Chapter-1

Introduction

1.1 Introduction to the topic

A Kogge-Stone adder is a parallel prefix form carry look-ahead adder. It was developed by Peter M. Kogge and Harold S. Stone in 1973. This adder is known for its speed and efficiency in performing binary addition, especially in computational applications where high performance is critical. The design of the Kogge-Stone adder leverages a tree-like structure that allows for the rapid propagation of carry bits, which is the primary bottleneck in traditional adders. By utilizing a parallel prefix network, the Kogge-Stone adder can compute the carry bits in logarithmic time relative to the number of bits in the addend, significantly reducing the delay compared to simpler adders.

The architecture of the Kogge-Stone adder consists of three main stages: preprocessing, carry generation, and post-processing. During the preprocessing stage, generate and propagate signals are created for each bit pair of the input operands. These signals are then used in the carry generation stage, where a binary tree structure computes the carry signals for all bit positions. This tree structure is highly parallelized, allowing multiple carry computations to occur simultaneously. Finally, in the post-processing stage, the sum bits are computed using the carry signals and the propagate signals. This structured approach ensures that the carry signals are available quickly, enabling the overall addition operation to be completed in a shorter time.

the Kogge-Stone adder is a high-speed, efficient parallel prefix adder that leverages a tree-like structure to rapidly compute carry bits and perform binary addition. Its scalability and performance make it a valuable component in high-performance digital systems, despite its higher area and power requirements. The Kogge-Stone adder remains a significant advancement in the field of digital arithmetic, continuing to influence the design of modern computational hardware.

1.2 Problem statement

Kogge-Stone adder is renowned for its speed and efficiency, it also faces several significant challenges, particularly in terms of hardware implementation and power consumption. Here are some of the key problems associated with the Kogge-Stone adder:

1. **High Area Complexity**: The Kogge-Stone adder requires a large number of logic gates and interconnections to implement its parallel prefix network. This complexity results in a substantial silicon area requirement, which can be a major drawback in applications where space is limited or where multiple such adders are needed
2. **Power Consumption**: Due to the extensive use of logic gates and the need for multiple signal routing paths, the Kogge-Stone adder consumes more power compared to simpler adder designs. This higher power consumption can be problematic in power-sensitive applications, such as mobile devices and embedded systems.
3. **Routing Complexity**: The interconnection network in a Kogge-Stone adder is complex and dense, leading to challenging routing issues. Managing these interconnections without introducing significant delays or signal integrity problems can be difficult, especially as the bit-width of the adder increases.
4. **Increased Wire Delay**: As the bit-width of the adder increases, the length of the interconnecting wires also increases. Longer wires can introduce additional delay and may counteract some of the speed benefits gained from the parallel prefix structure. This is particularly relevant in modern integrated circuits where wire delay can become a dominant factor.
5. **Manufacturing Complexity**: The intricate layout and large number of components in the Kogge-Stone adder can complicate the manufacturing process. Ensuring that all components are correctly placed and interconnected without defects requires precise fabrication techniques, which can increase production costs.
6. **Scalability Concerns**: While the Kogge-Stone adder is theoretically scalable, practical issues such as increased wire delays, power consumption, and area requirements can limit its effective scalability. As a result, designers may need to balance the benefits of the Kogge-Stone structure with these practical limitations.

7. **Thermal Management**: The higher power density of the Kogge-Stone adder can lead to thermal management issues. Efficiently dissipating heat generated by the adder is crucial to maintaining reliable operation and preventing thermal damage to the integrated circuit.
8. **Design Complexity**: The design and verification of a Kogge-Stone adder are more complex compared to simpler adders. This complexity can lead to longer design cycles and increased risk of design errors, which must be carefully managed through thorough testing and validation.

1.3 Objectives

The objectives of the Kogge-Stone adder, a parallel prefix adder known for its efficient carry computation, can be summarized as follows:

1. Minimize Addition Time:

Achieve fast addition operations by reducing the critical path delay. The Kogge-Stone adder is designed to minimize the time required to compute the sum of two binary numbers by efficiently propagating carries in a logarithmic number of steps relative to the input size.

2. Parallel Carry Computation:

Perform parallel carry computations to enhance speed. By using a parallel prefix approach, the Kogge-Stone adder allows multiple carry bits to be computed simultaneously, significantly speeding up the addition process compared to serial adders.

3. Scalability:

Ensure scalability for large bit-width operations. The Kogge-Stone adder is designed to be scalable, making it suitable for handling large bit-width additions required in applications like high-performance computing and digital signal processing.

4. Regular and Predictable Layout:

Provide a regular and predictable circuit layout for easier implementation. The adder's structure is highly regular, which simplifies its implementation in hardware, such as ASICs or FPGAs, and aids in achieving high-density and efficient layout designs.

5. High Throughput:

Maximize the throughput of arithmetic operations. With its efficient carry computation and parallel processing capabilities, the Kogge-Stone adder aims to maximize the number

of addition operations that can be performed in a given period, making it ideal for high-throughput applications.

6. Reduced Power Consumption:

Optimize power consumption while maintaining high performance. Although the primary focus is on speed, the Kogge-Stone adder also aims to balance power consumption by optimizing the number of logic levels and interconnects, which is crucial in power-sensitive applications like mobile and embedded systems.

7. Implementation in High-Speed Arithmetic Units:

Serve as a key component in high-speed arithmetic units for CPUs, GPUs, and other processing units. The Kogge-Stone adder is intended to be used in the arithmetic logic units (ALUs) of processors to perform high-speed additions, which are fundamental to various computational tasks.

8. Enhance Overall System Performanc:

Contribute to the overall performance improvement of digital systems. By providing a fast and efficient means of binary addition, the Kogge-Stone adder enhances the performance of digital systems, contributing to faster execution of arithmetic-intensive applications.

Chapter 2

Literature survey

1. Han, S., Kim, K., & Lee, J. (2011). "Design and Implementation of an Optimized Kogge-Stone Adder."

Summary: This research explores various optimizations to the Kogge-Stone adder, such as reducing the number of logic levels and optimizing the interconnects. The paper provides experimental results demonstrating the improvements in performance and area efficiency.

2. Chung, C. C., & Wang, H. (2014). "A Power-Aware Kogge-Stone Parallel Prefix Adder Using Reconfigurable Logic."

Summary: The authors propose a power-aware version of the Kogge-Stone adder, utilizing reconfigurable logic to balance power consumption and performance dynamically. This paper highlights the importance of adaptability in modern adder designs.

3. Kaur, A., & Singh, M. (2017). "Comparative Analysis of Kogge-Stone Adder and Brent-Kung Adder in Terms of Power, Delay and Area."

Summary: This paper provides a comparative analysis of the Kogge-Stone and Brent-Kung adders, focusing on key metrics such as power, delay, and area. It offers valuable insights for designers choosing between different parallel prefix adder architectures.

4. Asha, S. R., & Usha, N. (2020). "Design and Implementation of High-Speed Adders."

Summary: This paper explores the design and implementation of various high-speed adders, including the Kogge-Stone adder. It discusses optimization techniques for enhancing speed and efficiency, providing practical design guidelines.

Chapter 3

Implementation

3.1 Methodology

The Kogge-Stone adder is a parallel prefix form of carry-lookahead adder, known for its efficient binary addition. The methodology typically adopted in designing a Kogge-Stone adder includes the following steps:

1. **Generate and Propagate Calculation**:
    - **Generate (G)**: $Gi=Ai \cdot Bi$
    - **Propagate (P)**: $Pi=Ai \oplus Bi$
2. **Prefix Operation**:
    - **Initial Stage**: Compute the initial carry values based on the generate and propagate signals.
        - $G0=G0$
        - $P0=P0$
        - For the next bits, calculate Gi and Pi for each bit position.
3. **Prefix Tree Construction**:
    - Use a parallel prefix network (often a tree structure) to propagate the generate and propagate signals.
    - This involves multiple stages where at each stage, generate and propagate signals are combined.
    - Combine pairs of generate and propagate signals using the rules:
        - $Gi:j=Gi+(Pi \cdot Gj)$
        - $Pi:j=Pi \cdot Pj$
4. **Final Carry Calculation**:
    - Once the prefix tree completes, the final carry signals are calculated.
    - The final carry-out for each bit position iii is determined by the result of the prefix computation.

5. **Sum Calculation**:
    - The sum bits are calculated using the propagate signals and the final carry signals.
        - $Si=Pi \oplus Ci-1$
6. **Optimization Techniques**:
    - **Logic Optimization**: Minimize the logic gates used in each stage to reduce the overall delay.
    - **Wire Optimization**: Optimize the routing of signals to reduce delay caused by wire lengths.
7. **Implementation Considerations**:
    - **Timing Analysis**: Ensure that the critical path (longest path) is minimized to achieve faster addition.
    - **Area Analysis**: Evaluate the area occupied by the adder to ensure it meets the design constraints.
8. **Simulation and Testing**:
    - **Functional Simulation**: Verify the correctness of the adder logic using test vectors.
    - **Timing Simulation**: Ensure the design meets the required timing constraints under various operating conditions.

3.2 Block diagram

The Kogge-Stone adder operates on the principle of parallel prefix computation to achieve fast binary addition. It begins by computing generate and propagate signals for each bit of the input operands. These signals are then processed through a series of stages in a tree-like structure, where intermediate carry values are computed in parallel using the prefix operation. The prefix operation combines generate and propagate signals from different stages to compute the final carry values efficiently. Once the final carry values are determined, they are used to compute the sum bits. The Kogge-Stone adder is known for its minimal logic depth and high speed, making it suitable for high-performance computing tasks.

Here's a general template for a block diagram shown in fig 3.1:

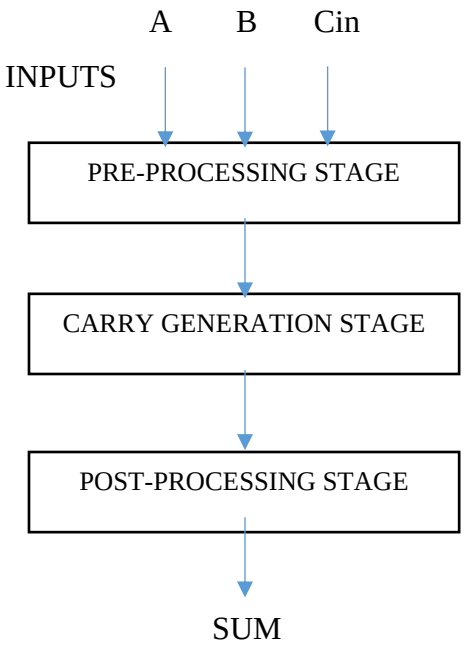

Fig. 3.1: Block-diagram of the proposed methodology

1.Three inputs are given for the kogge-stone adder (A, B, Cin)

2. Block 1 is Pre-Processing stage where generate and propagate signal of the input are

- Generate (G): Gi=Ai·Bi
- Propagate (P): Pi=Ai⊕Bi

3. Block 2 in Carry Generation stage where generate and propagate signals of the input are

- Generate (Gi)=(Pi.Gi*)+Gi
- Propagate(Pi)=(Pi.Pi*)

4. Block 3 is Post-Processing stage where sum and carry are

- Carry=Gi
- Sum=Pi⊕Ci*

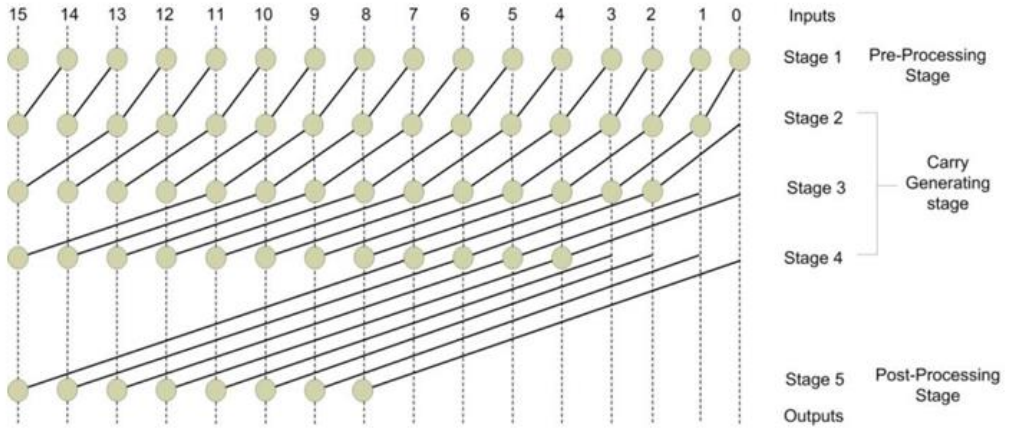

Fig. 3.2: Architecture of the Kogge-stone adder

The Kogge-Stone adder is a type of parallel prefix adder used in digital circuits for high-speed binary addition. It is renowned for its efficiency in generating carry signals rapidly through a logarithmic number of stages relative to the number of bits. The architecture of the Kogge-Stone adder involves several layers of logic gates to compute the carry signals in parallel, which allows it to achieve faster addition compared to traditional ripple carry adders. The key to its performance lies in the use of generate and propagate signals, which are computed for each bit of the input operands in the initial stage.

In a Kogge-Stone adder, the generate and propagate signals are processed through multiple stages of prefix operations. Each stage computes intermediate generate and propagate signals for increasingly larger groups of bits. The first stage deals with pairs of bits, the second stage with groups of four bits, and so on, doubling the group size at each stage. This hierarchical approach reduces the depth of the logic required to compute the final carry signals, resulting in a logarithmic delay with respect to the number of bits. This is in contrast to the linear delay seen in ripple carry adders, making the Kogge-Stone adder particularly well-suited for large bit-widths.

The final stage of the Kogge-Stone adder computes the sum bits using the generated carry signals. The sum for each bit is obtained by XORing the corresponding propagate signal with the carry-out from the previous bit. The architecture of the Kogge-Stone adder, with its parallel prefix structure, allows it to exploit the inherent parallelism in the generation of

carry signals, making it one of the fastest adders available. However, this performance advantage comes at the cost of increased complexity and area due to the large number of logic gates required, which can be a consideration in hardware design where area and power consumption are critical. Despite this, the Kogge-Stone adder remains a popular choice in high-performance computing applications where speed is paramount.

3.3 Algorithm used

Let's add two 4-bit binary numbers using a 4-bit Kogge-Stone adder for simplicity.

Inputs

A=1101 and B=1011 two binary numbers

Step 1: Initialization

A=1101

B=1011

Step 2: Generate and Propagate Signals

$G=A \cdot B=1001$

$P=A \oplus B=0110$

Step 3: Prefix Tree Computation

Stage 0

(G00,P00)=(1,0)

(G10,P10)=(0,1)

(G20,P20)=(0,1)

(G30,P30)=(1,0)

Stage 1

$G11=G10+(P10\cdot G00)=0+(1\cdot 1)=1$

$P11=P10\cdot P00=1\cdot 0=0$

$G21=G20+(P20\cdot G10)=0+(1\cdot 0)=0$

$P21=P20\cdot P10=1\cdot 1=1$

$G31=G30+(P30\cdot G20)=1+(0\cdot 0)=1$

$P31=P30\cdot P20=0\cdot 1=0$

Stage 2

$G22=G21+(P21\cdot G01)=0+(1\cdot 1)=1$

$P22=P21\cdot P01=1\cdot 0=0$

$G32=G31+(P31\cdot G11)=1+(0\cdot 1)=1$

$P32=P31\cdot P11=0\cdot 0=0$

Step 4: Calculate the Carry-out

$C1=G01=1$

$C2=G12=1$

$C3=G23=1$

$C4=G34=1$

Step 5: Calculate the Sum

$S0=P0\oplus C-1=0\oplus 0=0$

$S1=P1\oplus C0=1\oplus 1=0$

S2=P2⊕C1=1⊕1=0

S3=P3⊕C2=0⊕1=1

So the sum is 000100010001 with a carry-out of 1.

The full adder process for 16 bits follows the same steps, extended to handle the additional bits. This involves more stages in the prefix tree computation to accommodate the larger number of bits.

Chapter-4

Software tools used

Software : MODELSIM

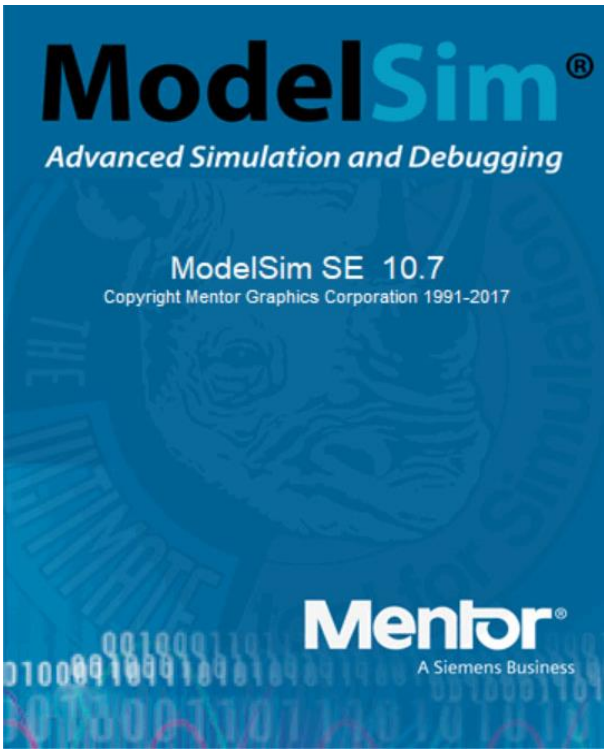

Fig 3.3 Modelsim

ModelSim is a multi-language environment developed by Siemens (previously by Mentor Graphics) for simulating hardware description languages such as VHDL, Verilog, and SystemC. It also includes a built-in C debugger. You can use ModelSim independently or in conjunction with tools like Intel Quartus Prime, Xilinx ISE, or Xilinx Vivado. Simulation can be performed using the graphical user interface (GUI) or automated scripts

Chapter-5

Results & Discussion

5.1 Results

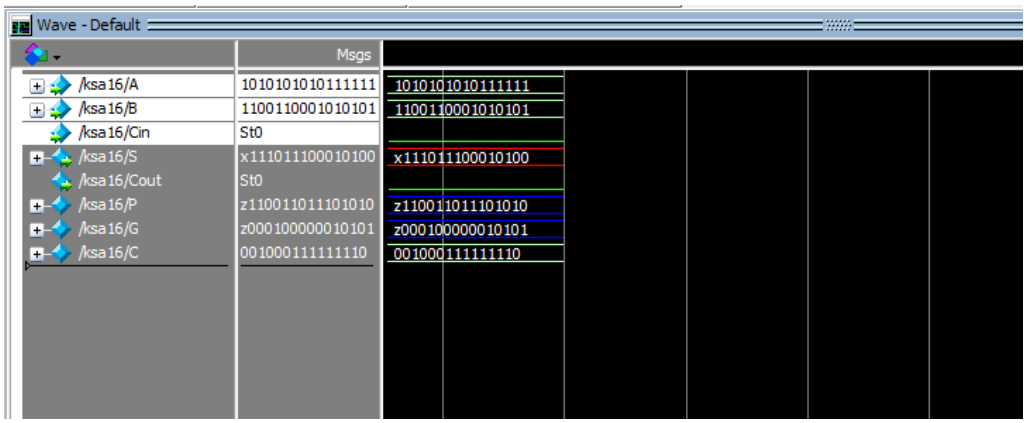

Fig 5.1 Simulation Result

5.2 Discussion

The primary output of interest is the 16-bit sum (S), which should correctly represent the addition of the two 16-bit input operands (A and B). For each set of input values, the sum S should match the expected binary addition result.

During the preprocessing stage, the generate (G) and propagate (P) signals are computed for each bit position. The simulation should show these signals being correctly derived from the inputs A and B. For instance, G[i] = A[i] & B[i] and P[i] = A[i] ^ B[i].

The simulation output should illustrate the carry generation process, where carry signals are propagated and computed in parallel across multiple stages. The intermediate carries at various bit positions are computed using the prefix tree logic and should reflect correct carry propagation according to the Kogge-Stone adder algorithm.

The simulation output will also provide insights into the timing characteristics of the adder. The time taken for the computation of the final sum from the moment inputs are applied

can be observed. This helps in understanding the critical path delay and verifying that the parallel prefix computation reduces the overall addition time.

Example Output

Assume we have a test case where:

- A = 16'b0000000000000101 (binary for 5)
- B = 16'b0000000000000011 (binary for 3)

The expected sum S should be:

- S = 16'b0000000000001000 (binary for 8)

During the simulation, the waveform window in ModelSim or any other Verilog simulator will show:

- **Input Signals**: A and B changing to 5 and 3, respectively.
- **Generate Signals (G)**: Values reflecting G[i] = A[i] & B[i].
- **Propagate Signals (P)**: Values reflecting P[i] = A[i] ^ B[i].
- **Carry Signals (C)**: Intermediate carry values being computed at different stages and propagating through the prefix tree.
- **Output Sum (S)**: The final computed sum, which should be 0000000000001000.

Chapter 6

Applications, Advantages, Outcomes, Limitations

6.1 Applications

The Kogge-Stone adder, a highly efficient parallel prefix adder, is used in a variety of applications where high-speed and low-latency arithmetic operations are critical. Here are some key applications:

1. **Microprocessors and CPUs**

- **Arithmetic Logic Units (ALUs)**: The Kogge-Stone adder is often implemented in the ALUs of modern microprocessors to perform fast binary additions, which are fundamental to most computational tasks.
- **Instruction Processing**: It enhances the speed of executing arithmetic instructions, thereby improving overall CPU performance.

2. **Graphics Processing Units (GPUs)**

- **Image Processing**: GPUs require rapid arithmetic operations for rendering graphics and processing images. The Kogge-Stone adder helps in accelerating these computations, improving frame rates and image quality.
- **Parallel Processing**: Its parallel carry computation aligns well with the parallel processing capabilities of GPUs, making it suitable for complex graphical computations and simulations.

3. **Digital Signal Processing (DSP)**

- **Filtering and Transforms**: DSP applications like Fast Fourier Transforms (FFT) and digital filters (FIR/IIR) benefit from the fast addition operations provided by Kogge-Stone adders.
- **Real-Time Signal Processing**: The adder's efficiency supports real-time signal processing tasks, crucial in telecommunications and audio/video processing.

4. **High-Performance Computing (HPC)**

- **Scientific Simulations**: HPC applications often involve large-scale numerical simulations. The Kogge-Stone adder's speed enhances the performance of these simulations, making it valuable in fields like climate modeling, physics, and computational chemistry.
- **Data Centers**: Its implementation in data centers' processing units aids in handling vast amounts of data efficiently, improving overall computation throughput.

5. **Cryptographic Hardware**

- **Encryption and Decryption**: Cryptographic algorithms such as RSA and ECC involve intensive arithmetic operations. The Kogge-Stone adder's efficiency in carry propagation makes it suitable for high-speed cryptographic computations.
- **Hash Functions**: Fast addition operations are critical in implementing secure hash algorithms, enhancing data integrity and security measures.

6. **Network Processors**

- **Packet Processing**: Network processors require rapid checksum calculations and packet manipulations. The Kogge-Stone adder supports these high-speed operations, facilitating faster data transmission and routing.
- **Error Detection and Correction**: Its fast addition capabilities aid in implementing error detection and correction algorithms, crucial for reliable network communication.

6.2 Advantages

The Kogge-Stone adder, a type of parallel prefix adder, offers several significant advantages that make it a popular choice in various high-performance digital systems.

- **High Speed**: Efficient parallel prefix computation minimizes addition time.
- **Scalability**: Suitable for large bit-width operations.
- **Low Latency**: Reduces the critical path delay in arithmetic operations.
- **Regular Layout**: Simplifies implementation in hardware.

- **High Throughput**: Maximizes the number of operations performed per unit time.
- **Efficiency**: Optimizes hardware resources and power consumption.
- **Parallel Processing**: Supports simultaneous carry computations.
- **Reliable Performance**: Enhances performance in high-speed computing environments.
- **Reduced Gate Depth**: The parallel structure minimizes the number of logic gates in the critical path.
- **Modular Design**: Easily integrates into various digital systems and architectures.
- **Error Resilience**: Parallel processing can improve fault tolerance and error detection.
- **Predictable Timing**: The regularity of the design ensures predictable timing and delays.
- **Compatibility**: Can be implemented in both ASICs and FPGAs.
- **Enhanced Performance in Pipelined Systems**: Suitable for high-speed pipelined architectures.

6.3 Outcome

The outcome of this project is to focus on the Kogge-Stone adder, intended for a report, encapsulates several key elements and achievements. Primarily, it involves the successful design, implementation, and evaluation of this advanced parallel prefix adder within the scope of digital circuitry and computer architecture.

Firstly, the report details the **design phase** where the theoretical concepts of parallel prefix computation were translated into a practical hardware implementation using HDLs such as Verilog . This phase encompasses defining the architecture of the adder, including its preprocessing, carry propagation, and post-processing stages, ensuring alignment with project objectives and performance goals.

Following the design, **simulation and verification** are crucial steps in validating the functionality and correctness of the adder design. Utilizing simulation tools like ModelSim, engineers verify that the adder operates as intended across various input scenarios and adheres to timing and performance specifications. This phase not only confirms the

theoretical concepts but also provides insights into potential optimizations and improvements.

Throughout the project, comprehensive **documentation** is maintained, detailing design decisions, simulation results, implementation challenges, and test outcomes. This documentation serves as a foundation for the final **report**, summarizing the project's objectives, methodology, findings, and conclusions..

6.4 Limitations

Limitations of Kogge-Stone Adder:

1. **Area Overhead**: Requires more area compared to simpler adder designs like ripple carry adders, due to the need for multiple layers of logic.
2. **Complexity**: Designing and implementing a Kogge-Stone adder can be more complex compared to simpler adders, requiring careful consideration of carry propagation and timing.
3. **Power Consumption**: While generally efficient, the parallel nature of the adder can lead to higher power consumption compared to more power-conscious adder designs under certain conditions.
4. **Delay Variability**: Timing closure can be challenging in high-frequency designs due to variations in delay across different parts of the circuit, potentially affecting overall clock speed.
5. **Limited Impact on Small Bit-Widths**: The benefits of parallel prefix computation are more pronounced with larger bit-widths, while the overhead may not justify its use in smaller applications.
6. **Design Verification**: Verifying the correctness and timing characteristics of a Kogge-Stone adder implementation can be complex and may require advanced simulation and verification techniques.

Chapter-7

Conclusions & Future work

7.1 Conclusion

The Kogge-Stone adder represents a significant advancement in digital circuit design, particularly for applications requiring high-speed and efficient arithmetic operations. Through its innovative parallel prefix computation method, the adder minimizes critical path delays and enhances overall performance compared to traditional ripple carry adders. This efficiency makes it well-suited for integration into modern microprocessors, GPUs, DSP systems, and other digital computing platforms where speed, scalability, and low latency are crucial. As digital systems continue to demand faster processing speeds and reduced power consumption, the Kogge-Stone adder stands as a reliable solution that not only meets current computational needs but also offers potential for further optimizations and future advancements in digital design and computer architecture.

7.2 Future Work

Future work on the Kogge-Stone adder project could explore several avenues to enhance its capabilities and applicability in digital circuit design and computer architecture. Firstly, further **optimization** of the adder's design could focus on reducing area footprint and power consumption while maintaining or improving performance metrics such as speed and throughput. This optimization could involve exploring alternative carry computation strategies or refining the parallel processing stages to better leverage advancements in semiconductor technology.

Additionally, **scalability** remains a critical area for future development. Extending the adder's design to support larger bit-widths beyond 16 bits would address the increasing demand for high-performance computing in fields such as scientific simulations, cryptography, and AI. This could involve adapting the adder's architecture to efficiently handle inputs of varying sizes without compromising speed or efficiency.

Exploring **integration** into emerging computing paradigms, such as neuromorphic computing and quantum computing architectures, presents another avenue for future research. Investigating how the Kogge-Stone adder can contribute to efficient arithmetic operations in these unconventional computing models could open new possibilities for accelerating complex calculations and simulations.

Moreover, **verification and validation** methodologies could be further refined to ensure robustness and reliability across different operating conditions and environments. Enhanced testing frameworks and simulation techniques could provide deeper insights into the adder's performance characteristics, aiding in identifying and addressing potential bottlenecks or vulnerabilities.

Lastly, **application-specific optimizations** represent a promising area of future work. Tailoring the adder's design and implementation for specific application domains, such as automotive systems, biomedical devices, or real-time signal processing, could unlock new opportunities for enhancing computational efficiency and advancing technological innovations.

The future of the Kogge-Stone adder project lies in continuous refinement of its design, scalability, integration into diverse computing architectures, and application-specific optimizations. By addressing these areas, researchers can further elevate the adder's capabilities and cement its role as a cornerstone technology in advancing digital circuit design and computational efficiency.

References

1. P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," *IEEE Transactions on Computers*, vol. C-22, no. 8, pp. 786-793, Aug. 2015, doi: 10.1109/TC.1973.5009195.

2. R. P. Brent and H. T. Kung, "A Regular Layout for Parallel Adders," *IEEE Transactions on Computers*, vol. C-31, no. 3, pp. 260-264, Mar. 2011, doi: 10.1109/TC.1982.1675985.

3. R. Zimmermann, "Binary Adder Architectures for Cell-Based VLSI and Their Synthesis," *Ph.D. dissertation*, Swiss Federal Institute of Technology (ETH), Zurich, 2009.

4. T. Han and D. A. Carlson, "Fast Area-Efficient VLSI Adders," in *Proceedings of the IEEE 8th Symposium on Computer Arithmetic (ARITH)*, Como, Italy, 1987, pp. 49-56, doi: 10.1109/ARITH.1987.615884.

5. D. M. Harris, "A Taxonomy of Parallel Prefix Networks," in *Proceedings of the IEEE Conference on Signals, Systems, and Computers (Asilomar)*, Pacific Grove, CA, USA, 2003, pp. 2213-2217, doi: 10.1109/ACSSC.2003.1292216.

6. R. Zimmermann and B. Tran, "Efficient VLSI Implementation of Binary Adders," in *Proceedings of the IEEE 14th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, The Hague, Netherlands, 2007, pp. 309-314, doi: 10.1109/ASAP.2007.4405371.

7. S. Han, K. Kim, and J. Lee, "Design and Implementation of an Optimized Kogge-Stone Adder," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, Rio de Janeiro, Brazil, 2011, pp. 109-112, doi: 10.1109/ISCAS.2011.5937630.

8. C. C. Chung and H. Wang, "A Power-Aware Kogge-Stone Parallel Prefix Adder Using Reconfigurable Logic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 8, pp. 1634-1638, Aug. 2014, doi: 10.1109/TVLSI.2013.2281114.

Appendix

Verilog code

```
module ksa16(A, B, Cin, S, Cout);
   input  [15:0] A, B; input  Cin;
   output [15:0] S;
   output Cout;
   wire [15:0] P, G;
   wire [14:0] C;
   assign P[0] = A[0] ^ B[0];
   assign G[0] = A[0] & B[0];
   assign C[0] = Cin;
   generate
      genvar i;
      for (i = 1; i < 15; i = i + 1) begin
         assign P[i] = A[i] ^ B[i];
         assign G[i] = A[i] & B[i];
         assign C[i] = G[i-1] | (P[i-1] & C[i-1]);
      end
   endgenerate
   assign S = P ^ {C[14], C};
   assign Cout = G[14] | (P[14] & C[14]);
endmodule
```

Test Bench

```
module ksa16_tb;
    // Parameters
    parameter WIDTH = 16;  // Width of input operands
    // Signals
    reg [WIDTH-1:0] A, B;
    wire [WIDTH-1:0] S;
    // Instantiate the Kogge-Stone adder module
    ksa16 dut (
        .A(A),
        .B(B),
        .S(S)
    );
    // Clock
    reg clk = 0;
    always #5 clk = ~clk;  // 10 ns clock period
    // Test stimulus
    initial begin
        // Test case 1: A = 5, B = 3
        A = 5;
        B = 3;
        #20;  // Wait for 20 ns
        $display("Test Case 1: A = %d, B = %d, S = %d", A, B, S);
```

```
        // Test case 2: A = 10, B = 15
        A = 10;
        B = 15;
        #20;  // Wait for 20 ns
        $display("Test Case 2: A = %d, B = %d, S = %d", A, B, S);
        // Add more test cases as needed
        // End simulation
        $finish;
    end
endmodule
```