Project 62: Asynchronous 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

The Asynchronous Divider project focuses on designing a hardware divider that operates asynchronously, without the need for a clock signal to control the timing of division operations. The divider will process input numbers in a sequence, performing division through a combination of shifts, adds, and subtracts. This project aims to optimize efficiency in digital circuits by reducing clock dependency, improving speed, and minimizing power consumption. The design will be implemented using Verilog or VHDL for hardware description and simulation.

2 Asynchronous Divider

2.1 Key Concept of Asynchronous Divider

The key concept of an Asynchronous Divider is that it performs division operations without relying on a synchronized clock signal. Unlike synchronous circuits, which require a clock to drive operations at fixed time intervals, an asynchronous divider operates using a self-timed mechanism, where the circuit's actions depend on the data inputs and internal states rather than a global clock.

Key features include:

1. **Self-timed operation:** The divider processes input values and generates the quotient based on internal signal propagation delays.

2. **Serial processing:** Division is typically carried out through bit-by-bit shifts and subtracts, much like in long division, allowing it to function asynchronously.

3. Efficiency: Asynchronous dividers can be more power-efficient than their synchronous counterparts, as they don't waste power on clocking operations.

This design is more complex due to challenges in timing and controlling the flow of data without a clock, but it offers potential advantages in terms of performance and power efficiency in certain applications.

2.2 Working of Asynchronous Divider

Initialization:

1. The dividend (numerator) and divisor (denominator) are fed into the divider circuit.
2. A register stores the quotient (result of the division) and the remainder (what's left after the division).
3. Initially, the quotient is set to zero, and the remainder is set to the dividend.

Shift and Subtract:

1. The divider works by shifting the remainder left (in binary) and then comparing it to the divisor.
2. If the remainder is greater than or equal to the divisor, the divisor is subtracted from the remainder.
3. The quotient is incremented by 1 for each successful subtraction.

Bit-by-Bit Operation:

1. For each bit of the quotient (starting from the most significant bit), the remainder is shifted left, and a subtraction is attempted.
2. The result of the subtraction determines whether a 1 or 0 is placed in the corresponding bit of the quotient.
3. If the subtraction is successful, the remainder is updated; otherwise, it is left unchanged.

Completion:

1. The process repeats for each bit of the quotient until the division is complete (i.e., all bits of the quotient are determined).
2. The final quotient and the remainder are stored as the result of the division.

Asynchronous Nature:

1. Unlike synchronous dividers, which are controlled by a clock, asynchronous dividers operate without a clock. The data signals themselves control the timing of the operations.
2. The time it takes to complete the division depends on the size of the inputs (dividend and divisor) and the propagation delays through the gates and logic elements.

2.3 RTL Code

Listing 1: Asynchronous Divider

```
1
2  module AsynchronousDivider #(parameter WIDTH = 8) (
3      input logic start, reset,
4      input logic [WIDTH-1:0] dividend, divisor,
5      output logic [WIDTH-1:0] quotient, remainder,
6      output logic valid
7  );
8      always_comb begin
9          if (reset) begin
10             quotient = 0; remainder = 0; valid = 0;
11         end else if (start) begin
12             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 else begin
20             valid = 0; // Not valid if not started
21         end
22     end
23 endmodule
```

2.4 Testbench

Listing 2: Asynchronous Divider

```
1
2  module AsynchronousDivider_tb;
3      parameter WIDTH = 8;
4      logic start, reset;
5      logic [WIDTH-1:0] dividend, divisor, quotient, remainder;
6      logic valid;
7
8      AsynchronousDivider #(.WIDTH(WIDTH)) uut (.start(start),
           .reset(reset),
9                                                .dividend(dividend),
                                                     .divisor(divisor),
10                                               .quotient(quotient),
                                                     .remainder(remainder),
11                                               .valid(valid));
12
13     initial begin
14         reset = 1; start = 0; #10 reset = 0;
15
16         // Test case 1
17         dividend = 20; divisor = 4; start = 1; #10;
18         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
19
20         // Test case 2: Division by zero
21         dividend = 15; divisor = 0; #10;
22         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
23
24         // Test case 3
25         dividend = 30; divisor = 6; start = 1; #10;
26         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
27
28         // Resetting
29         reset = 1; #10 reset = 0; start = 1; dividend = 10; divisor =
               2; #10;
30         $display("Div=%0d, Divisor=%0d, Quot=%0d, Rem=%0d, Valid=%b",
               dividend, divisor, quotient, remainder, valid);
31
32         $finish;
33     end
34 endmodule
```

3 Results

3.1 Simulation

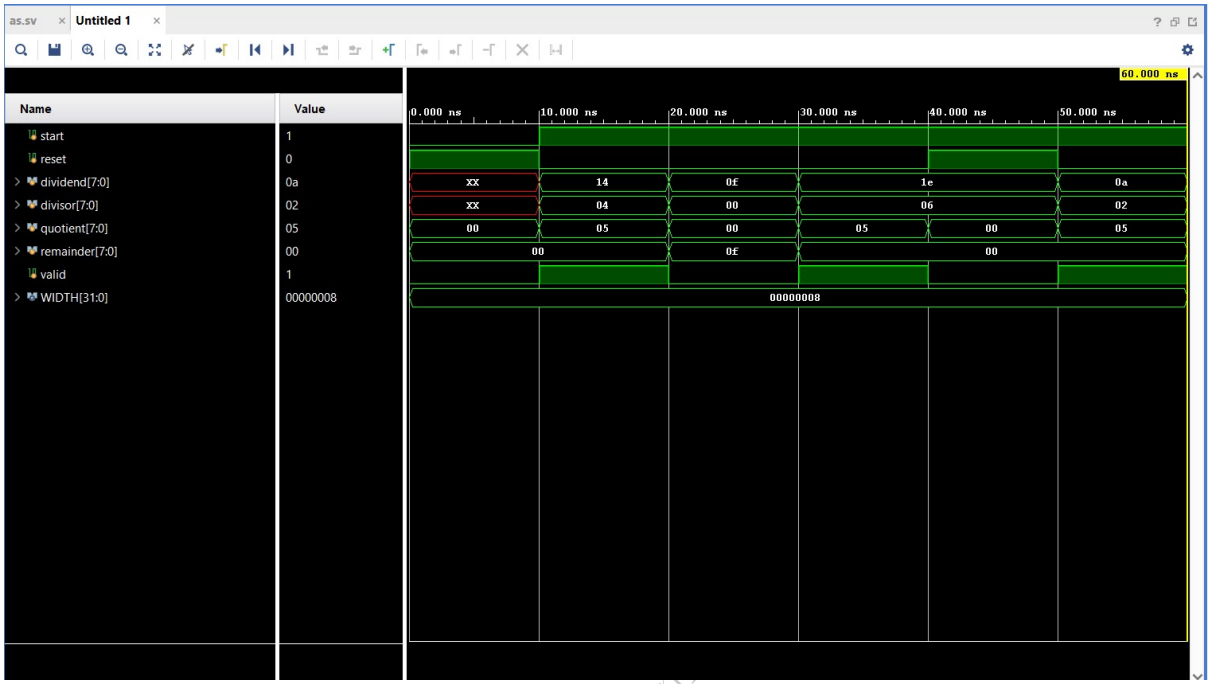

Figure 1: Simulation of Asynchronous Divider

3.2 Schematic

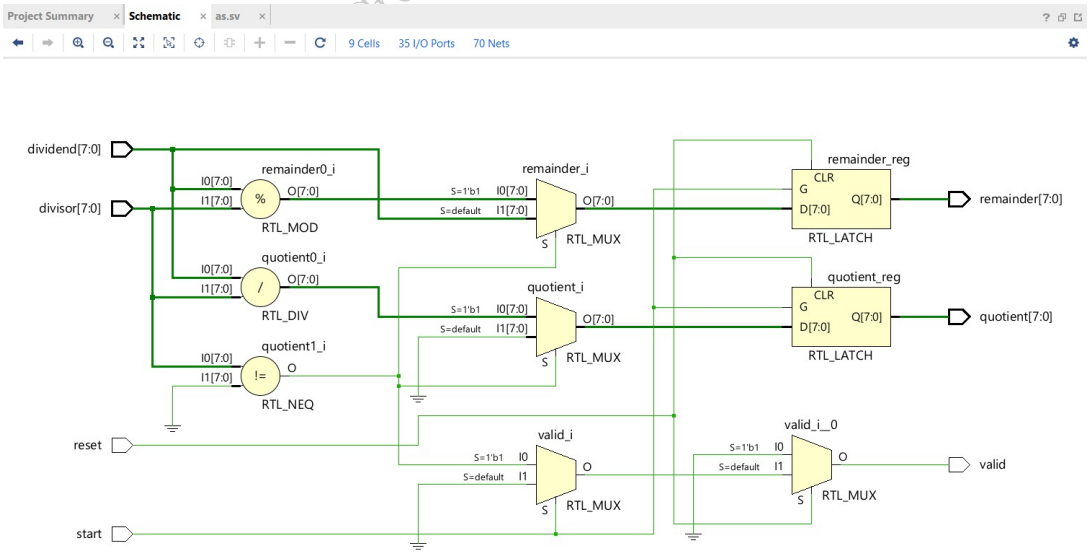

Figure 2: Schematic of Asynchronous Divider

3.3 Synthesis Design

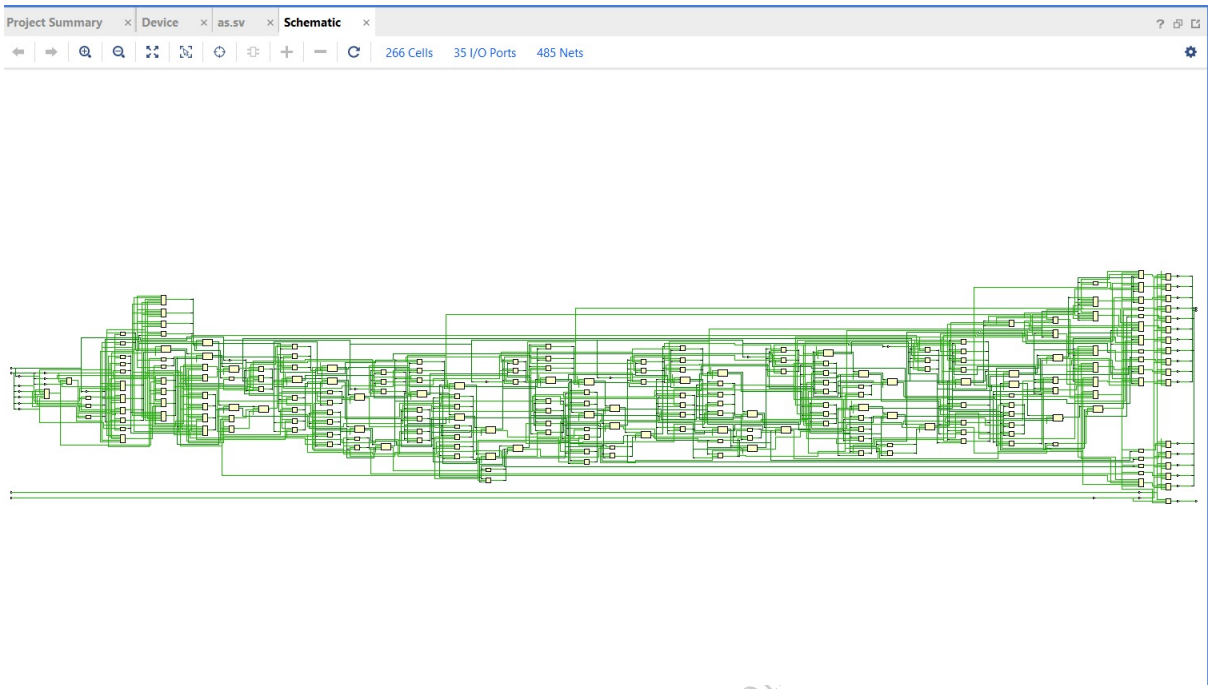

Figure 3: Synthesis Design of Asynchronous Divider

4 Advantages of Asynchronous Divider

- **Power Efficiency:**
  Asynchronous circuits do not require a clock signal to operate, eliminating the power consumption associated with clock generation and distribution. This can result in lower overall power consumption, which is particularly advantageous for low-power applications like battery-operated devices.

- **Reduced Clock Skew and Latency:**
  Since the operations are not synchronized to a clock, there is no clock skew (the time difference between the clock signal reaching different parts of the circuit). This reduces latency in some cases, as the divider operates based on the data's own timing.

- **Fewer Components:**
  Asynchronous dividers do not need a clock distribution network, which can simplify the design of the circuit and reduce the number of components required. This can lead to smaller and less complex systems.

- **Speed:**
  For certain applications, asynchronous dividers can achieve faster operation because they do not have to wait for a clock pulse to complete each division step. The system can proceed as quickly as the data changes, potentially speeding up operations compared to clocked designs.

- **Adaptability:**
  Asynchronous dividers can be adapted to handle varying input sizes and changing operational conditions without being constrained by clock cycles. This makes them more flexible in certain dynamic or irregular timing applications.

- **Clock-less Operation:**
  The absence of a global clock means asynchronous dividers are not subject to timing issues like clock skew or race conditions, which are common in synchronous circuits. This can improve reliability in systems where clock synchronization is challenging or expensive.

- **Simpler Timing Management:**
  In some designs, asynchronous circuits may allow easier control over timing, as they rely on data flow rather than a global clock, which can make them easier to manage in systems where precise clock timing is difficult or unnecessary.

- **Improved Scalability in Some Applications:**
  Asynchronous circuits can scale more effectively in certain contexts, especially in systems where the size of the input data varies or is unpredictable. They do not need additional clock cycles to handle larger numbers, and the delay only depends on the data rather than fixed clock cycles.

5 Disadvantages of Asynchronous Divider

- **Design Complexity:**
  Harder to design and verify due to the lack of a clock signal.

- **Increased Propagation Delay:**
  Delays vary based on data and circuit size.

- **Timing Issues:**
  Susceptible to race conditions and timing hazards.

- **Unpredictable Performance:**
  Harder to predict completion time for large inputs.

- **Integration Challenges:**
  Difficult to interface with synchronous components.

- **Scalability Limits:**
  Performance can degrade with larger inputs.

- **Fault Sensitivity:**
  More prone to errors and harder to debug.

6 Applications of Asynchronous Divider

- **Low-Power Systems:**
  Asynchronous dividers are ideal for battery-powered devices like mobile phones, wearable electronics, and IoT devices, where minimizing power consumption is critical.

- **Embedded Systems:**
  In embedded systems that require simple, low-cost circuits without the overhead of a clock distribution network, asynchronous dividers can be used to save power and reduce complexity.

- **Digital Signal Processing (DSP):**
  Asynchronous dividers can be applied in DSP circuits, particularly where low-power, low-latency operations are needed for filtering or signal manipulation, especially in audio or video processing.

- **Hardware Division in Custom Circuits:**
  Asynchronous dividers are often used in custom hardware solutions where a simple divider is required, and the overhead of clock synchronization is not necessary, such as in digital controllers or arithmetic units in FPGAs.

- **Communication Systems:**
  In communication systems, where efficient data transmission is needed, asynchronous dividers can help with tasks like frequency division or data rate conversion in low-power, low-latency designs.

- **Clockless Circuits:**
  In clockless or pulsed circuits, where avoiding a global clock signal is advantageous, asynchronous dividers are used to perform arithmetic tasks without the need for clock synchronization.

- **Real-Time Processing Systems:**
  Asynchronous dividers are used in real-time processing systems where operations must occur as fast as the data changes, and there is no need to synchronize with a clock. This is useful in certain high-speed applications like sensor data processing.

- **FPGA and ASIC Design:**
  Asynchronous dividers are employed in FPGA or ASIC designs where custom low-power, high-speed arithmetic operations are needed without the overhead of clocking networks.

- **Digital Control Systems:**
  In digital control systems (e.g., robotics, automotive), asynchronous dividers can be used to perform division tasks where speed and low power are priorities, and where division timing can be unpredictable.

- **Cryptography and Security:**
  Certain cryptographic algorithms and security applications might use asynchronous division circuits to achieve secure, low-latency encryption or decryption without the need for centralized timing control.

7 Summary

Asynchronous dividers are used in low-power systems, embedded designs, and digital signal processing, where minimizing clock overhead is crucial. They enable efficient division in applications like communication systems, real-time processing, and digital control. These dividers are also valuable in FPGA, ASIC, cryptography, and clockless circuits for low-latency operations.

8 FAQs

1. **How do asynchronous dividers differ from synchronous dividers?** Synchronous dividers rely on a clock signal to coordinate operations, while asynchronous dividers operate based on internal data flows, without the need for synchronization to a global clock. Asynchronous dividers can be more power-efficient but are harder to design and verify.

2. **Are asynchronous dividers scalable?** Asynchronous dividers may have scalability limitations because their performance can degrade as the size of the input increases. Larger inputs may lead to longer delays, reducing their efficiency in high-performance applications compared to synchronous dividers.

3. **What challenges exist when designing asynchronous dividers?** Challenges include managing timing to avoid race conditions, ensuring data integrity, handling propagation delays, and testing and debugging the design without a clock signal to monitor operations.

4. **Can asynchronous dividers be integrated with synchronous systems?**
   Integrating asynchronous dividers with synchronous systems can be challenging. Additional logic may be required to interface the two, such as timing synchronization mechanisms, to prevent issues with timing mismatches or data consistency.

5. **What are the advantages of using an asynchronous divider?** Advantages include lower power consumption (due to no clock signal), reduced design complexity (no need for clock distribution), and faster operation in certain applications where timing is data-driven. It is also flexible and suitable for low-power or battery-operated devices.

6. **What are the disadvantages of asynchronous dividers?**
   Disadvantages include potential timing issues, increased propagation delay, susceptibility to race conditions, difficulty in design and verification, and unpredictability in performance, especially with large inputs or complex operations.