High-Speed and Low-Power 16-bit Carry Skip Adder Implementation in Verilog

Abstract

Adders are fundamental components of digital systems, widely utilized in digital signal processors (DSPs), filters, and arithmetic logic units (ALUs). Conventional carry-skip adders (CSKA) balance speed and area efficiency but exhibit limitations in terms of propagation delay, power consumption, and design area. To address these issues, a novel hybrid carry-skip adder (Hybrid CSKA) is proposed. This design integrates a hybrid multiplexer (MUX) for skip logic, significantly improving performance in terms of speed, energy efficiency, and area utilization.

Introduction

The increasing demand for mobile and high-performance electronic devices necessitates power-efficient, high-speed, and compact VLSI circuits. Adders, being critical building blocks in arithmetic units, heavily influence system performance. While ripple carry adders (RCA) offer simplicity, they suffer from high propagation delay. Carry-skip adders (CSKA) are widely preferred due to their improved speed and area efficiency.

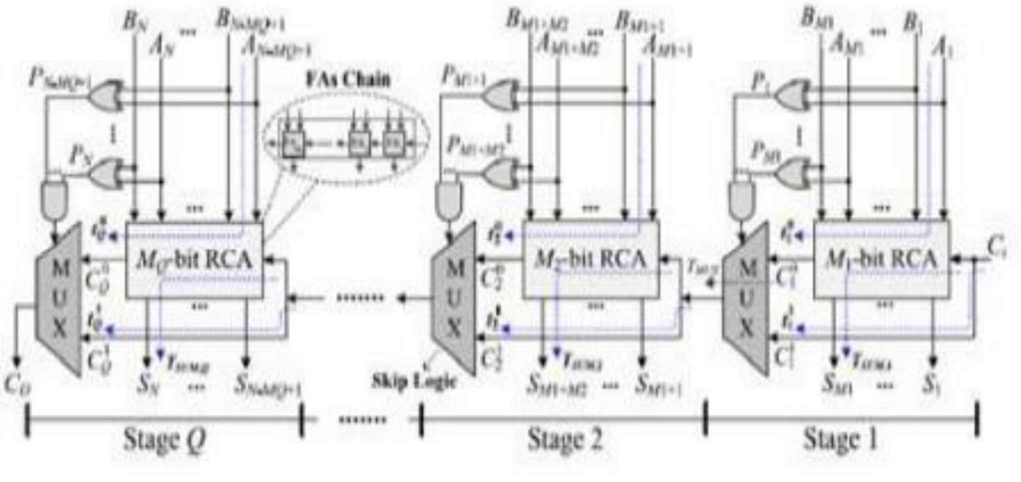
Therefore, a new hybrid CSKA design is proposed, leveraging a hybrid MUX to address these limitations.

Limitations of Conventional CSKA

1. **Propagation Delay**: Traditional multiplexers in skip logic require 12 transistors, contributing to higher delay in carry propagation.
2. **Power Consumption**: Compound gates, though slightly better than conventional MUX in terms of power, still exhibit higher energy usage, especially in high-performance applications.
3. **Design Area**: Both conventional MUX and compound gates increase the silicon footprint, making them less suitable for compact designs.

Conventional Carry Skip Adder

The conventional carry-skip adder (CSKA) is a widely used adder design that balances speed and area efficiency. It combines multiple ripple carry adder (RCA) blocks with skip logic, which allows the carry signal to bypass certain stages to reduce overall propagation delay.



16bit Carry skip Adder Code

```
module CSK_16bit( input[15:0]a,input
c,input[15:0]b,output[16:0]s);
   wire t1, t6, t7;
wire [3:0] xor_ab;
RCA_4bit x1(.a(a[3:0]), .b(b[3:0]), .cin(c), .cout(t1),
.sum(s[3:0]));
genvar i;
generate
    for (i = 0; i < 4; i = i + 1) begin : xor_gen
        assign xor_ab[i] = a[i] ^ b[i];
    end
endgenerate
assign t6 = &xor_ab;
assign t7 = t6 ? c : t1;
    wire q1, q6, q7;
wire [3:0] xor_ab2;
RCA_4bit x2(.a(a[7:4]), .b(b[7:4]), .cin(t7), .cout(q1),
.sum(s[7:4]));
genvar j;
generate
    for (j = 0; j < 4; j = j + 1) begin : xor_gen2
```

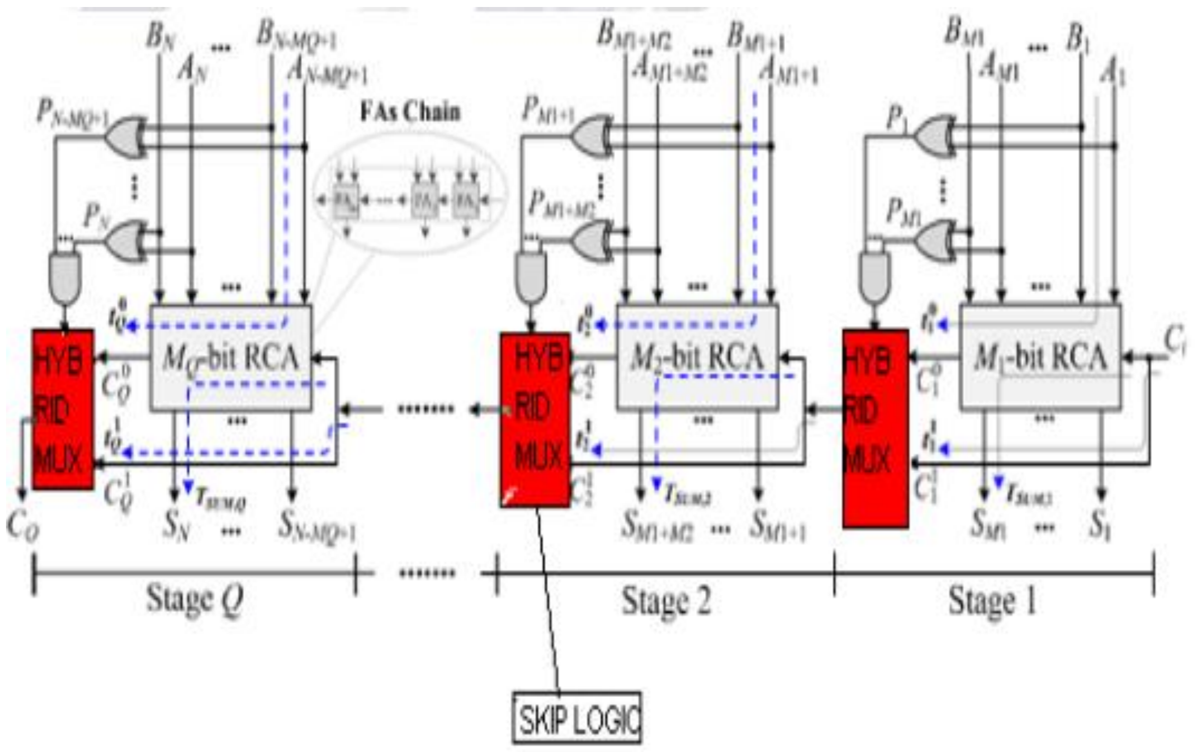
```
        assign xor_ab2[j] = a[4+j] ^ b[4+j];
    end
endgenerate
assign q6 = &xor_ab2;
assign q7 = q6 ? t7 : q1;



    wire m1, m6, m7;
wire [3:0] xor_ab3;
RCA_4bit x3(.a(a[11:8]), .b(b[11:8]), .cin(q7), .cout(m1),
.sum(s[11:8]));
genvar k;
generate
    for (k = 0; k < 4; k = k + 1) begin : xor_gen3
        assign xor_ab3[k] = a[8+k] ^ b[8+k];
    end
endgenerate
assign m6 = &xor_ab3;
assign m7 = m6 ? q7 : m1;
```

```
   wire n1, n6;
wire [3:0] xor_ab4;
RCA_4bit x4(.a(a[15:12]), .b(b[15:12]), .cin(m7), .cout(n1),
.sum(s[15:12]));
genvar l;
generate
    for (l = 0; l < 4; l = l + 1) begin : xor_gen4
        assign xor_ab4[l] = a[12+l] ^ b[12+l];
    end
endgenerate
assign n6 = &xor_ab4;
assign s[16] = n6 ? m7 : n1;


endmodule



 module RCA_4bit(
    input [3:0] a, b, input cin,output cout,output [3:0] sum);
    wire [3:0] carry;
   genvar i;



   generate
       for (i = 0; i < 4; i = i + 1) begin: adder_block
           if (i == 0)
              FA_1bit ff (
.a(a[i]),.b(b[i]),.c(cin),.carry(carry[i]),.s(sum[i]));


           else if (i == 3)
               FA_1bit ff (.a(a[i]),.b(b[i]),.c(carry[i-
1]),.carry(cout),.s(sum[i]));
           else
                FA_1bit ff (.a(a[i]),.b(b[i]),.c(carry[i-
1]),.carry(carry[i]),.s(sum[i]));
            end
   endgenerate


endmodule


module FA_1bit(
   input a, b, c,output carry, s );
```

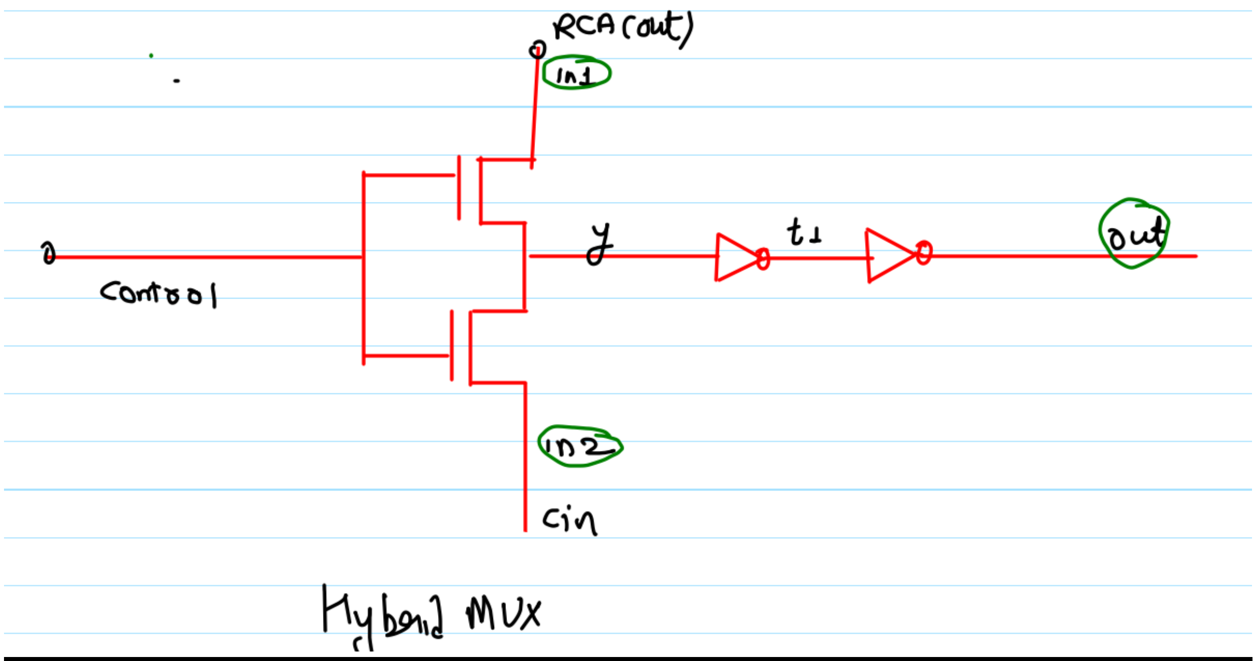
```
    assign s = a ^ b ^ c;
    assign carry = (a & b) | (b & c) | (c & a);
endmodule
```

Proposed Hybrid CSKA

The proposed hybrid CSKA uses a new approach for skip logic by replacing the conventional multiplexer and compound gates with a "Hybrid MUX." This modification enhances speed, reduces power consumption, and minimizes the design area.



HYBRID MUX



Working Principle of Hybrid MUX

Hybrid MUX is similar to a CMOS inverter and consists of one NMOS, one PMOS, and two NOT gates. It takes two inputs, one output, and a conditional signal. The conditional signal is generated using partial products of the input bits through an AND gate. Depending on the conditional signal:

- If the signal is 1, the NMOS transistor turns ON and directly passes the input carry (Cin) to the next stage.
- If the signal is 0, the PMOS transistor turns ON, and the RCA block's carry-out (Cout) serves as the carry for the next stage.

Hybrid MUX Code :

module inverter_cmos (

```
    input  a,
    output  out  ,
    input vdd, vss
);
wire t1,y;
 pmos p1 (y, vdd, a);
 nmos n2 (y, vss, a);
 assign t1=~y;
 assign out =~t1;

endmodule
```

Proposed Hybrid CSKA Code

```
module newCSKA( input[15:0]a,input
c,input[15:0]b,output[16:0]s);



   wire t1, t6, t7;
wire [3:0] xor_ab;
```

```
RCA_4bit x1(.a(a[3:0]), .b(b[3:0]), .cin(c), .cout(t1),
.sum(s[3:0]));
genvar i;
generate
    for (i = 0; i < 4; i = i + 1) begin : xor_gen
        assign xor_ab[i] = a[i] ^ b[i];
    end
endgenerate
assign t6 = &xor_ab;
 inverter_cmos hyb_mux1(.control(t6),.in1(t1) ,.in2(c),.out(t7));



    wire q1, q6, q7;
wire [3:0] xor_ab2;
RCA_4bit x2(.a(a[7:4]), .b(b[7:4]), .cin(t7), .cout(q1),
.sum(s[7:4]));
genvar j;
generate
    for (j = 0; j < 4; j = j + 1) begin : xor_gen2
        assign xor_ab2[j] = a[4+j] ^ b[4+j];
    end
```

```
endgenerate
assign q6 = &xor_ab2;

inverter_cmos hyb_mux2(.control(q6),.in1(q1)
,.in2(t7),.out(q7));



    wire m1, m6, m7;
wire [3:0] xor_ab3;
RCA_4bit x3(.a(a[11:8]), .b(b[11:8]), .cin(q7), .cout(m1),
.sum(s[11:8]));
genvar k;
generate
    for (k = 0; k < 4; k = k + 1) begin : xor_gen3
        assign xor_ab3[k] = a[8+k] ^ b[8+k];
    end
endgenerate
assign m6 = &xor_ab3;
inverter_cmos hyb_mux3(.control(m6),.in1(m1)
,.in2(q7),.out(m7));

   wire n1, n6;
wire [3:0] xor_ab4;
RCA_4bit x4(.a(a[15:12]), .b(b[15:12]), .cin(m7), .cout(n1),
.sum(s[15:12]));
genvar l;
generate
    for (l = 0; l < 4; l = l + 1) begin : xor_gen4
        assign xor_ab4[l] = a[12+l] ^ b[12+l];
    end
endgenerate
assign n6 = &xor_ab4;
inverter_cmos hyb_mux4(.control(n6),.in1(n1)
,.in2(m7),.out(s[16]));

endmodule



 module RCA_4bit(
    input [3:0] a, b, input cin,output cout,output [3:0] sum);
    wire [3:0] carry;


    genvar i;
 generate
```

```
        for (i = 0; i < 4; i = i + 1) begin: adder_block
            if (i == 0)
              FA_1bit ff (
.a(a[i]),.b(b[i]),.c(cin),.carry(carry[i]),.s(sum[i]));


            else if (i == 3)
               FA_1bit ff (.a(a[i]),.b(b[i]),.c(carry[i-
1]),.carry(cout),.s(sum[i]));
            else
               FA_1bit ff (.a(a[i]),.b(b[i]),.c(carry[i-
1]),.carry(carry[i]),.s(sum[i]));
             end
    endgenerate
endmodule
module FA_1bit(
    input a, b, c,output carry, s );
    assign s = a ^ b ^ c;
    assign carry = (a & b) | (b & c) | (c & a);
endmodule



module inverter_cmos (input  control,in1 , in2, output  out);
```

```
wire t1,y;
 pmos p1 (y,in1,control);
 nmos n2 (y, in2,control);
 assign t1=~y;
 assign out =~t1;

endmodule
```

Testbench

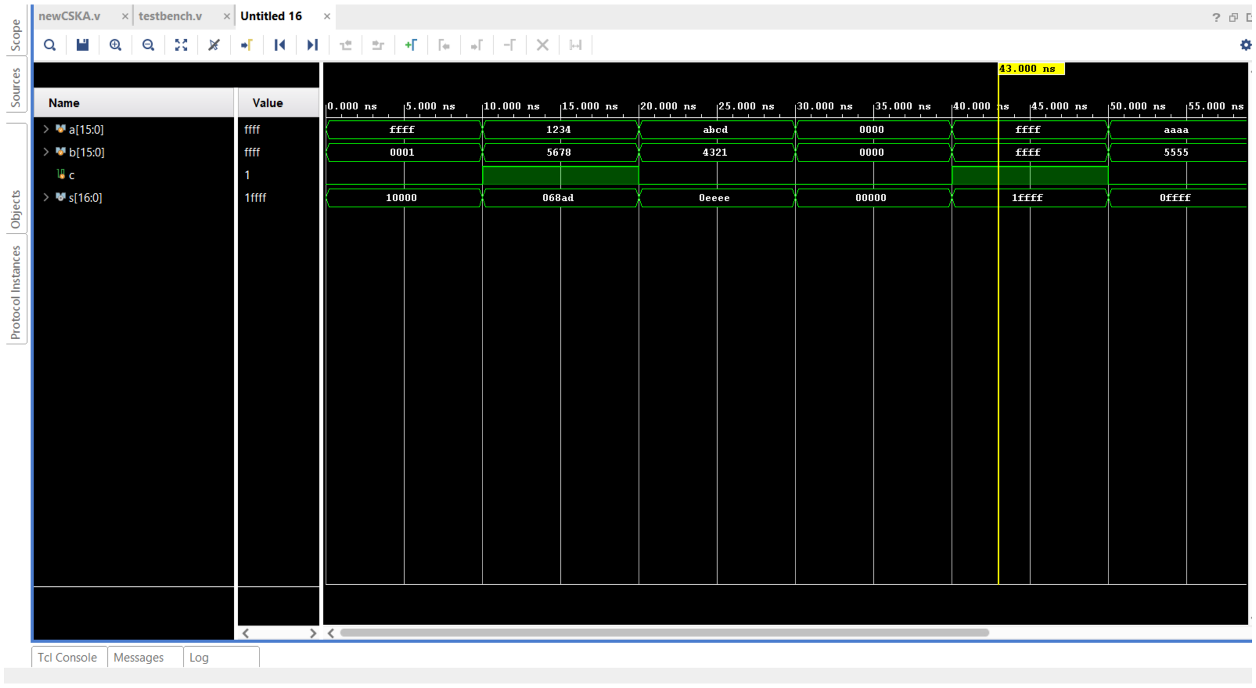
```
module testbench;
reg [15:0] a;
  reg [15:0] b;
  reg c;
 wire [16:0] s;
  newCSKA uut (.a(a),.b(b), .c(c),.s(s) );
  initial begin

    $monitor("a = %h, b = %h, c = %b, s = %h", a, b, c, s);
     a = 16'hFFFF; b = 16'h0001; c = 0;
    #10;
    a = 16'h1234; b = 16'h5678; c = 1;
    #10;
    a = 16'hABCD; b = 16'h4321; c = 0;
    #10;
    a = 16'h0000; b = 16'h0000; c = 0;
    #10;
    a = 16'hFFFF; b = 16'hFFFF; c = 1;
    #10;
    a = 16'hAAAA; b = 16'h5555; c = 0;
    #10;
     a = 16'hFFFF; b = 16'hFFFF; c = 1;
    #10;
    $finish;
  End
endmodule
```

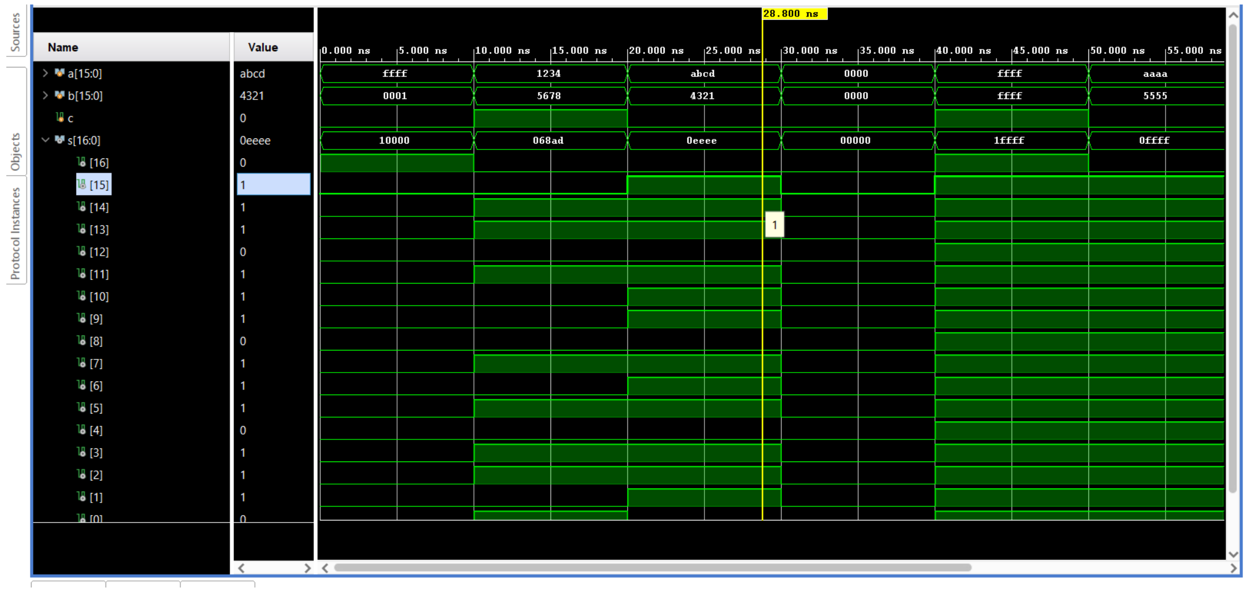
Simulation Results

```
a = ffff, b = 0001, c = 0, s = 10000
a = 1234, b = 5678, c = 1, s = 068ad
a = abcd, b = 4321, c = 0, s = 0eeee
a = 0000, b = 0000, c = 0, s = 00000
a = ffff, b = ffff, c = 1, s = 1ffff
```

a = aaaa, b = 5555, c = 0, s = 0ffff

a = ffff, b = ffff, c = 1, s = 1ffff





Simulations were conducted using Xilinx tools with a 45nm CMOS technology node. The hybrid CSKA was compared against conventional CSKA and other existing designs. Key findings include:

The hybrid CSKA demonstrates significant improvements, achieving a 45% reduction in delay and a 40% reduction in power consumption compared to the conventional CSKA.

Comparison	Delay (ns)	Power Consumption (%mv)
Conventional CSKA	16	17
Proposed Hybrid CSKA	7.8	6

Table 1: Comparison of Conventional and Proposed Hybrid CSKA

The hybrid CSKA demonstrates significant improvements, achieving a 45% reduction in delay and a 40% reduction in power consumption compared to the conventional CSKA.

Conclusion

The proposed hybrid CSKA effectively addresses the limitations of conventional designs, achieving higher speed, lower power consumption, and reduced design area. These characteristics make it an ideal choice for applications requiring energy-efficient and high-performance arithmetic units.

References

1. Zimmermann, R. (1998). Binary Adder Architectures for Cell-Based VLSI and Their Synthesis. *Ph.D. Dissertation,*

Swiss Federal Institute of Technology (ETH), Zürich, Switzerland.

2. Koren, I. (2002). *Computer Arithmetic Algorithms* (2nd ed.). Natick, MA: A K Peters, Ltd.
3. Oklobdzija, V. G., Zeydel, B. R., et al. (2005). Comparison of High-Performance VLSI Adders in the Energy-Delay Space. *IEEE Transactions on VLSI Systems*, 13(6), 754-758.
4. Alioto, M., & Palumbo, G. (2003). A Simple Strategy for Optimized Design of One-Level Carry-Skip Adders. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 50(1), 141-148.
5. Kantabutra, V. (1993). Designing Optimum One-Level Carry-Skip Adders. *IEEE Transactions on Computers*, 42(6), 759-764.
6. Konduru, L. B. P., & Kumar, S. V. (2017). High-Speed, Low Area, and Energy Efficient 32-bit Carry Skip Adder using Verilog HDL. *International Journal of Engineering Research in Electronics and Communication Engineering (IJERECE)*, 4(3), 205-209.
7. Suzuki, H., Jeong, W., & Roy, K. (2004). Low-Power Carry-Select Adder Using Adaptive Supply Voltage Based on Input Vector Patterns. *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, 313-318.

8. Turrini, S. (1989). Optimal Group Distribution in Carry-Skip Adders. *Proceedings of the 9th IEEE Symposium on Computer Arithmetic*, 96-103.