8 Bit Binary Multiplier

A binary multiplier is a digital circuit or mathematical operation that performs multiplication of two binary numbers, which consist of 0s and 1s. In binary multiplication, each digit of one binary number is multiplied by each digit of the other binary number, similar to how multiplication works in the decimal system. The results of these multiplications are then added together to obtain the final binary product.

Let's multiply two binary numbers, 1101 and 101:

```
  1101
x  101
-------
  1101   (This is 1101 * 1)
 0000    (This is 1101 * 0, shifted one position to the left)
1101     (This is 1101 * 1, shifted two positions to the left)
-------
1001001  (The final result is the sum of the partial products.
```

Types of multipliers

There are several types of multipliers used in digital circuits and computer architectures, each with its own advantages and disadvantages. Here are some common types of multipliers:

1. Binary Multiplier: This is the simplest form of multiplication, where each bit of the multiplier is used to select or reject the corresponding bit of the multiplicand. The results are then added together. It's straightforward but not very efficient for large operands.

2. Shift-and-Add Multiplier: In this method, the bits of the multiplier are shifted left one by one, and if the current bit is 1, the multiplicand is added to an accumulating sum. This is also known as the "repeated addition" method and is relatively slow.

3. Array Multiplier: This is a combinational circuit where a grid of AND gates performs partial products generation, and a tree of adders adds them up. It's faster than shift-and-add but requires a large number of gates.

4. Wallace Tree Multiplier: This is an improvement over the array multiplier that reduces the number of partial products by grouping them into larger partial products before adding. It's more efficient than the array multiplier in terms of gate count but still not the fastest.

5. Dadda Multiplier: The Dadda multiplier is another improvement over the Wallace tree multiplier, aiming to reduce the number of adders by optimizing the generation of partial products. It's more area-efficient than the Wallace tree multiplier.

6. Booth Multiplier: The Booth algorithm is an algorithmic approach to multiplication that reduces the number of partial products generated and is more efficient than the above methods. It works by encoding adjacent bits of the multiplier into a 2-bit code and using this code to select between adding, subtracting, or ignoring the multiplicand.

7. Modified Booth Multiplier: This is an enhancement of the Booth multiplier that further reduces power consumption by eliminating unnecessary add/subtract operations.

8. Radix-4 Multiplier: Radix-4 multipliers operate on four-bit groups of the multiplier and are more efficient than binary multipliers for large operands.

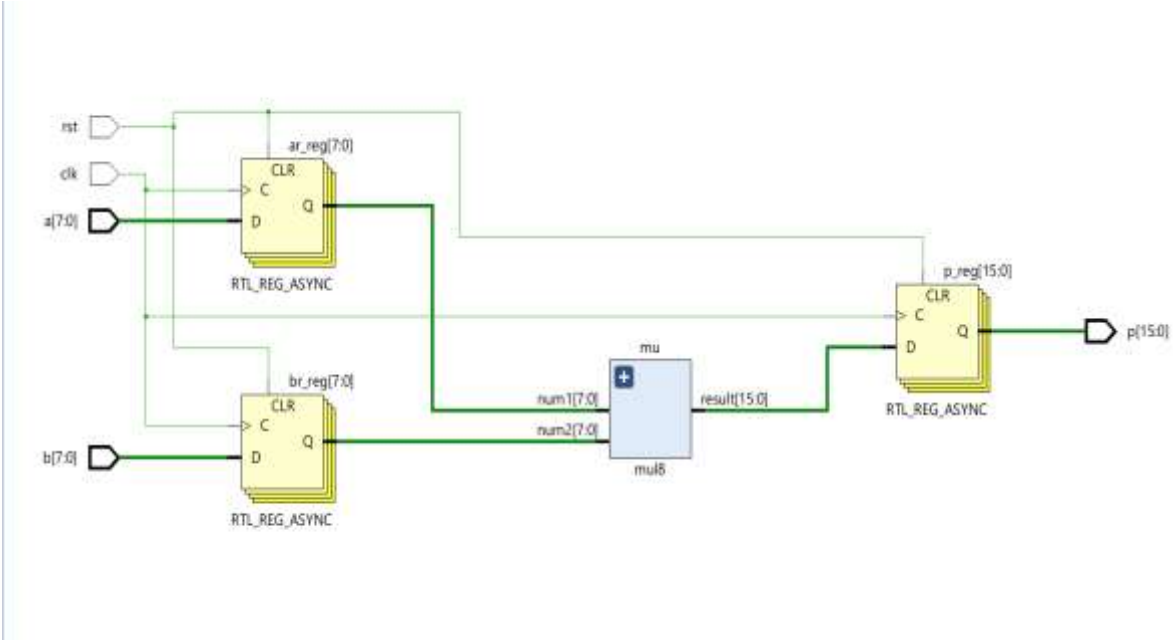
9.Tree Multiplier: Tree multipliers use a tree structure to add partial products, which can reduce the delay and power consumption compared to array or Wallace tree multipliers.

10. Pipelined Multiplier: Pipelined multipliers break down the multiplication process into stages and process multiple operands simultaneously to improve throughput. This is common in high-performance processors.

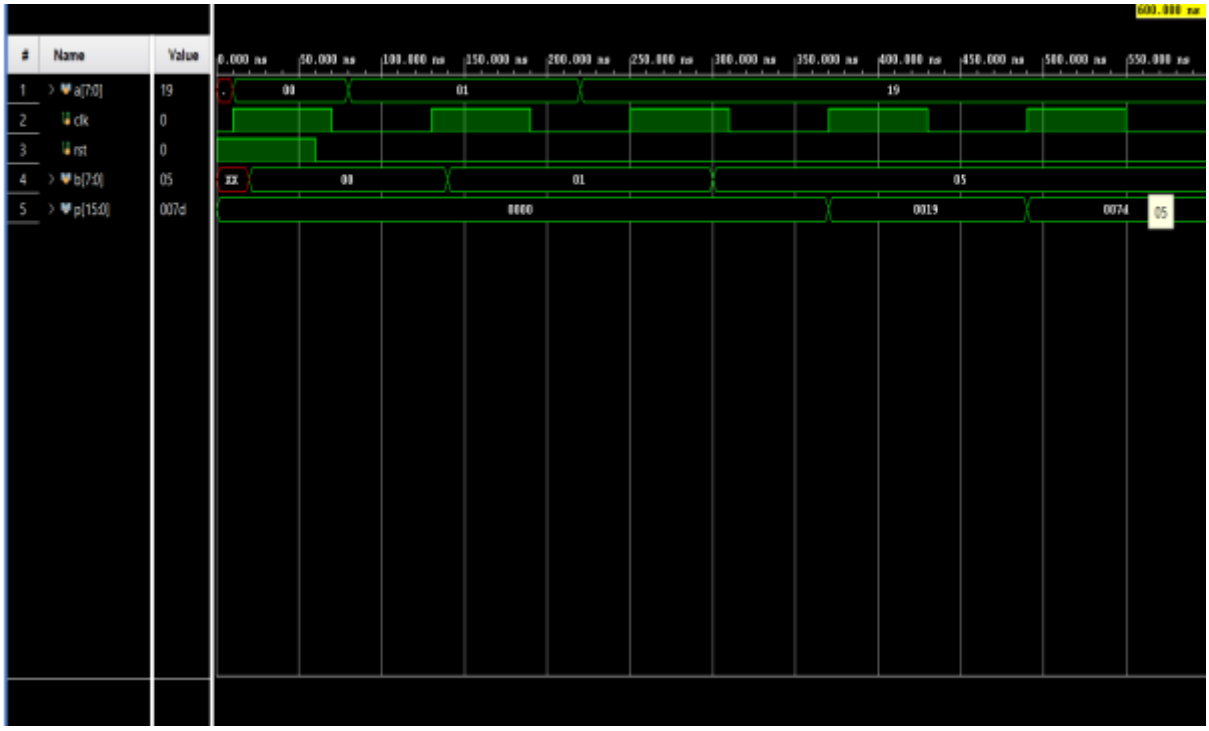
11. Parallel Multiplier: Parallel multipliers employ various techniques to compute multiple bits of the result in parallel, reducing the overall multiplication time.

12. Floating-Point Multiplier : These multipliers are specifically designed for performing multiplication operations on floating-point numbers according to IEEE 754 standards.

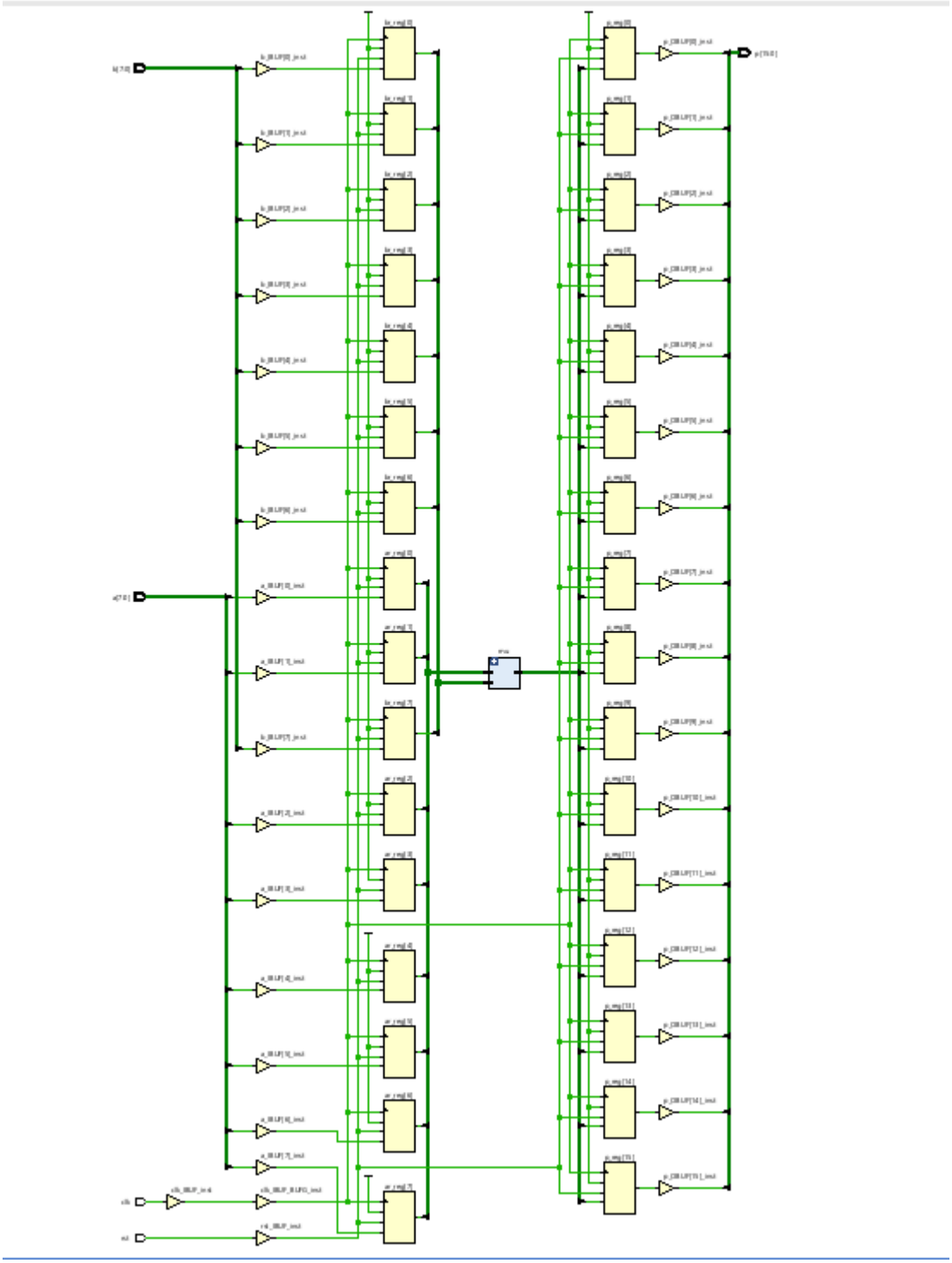
Elaborated design of binary multiplier



Behavioural simulation of binary multiplier



Implemented design of binary adder



Setup and Hold slacks

Intra-Clock Paths - clk - Setup

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	4.125	10	15	br_reg[4]/C	p_reg[12]/D	5.875	2.098	3.777	10.0	clk	clk	
Path 2	4.186	10	15	br_reg[4]/C	p_reg[11]/D	5.786	2.164	3.622	10.0	clk	clk	
Path 3	4.208	10	15	br_reg[4]/C	p_reg[13]/D	5.793	2.135	3.658	10.0	clk	clk	
Path 4	4.316	10	15	br_reg[4]/C	p_reg[14]/D	5.656	2.258	3.398	10.0	clk	clk	
Path 5	4.371	10	15	br_reg[4]/C	p_reg[15]/D	5.630	2.223	3.407	10.0	clk	clk	
Path 6	4.482	10	15	br_reg[4]/C	p_reg[10]/D	5.490	2.093	3.397	10.0	clk	clk	
Path 7	4.725	10	15	br_reg[4]/C	p_reg[9]/D	5.279	1.920	3.359	10.0	clk	clk	
Path 8	4.939	9	15	br_reg[4]/C	p_reg[7]/D	5.065	1.617	3.448	10.0	clk	clk	
Path 9	4.969	9	15	br_reg[4]/C	p_reg[8]/D	5.006	1.819	3.187	10.0	clk	clk	
Path 10	5.387	7	18	br_reg[3]/C	p_reg[6]/D	4.588	1.493	3.095	10.0	clk	clk	

Intra-Clock Paths - clk - Hold

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 11	0.258	2	19	ar_reg[2]/C	p_reg[3]/D	0.340	0.179	0.161	0.0	clk	clk	
Path 12	0.282	2	19	ar_reg[2]/C	p_reg[4]/D	0.364	0.203	0.161	0.0	clk	clk	
Path 13	0.287	1	10	br_reg[7]/C	p_reg[7]/D	0.373	0.131	0.242	0.0	clk	clk	
Path 14	0.291	1	10	br_reg[7]/C	p_reg[12]/D	0.374	0.130	0.244	0.0	clk	clk	
Path 15	0.295	1	10	br_reg[7]/C	p_reg[13]/D	0.378	0.133	0.245	0.0	clk	clk	
Path 16	0.296	1	10	br_reg[7]/C	p_reg[9]/D	0.382	0.130	0.252	0.0	clk	clk	
Path 17	0.299	1	10	br_reg[7]/C	p_reg[6]/D	0.370	0.128	0.242	0.0	clk	clk	
Path 18	0.304	1	10	br_reg[7]/C	p_reg[10]/D	0.372	0.128	0.244	0.0	clk	clk	
Path 19	0.305	1	10	br_reg[7]/C	p_reg[11]/D	0.373	0.128	0.245	0.0	clk	clk	
Path 20	0.309	1	10	br_reg[7]/C	p_reg[8]/D	0.380	0.128	0.252	0.0	clk	clk	

Critical paths

Path 1

From = br_reg[4]/c

To = p_reg[12]/d

Source clock path = 4.757

Data path (arrival time) = 10.632

Destination clock path(required time) = 14.758

Salck = required time – arrival time = 4.125

Path 2

From = br_reg[4]/c

To = p_reg[11]/d

Source clock path = 4.757

Data path (arrival time) = 10.543

Destination clock path(required time) = 14.729

Salck = required time – arrival time = 4.186

Path 3

From = br_reg[4]/c

To = p_reg[13]/d

Source clock path = 4.757

Data path (arrival time) = 10.550

Destination clock path(required time) = 14.758

Salck = required time – arrival time = 4.208

Path 4

From = br_reg[4]/c

To = p_reg[14]/d

Source clock path = 4.757

Data path (arrival time) = 10.413

Destination clock path(required time) = 14.730

Salck = required time – arrival time = 4.316

Path 5

From = br_reg[4]/c

To = p_reg[15]/d

Source clock path = 4.757

Data path (arrival time) = 10.387

Destination clock path(required time) = 14.758

Salck = required time – arrival time = 4.317

Path 6

From = br_reg[4]/c

To = p_reg[10]/d

Source clock path = 4.757

Data path (arrival time) = 10.247

Destination clock path(required time) = 14.730

Salck = required time – arrival time = 4.482

Path 7

From = br_reg[4]/c

To = p_reg[9]/d

Source clock path = 4.757

Data path (arrival time) = 10.036

Destination clock path(required time) = 14.761

Salck = required time – arrival time = 4.725

Path 8

From = br_reg[4]/c

To = p_reg[7]/d

Source clock path = 4.757

Data path (arrival time) = 9.822

Destination clock path(required time) = 14.761

Salck = required time – arrival time = 4.939

Path 9

From = br_reg[4]/c

To = p_reg[8]/d

Source clock path = 4.757

Data path (arrival time) = 9.763

Destination clock path(required time) = 14.732

Salck = required time – arrival time = 4.969

Path 10

From = br_reg[4]/c

To = p_reg[6]/d

Source clock path = 4.757

Data path (arrival time) = 9.345

Destination clock path(required time) = 14.733

Salck = required time – arrival time = 5.389

Applications

Multipliers are fundamental components in digital systems and have a wide range of applications across various domains. Here are some common applications of multipliers:

1. Arithmetic Operations: Multipliers are used for basic arithmetic operations such as multiplication and division. In digital computers and processors, they perform these operations quickly and efficiently.

2. Digital Signal Processing (DSP): DSP applications, such as audio and image processing, often involve multiplying data by coefficients or other data points. Multipliers are crucial in implementing various DSP algorithms like filters, transforms, and modulation/demodulation.

3. Fixed-Point and Floating-Point Arithmetic: In numerical computing, multipliers are used to perform fixed-point and floating-point arithmetic, which is essential for scientific and engineering simulations.

4. Graphics Processing: In GPUs (Graphics Processing Units), multipliers are used for rendering graphics, including transformations, shading, and texture mapping. They are critical for fast and parallel computation in gaming and professional graphics applications.

5. Wireless Communication: Multipliers play a vital role in wireless communication systems, including modulation, demodulation, encoding, and decoding of signals. They enable the encoding of data onto carrier signals and subsequent decoding.

6. Cryptography: In encryption and decryption algorithms, multipliers are used for cryptographic operations such as key generation, public-key cryptography, and secure data transmission.

7. Control Systems: In control systems engineering, multipliers are used for system modelling and control algorithms. They help calculate control signals, control gains, and system responses.

8. Scientific Simulations: Multipliers are used in scientific simulations, particularly in solving partial differential equations (PDEs) and other mathematical models that require iterative calculations.

9. Digital Filters: Digital filters, used in various applications like audio processing and communications, require multipliers for convolution operations and filtering.

10. Computer Graphics: Multipliers are used in 3D graphics rendering, transformations, lighting calculations, and rendering algorithms to generate realistic images and animations.

11. Artificial Intelligence (AI) and Machine Learning: Multipliers are employed in machine learning models and deep neural networks for weight updates during training and inference stages.

12. Signal Analysis: In spectrum analysis and Fourier analysis, multipliers are used to calculate cross-correlation and convolution operations between signals.

13. Robotics: Multipliers can be found in robotics applications for kinematics and dynamics calculations, including forward and inverse kinematics.

14. Communication Systems: In communication systems, multipliers are used for channel equalization, beamforming, and adaptive filtering to enhance the quality of received signals.

15. Medical Imaging: Multipliers are used in medical imaging systems, such as MRI and CT scanners, for signal processing and reconstruction of images.

16. Financial Modelling: In finance, multipliers are used for calculating financial derivatives, option pricing, risk assessment, and portfolio management.

17. Astronomy and Astrophysics: Multipliers are used in data analysis and processing in astronomical observations and simulations.

References

1) Implementation for Multiplying IEEE 754-2008 Binary 32 Bit Number Using Verilog

   Published in: 2014 International Conference on Computational Intelligence and Communication Networks

2) A binary high speed floating point multiplier

   Published in: **2017 International Conference on Nextgen Electronic Technologies: Silicon to Software (ICNETS2)**

3) **FPGA Implementation of 32 Bit Complex Floating Point Multiplier Using Vedic Real Multipliers with Minimum Path Delay**

   Published in: 2018 5th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)