**RESTORING DIVISION ALGORITHM**

**Introduction to Restoring Division Algorithm**

In digital arithmetic and computer architecture, division is a fundamental operation performed on binary numbers. One of the algorithms used to implement division is the restoring division algorithm. This algorithm is designed to efficiently compute the quotient and remainder when dividing two binary numbers.

The restoring division algorithm operates by iteratively subtracting the divisor from the dividend to obtain the quotient and remainder. Unlike non-restoring division, where adjustments are made based on the sign of the intermediate result, restoring division always restores the remainder to a non-negative value after each subtraction step.

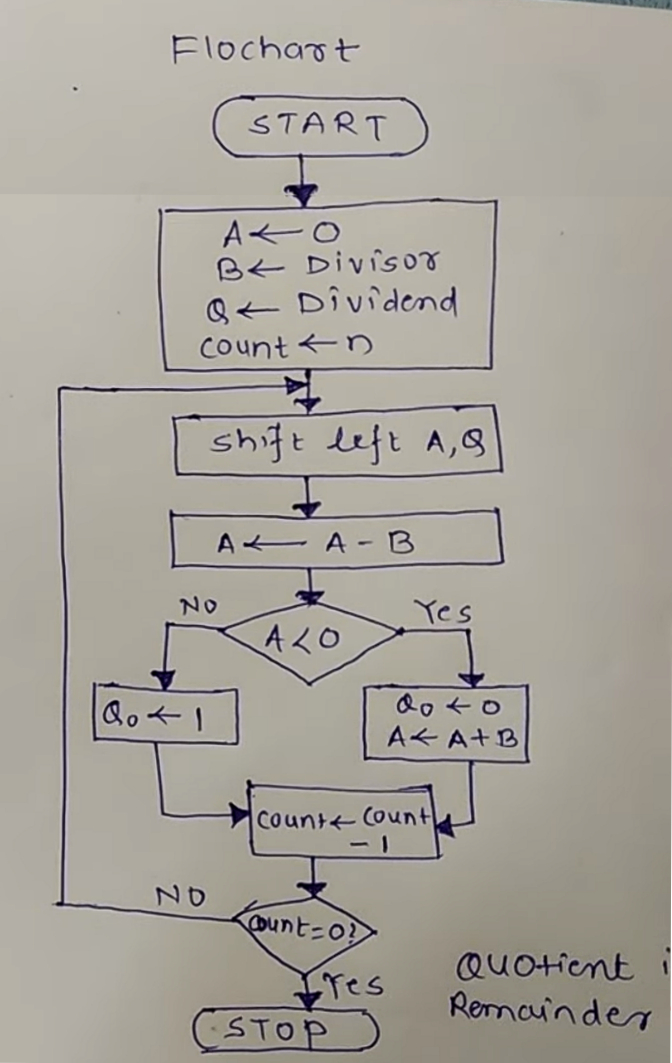
The algorithm involves steps such as initialization, shifting, subtraction, normalization, and updating quotient and remainder registers. By carefully managing the subtraction and restoration steps, the algorithm ensures accuracy in the division process.

Restoring division finds applications in digital signal processing, microprocessor design, and various computational tasks where division operations are required. Understanding its principles and implementation is crucial for digital system designers and computer engineers working with arithmetic units and computational algorithms.

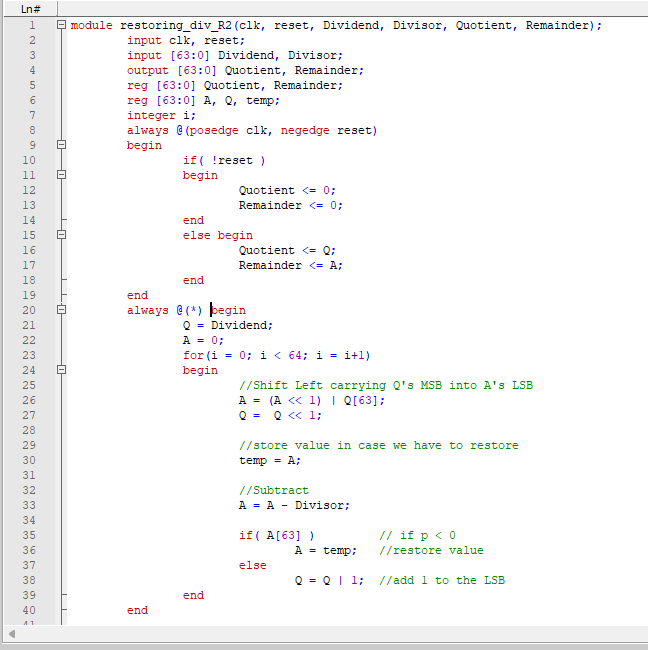
**Steps to Perform:**

1. **Initialize:**
   * Load the divisor and dividend into their respective registers.
   * Set up other necessary registers such as the quotient (Q) and remainder (A) registers.
2. **Initialize Quotient and Remainder:**
   * Set the quotient (Q) register to zero initially.
   * Load the dividend into the remainder (A) register.
3. **Repeat for N cycles (N is the number of bits in the dividend and divisor):**
   * Left-shift the contents of the quotient (Q) and remainder (A) registers by 1 bit.
   * Subtract the divisor from the shifted remainder (A).
   * If the result of the subtraction is non-negative:
     + Set the least significant bit (LSB) of the quotient (Q) to 1.
   * If the result is negative:
     + Restore the remainder (A) by adding the divisor back to it.
     + Set the LSB of the quotient (Q) to 0.
4. **Adjust the Remainder:**
   * If the quotient bit was set to 0 in the previous step (indicating a negative result), add the divisor to the remainder (A) again to restore it.
5. **Normalization:**
   * If the most significant bit (MSB) of the remainder (A) is 1 after adjustment, subtract the divisor from the remainder to normalize it.
6. **Repeat:**
   * Repeat the above steps until N cycles are completed (for an N-bit division).
7. **Final Result:**
   * At the end of N cycles, the quotient (Q) register contains the quotient of the division operation, and the remainder (A) register contains the final remainder.

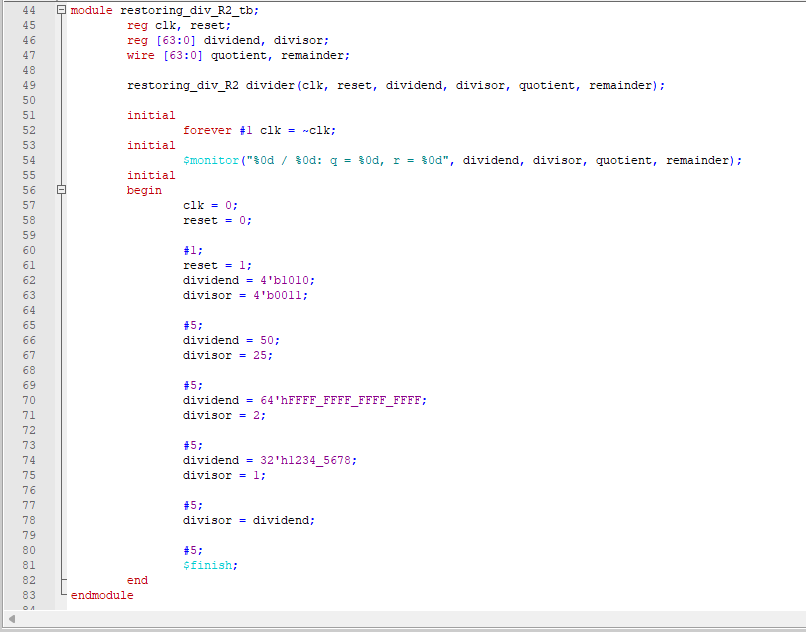
**Flow Chart:**

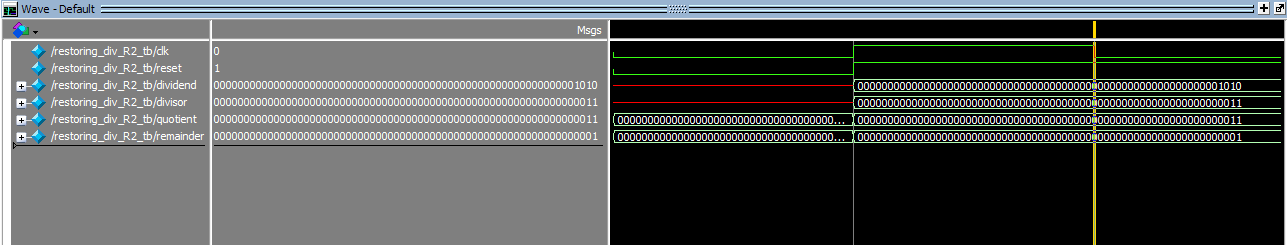
****

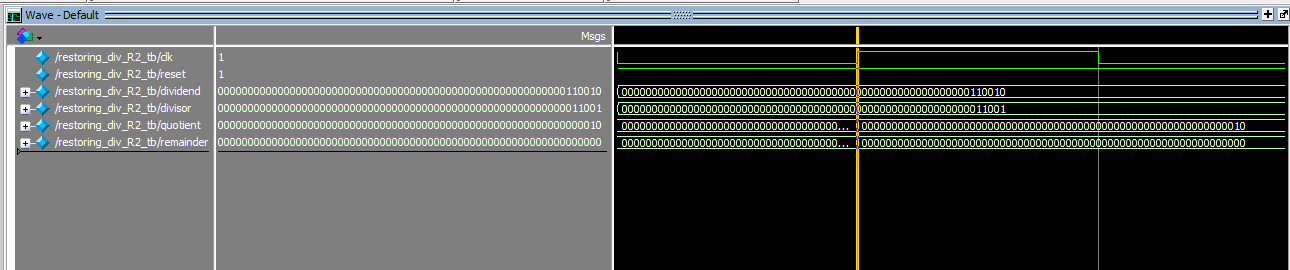
**Verilog Code:**

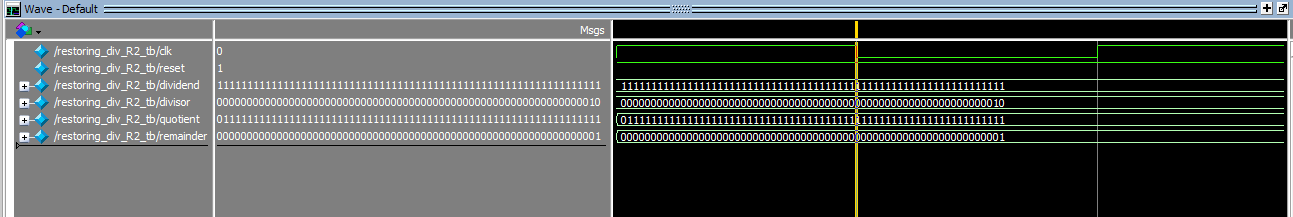
****

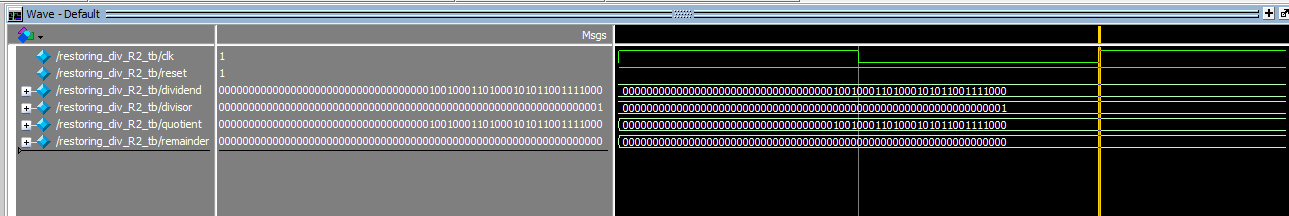
**Test Bench Code:**

****

**Outputs: **

****

****

****

**Conclusion:**

The restoring division algorithm stands as a robust method for performing division operations in digital arithmetic and computer architecture. Through its iterative approach of subtracting the divisor from the dividend while ensuring a non-negative remainder, the algorithm guarantees accuracy and reliability in computing both the quotient and remainder.

In conclusion, our exploration of the restoring division algorithm has revealed its importance in various computational tasks, including digital signal processing and microprocessor design. Its efficient handling of division operations underscores its relevance in modern computing systems, making it a fundamental concept for digital system designers and computer engineers.

By mastering the principles and intricacies of the restoring division algorithm, practitioners can enhance their ability to design optimized arithmetic units and computational algorithms, contributing to advancements in digital technology and computational efficiency.