ECE 310
Lab 3 Report: 4-Bit Kogge-Stone Adder

Ohm Patel

September 25, 2025

Abstract

This lab focused on the design and implementation of a 4-bit Kogge-Stone Adder (KSA) using dataflow modeling in Verilog. The KSA is a parallel prefix adder architecture designed to minimize carry propagation delay compared to ripple-carry adders. The experiment required implementing the 4-bit KSA, creating a comprehensive testbench, and verifying the correctness of the design through simulation waveforms. The results demonstrated that the KSA produces correct outputs across representative test cases and achieves faster carry resolution compared to ripple-carry implementations.

1 Introduction, Background, and Theory

Binary addition is one of the most fundamental operations in digital systems. While ripple-carry adders (RCAs) are simple to implement, their delay grows linearly with the number of bits due to serial carry propagation. The Kogge-Stone Adder (KSA) is a parallel prefix adder architecture that reduces delay by computing carry signals in parallel. It uses generate (g) and propagate (p) signals for each bit and applies a tree-based structure to compute carries.

The KSA thus significantly reduces critical path delay compared to an RCA, especially for larger bit-widths.

For a 4-bit KSA:

$$P = A_i \oplus B_i, \quad G = A_i \cdot B_i$$
$$P_i = P_i + P_{iprev}, \quad G_i = (P_i \cdot C_{iprev}) + G_i$$
$$S_i = P_i \oplus C_{i-1}, \quad C_i = G_i$$

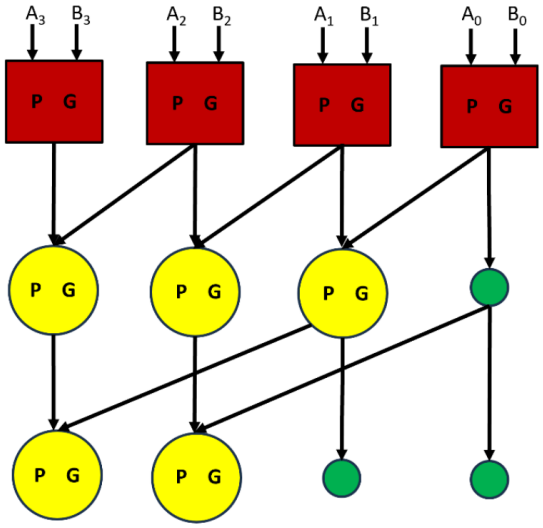

Figure 1: Diagram of the 4-bit Kogge-Stone Adder showing stages (Taken from lab manual).

2 Design and Implementation

The design was implemented using dataflow modeling in Verilog.

2.1 4-Bit Kogge-Stone Adder

The 4-bit KSA design computes generate and propagate signals, applies prefix computation, and produces the final sum and carry outputs. A Verilog implementation is shown in Listing 1.

Listing 1: 4-bit Kogge-Stone Adder (dataflow model)

```
module ksa_4bit_df(
    input wire [3:0] A, B,
    output wire [3:0] S,
    output wire Cout
    );

wire [3:0]  g0, p0,
            g1, p1,
            g2, p2,
            C;

assign p0 = A ^ B;
assign g0 = A & B;

// Stage 1

assign p1[0] = p0[0];
assign g1[0] = g0[0];

assign p1[1] = p0[1] & p0[0];
assign g1[1] = (p0[1] & g0[0]) | g0[1];
assign p1[2] = p0[2] & p0[1];
assign g1[2] = (p0[2] & g0[1]) | g0[2];
assign p1[3] = p0[3] & p0[2];
assign g1[3] = (p0[3] & g0[2]) | g0[3];

// Stage 2

assign p2[0] = p1[0];
assign g2[0] = g1[0];
assign p2[1] = p1[1];
assign g2[1] = g1[1];

assign p2[2] = p1[2] & p1[0];
assign g2[2] = (p1[2] & g1[0]) | g1[2];
assign p2[3] = p1[3] & p1[1];
assign g2[3] = (p1[3] & g1[1]) | g1[3];

// Stage 3

assign C[0] = 1'b0;  // no carry-in
assign C[1] = g1[0]; // carry into bit1
assign C[2] = g1[1]; // carry into bit2
assign C[3] = g2[2]; // carry into bit3
assign Cout = g2[3]; // carry out

// Assign Sum output

assign S[0] = p0[0] ^ C[0];
assign S[1] = p0[1] ^ C[1];
assign S[2] = p0[2] ^ C[2];
assign S[3] = p0[3] ^ C[3];

endmodule
```

2.2 Testbench

The testbench was constructed to cover representative test cases, as shown in Listing 2. The selected inputs ensure testing of baseline, carry propagation, overflow, and mixed intermediate conditions.

Listing 2: Testbench for 4-bit KSA

```
module ksa_4bit_tb;
    reg [3:0] A, B;
    wire [3:0] S;
    wire Cout;

    ksa_4bit_df dut(A, B, S, Cout);

    initial
    begin

        A = 4'b0000; B = 4'b0000;
        A = 4'b0001; B = 4'b0010;
        A = 4'b0001; B = 4'b0001;
        A = 4'b0111; B = 4'b0001;
        A = 4'b1111; B = 4'b0001;
        A = 4'b1010; B = 4'b0101;
        A = 4'b1111; B = 4'b1111;

        // Line after each testcase (removed for simplicity)
        #10 $display("A: %b B: %b S: %b Cout: %b", A, B, S, Cout);

        $finish;
    end
endmodule
```

3 Results and Analysis

Seven test cases were used to validate the KSA:

- **0000 + 0000:** Baseline check to confirm no spurious outputs.
- **0001 + 0010:** Simple addition with no carry.
- **0001 + 0001:** Carry generated at the LSB.
- **0111 + 0001:** Carry propagation into higher-order bits.
- **1111 + 0001:** Full carry propagation resulting in Cout.
- **1010 + 0101:** Mixed bit addition producing maximum value without overflow.
- **1111 + 1111:** Maximum inputs producing overflow (Cout = 1).

These results confirm the correctness of the design and demonstrate the KSA's ability to resolve carries in parallel. The waveform output (Figure 2) verifies that the adder behaved as expected for all test cases.

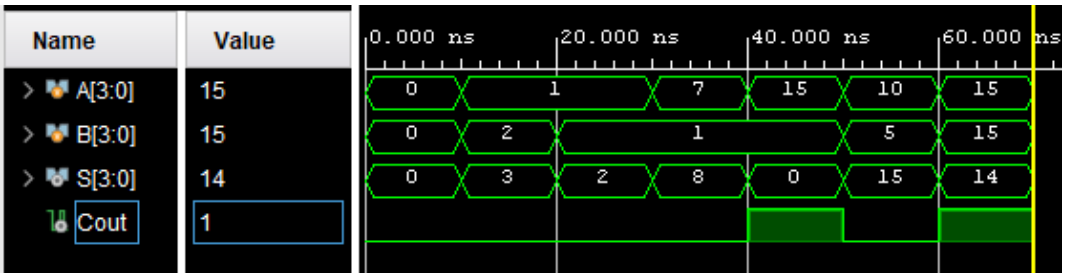

Figure 2: Simulation waveform of the 4-bit Kogge-Stone Adder showing selected test cases.

4 Conclusion

A 4-bit Kogge-Stone Adder was successfully implemented in Verilog using dataflow modeling. Simulation results verified the correctness of the design across representative test cases, including baseline, carry generation, carry propagation, and overflow. The experiment highlighted the efficiency of prefix adders such as the KSA in reducing carry propagation delay compared to ripple-carry designs, reinforcing their importance in high-performance arithmetic circuits.

This report was compiled using LaTeX .