Project 58: Pipeline Divider

A Comprehensive Study of Advanced Digital Circuits

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

Documentation Specialist: Dhruv Patel & Nandini Maheshwari

Contents

1 Project Overview

A pipeline divider is a type of digital circuit designed to perform division operations on binary numbers efficiently, using a pipelining technique. Pipelining is a method used in digital circuits to improve performance by allowing multiple operations to be performed simultaneously, each at a different stage of completion. This approach increases the throughput of the system while potentially increasing the latency of individual operations.

2 Pipeline Divider

2.1 Key Features of Pipeline Divider

Pipelining Technique:

- In a pipeline divider, the division operation is broken down into several stages, allowing different parts of the operation to be processed simultaneously. For example, in a typical implementation, the division process might be divided into stages like estimation, subtraction, and shifting.
- Each stage of the pipeline completes a portion of the overall division operation, passing its results to the next stage. This allows the divider to start processing a new division request before the previous one has completed, improving overall throughput.

Parallel Processing:

- By dividing the division operation into smaller, more manageable parts, multiple divisions can be processed in parallel. This parallelism is essential for high-speed computations, particularly in applications that require frequent division operations.

Reduction of Critical Path:

- Pipelining can help reduce the critical path delay in the division operation, which is often one of the slowest processes in digital arithmetic. By breaking the division into stages, the time taken for each stage can be shorter, allowing the system to operate at higher clock frequencies.

2.2 Basic operation of Pipeline Divider

Initialization:
The inputs (dividend and divisor) are loaded into the pipeline. This may involve preliminary computations or setup for the division.

Estimation:
The divider estimates the initial quotient based on the leading bits of the dividend and divisor. This could involve some form of approximation to determine how many times the divisor can fit into the current portion of the dividend.

Subtraction:
The estimated quotient is then used to subtract an appropriate multiple of the divisor from the dividend. This step reduces the dividend iteratively.

Shift and Repeat:
After subtraction, the divider may shift the dividend (and possibly the quotient) left or right depending on the division algorithm used. The process is repeated for the remaining bits of the dividend until the entire dividend has been processed.

Finalization:
Once all bits have been processed, the final quotient and remainder are produced. The output is then made available to the next stage of computation or to the user.

2.3 RTL Code

Listing 1: Pipeline Divider

```
1
2  module PipelineDivider #(parameter WIDTH = 8, STAGES = 3) (
3      input logic clk, reset,
4      input logic [WIDTH-1:0] dividend, divisor,
5      output logic [WIDTH-1:0] quotient, remainder,
6      output logic valid
7  );
8      logic [WIDTH-1:0] div_reg [0:STAGES-1], quo_reg [0:STAGES-1],
           rem_reg [0:STAGES-1];
9      logic valid_reg [0:STAGES-1];
10
11     always_ff @(posedge clk or posedge reset) begin
12         if (reset) valid_reg <= '{default: 0}; else begin
13             div_reg[0] <= dividend; valid_reg[0] <= (divisor != 0);
14             for (int i = 1; i < STAGES; i++) begin
15                 div_reg[i] <= div_reg[i-1];
16                 quo_reg[i] <= div_reg[i-1] / divisor;
17                 rem_reg[i] <= div_reg[i-1] % divisor;
18                 valid_reg[i] <= valid_reg[i-1];
19             end
20             quotient <= quo_reg[STAGES-1];
21             remainder <= rem_reg[STAGES-1];
22             valid <= valid_reg[STAGES-1];
23         end
24     end
25 endmodule
```

2.4 Testbench

Listing 2: Pipeline Divider

```
1
2  module PipelineDivider_tb;
3      parameter WIDTH = 8, STAGES = 3;
4      logic clk = 0, reset;
5      logic [WIDTH-1:0] dividend, divisor, quotient, remainder;
6      logic valid;
7
8      PipelineDivider #(.WIDTH(WIDTH), .STAGES(STAGES)) uut (.clk(clk),
           .reset(reset), .dividend(dividend),.divisor(divisor),
           .quotient(quotient),.remainder(remainder), .valid(valid));
9
10
11     always #5 clk = ~clk; // Clock generation
12
13     initial begin
14         reset = 1; #10 reset = 0;
15         dividend = 15; divisor = 4; #20;
16         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
17
18         dividend = 20; divisor = 5; #20;
19         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
```

```
20
21          dividend = 10; divisor = 0; #20;
22          $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
23          $finish;
24      end
25  endmodule
```

3 Results

3.1 Simulation

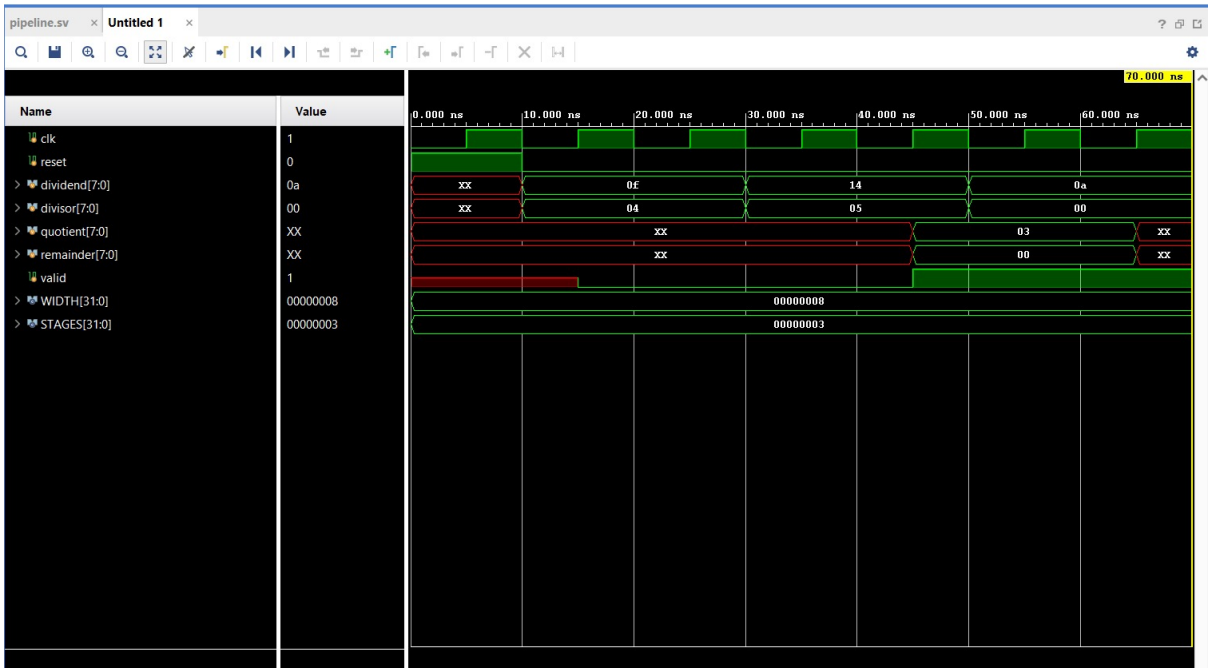

Figure 1: Simulation of Pipeline Divider

3.2 Schematic

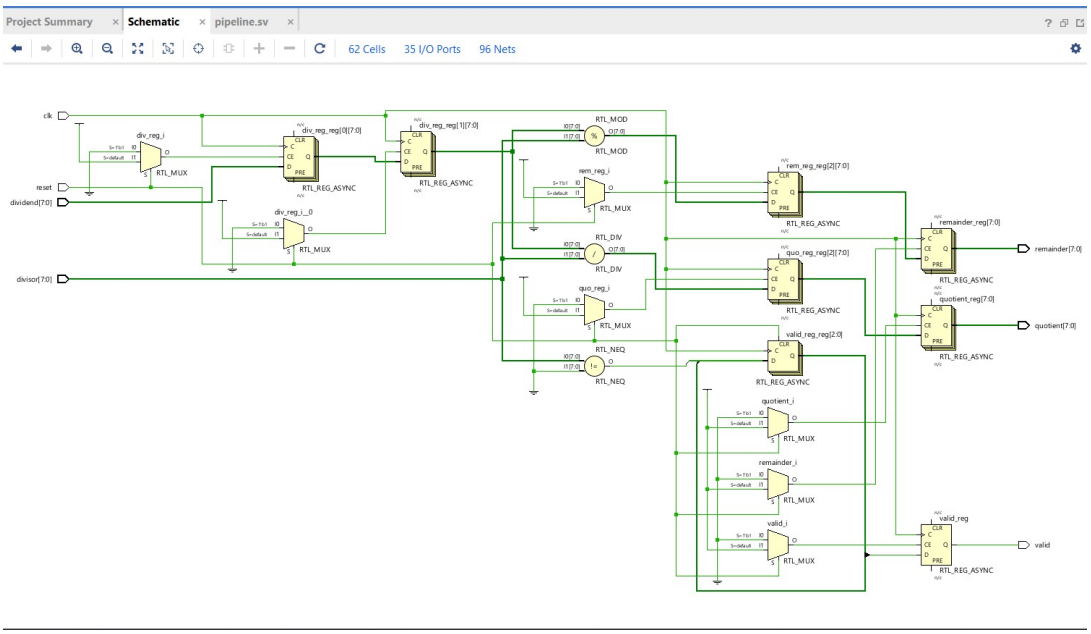

Figure 2: Schematic of Pipeline Divider

3.3 Synthesis Design

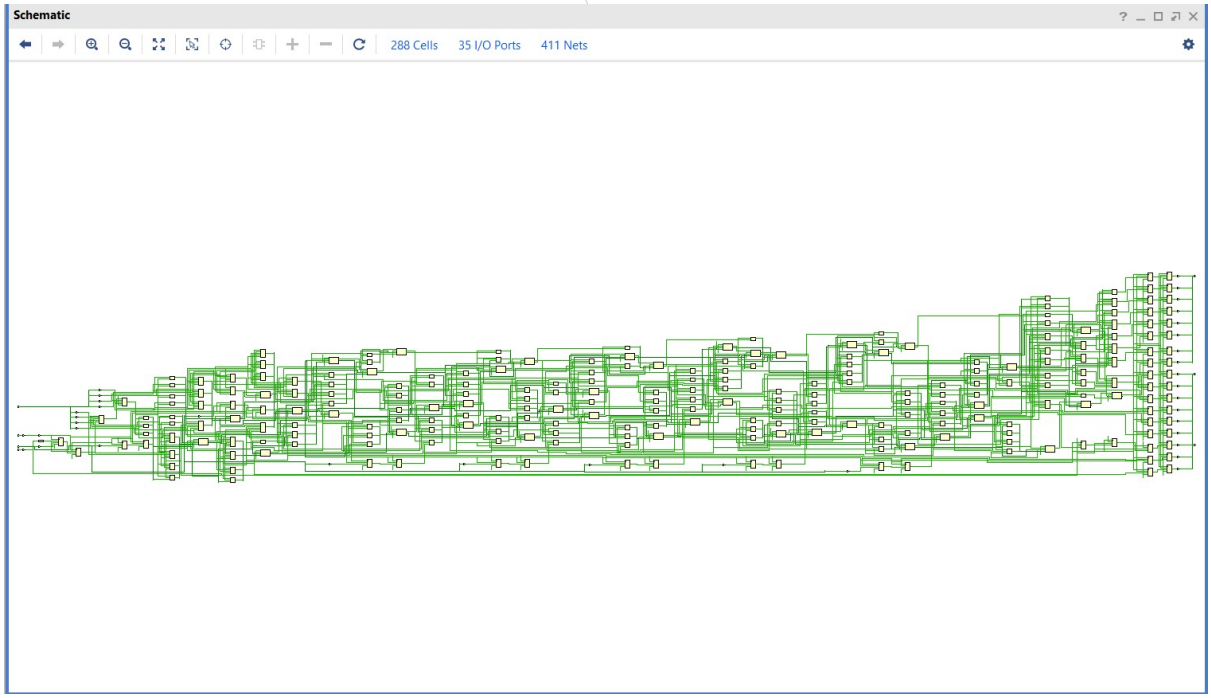

Figure 3: Synthesis Design of Pipeline Divider

4 Advantages of Dynamic Divider

Increased Throughput:
Multiple division operations can be processed simultaneously due to the pipelining technique, leading to higher overall throughput.

Reduced Latency for Multiple Operations:
While individual operations may take longer, the system can start new division tasks before previous ones complete, reducing the effective latency for multiple operations.

Improved Resource Utilization:
Different stages of the divider can utilize hardware resources more effectively, allowing for better overall performance and efficiency.

Higher Clock Speeds:
By dividing the operation into smaller stages, the critical path delay is reduced, enabling the circuit to operate at higher clock frequencies.

Flexibility:
The pipelining approach allows for easy adjustments to the number of stages based on specific performance and resource requirements.

5 Disadvantages of Dynamic Divider

Increased Design Complexity:
Pipelined designs can be more complex than non-pipelined counterparts, requiring careful management of timing and data dependencies between stages.

Latency for Individual Operations:
The time taken for a single division operation may increase due to the overhead of managing multiple pipeline stages.

Data Hazards:
Pipelined dividers can face data hazards, such as read-after-write or write-after-read issues, which may require additional control logic to handle.

Resource Overhead:
More registers and additional control circuitry are needed to manage the stages, which can increase area and power consumption.

Latency in Completion:
If one stage takes longer than expected, it can stall the entire pipeline, leading to inefficient processing and potential bottlenecks.

6 Applications of Dynamic Divider

Digital Signal Processing (DSP):
Used in applications like filtering, modulation, and transformation, where division is frequently required for algorithm computations.

Microprocessors and CPUs:

Modern processors incorporate pipelined dividers to enhance arithmetic performance and speed up complex calculations.

Graphics Processing Units (GPUs):
GPUs utilize pipeline dividers for rendering images and performing real-time graphics transformations that require high-speed arithmetic.

Communications:
In digital communication systems, pipeline dividers are used in algorithms for error correction, modulation, and demodulation.

Embedded Systems:
Used in various embedded applications where efficient division is required, such as control systems and signal processing units.

Scientific Computing:
High-performance computing applications often require rapid division capabilities for simulations and calculations, making pipeline dividers ideal for these environments.

7 Conclusion

Pipeline dividers represent a significant advancement in digital circuit design, enabling fast and efficient division operations essential for high-performance computing applications. Their ability to parallelize operations through pipelining makes them a crucial component in modern digital systems.

8 FAQs

1. What is a pipeline divider?
A pipeline divider is a digital circuit that performs division operations by breaking the process into multiple stages, allowing for simultaneous processing of multiple divisions and improving throughput.

2. Why is pipelining used in dividers?
Pipelining increases throughput by enabling multiple division operations to overlap, thereby speeding up the overall division process in high-performance systems.

3. How does a pipeline divider improve performance?
By dividing the operation into stages and allowing multiple operations to run concurrently, a pipeline divider reduces the effective time per division operation, leading to higher throughput.

4. What are the key stages in a pipeline divider?
The main stages are input loading, estimation, subtraction, shifting, and finalization, with each stage handling part of the division operation sequentially.

5. What is the main advantage of using a pipeline divider?
The main advantage is increased throughput, as it allows multiple division operations to be processed simultaneously, making it ideal for high-speed applications.

6. What are some disadvantages of pipeline dividers?
Pipeline dividers can be complex to design, have increased latency for single operations, and may face data hazards that require additional control logic.

7. What is meant by "critical path" in pipeline dividers?
The critical path is the longest delay path in the circuit. Pipelining reduces the critical path length by breaking the division operation into smaller, faster stages.

8. What type of applications benefit from pipeline dividers?

Applications in DSP, microprocessors, GPUs, and communication systems benefit greatly, as they often require fast and repetitive division operations.

9. How does a pipeline divider handle data hazards?

Data hazards are managed using control logic that may stall stages or forward data to ensure accurate computation, preventing incorrect results due to dependencies.

10. What's the difference between pipeline dividers and non-pipelined dividers?

Pipeline dividers can process multiple divisions in parallel, while non-pipelined dividers complete one operation at a time, making them slower in high-throughput scenarios.