Project 59: Adaptive 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 3

2 Adaptive Divider 3
- 2.1 Key Features of Adaptive Divider . . . 3
- 2.2 Working of Adaptive Divider . . . 3
- 2.3 RTL Code . . . 4
- 2.4 Testbench . . . 5

3 Results 6
- 3.1 Simulation . . . 6
- 3.2 Schematic . . . 6
- 3.3 Synthesis Design . . . 7

4 Advantages of Adaptive Divider 7

5 Disadvantages of Adaptive Divider 7

6 Applications of Adaptive Divider 8

7 Conclusion 8

8 FAQs 8

1 Project Overview

An Adaptive Divider is a division circuit in digital design that adjusts its calculation process dynamically to improve performance and efficiency. Unlike traditional fixed-structure dividers, adaptive dividers can modify their internal behavior based on the inputs or intermediate results, often reducing the number of required computation steps.

2 Adaptive Divider

2.1 Key Features of Adaptive Divider

Control Unit: Manages the adaptive behavior, deciding how many iterations to perform and adjusting the computation path based on intermediate results.

Quotient Prediction Logic: Predicts quotient bits dynamically based on the divisor, dividend, or partial remainder, aiming to minimize the steps required for each division cycle.

Partial Remainder Calculator: Computes partial remainders at each step, providing feedback for the control unit to adjust further operations.

Lookup Table (if using SRT or similar methods): Stores precomputed values to expedite the selection of quotient digits, enabling faster decision-making.

Error Handling and Approximation Logic: Ensures precision by handling potential errors in approximate calculations, balancing between speed and accuracy.

Termination Control: Determines when the division has reached the desired accuracy or threshold precision, allowing the divider to stop iterating early if the remainder is within an acceptable range.

Dynamic Precision Adjustment: Adjusts the precision of the calculations based on system requirements, trading off between computation speed and result accuracy.

Iteration Counter: Tracks the number of cycles to avoid unnecessary steps, aiding in power and resource management.

Pipeline Stages: For high-speed operations, pipelining stages may be added to process different parts of the division concurrently, enhancing throughput.

Clock Gating/Power Management: Deactivates unused portions of the circuit during certain steps, optimizing power efficiency.

2.2 Working of Adaptive Divider

Initialization:

- The dividend and divisor are loaded into the circuit.
- The control unit initializes variables, including setting the iteration counter and determining initial precision requirements.

Quotient Prediction:

- Based on the dividend and divisor values, the quotient prediction logic estimates an initial quotient digit or a set of quotient bits.
- If using a lookup table, it retrieves values that help to quickly determine these quotient digits.

Partial Remainder Calculation:

- The partial remainder is calculated by subtracting the predicted product (quotient * divisor) from the dividend or the intermediate result.
- This remainder is fed back to the control unit for the next iteration's adjustment.

Adaptive Adjustment:

- The control unit analyzes the partial remainder and dynamically adjusts the calculation path (e.g., refining or skipping steps).
- If the partial remainder is small enough, the control unit might choose a shortcut or approximation for the next steps.

Precision Check and Iteration Control:

- The divider checks if the current precision meets the desired accuracy or if further iterations are required.
- If acceptable accuracy is reached, the division operation can terminate early, saving time.

Approximation or Error Handling:

- If precision can be relaxed (for faster results), the control unit enables approximation logic.
- If higher precision is needed, additional steps or recalculations may be triggered to reduce error.

Termination:

- When the desired quotient precision is achieved, the control unit stops the iteration process.
- The quotient and remainder are finalized and outputted.

Output of Result:

- The final quotient and remainder values are outputted, completing the division process.
- Unused parts of the circuit are powered down if power management is implemented.

2.3 RTL Code

Listing 1: Adaptive Divider

```
1
2
3  module AdaptiveDivider #(parameter WIDTH = 8) (
4      input logic clk, reset,
5      input logic [WIDTH-1:0] dividend, divisor,
6      output logic [WIDTH-1:0] quotient, remainder,
7      output logic valid
8  );
9      always_ff @(posedge clk or posedge reset) begin
10         if (reset) begin
11             quotient <= 0; remainder <= 0; valid <= 0;
12         end else if (divisor != 0) begin
13             quotient <= dividend / divisor;
14             remainder <= dividend % divisor;
15             valid <= 1;
16         end else begin
17             quotient <= 0; remainder <= dividend; valid <= 0; //
                   Handle division by zero
18         end
19     end
20 endmodule
```

2.4 Testbench

Listing 2: Adaptive Divider

```
1
2  module AdaptiveDivider_tb;
3      parameter WIDTH = 8;
4      logic clk = 0, reset;
5      logic [WIDTH-1:0] dividend, divisor, quotient, remainder;
6      logic valid;
7
8      AdaptiveDivider #(.WIDTH(WIDTH)) uut (.clk(clk), .reset(reset),
           .dividend(dividend), .divisor(divisor),
           .quotient(quotient),.remainder(remainder), .valid(valid));
9
10     always #5 clk = ~clk; // Clock generation
11
12     initial begin
13         reset = 1; #10 reset = 0;
14         dividend = 30; divisor = 6; #10;
15         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
16
17         dividend = 17; divisor = 0; #10;
18         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
19
20         dividend = 25; divisor = 5; #10;
21         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
22         $finish;
23     end
24 endmodule
```

3 Results

3.1 Simulation

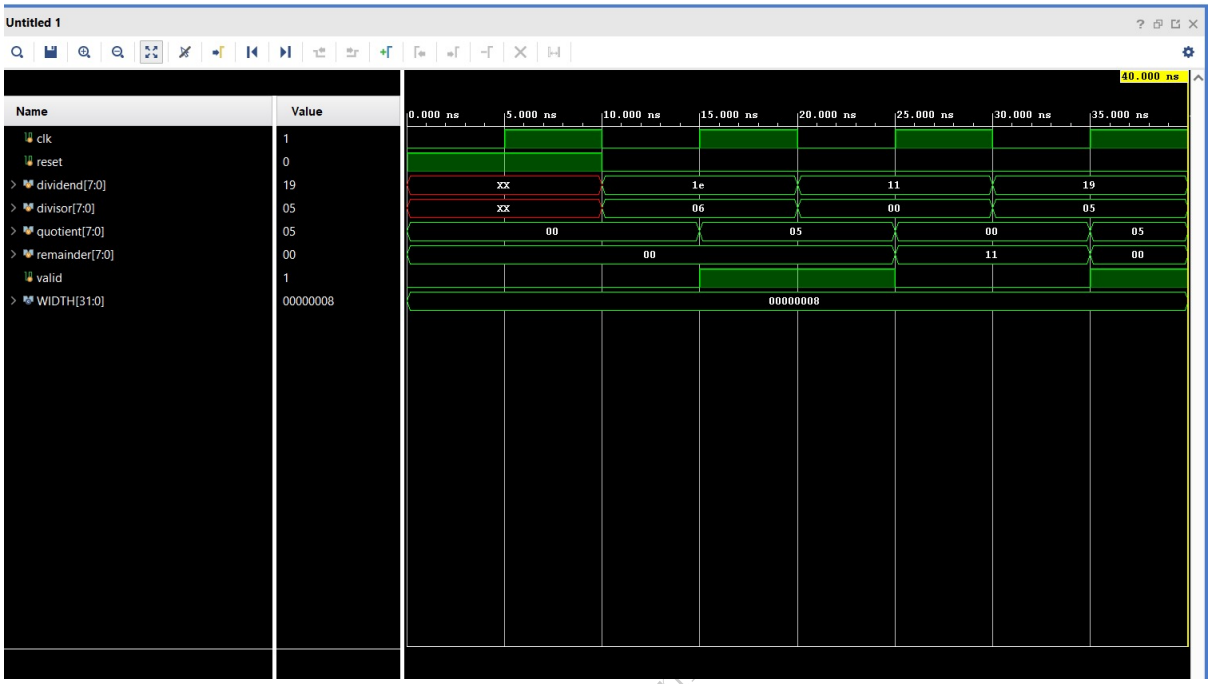

Figure 1: Simulation of Adaptive Divider

3.2 Schematic

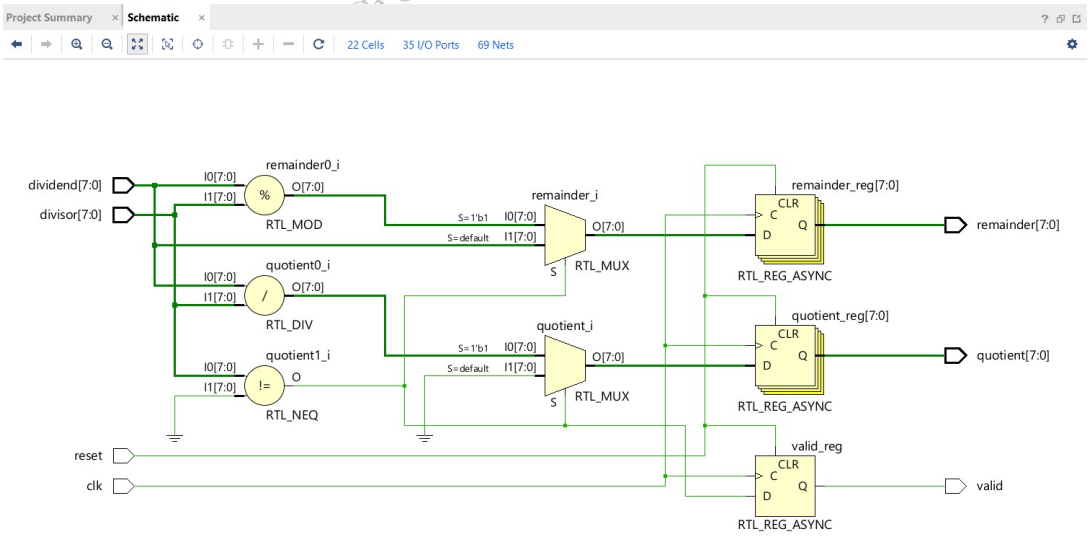

Figure 2: Schematic of Adaptive Divider

3.3 Synthesis Design

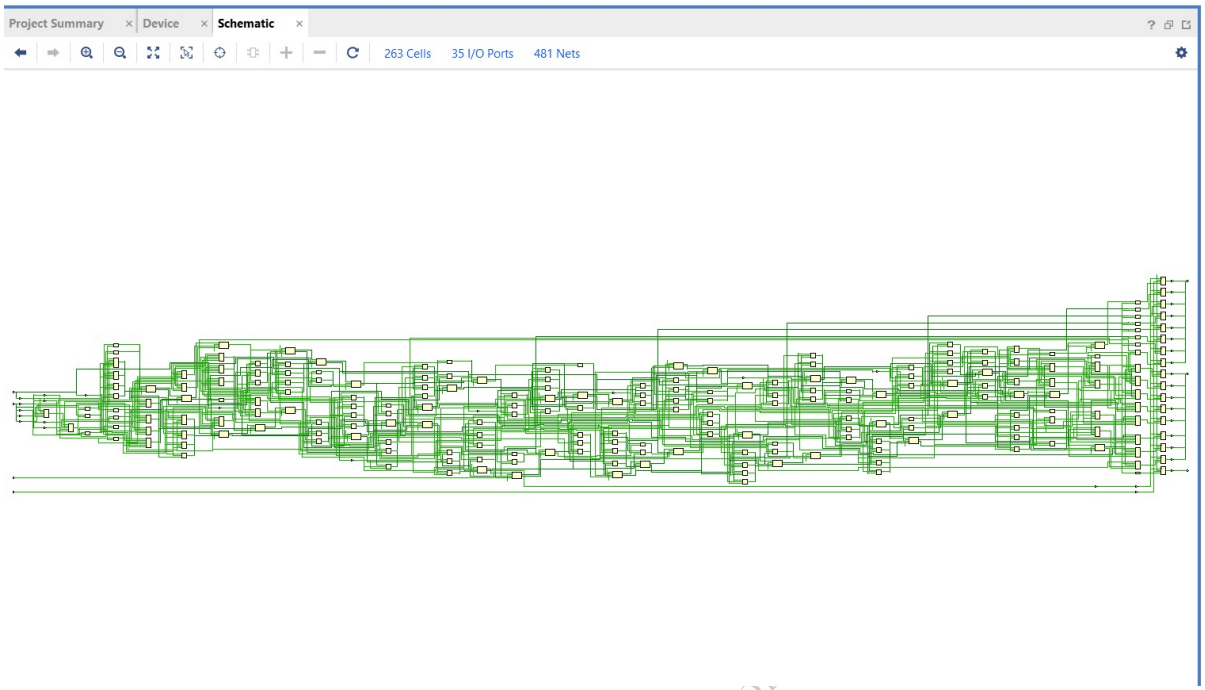

Figure 3: Synthesis Design of Adaptive Divider

4 Advantages of Adaptive Divider

Higher Speed: Adaptive dividers can dynamically skip unnecessary iterations, making the division process faster compared to fixed dividers.

Lower Power Consumption: Fewer cycles mean reduced energy usage, which is ideal for power-sensitive applications, such as mobile and embedded systems.

Improved Efficiency: Adaptive behavior allows for optimal use of hardware resources, with less redundant computation.

Dynamic Precision Control: The divider can adjust precision based on the application's needs, balancing speed and accuracy.

Scalability: Adaptive dividers can handle a range of operand sizes and precision requirements, making them versatile in various computational settings.

5 Disadvantages of Adaptive Divider

Complex Control Logic: The adaptive mechanisms require sophisticated control units and logic circuits, increasing design complexity.

Potential Accuracy Trade-offs: When using approximation or early termination, there may be a loss in accuracy, which can be problematic for high-precision applications.

Higher Design and Testing Costs: More complex control and error-handling logic necessitate rigorous testing, increasing development time and cost.

Increased Circuit Area: While it saves computation cycles, the additional components (like lookup tables and control units) might increase the overall circuit area.

Risk of Unpredictable Latency: Adaptive dividers may vary in computation time based on the input values, which can be unsuitable for real-time applications requiring predictable processing times.

6 Applications of Adaptive Divider

Digital Signal Processing (DSP): Adaptive dividers are useful in DSP for efficient real-time processing, where division operations are frequent.

Graphics and Image Processing: They help in 3D rendering and image scaling, where division speed can impact performance.

Communication Systems: In digital communication, adaptive dividers optimize error-correction and modulation schemes that require frequent division.

AI and Machine Learning: They are advantageous in neural networks and inference processes that benefit from approximate calculations or adjustable precision.

Cryptography: Used in cryptographic algorithms that require division and modular arithmetic for secure computations.

Embedded Systems and IoT Devices: Adaptive dividers suit low-power, resource-constrained environments by offering efficient and power-saving division operations.

7 Conclusion

Adaptive Dividers enhance division operations by dynamically adjusting their computation process based on input characteristics or intermediate results. By optimizing the division process, they reduce the number of cycles, improve speed, and save power, making them suitable for high-performance and resource-constrained environments.

8 FAQs

1. What is an adaptive divider, and how does it differ from a traditional divider?

An adaptive divider is a type of digital divider that dynamically adjusts its computation steps based on the input values or intermediate results. Unlike traditional dividers that use a fixed number of steps, adaptive dividers may skip or adjust iterations to improve speed, reduce power consumption, and optimize efficiency.

2. How does an adaptive divider improve computation speed?

By dynamically predicting quotient bits, reducing unnecessary iterations, and potentially terminating calculations early, adaptive dividers minimize the number of cycles required for division. This adaptive process accelerates computation compared to traditional methods.

3. In what scenarios should an adaptive divider be used?

Adaptive dividers are ideal for applications where speed, efficiency, and power savings are essential. They are commonly used in DSP, graphics processing, communication systems, and low-power embedded devices where division calculations are frequent and performance-sensitive.

4. What are the main components of an adaptive divider?

The main components include a control unit, quotient prediction logic, partial remainder calculator, lookup table (for certain algorithms), approximation/error handling logic, precision control, and an iteration counter.

5. What types of algorithms are commonly used in adaptive dividers?

Common algorithms include the SRT (Sweeney-Robertson-Tocher) division, which uses a lookup table for quotient selection, as well as variable iteration methods and approximate division techniques that trade off precision for speed when necessary.

6. Can an adaptive divider achieve the same level of accuracy as a traditional divider?

Yes, adaptive dividers can achieve high accuracy; however, they may employ approximation in certain cases to prioritize speed. The precision can often be controlled based on the application's needs, allowing the user to balance accuracy and performance.

7. What are the main disadvantages of adaptive dividers?

Disadvantages include increased complexity in control logic, potential trade-offs in accuracy, higher testing and validation costs, and unpredictable latency due to variable iteration times. These factors make them more complex to design and test than traditional dividers.

8. Are adaptive dividers suitable for real-time applications?

It depends. Adaptive dividers offer high efficiency and speed but may have variable latency, which can be problematic for real-time systems that require predictable processing times. In such cases, a carefully designed adaptive divider with bounded latency may still be suitable.

9. How does dynamic precision adjustment work in adaptive dividers?

Adaptive dividers can alter the number of calculation steps or accuracy based on the required output precision. For example, they might reduce iterations if a close approximation is acceptable or extend calculations for high-precision applications.

10. What are common applications for adaptive dividers in AI and ML?

In AI and ML, adaptive dividers help optimize operations like scaling, normalization, and division-heavy computations in neural networks. They're beneficial when processing power is limited, or variable precision can help speed up model inference.

11. How is power consumption reduced in adaptive dividers?

By eliminating unnecessary calculations, adaptive dividers save energy. Components of the divider may also be powered down dynamically during unused cycles, further reducing power consumption, which is especially beneficial in embedded and IoT devices.

12. Is it difficult to design an adaptive divider?

Yes, designing an adaptive divider can be complex due to the need for adaptive control logic, error handling, and dynamic precision adjustments. Proper testing and optimization are also essential to ensure the divider operates efficiently and reliably across different input values.