Project 19: Carry Bypass Adder (CBA)

A Comprehensive Study of Advanced Digital Circuits

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

Contents

1 Project Overview

The *Carry Bypass Adder (CBA), also known as **Carry-Skip Adder, is a type of adder that aims to speed up addition by bypassing (skipping) carry propagation over certain groups of bits. This makes it faster than a traditional **Ripple Carry Adder (RCA)* but still simpler than a *Carry-Lookahead Adder (CLA)*.

2 Carry Bypass Adder

2.1 Description

A Carry Bypass Adder (CBA), also known as a Carry Skip Adder, is designed to improve the speed of binary addition by reducing the carry propagation delay seen in simple ripple carry adders. The primary idea is to bypass the carry propagation in certain conditions, allowing the adder to operate faster.

2.2 Key Concepts of Carry Bypass Adder (CBA)

1. *Goal*: Reduce the delay caused by the carry propagation in a typical adder. The carry bypasses groups of bits when it is determined that the carry from a previous stage will not affect the current group.

2. *Structure*: - *Groups of Bits*: The adder is divided into several groups or blocks of bits. - *Ripple Carry within Groups*: Within each group, the addition is performed using ripple carry logic. - *Bypass Logic*: If a group does not generate or propagate a carry, the carry can "skip" to the next group without waiting for the carry to ripple through all the bits in the current group.

3. *Carry Propagation: For each group, a **propagate signal* (P) is generated. If all bits in a group are "propagating" (i.e., the carry simply passes through the group without affecting the result), the carry is bypassed directly to the next group.

4. *Speed Improvement*: By skipping unnecessary carry propagation across groups, the delay in a carry-bypass adder is shorter than a ripple-carry adder but slightly longer than a carry-lookahead adder.

2.3 RTL Code

Listing 1: Carry Bypass Adder

```
module carry_bypass_adder (
    input logic [15:0] A, // 16-bit input A
    input logic [15:0] B, // 16-bit input B
    output logic [15:0] Sum, // 16-bit sum output
    output logic CarryOut // Final carry output
);

    logic [3:0] carry_in, carry_out; // Carry in and carry out for
        each group
    logic [3:0] propagate; // Propagate signals for each group

    // Divide the 16-bit inputs into 4-bit groups and ripple carry add
        them
    // Group 0 (bits 3:0)
    assign {carry_out[0], Sum[3:0]} = A[3:0] + B[3:0] + 1'b0;

    // Group 1 (bits 7:4)
    assign propagate[0] = &(A[3:0] ^ B[3:0]); // If all bits propagate
        carry
    assign carry_in[1] = propagate[0] ? carry_out[0] : 1'b0;
    assign {carry_out[1], Sum[7:4]} = A[7:4] + B[7:4] + carry_in[1];

```

```
    // Group 2 (bits 11:8)
    assign propagate[1] = &(A[7:4] ^ B[7:4]); // If all bits propagate
        carry
    assign carry_in[2] = propagate[1] ? carry_out[1] : 1'b0;
    assign {carry_out[2], Sum[11:8]} = A[11:8] + B[11:8] + carry_in[2];

    // Group 3 (bits 15:12)
    assign propagate[2] = &(A[11:8] ^ B[11:8]); // If all bits
        propagate carry
    assign carry_in[3] = propagate[2] ? carry_out[2] : 1'b0;
    assign {CarryOut, Sum[15:12]} = A[15:12] + B[15:12] + carry_in[3];

endmodule
```

2.4 Testbench

Listing 2: Carry Bypass Adder Testbench

```
### Testbench for Carry Bypass Adder

systemverilog
module tb_carry_bypass_adder;

    // Inputs
    logic [15:0] A;
    logic [15:0] B;

    // Outputs
    logic [15:0] Sum;
    logic CarryOut;

    // Instantiate the Carry Bypass Adder
    carry_bypass_adder uut (
        .A(A),
        .B(B),
        .Sum(Sum),
        .CarryOut(CarryOut)
    );

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

        // Test case 1
        A = 16'd12345;
        B = 16'd54321;
        #10;
        $display("A = %d, B = %d, Sum = %d, CarryOut = %b", A, B, Sum,
            CarryOut);

        // Test case 2
        A = 16'd1000;
        B = 16'd500;
        #10;
        $display("A = %d, B = %d, Sum = %d, CarryOut = %b", A, B, Sum,
            CarryOut);

```

```
38          // Test case 3: Overflow case
39          A = 16'd65535; // Maximum 16-bit value
40          B = 16'd1;
41          #10;
42          $display("A = %d, B = %d, Sum = %d, CarryOut = %b", A, B, Sum,
                CarryOut);
43
44          $stop;
45      end
46  endmodule
```

3 How it works ?

3.1 Steps in Carry Bypass Adder

- Step 1: Divide the adder into fixed-sized groups of bits.
- Step 2: Perform ripple carry addition within each group.
- Step 3: For each group, compute a propagate signal P. If the propagate signal is 1, the carry bypasses the group, and the carry from the previous stage is used directly for the next stage.
- Step 4: Continue to the next group, where the ripple carry adder starts again.

Explanation

A Carry Bypass Adder is divided into blocks or groups of bits. Each block contains a simple Ripple Carry Adder (RCA) that adds the corresponding bits of the input operands within the block. For each block, a special circuit called the carry bypass logic decides whether the carry generated in a previous block should propagate directly to the next block (bypassing the current block) or propagate through the individual bits of the block.

3.2 Key Components

The major components of a Carry Bypass Adder include:

Ripple Carry Adders in Each Block: Each block has its own internal RCA that computes the sum and the carry.

Carry Propagation Logic: This logic determines whether the carry should propagate through each bit of the block.

Bypass Logic: If the carry can propagate directly across the entire block, the carry bypass logic will allow the carry to "skip" the block and go directly to the next one.

3.2.1 Propagate and Generate Signals:

To understand how carry bypass works, it's essential to understand the generate and propagate signals for each bit:

Generate (G): A bit generates a carry when both input bits are 1 (A = 1, B = 1), regardless of the carry input. This produces a carry output of 1.

Propagate (P): A bit propagates a carry if one of the input bits is 1 and the other is 0 (A = 1, B = 0 or A = 0, B = 1). In this case, the carry input to this bit propagates to the next bit.

3.2.2 For a group of bits (or block):

The block will propagate the carry if all the bits within the block propagate the carry (i.e., all the bits have the propagate signal set to 1). The block will generate a carry if one of the bits generates a carry (i.e., has the generate signal set to 1).

3.3 Comparison with Other Adders:

Key Differences of Carry Bypass Adder Compared to Other Adders:

1. *Ripple Carry Adder (RCA)*: - *CBA Advantage: CBA improves on the **ripple carry adder* by reducing the carry propagation delay across bit groups. - *Speed*: Ripple carry adders propagate the carry through all bits in sequence, making them slower. The carry bypass mechanism in the CBA allows it to skip over groups, speeding up the process.

2. *Carry-Lookahead Adder (CLA)*: - *CBA Disadvantage: A **carry-lookahead adder* is faster than a CBA because it generates the carry signals in parallel for all bits. However, CLA is more complex and uses more hardware. - *Hardware*: CLA requires more gates and complex wiring compared to CBA, which has a simpler design but with reduced speed.

3. *Wallace Tree / Dadda Tree Adders*: - *Different Purpose: Wallace Tree and Dadda Tree adders are used in multipliers to sum multiple operands in stages. The CBA, on the other hand, is designed to speed up **two-operand addition*.

4. *Carry-Save Adder (CSA)*: - *Different Purpose*: Carry-Save Adders are used to add multiple numbers in parallel by delaying the carry propagation until the final stage, whereas CBA is used for two numbers, skipping unnecessary carry propagation within groups.

3.4 Advantages

The Carry Bypass Adder (CBA) offers several advantages that make it an attractive choice for specific digital circuit designs, especially when considering performance and complexity trade-offs. Below are the key advantages:

- Reduced Carry Propagation Delay:

In a Ripple Carry Adder (RCA), the carry must ripple through each bit, leading to a significant delay for large bit-widths. The CBA reduces this delay by allowing the carry to skip entire blocks of bits when possible, effectively shortening the overall carry propagation path. This leads to faster addition compared to a Ripple Carry Adder, especially for adders with more bits.

- Balanced Trade-off Between Speed and Complexity:

Faster than Ripple Carry Adders: By reducing carry propagation time through the bypass mechanism, the CBA performs faster than a standard RCA. Simpler than Carry Look-Ahead Adders: While Carry Look-Ahead Adders (CLA) can be faster, they are more complex and require more hardware. The CBA offers a simpler design with less area overhead, making it easier and cheaper to implement. This balance makes it a good choice when looking for a moderate increase in speed without the complexity of more advanced adders like CLAs.

- Efficient for Medium-Sized Adders:

The CBA is well-suited for adders of medium size (e.g., 16-bit or 32-bit adders). For such sizes, it achieves a significant speed improvement compared to RCAs without incurring the higher complexity and power consumption of CL

3.5 Disadvantages

While the Carry Bypass Adder (CBA) offers advantages in terms of speed and simplicity, it also has certain drawbacks. These disadvantages include:

- Limited Speed Improvement:

Suboptimal for Large Bit Widths: For very large adders (e.g., 64-bit or more), the performance gains of a CBA are not as significant compared to more advanced adders like Carry Look-Ahead Adders (CLA) or Carry Select Adders (CSA). As the number of bits increases, the bypass mechanism may still encounter delays because the carry propagation still occurs within each block, limiting its speed improvement in large-scale operations.

- Bypass Logic Overhead:

The addition of bypass logic introduces extra circuitry. This increases the hardware complexity compared to a Ripple Carry Adder (RCA), although it remains simpler than a CLA. The bypass logic itself needs to be evaluated, which can add delays, especially when not all blocks allow carry bypassing. This overhead might limit the speed gains, particularly in certain cases where the carry must propagate through many blocks.

- Uneven Delay:

The delay in a CBA is not uniform. While some blocks can be bypassed quickly, others may need to propagate the carry through all their bits. This leads to varying delays in different sections of the adder, which can make the timing of the adder unpredictable. Worst-case Delay: If many blocks generate a carry, the CBA essentially becomes like a Ripple Carry Adder for those blocks, resulting in delays that are similar to a ripple carry scenario.

- Less Efficient for Small Bit Widths:

For small bit-widths (e.g., 4-bit or 8-bit adders), the complexity added by the bypass logic may not justify the minor speed improvements. In these cases, simpler designs like Ripple Carry Adders (RCA) are more efficient due to their minimal hardware and simpler implementation.

- Block Size Selection:

Design Complexity in Block Sizing: The choice of how to divide the adder into blocks (block size) is critical to achieving optimal performance. Poor block size selection can lead to suboptimal speed improvements or even degraded performance. Larger blocks reduce the number of bypass stages but increase the propagation delay within the block, while smaller blocks increase the number of bypass stages but reduce the propagation delay within each block. This adds design complexity.

- Power Consumption:

Although less power-consuming than more complex adders like CLAs, the CBA consumes more power than a simple RCA due to the additional bypass circuitry. This extra power may not be acceptable in low-power applications.

3.6 Applications

The Carry Bypass Adder (CBA) finds its use in various digital circuits and systems where a balance between speed and complexity is desired. Its moderate improvement in performance over Ripple Carry Adders (RCAs) and lower complexity compared to Carry Look-Ahead Adders (CLAs) make it suitable for certain applications. Below are some key applications:

- Medium-Sized Arithmetic Units:

CBAs are well-suited for medium-sized arithmetic circuits (e.g., 16-bit or 32-bit adders), such as those found in: Microcontrollers Embedded systems In these applications, CBAs offer a good trade-off between speed and hardware complexity, making them effective for arithmetic operations without consuming too much power or silicon area.

- General-Purpose Processors:

ALUs (Arithmetic Logic Units) in general-purpose processors can use CBAs for addition and subtraction operations. Processors with moderate performance requirements (e.g., consumer-grade CPUs or embedded processors) can benefit from the speed improvements provided by CBAs without the complexity and power cost of faster adders like CLAs or Carry Select Adders (CSAs).

- Signal Processing Units:

In digital signal processing (DSP) applications, such as audio and video processing, speed is critical, but the design needs to remain relatively simple to reduce power consumption and cost. CBAs are ideal in DSP blocks where moderate-sized additions (such as 16-bit or 32-bit operations) are frequently performed, striking a balance between performance and efficiency.

- Floating-Point Arithmetic Units:

Floating-point units (FPUs) that perform arithmetic operations on large numbers often require fast adders for the mantissa addition part. For certain precision levels (like single-precision floating-point), CBAs can be used in FPUs to provide an efficient addition mechanism that ensures adequate speed without the complexity overhead of more advanced adders.

- Low-Power and Low-Cost Systems:

CBAs are ideal for applications where power consumption and cost are important constraints, but faster operation than a Ripple Carry Adder is still desired. Examples include battery-operated devices, wearables, and IoT (Internet of Things) devices, where the CBA offers better performance while keeping power consumption low.

- ASIC (Application-Specific Integrated Circuits):

ASICs, which are tailored for specific applications, often incorporate CBAs in arithmetic circuits when moderate speed improvements are required without significantly increasing the design complexity or area. They are used in applications like networking chips, encryption/decryption hardware, and signal processing units.

- Multipliers:

CBAs are sometimes used in the partial sum adders within multipliers. Multiplication involves multiple addition steps (e.g., in array multipliers), and using CBAs in these steps can improve the overall speed while keeping the design simpler compared to using CLAs.

- Data Path Design in FPGAs:

In FPGA (Field-Programmable Gate Array) designs, where flexibility and power consumption are key concerns, CBAs can be used in data path units to perform arithmetic operations efficiently. They can be part of custom arithmetic units or integer processing blocks in FPGAs, providing a balance between fast operation and resource utilization.

4 Results

4.1 Simulation Results

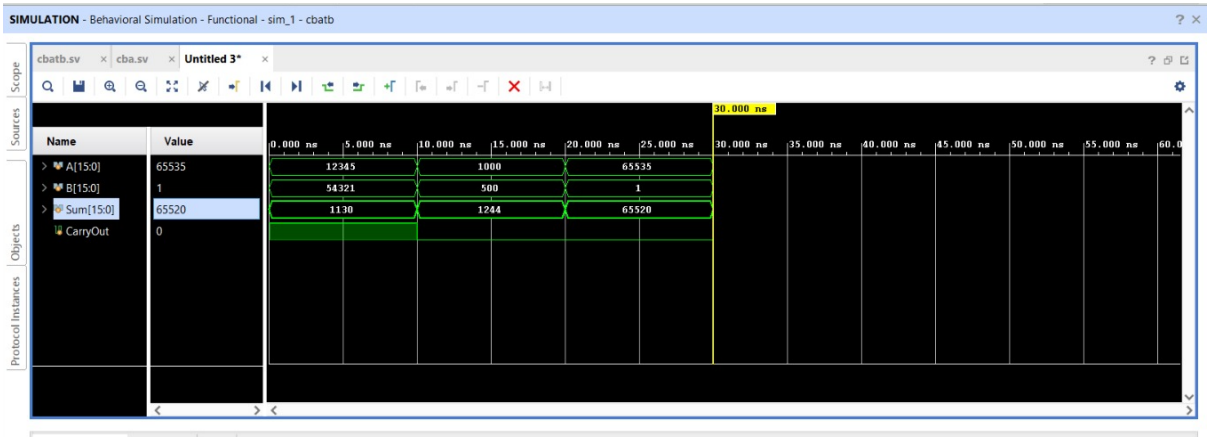

Figure 1: Simulation results of Carry Bypass adder

4.2 Schematic

4.3 Synthesis Design

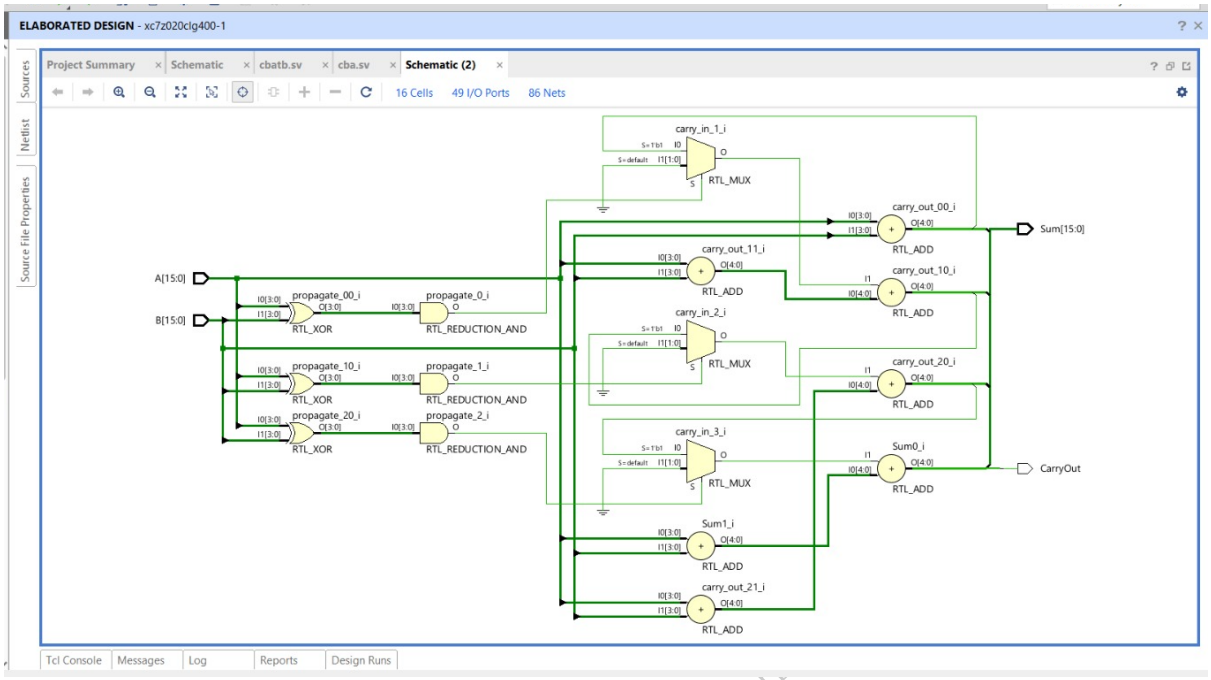

Figure 2: Schematic of Carry Bypass Adder

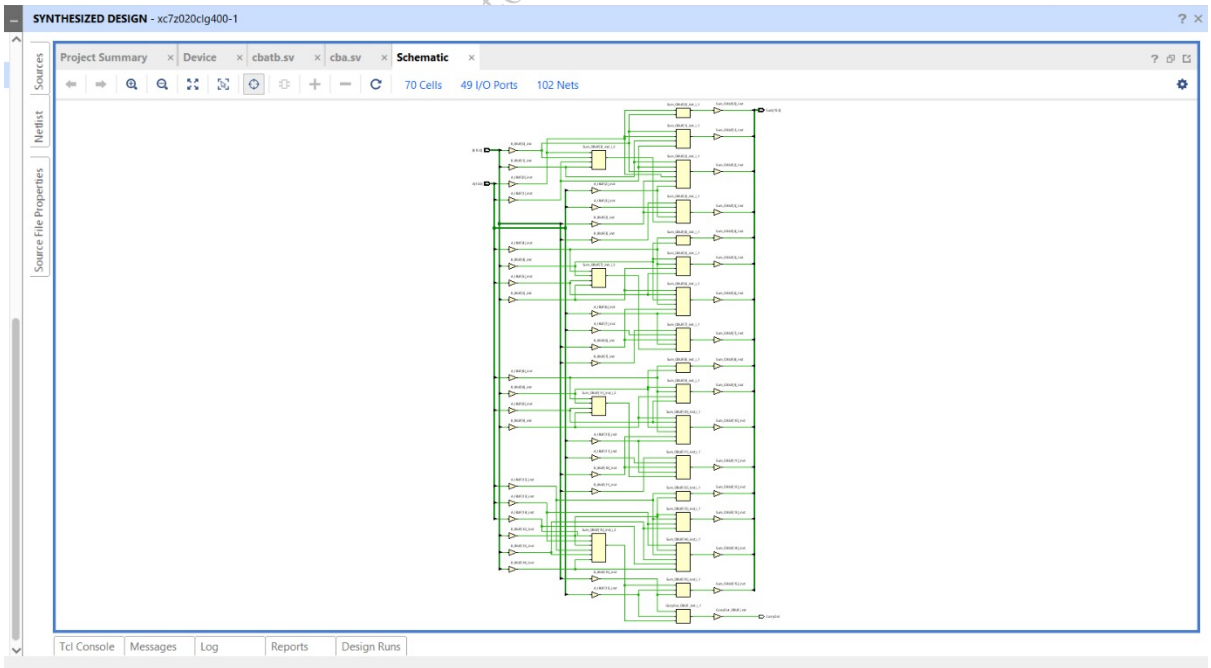

Figure 3: Synthesis Design of Carry Bypass Adder