- Latency at High Fan-out Nodes:

The unequal distribution of carry propagation paths can lead to latency at certain stages, especially at nodes with high fan-out. This can degrade performance slightly if not addressed carefully during circuit design.

3.6 Applications

- High-Speed Arithmetic Units:

The Sklansky Adder is commonly used in Arithmetic Logic Units (ALUs) for performing high-speed addition. Its parallel structure makes it suitable for processors requiring fast arithmetic operations, such as in general-purpose CPUs or floating-point units (FPUs).

- Digital Signal Processing (DSP):

In DSP applications, where fast arithmetic operations on large datasets are crucial, the Sklansky Adder's ability to efficiently handle wide bit-widths makes it ideal for filters, multipliers, and transformations that involve repeated additions.

- Multipliers and Accumulators:

The Sklansky Adder is often used in multipliers and accumulators, where multiple partial sums need to be added quickly. It helps reduce the delay in summing partial products, improving the overall performance of these units.

- Application-Specific Integrated Circuits (ASICs):

The Sklansky Adder is a good choice for ASICs that require high-speed arithmetic processing with optimized performance. It is used in industries like telecommunications, where fast data processing is critical.

- High-Performance Processors:

In processors that require high-performance integer or floating-point arithmetic operations, such as gaming processors, graphics processing units (GPUs), and scientific computing processors, the Sklansky Adder offers faster addition times.

- Floating-Point Units (FPUs):

The Sklansky Adder is often utilized in the mantissa addition step of FPUs, where high-speed arithmetic is required. Its ability to handle large bit-widths efficiently helps improve the overall speed of floating-point operations.

3.7 Characteristics of Sklansky Adder

1. *Speed*: The Sklansky adder is faster than a ripple carry adder, as it reduces the carry propagation delay by generating carries in parallel.

2. *Complexity: It has a **logarithmic delay, as it requires log2(n) stages for n-bit inputs. However, it has **higher fan-out* at each stage compared to other parallel-prefix adders like the Kogge-Stone adder, which may affect wiring complexity and power consumption.

3. *Trade-Off*: - *Speed: Faster than ripple carry adders but slower than the **Kogge-Stone adder*, which is the fastest parallel-prefix adder. - *Hardware*: Simpler than Kogge-Stone because it requires fewer wiring connections, though it has higher fan-out at each stage.

4 Results

4.1 Simulation Results

4.2 Schematic

4.3 Synthesis Design

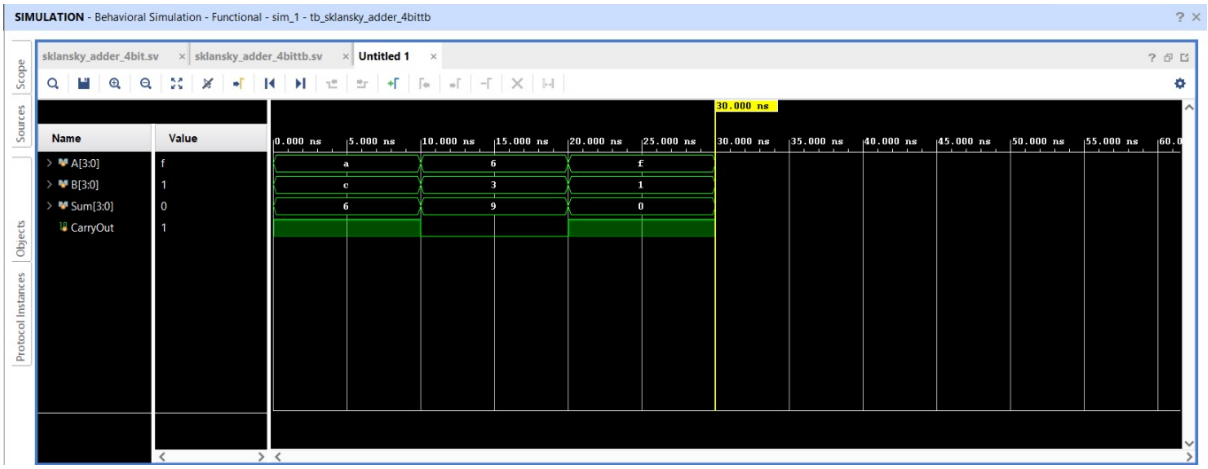

Figure 1: Simulation results of Sklansky adder

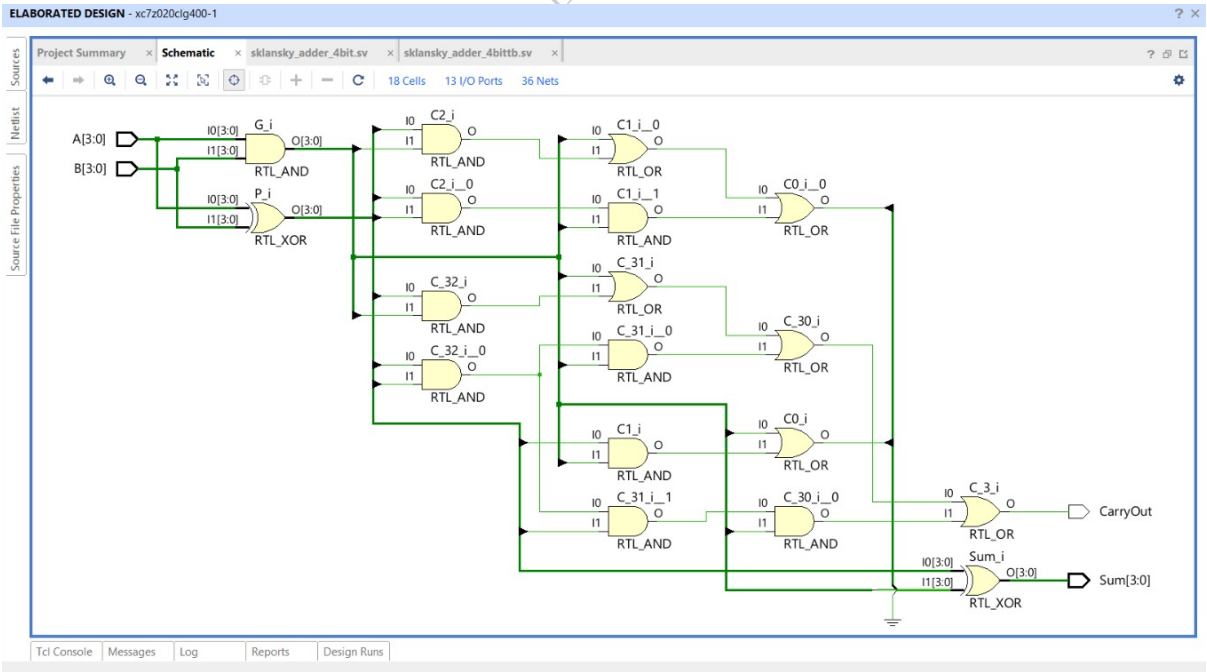

Figure 2: Schematic of Sklansky Adder

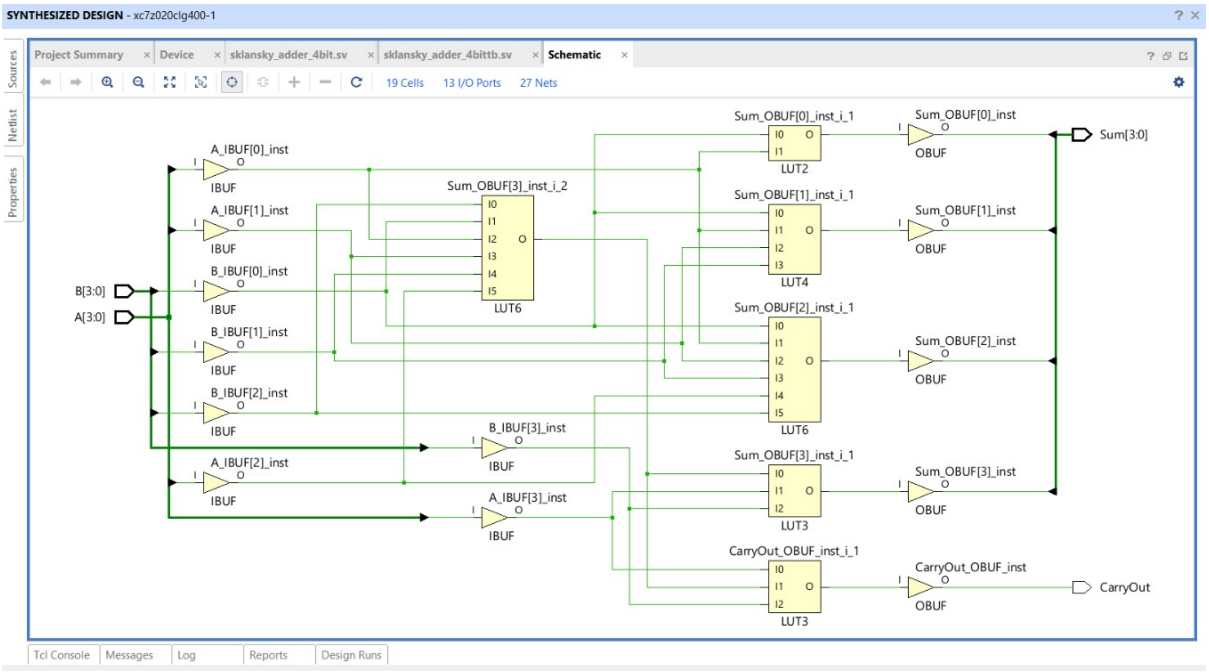

Figure 3: Synthesis Design of Sklansky Adder