Chapter: 6
Applications of Vedic Mathematics in Computer Arithmetic

6.1 Introduction to Computer:
The electronic circuitry within a computer, the Processing Unit the (CPU) of the computer implements instructions given by a computer program by performing basic arithmetic, logical, control and input/output (I/O) operations. These operations are specified by the instructions from which the actual mathematical operation like addition, Multiplication, Division, squaring and cubing for each instruction is performed. The combinational logic circuit responsible for the above function is known as the Arithmetic logic unit or ALU. Mathematical operations find applications in shifting and recovery of distorted streams of information. Squaring & Cubing has the most essential part in Cryptography, Animation, DSP & image Processing etc. where the speed is a crucial performance characteristic. Speed of DSP is a function of speed of Multiplier, adder, Adder- Subtractor, Division and also square and cube architecture. Thus, it is essential to have a faster ALU to meet the needs of this high performance applications.

6.1.1 Multiplier:
The 3 crucial processes of multiplication are creating, reducing and finally adding the Partial product. If a multiplier has to be efficient, it should have features like Accuracy, speedy performance, and energy efficient system & must occupy few number of slices. Based on time taken and area consumed, the construction of multipliers are divided into 3 classes: The Serial multiplier, Parallel Multiplier & Serial Parallel Multiplier. The first stresses on the optimum usage of the hardware specially the chip area. The second does speedy mathematical operations and the third is a transaction between the first two classes.

Different types of Multipliers:
In signed-2’s complement Booth multiplier represents a method for multiplying binary integers. The process of multiplication of two unsigned binary numbers as well as multiplication of two signed numbers is done by Combinational multipliers. An effective hardware application of a
digital circuit is depicted by the Wallace tree multiplier that multiplies two integers which involves reducing the no. of part products and the addition of the part products is carried out by using carry select adder. The regular structure of Array multiplier it is best suited for the process of repetitive addition and shifting. For low area requirement, Sequential Multiplier is well known.

6.1.2 Adder:
An adder is a digital circuit which can be classified into two types- half Adder and full Adder. The difference between both the type is that by taking sum of 2 digits the first type generates two outputs of a half adder as a carry and sum; whereas the full adder not only adds the sum of the numbers but also keeps record for those values which carried in as well as out. The circuit of 2nd type has varied responsibilities including:

\[ A + B + C_{in} = C_{out} + S \]

Given that input circuit on left side all represent single bit quantities whereas the output on the right is characterized by double bit. S & C stand for Sum and Carry respectively.

The other adders supporting the multiple bits are Carry Save Adder (CSA), Carry look ahead Adder and Ripple Carry Adder (RCA). Among them CSA adds 3 or greater than 3 binary numbers (n-bit) and obtains two outputs of the same dimension as the input in the form of a sequence of carry & partial sum bits; CLA is used to add two binary numbers to reduce the computation time and generates two o/p signals P (Sum-type) & G (Carry-type).

Kogge-Stone adder (KSA) is one of the advanced carry-look ahead architecture. RCA uses multiple full adder for finding the summation of n-bit numbers, in which each full adder depends on the previous one to carry bit for the calculation.

Therefore, RCA is very slow compared to the above two adders.

6.1.3 Types of division algorithm:
In division algorithm, division is of the form \( N/D = (Q, R) \) where inputs are N (Numerator) and Divisor D (Denominator) and Outputs are Q (Quotient) and R (Remainder). There are two main categories of division algorithms: Slow & Fast. Every iteration of the slow division algorithm gives 1 digit of the quotient answer, while Fast division algorithm generates twice as many digits of the final quotient on each iteration because it begins with a near estimation of the
quotient answer. Examples of slow division are Restore type, Non-Restore type and SRT division. Out of them, Non restore type is better than others. Examples of fast division are Newton-Raphson and Goldschmidt.

6.2 Binary Numbering System and Decimal Numbering System:
The binary system is the internal language of electronic computers. Decimal numbering systems has base 10 i.e. It uses 10 digits from 0 to 9. Binary numbering system has base 2 i.e. it uses only two digits 0 & 1.

6.2.1 Conversion Method of Binary number to Decimal number:
This unique numbering method involves the rise of values from right to left digits by positive integer powers starting from zero on the unit's place raised on the common base of two. By neglecting the corresponding digital value of binary digit value 0 and adding the corresponding digital value of binary digit value 1 the decimal number can be obtained.

Convert binary number 10011101 into decimal number equivalent:

<table>
<thead>
<tr>
<th>Decimal Digit Value</th>
<th>$2^7$</th>
<th>$2^6$</th>
<th>$2^5$</th>
<th>$2^4$</th>
<th>$2^3$</th>
<th>$2^2$</th>
<th>$2^1$</th>
<th>$2^0$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>=128</td>
<td>=64</td>
<td>=32</td>
<td>=16</td>
<td>=8</td>
<td>=4</td>
<td>=2</td>
<td>=1</td>
</tr>
<tr>
<td>Binary Digit Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table: 6.1

$$ (10011101)_2 = 1 \times 2^7 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 $$
$$ = 1 \times 128 + 1 \times 16 + 1 \times 8 + 1 \times 4 $$
$$ = 128 + 16 + 8 + 4 + 1 $$
Thus, $$ (10011101)_2 = (157)_{10} $$

Examples:
$$ (10)_2 = 1 \times 2^1 + 0 \times 2^0 = 2, \quad (100)_2 = 2^2 = 4, \quad (1000)_2 = 2^3 = 8, \quad (10000)_2 = 2^4 = 16 $$
$$ (101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5, $$
$$ (1001)_2 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8 + 1 = 9 $$
\[(11111111)_2 = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]
\[= 128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255\]

### 6.2.2 Conversion Method of Decimal Number to Binary Number:

We begin by dividing the said decimal number by 2 continuously and write the result. Since it is being divided by 2, for an even dividend -binary remainder will be Zero & for odd dividend binary remainder will be one. Thus a remainder of either 1 or 0 until the final result equals zero.

Starting with the bottom remainder, read the sequence of remainders upwards to the top.

Convert Decimal number 158 in binary number system equivalent:

**Explanation:**

<table>
<thead>
<tr>
<th>Divisor</th>
<th>Dividend</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>158</td>
<td>2</td>
<td>79</td>
<td>0</td>
</tr>
<tr>
<td>79</td>
<td>2</td>
<td>39</td>
<td>1</td>
</tr>
<tr>
<td>39</td>
<td>2</td>
<td>19</td>
<td>1</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Thus, \((158)_{10} = (10011110)_2\)

Table: 6.2

**OR**

\[(158)_{10} = 128 + 16 + 8 + 4 + 2 = 2^7 + 2^4 + 2^3 + 2^2 + 2^1 = (10011110)_2\]

For unsigned number to convert decimal into binary is easy. But for signed numbers i.e. negative number it is difficult to convert in binary number. In the computer to convert negative number in binary number negative sign can also be represented as bit. Negative binary numbers are represented by Two’s complement notation.
6.2.3 Two’s complement of Binary No.:
Two’s complement is formed as a result of inverting all the bits (i.e. changing each 1 to 0 and each 0 to 1) and then adding 1 to it. The higher order bit of the number (i.e. the leftmost bit) indicates the sign. It indicates positive sign if the leftmost bit is 0 and negative sign if the leftmost bit is 1. The remaining bits represent the magnitude. Thus, using Two’s complement all of the arithmetic operations can be performed by the same hardware whether the numbers are signed or unsigned.

Shortcut method to find the 2’s complement:
Starting from the right i.e. least significant digit to the left i.e. most significant digit, select digit 1 which first time.
Reverse (i.e. change 0 to 1 and 1 to 0) all of the bits to the left of that ‘1’
E.g.
Find 2’s complement of 11010100.
Starting from the right 3rd digit is first 1, so invert all the bits after that 1.
Thus, 2’s complement of 11010100 is 00101100

OR
Invert all the bits and then add 1 to it. i.e. 00101011 + 1 = 00101100

6.3 Arithmetic operations in Binary Numbering System by Vedic Sūtras:

In binary arithmetic operations; Addition, Subtraction, Multiplication and Division are performed by using two bits 0 and 1.
By taking one example of each operation it is explained by the researcher.

6.3.1 Binary Addition:
11100011 + 01111011 = 101011110

11100011
+ 01111011
101011110
6.3.2
Binary Subtraction:
1111000 – 01010100 = 10100100

\[
\begin{array}{c}
1111000 \\
- 01010100 \\
10100100
\end{array}
\]

6.3.3
Binary Multiplication:
111001 \times 101011

\[
\begin{array}{c}
111001 \\
\times 101011 \\
111001 \\
111001 \\
000000 \\
111001 \\
000000 \\
111001
\end{array}
\]

\[
\begin{array}{c}
\text{-------------------} \\
11010011
\end{array}
\]

6.3.4
Binary Division:
111111 ÷ 1001 = 111

\[
\begin{array}{c}
\text{111} \\
1001 \overline{|111111} \\
\text{1001} \\
\text{01101} \\
\text{1001} \\
\text{01001} \\
\text{1001}
\end{array}
\]
Thus, $111111 \div 1001$ then Quotient $Q = 111$ and Remainder $R = 0$

In Octal base is 8 i.e. it uses 8 digits 0,1,2,3,4,5,6, 7 and 8 becomes 10.

$\,(10)_8 = 1 \times 8^1 + 0 \times 8^0 = 8.$

In hexadecimal base is 16 i.e. it contains digits 0, 1, 2,3,5,6,7,8,9

$A = 10, \, B = 11, \, C = 12, \, D = 13, \, E = 14 \, \text{and} \, F = 15.$

**Vedic Sūtras: computer applications:**

In most DSP processor multiplier is one important part of hardware because of that speed of processor increases. Finite Impulse Response filters normally called as a convolution filter (FIR and IIR Filter) includes a multiplier in it. Execution of FFT needs huge no. of multiplications & additions and because of that it becomes very complicated. So it is crucial to have high speed and power consuming multiplier to make it simple and rapid.

In RSA cryptography fix and floating point division is performed. Division is very time consuming process. So there is a need for efficient division architecture to achieve high speed cryptography algorithm in secure transaction.

In Elliptic Curve Cryptography point additions and doubling as exponential operations like squaring, cubing and 4th power occur in these operations as they are more time intensive arithmetic processes. So there is a need to have time, Area and Power efficient Square and Cube architecture.

To overcome the above needs Vedic Mathematics Sūtras are very helpful.

Vedic Mathematics Sūtras are applied in decimal number system as well as binary number system also.

In this chapter Vertically & Crosswise Sūtra based multiplier for all types of multiplication, Nikhila Sūtra based multiplier for the special type of multiplication, Ekanyūnena Pūrveśa Vedic Multiplier and Ānurūpyeśa Sub-Sūtra based multiplier for special type of multiplication increase the modularity while reducing complexity of the design required to input bigger bit
numbers. Binary division by Nikhila Śūtra method for the numbers which are near to base 10 further more simplifying the problem and reducing the number of elements, Parāvartya Yojayet Śūtra as against Nikhila Śūtra for helping in solving bigger problems of binary division, Dhwaja Śūtra a generic method applicable to all type of binary division, squaring by Dwandwayoga means Duplex for a multiplication of number by itself (i.e. for squaring) & also Yāvadūnam Tāvadūnī Krūtya Vargañaca Yojayet Sub-Śūtra for squaring and cubing architecture and Ānurūpye Śūtra for cubing are discussed by the researcher.

Vedic Multiplier, Vedic Squaring and Cubing architecture are the most efficient multiplier output relating to space, less power consumption & rapidity than the conventional multipliers like Sequential, Booth, Array, Wallace Tree and Combinational multipliers.

6.4 Binary Multiplication by using Vedic Śūtra:

6.4.1 By using Vertically & Crosswise Śūtra based Vedic Multiplier:

In DSP Vedic Mathematics algorithm is applied to digital multiplier. Ārdhva-Tiryagbhyām algorithm applied to decimal number system is also applied to binary number system. In binary system base is two. i.e. only 0 and 1 (two bits) are used, therefore multiplication by using Vertically & Crosswise Algorithm can be exchanged by AND logic where every AND would be a bit broad. To generate cross-product these bits are added together.

Algorithm:

Single bit vertical product is carried out between bits of minimum significance (LSBs) of multiplicand and multiplier.

Further calculation is done with 2-bits crosswise product and by incrementally adding by 1 bit until all bits are exhausted.

Moreover decreasing bits from LSB for cross-multiplication and continuing this until MSB (only) is useful for vertical multiplication.

Concatenate all the result by carry over extra bit which is the final answer.

Architecture of Vedic multiplier by Vertically & Crosswise Śūtra:
Vertically & Crosswise Sūtra based Vedic multiplier is applicable for multiplication and it reveals the efficiency of reducing the N × N bit multiplier to 2 × 2 bit structure. The N × N bit multiplier structure is executed by N × N gates and N half adders and (N-2)*N Full adder i.e. total (N-1) * N adders. The 2 × 2 multiplier is executed by 4 input AND gates and 2 half adders. The 4 × 4 multiplier is executed by 16 input AND gates 4 half adders and 8 full adders. An 8 × 8 - bit multiplier is executed by Four 4×4 -bit multipliers and Three 16 × 16 bit adders. With the help of the basic 2 × 2 bit multiplier, multiplier with 4 by 4 bit developed, with the help of 4 × 4 bit block, 8 by 8 bit block, 16 by 16 bit multiplier and then finally 32 by 32 bit multiplier based on Vedic Sūtra has been designed and thus ALU design with Vedic overlay algorithm is efficient related to rapidity, area and power consumption.

Binary Multiplication of 2 x 2 bit:

(By using Vertically & Crosswise Sūtra based Vedic Multiplier)

Let A = A₁ A₀ and B = B₁ B₀ are given two bit nos.

For finding A × B

\[
\begin{array}{c}
A_1 \\
\times \\
B_1 \\
\end{array}
\]

\[
A_1B_1 + A_0B_1 + A_1B_0 + A_0B_0
\]

\[
C_1
\]

\[
\begin{array}{c}
C_2S_2 \\
\downarrow \\
C_1S_1 \\
\downarrow \\
S_0
\end{array}
\]

Thus, \(A_1A_0 \times B_1B_0 = C_2S_2S_1S_0\)

It can be explained by figure using Vertically & Crosswise Sūtra:
Algorithm:
First, start with $1 \times 1$ bit vertically multiplication of the multiplicand ($A_0$) with the least significant Bit of multiplier ($B_0$). i.e. $A \times B_0 = S_0$ which is the LSB of the answer.

Find crosswise multiplication of $2 \times 2$ bit of LSB of multiplier with next higher bit of the multiplicand and evaluate another product similarly by interchanging digits in vice versa format.

Then add both the product and the obtained result gives the second bit of the final answer.

For $2 \times 2$ bit maximum cross product width is 2. i.e. $A_0B_1 + A_1B_0 = C_1S_1$ where the $S_1$ is the second bit of the final answer and carry $C_1$ add to the next step.

Now $1 \times 1$ bit vertically product of the most significant bit (MSB) of multiplicand ($A_1$) with MSB of multiplier ($B_1$) and then add the carry $C_1$ to that product.

i.e. $A_1 \times B_1 + C_1 = C_2S_2$. Here sum $S_2$ is the third bit of the final answer and carry $C_2$ becomes the most significant bit of the final answer.

Thus the final answer is $A_1A_0 \times B_1B_0 = C_2S_2S_1S_0$.

Binary multiplication of 4x4 bit by using Vedic Multiplier:

```
  1111
× 1010
```

```
11111010100110110
111001011010110
11011101010110101
carry 10 0 1 0 1 1 0
Thus, 1111 × 1010 = 10010110
```

Explanation:
Starting multiplication either from LSB or MSB with $1 \times 1$ bit, then continuing $2 \times 2$, $3 \times 3$ and till $4 \times 4$ and again decreasing order of bit $3 \times 3$, $2 \times 2$ and $1 \times 1$ bit.
Thus, $1111 \times 1010 = 10010110$

In $N \times N$ bit multiplication maximum cross product width will be $\log_2 N + 1$.

In $8 \times 8$ bit multiplication maximum cross product width will be $\log_2 8 + 1 = 4$ and for $4 \times 4$ it will be $\log_2 4 + 1 = 3$ for $2 \times 2$ it will be $\log_2 2 + 1 = 2$.

There is alternative way for finding multiplication by using Vertically & Crosswise Sūtra which is explained by the following figure.

An alternative way of $4 \times 4$ bit multiplication: Calculate $1111 \times 1010$
Thus, $1111 \times 1010 = 10010110$

As the figure above bits of multiplier and bits of multiplicand are written to the consecutive sides of a square.

The horizontal and vertical arrangements of numbers carry information about the bits of quantities whose product is to be found.
These multiplier bits are multiplied with each bits of multiplicand independently and write the product in the common box.

Initially, carry for RHS start is assumed 0 as per the standard format.

Addition is done diagonally along the slant path highlighted along with an additional entity called the preceding carry-over from neighbour.

From obtained product, the part of the final answer possessing minimum significance is retained whereas the remaining is passed to next neighbour as carry-over and this operation continues till it reaches the bit of greatest significance & thus, the desired product is obtained.

**Binary multiplication of 8x8 bit by using Vedic multiplier:**

Calculate: $10111111 \times 11000011$

\[
\begin{array}{cccccccccc}
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\times & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
\hline
1 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\end{array}
\]

Thus, $10111111 \times 11000011 = 1001000101111101$

**Explanation:**

\[
\begin{array}{cccccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
\times & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
\hline
1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

Thus, $10111111 \times 11000011 = 1001000101111101$
Example of 8 × 8 bit Binary Multiplication:

Calculate: 11111111 × 11111111

\[
\begin{array}{cccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
\end{array}
\]

Thus, 11111111 × 11111111 = 1111111000000001

16×16 bit and 32×32 bit multiplication can also be done by using Vedic multiplier.

6.4.2 Binary multiplication by using Nikhilam Sūtra based Multiplier:

Nikhila Sūtra based multiplier is more efficient in the multiplication of large numbers as it reduces the multiplication of two large numbers to that of smaller.

The above Sūtra can also be applicable to binary number system. In binary system deviation from base can also be found by taking 2’s complement of that number. For an eight bit Vedic Multiplier, the R.H.S. part of the product is implemented using 8-bit carry save adder. Hence multiplication of two 8-bit numbers is reduced to the multiplication of their compliments and addition. Nikhila Sūtra is basically more efficient when both the multiplier and multiplicand are near to same base power. For binary numbers base is in the power of 2.

e.g. \( 2^1 = 2 = (10)_2, 2^2 = 4 = (100)_2, 2^3 = 8 = (1000)_2 \)

Algorithm:

First find the nearest same base bits of the given multiplier and multiplicand both.
Then find deficit bit by subtracting the base bit from both multiplier and multiplicand.
Obtain the adjusted insufficiency and place these results across corresponding row of multiplier and multiplicand.
Multiply that two deficit bits which is the R. H.S. of the final answer.
Then either add the deficit of multiplicand to the given multiplier or add the deficit of multiplier to given multiplicand. The obtained result is the L.H.S. of the final answer.

2×2 bit:
Calculate: 11 × 11

Here base is 10

\[
\begin{array}{c|c}
11 & \text{(11-10) = 1} \\
\times 11 & \text{(11-10) = 1} \\
\hline
(11 + 1) \ | (1 \times 1) \\
100 \ | \ 1
\end{array}
\]

Thus, 11 × 10 = 1001

3×3 bit:
Calculate: 101 × 110

Here base is 100

\[
\begin{array}{c|c}
101 & \text{(101-100) = 1} \\
\times 110 & \text{(110-100) = 10} \\
\hline
(110 + 1) \ | (1 \times 10) \\
111 \ | \ 10
\end{array}
\]

Thus, 101 × 110 = 11110

4×4 bit:
Calculate: 1111 × 1111

Here base is 1000

\[
\begin{array}{c|c}
1111 & \text{(1111-1000) = 111} \\
\times 1111 & \text{(1111-1000) = 111} \\
\hline
(1111+111) \ | (111 \times 111) \\
10110 \ | (111 \times 111) \\
10110 \ | 110001 (111 \times 111 = 110001)
\end{array}
\]
Thus, \(1111 \times 1111 = 11100001\)

**Explanation of \(111 \times 111\) used in above example:**

\[
111 \times 111 \text{ (Base is 100)} \\
111 \times 111 \text{ (111-100) = 11} \\
\times 111 \times 111 \text{ (111-100) = 11} \\
(111+11) \times (11 \times 11) \\
1010 \times (11 \times 11) \\
1010 \times (1001) \\
1010 \times 101 \\
\underline{10} \text{ Carry (base 100)} \\
1100001 \text{ Thus } 111 \times 111 = 1100001
\]

**Explanation of \(11 \times 11\) used in above example:**

\[
11 \times 11 \text{ (base 10)} \\
11 \times 11 \text{ (11-10) = 1} \\
\times 11 \times 11 \text{ (11-10) = 1} \\
(11 + 1) \times (1 \times 1) \\
100 \times 1 \text{ Thus, we get } 11 \times 11 = 1001
\]

**Explanation by alternative way:**

<table>
<thead>
<tr>
<th>Calculation for 4 x 4 bit</th>
<th>Base difference</th>
<th>Next difference</th>
<th>Next difference</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Calculation for 4 x 4 bit</th>
<th>Base difference</th>
<th>Next difference</th>
<th>Next difference</th>
</tr>
</thead>
</table>
\[
\begin{align*}
1111 \times 1111 &= (111 + 111) \times 1000 + (11 \times 111) \\
&= 10110 \times 1000 + 11001 \\
&= 10110000 + 110001 \\
&= 11100001 \\
\text{Thus,} \\
1111 \times 1111 &= 11100001
\end{align*}
\]

\[
\begin{align*}
(1111 - 1000) &= 111 \\
(11 + 11) \times 100 + (11 \times 11) &= 1010 \times 100 + 1001 \\
&= 101000 + 1001 \\
&= 110001 \
\text{Thus,} \\
111 \times 111 &= 110001
\end{align*}
\]

\[
\begin{align*}
111 - 100 &= 11 \\
(11 + 1) \times 10 + 1 &= 100 \times 10 + 1 \\
&= 1000 + 1 \\
&= 1001 \\
\text{Thus,} \\
11 \times 11 &= 1001
\end{align*}
\]

\[
\begin{align*}
11 - 10 &= 1 \\
1 \times 1 &= 1
\end{align*}
\]

Table: 6.3

In the same way 11111111 \times 11111111 \times (8 \times 8 \text{ bit}) can be solved as below in table: 6.4

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{Next difference} & 11 - 10 &= 1 & 1 \times 1 &= 1
\hline
\text{Next diff.} & 111 - 100 &= 11 & (11 + 1) \times 10 + 1 &= 100 (10) + 1 \\
\hline
\text{Next diff.} & 1111 - 1000 &= 111 & (111 + 111) \times 10 + 1 &= 10100 \times 100 + 1 \\
\hline
\text{Next diff.} & 11111 - 10000 &= 1111 & (1111 + 1111) \times 100 + 1 &= 101110 \times 1000 + 1 \\
\hline
\text{Next diff.} & 111111 - 100000 &= 11111 & (11111 + 11111) \times 1000 + 1 &= 1101100 \times 110000 + 1 \\
\hline
\end{array}
\]
With rise in technological advances and research in material science, future trends predict complete eradication of outdated irreversible operations of logic elements which thrive on deleting valuable information during processing. The inherent characteristic of these futuristic inventions will rely on the Nikhila Sūtra algorithm helping in fast and effective computational performance.

### 6.4.3 Binary multiplication by Ānurūpyeṇa Sub-Sūtra based Multiplier:

When both the numbers are near the same base but other than the theoretical base like 10, 100, 1000, 10000…… and if we apply Nikhila Sūtra then after subtracting the given multiplier and multiplicand from the base we get the result very big number. By using Ānurūpyeṇa Sub-Sūtra means proportionately find the working base and compare it with theoretical base and find the proportion between the two bases. In the final answer only left part is proportionately multiply.
By Nikhilam Method

1101 \times 1100

By Nikhilam nearest base is 1000

1101 \quad (1101 - 1000) = 101
\times 1100 \quad (1100 - 1000) = 100

(1101+100) or (1100 +101) \mid 101 \times 100

10001110100

10001+10 \mid 100

10011000 \quad \text{Thus, } 1101 \times 1100 = 10011100

By Anurūpyeṣa Method

By using Anurūpyeṣa, theoretical base is 100

Working base is 1100 = 11 \times 100

1101 \quad (1101-1100) = 01
\times 1100 \quad (1100 -1100) = 00

(1100 + 01)(1100 + 00) \mid 100

1101 \mid 100

1101 \times 11 \mid 00

10011100 \quad \text{Thus, } 1101 \times 1100 = 10011100

6.4.4 Binary multiplication by Ekanyūnena Pūrveṣa Sūtra based multiplier:

Ekanyūnena Pūrveṣa Sub-Sūtra means, “one less than the previous one”. The greatest benefit is that this sūtra comes handy during hard calculations involving figures of type 9, 999, 999, 9999 & so on.

Avoiding unnecessary complexity significantly reduce the time delay and makes the system faster.

In Binary system base is 2. This Sūtra is useful for multiplication of numbers with the multiplier which are in the form of $2^n – 1$.

i.e. Multiplication of binary number with multiplier like 1, 11, 111, 1111, 11111……..

First subtract 1 from the multiplicand i.e. given binary number other than multiplier. The obtained result is the left part of the final answer.
The obtained result subtract from the multiplier (which is in the form of 1, 11, 111, 1111…….). That result is the right part of the final answer.

E.g.

Calculate: 1111 × 1000

(1000 - 1) = 0111, which is the left part of the final answer.

(1111 - 0111) = 1000, which is the right part of the final answer.

```
  1 1 1 1
× 1 0 0 0
0 1 1 1 1 0 0 0
```

The final answer 1111 × 1000 = 01111000.

Calculate: 10101 × 11111

(10101 - 1) = 10100

(11111 - 10100) = 01011

```
  1 0 1 0 1
× 1 1 1 1 1
1 0 1 0 0 1 0 1 1
```

Thus, 10101 × 11111 = 101001011

6.5 Binary Division by using Vedic Sūtras:

6.5.1 Binary Division Algorithm by using Nikhila Sūtra:

Let A be the dividend (numerator), B is the divisor (denominator) and N represents number of bits of the divisor. First find the deficit bits by subtracting 10N from the divisor. Separate the dividend bits into two parts one for the quotient bits and another for remainder bits such that Remainder bits is equal to the divisor bits and the number of dividend bits – number of divisor bits = number of quotient bits.

Write the dividend bits and remainder bits in the first row and write first bit of the dividend part under the first bit below the last row in the first column which is the first bit $Q_1$ of the final
answer of the quotient. Then find product of deficit bits $D_1 D_2$ and $Q_1$ which is $P_1 P_2$ write it below the second and third bits of dividend.

Add the second bit of dividend to $P_1$ and write it below the last row as $Q_2$.

Till the last bit continue this process. The last quotient bit considered for finding remainder $R_1$, $R_2$.

**Example of binary division by Nikhila Sūtra:**

[1] $1100110 \div 1001$

Nearest base of $1001 = 10000$, Deficit bits = $10000 - 1001 = 0111$

OR Deficit Bits = 2’s complement of $1001 = 0111$

<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>Dividend bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1</td>
<td>1 1 0 0 1 1 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deficit Bits:</th>
<th>Bits allotted for finding quotient bits</th>
<th>Bits allotted for finding remainder bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 1</td>
<td>1 1 0</td>
<td>0 1 1 0</td>
</tr>
<tr>
<td></td>
<td>0 1</td>
<td>1 1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1 1 1</td>
</tr>
<tr>
<td></td>
<td>0 1 1 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 1 1 1</td>
<td>10 100 11 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10 1</td>
</tr>
<tr>
<td></td>
<td>1 1 1 1</td>
<td>100 1 1 1</td>
</tr>
</tbody>
</table>

Therefore, $Q = 1 1 1$ $R = 100 111$

Table: 6.5

Here, Remainder is greater than Divisor. So considering Remainder as dividend divide it by the divisor same process as above.

<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>Dividend bits = Previous Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1</td>
<td>1 0 0 1 1 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deficit Bits:</th>
<th>Bits allotted for finding</th>
<th>Bits allotted for finding</th>
</tr>
</thead>
</table>
Table: 6.6

Therefore, \( Q = 111 + 10 = 1101 \) \( R = 10101 \)

Remainder is also greater than the divisor. Repeat the above procedure.

Table: 6.7

Therefore, \( Q = 111 + 10 + 1 = 1010 \) \( R = 1100 \)

Here also remainder is greater than divisor. Considering the remainder as dividend repeat the procedure till remainder is either equal or less than the divisor.
Deficit Bits: 0 1 1 1  

<table>
<thead>
<tr>
<th>Bits allotted for finding quotient bits</th>
<th>Bits allotted for finding remainder bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 1 0 0</td>
<td>0 1 1 1</td>
</tr>
<tr>
<td></td>
<td>0 1 1 1</td>
</tr>
<tr>
<td></td>
<td>0 1 1 1</td>
</tr>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0 1 0 1 1</td>
</tr>
<tr>
<td></td>
<td>1 0 0 1 1</td>
</tr>
<tr>
<td></td>
<td>1 0 0 1 1</td>
</tr>
</tbody>
</table>

Therefore, Final quotient $Q = 111 + 10 + 1 + 1 = 1011$, Final $R = 0 0 1 1$

Table: 6.8

Here, remainder is less than divisor so Final answer is $Q = 1011$ and $R = 0011$

[2] $1101011 \div 1101$

<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>Dividend bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1</td>
<td>1 1 0 1 0 1 1</td>
</tr>
</tbody>
</table>

Deficit Bits: 0 0 1 1

<table>
<thead>
<tr>
<th>Bits allotted for finding quotient bits</th>
<th>Bits allotted for finding remainder bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>0 0</td>
<td>1 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 0 0 0 0</td>
<td>1 0 1 1</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>10 10 10 1</td>
</tr>
<tr>
<td></td>
<td>1 1</td>
</tr>
<tr>
<td>1 1 1 0</td>
<td>11 1 0 1</td>
</tr>
</tbody>
</table>

Therefore, $Q = 110$ and Remainder $= 11101$

Table: 6.9

Here remainder is greater than the divisor. Considering the obtained remainder as dividend continuing the above procedure.
<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>Dividend bits = previous remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1</td>
<td>1 1 1 0 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deficit Bits:</th>
<th>Bits allotted for finding quotient bits</th>
<th>Bits allotted for finding remainder bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1</td>
<td>1 1 1 0 1</td>
<td>1 1 1 1 0</td>
</tr>
<tr>
<td></td>
<td>0 0 1 1</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td></td>
<td>1 1 1 1 10</td>
<td>1 1 1</td>
</tr>
<tr>
<td></td>
<td>1 1 1</td>
<td>1 0 0 0 0</td>
</tr>
</tbody>
</table>

Therefore, \( Q = 110 + 1 = 111 \quad \& \quad R = 10000 \)

Table: 6.10

Remainder is greater than the divisor. So considering the remainder as divisor continuing the same procedure as above.

<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>Dividend bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1</td>
<td>1 0 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deficit Bits:</th>
<th>Bits allotted for finding quotient bits</th>
<th>Bits allotted for finding remainder bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 1 1</td>
<td>1 0 0 0 0</td>
<td>0 0 1 1</td>
</tr>
<tr>
<td></td>
<td>0 0 1 1</td>
<td>1 0 0 0 0</td>
</tr>
<tr>
<td></td>
<td>1 0 0 1 1</td>
<td>1 0 0 1 1</td>
</tr>
</tbody>
</table>

Therefore, Final Quotient \( Q = 111 + 1 = 1000 \) and Final \( R = 0 0 1 1 \)

Table: 6.11

### 6.5.2 Binary Division by Parāvartya Yojayet Sūtra:

Process of Parāvartya division is similar to the Nikhila\(\) method but here the Deficit bits are replaced with transposed bits of denominator bits. Parāvartya Yojayet Sūtra means transpose and Apply.
Take the transpose of the bits of the divisor by keeping first bit of divisor as it is and change the sign of the remaining bits of the divisor and dividing the dividend bits by that transposed bits, Quotient and Remainder is obtained.

**Examples:**

[1] $1100110 \div 1001$

<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>Dividend bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1</td>
<td>1 1 0 0 1 1 0</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>Bits allotted for finding quotient bits</td>
</tr>
<tr>
<td>0 0 -1</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td></td>
<td>0 0 -1</td>
</tr>
<tr>
<td></td>
<td>0 0 -1</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 0 1</td>
</tr>
<tr>
<td></td>
<td>1 1 0 -1</td>
</tr>
<tr>
<td></td>
<td>0 1 1</td>
</tr>
</tbody>
</table>

**Therefore,** Quotient $Q = 1000 + 100 - 1 = 1101$ and $R = 0 1 1$

Table: 6.12

[2] $1101011 \div 1101$

<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>Dividend bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1 0 1</td>
<td>1 1 0 1 0 1 1</td>
</tr>
<tr>
<td>1 1 0 1</td>
<td>Bits allotted for finding quotient bits</td>
</tr>
<tr>
<td>-1 0 -1</td>
<td>1 1 0 1</td>
</tr>
<tr>
<td></td>
<td>-1 0 -1</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

**Therefore,** Quotient $Q = 1000 + 100 - 1 = 1101$ and $R = 0 1 1$
Table: 6.1

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quotient Q = 10000 and R = 011</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.5.3 Binary Division or Direct Flag Division by using Dhwajaṅka property of Vertically & crosswise Sūtra:

Even though conventional division works fine for low to moderate range numbers, the traditional method becomes notoriously difficult when dealing with extreme figures. In such cases, Dhwajaṅka feature of VM can come to our rescue working as a short cut to lengthy calculations.

In this process, division is carried out in parts & thus, a big clumsy operation reduces to several small easier ones. This splitting is carried out among all parameters of remainder, divisor, quotient and dividend.

Division takes place bitwise in a specific order & according to standard guidelines. Remaining bits of the divisor are defined as flag type. Since we are not engaging with the entire combination of bits at one time but flagging after specific intervals, the overall effort required drastically drops down.

Intermediate dividends like Flag (FD), Gross (GD), Net (ND) are evaluated which in turn are employed to find the desired solution in terms of Quotient & Remainder.

Mean computation period for this novel method can be verified to be relatively much lower than regular technique especially for larger quantities.

Also for binary numbers the division method is similar to the division process for decimal numbers,

Examples:

[1] 11011101 ÷ 1101

<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>1 1 0 1</th>
<th>Dividend bits</th>
<th>1 1 0 1 1 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Therefore, Quotient $Q = 1000$ and $R = 101$

Table 6.14

Remainder = $101 - [100(0) + 10(0) + 0] = 101 - 0 = 101$

[2] $1100110 \div 1001$

<table>
<thead>
<tr>
<th>Divisor bits</th>
<th>Dividend bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 1</td>
<td>1 1 0 0 1 1 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 1 0 1 0 0 0</td>
</tr>
</tbody>
</table>

Therefore, Quotient $Q = 1100$ and $R = 010$

Table: 6.15

Remainder $= 110 - [100(1) + 10(0) + 0] = 110 - 100 = 010$

6.6 Binary Squaring:

6.6.1 Algorithm for finding Binary Square by using Yāvadūnam TāvadūnīkarītyaVarganca yojayet Sub-Sūtra:

The Yāvadūnam Sūtra with the same procedure as in decimal system is applied for the binary number system (Base $= 2^n$) in which for finding squares. Algorithm for finding square is as follows.

As the base is always greater than n-bit number, find this deficiency. Square of that deficiency is the R.H.S. of the final answer.

This methodology of defecit estimation resembles the steps involved in twos complement calculation.

The L.H.S. of the final answer is obtained by subtracting the deficiency of the given n-bit number (which is already found in step-1) from the given n-bit number of which we want to find
out the square. The number of bit in the R.H.S. of the final answer depend on the base. E.g. If 
base is $2^2$ (i.e. 4 = 100) then R.H.S. is of 2 bits.
The subtraction operation of the deficit of the n-bit number can also be competent by a left shift 
operation. This can be explained as left shift by a single bit ignoring the 
(n-1)$^{th}$ bit and assigning the value ‘0’ to LSB.
By taking an example the algorithm of square architecture by using Yāvadūnam Sūtra can be 
explained:

E.g. Calculate $(1101)^2$

**Algorithm of Square Architecture by using Yāvadūnam:**

Considering a 4-bit binary number say $(1101)_2 = (13)_{10}$, then the nearest base of $(13)_{10}$ in the 
form of power-2 is $2^4 = (16)_{10}$, i.e. $2^4 = (16)_{10} = (10000)_2$

**Step-1:** $10000 - 1101 = 0011$ which is the deficit from the nearer base which is exactly equal to 
complement of two.

**Step-2:** The square of Deficit i.e. $(1101)^2 = 1001$ which is the R.H.S. of the final answer.

**Step-3:** Subtract the deficit from 1101 i.e. 1101-0011=1010 which is the L.H.S. of the final 
answer. This can be explained as left shift operation of the given 4-bit binary number 1101. By 
ignoring the $(4-1)^{th}$ bit i.e. 3$^{rd}$ bit and assigning LSB to 0 we get the result as 1010 which is 
exactly same as the output what we obtained by subtracting the deficit 0011 from the 4-bit 
number 1101.

**Step-4:** Concatenate the L.H.S. and the R.H.S. and by carry over the extra digit starting from the 
right side the final answer can be obtained.

Thus, $(1101)^2 = 10101001$

The Yāvadūnam Sūtra based squaring architecture is time and area efficient than the 
conventional method of squaring. So it may be used in computer graphics, Cryptography and 
implementation in ALU circuits.

6.6.2 Duplex property of Ūrdhva- Tiryagbhyām Sūtra:

First of all, the figure in the problem is analyzed & identified whether the string length is of even 
nature.Next, pairing is done from the outward to inward direction. These newly formed pairs are
multiplied by two and the results are noted. In case of odd string length, the unpaired central term is squared. These individual results are combined to evaluate the value of final solution.

**Algorithm:**

For n-bit number starting from MSB to find the duplex with 1 bit.
Then find duplex by decreasing 1 bit continuously till we reach the LSB with 1-bit. Concatenate all the result to get the final answer.

Squaring is done by using Vertically & Crosswise Sūtra by multiplication of given binary number with itself.
Comparing the answers of the Vertically & Crosswise Sūtra starting with the Least Significant Bit of each steps one by one with the answers of Duplex property starting with 1-bit, it can be found that both the answers are same.
Therefore, the duplex property is known as the duplex property of Vertically & Crosswise Sūtra.

**Algorithm for Binary number for calculating **(1101)**²:**

**Duplex Property:**

<table>
<thead>
<tr>
<th>Step</th>
<th>Calculation</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>D (1) = 1²</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>D (11) = 2<em>1</em>1=2 = (10)₂</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>D (110) = 2<em>1</em>0 +1² = 1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>D (1101) = 2<em>1</em>1 +2*1 *0 =2+0 = 2 = (10)₂</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>D (101) = 2<em>1</em>1 + 0² =2= (10)₂</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>D (01) = 2<em>1</em>0 = 0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>D (1) = 1² =1</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ (1101)^2 = \begin{array}{cccccccc}
1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\end{array} \]

Thus, \((1101)^2 = 10101001\)
By using \textit{Ūrdhva - Tiryagbhyām Sūtra}:

\[
\begin{array}{cccc}
1 & 1 & 0 & 1 \\
\times & 1 & 1 & 0 & 1 \\
\hline
1 & 10 & 1 & 10 & 10 & 0 & 1 \\
\hline
1 & 1 & 1 & 1 & \text{carry} \\
10 & 1 & 0 & 1 & 0 & 0 & 1 \\
\end{array}
\]

Thus, \((1101)^2 = 10101001\)

A novel technique of sūtra squaring has been devised to compliment the pros of speed algorithms and robustness of traditional multipliers

Similarly, Binary Square of 8-bit, 16-bit and 32-bit can be find out by the same procedure.

\textbf{6.7 Binary cubing:}

\textbf{6.7.1 Yāvadūnam Sūtra based cubic algorithm:}

Just like the square of a number cube of a number can be find out by using Yāvadūnam

\textbf{Algorithm for finding cube of a binary number:}

First find the deficit of the number double the deficit and add to the original number. New number can be obtained which is the L.H.S. of the final answer.

I.e. New number = original number + \((2 \times \text{deficit})\)

Find new deficit from the new number obtained in the above step

And multiply new deficit and deficit of the original number which is the middle part of the final answer.

Take the cube of the old deficit which is the R.H.S. of the final answer.

Thus we get,

\[(\text{Original number})^3 = \text{original number} \]
\[+ (2 \times \text{deficit}) | (\text{original deficit} \times \text{new deficit}) | (\text{original deficit})^3\]

Concatenate all the three parts final answer can be obtained.
E.g: \((101)^3\)

Nearest base is Original deficit = 101 - 100 = 001
New number = 101 + 10 \times (001) = 111
New deficit = 111 - 100 = 011
New Deficit \times Original deficit = 1 \times 011 = 011
Cube of original deficit = (1)^3 = 01
Thus,
\[(101)^3 = 11011101\]

The final answer = 1111101

6.7.2 Ānurūpye a cubic algorithm:

Algebraic identity:
\[(a + b)^3 = (a^3 + 3a^2b + 3ab^2 + b^3)\]

```
a^3    a^2b   ab^2   b^3
   2(a^2b) 2(ab^2)     
```
(Double the middle two terms of the first row)
```
a^3  3a^2b  3(ab^2)   b^3
```

Algorithm for finding cube of a binary number by Anurūpye a:

Let n-bit binary number M is given. First divide the given number in two partition of n/2 bits which is A and B.

Find the cube of A (M.S.B.) which is L.H.S. of the final answer and the cube of B (L.S.B.) which is the R.H.S. part of the final answer.

For finding intermediate terms calculate the square of L.S.B and multiply it with M.S.B. and calculate the square of M.S.B. and multiply it with L.S.B.

Multiply the intermediate term with 11 (i.e. in decimal by 3).
Concatenate all the three parts which is the final result.

E.g.: Find \((11)^3\)
Let \(A = 1\) and \(B = 1\)
\[(\text{M.S.B.})^3 = 1^3 = 1 = \text{L.H.S. of the final answer and}\]
\[(\text{L.S.B.})^3 = 1^3 = 1 = \text{R.H.S. of the final answer}.\]

For finding two intermediate terms by using vertically and crosswise Sūtra:

\[
\begin{array}{c}
\text{(M.S.B.) } = 1 \\
\text{L.S.B. } = 1 \\
\text{(M.S.B.) }^2 = 1 \\
\text{(L.S.B.) }^2 = 1
\end{array}
\]

Figure: 6. 6

For the final result concatenate all the parts.

\[
\begin{array}{c}
1 \mid 1 \mid 1 \mid 1 \\
10 \mid 10
\end{array}
\]

\[
\begin{array}{c}
10 \mid 1 \\
1 \mid 0 \mid 1 \mid 1
\end{array}
\]

Final answer = \((11)^3 \) = 11011

6.8 Applications and advantages of Vedic Sūtras in computer:

Vedic Mathematics Sūtras for multiplication, division and also for squaring and cubing operations are applied by many scholars belonging to areas like DSP, DFT, VLSI algorithm (with less power & greater speed), Algorithm to prepare new design of chip & RSA Encryption system.

FPGA implementation of block convolution, new multiplier architecture developed and embedded into OLA and OLS method by using Urdhva -Tiryagbhyam has improved efficiency in terms of area.
After comparing 4×4 bit Vedic multiplier on two different adder architectures concluded that the 4×4 bit Vedic multiplier with **carry look adder** is faster than 4×4 bit Vedic multiplier with ripple carry adder.

High speed reconfigurable FFT designed and implemented by using Urdhv-Tiryagbhyam algorithm boast rapid operations, lossless propagation and compact aesthetics.

After studying performance analysis of integrated multiplier architecture using **Urdhv-Tiryagbhyam & Nikhila Sūtra**, it is found that **Urdhv-Tiryagbhyam** works faster for small input and as the size of multiplication increases **Nikhila** multiplier works better. Based on initial condition only one multiplier Sūtra performs the multiplication at any given time.

The speed of 4×4 bit multiplier with reversible logic by using Vedic Sūtra **Urdhv-Tiryagbhyam** increased and negligible power can be achieved by reversible logic gate. After the performance analysis among multipliers like **Urdhv-Tiryagbhyam, Nikhila & Karatsuba** multiplier; for 8 × 8 bit Vedic Urdhva-Tiryagbhyam multiplier and for 16 × 16 bit Vedic Nikhila multiplier is faster. The 8 × 8 bit multiplier by Vertically & Crosswise Sūtra is more efficient than Array and Booth multiplier.

After comparison of various Vedic multiplication techniques like **Urdhv-Tiryagbhyam, Nikhila & Ānurūpya Sūtra** it is found that **Urdhv-Tiryagbhyam** is most efficient Sūtra giving minimum delay for all types of numbers and reduces the area and speed up the computation. Design of 64 bit Vedic multiplier by using Nikhila Sūtra has less delay and area compared to array and booth multiplier. The hardware implementation of 64 × 64 bit Vedic Nikhila multiplier using Barrel shifter contributes to adequate improvement of the speed in order to achieve high outturn than the conventional array multiplier.

A particular FFT which is designed by Vedic adder, subtractor and multiplier by using Vertically & Crosswise Sūtra provides feasible & rapid performance in the area of Discrete Fourier Transform and also rarely experiences any lag complaints.
RSA encryption and decryption algorithm by using Vertically & Crosswise Sūtra for multiplication and Dhwaja ka property for division reduces time and reduces delay prominently as compared to the RSA implemented using traditional multipliers and division algorithm.

Implementation of fix and floating point division by Dhwaja ka property of Vertically & Crosswise Sūtra is more efficient to achieve high speed cryptography algorithm in secure transaction.

After comparison results shows that the proposed Vedic square architecture by using Vedic Yāvadūnam Sūtra have both speed and area improvement over existing architecture.

Thus, Application of Vedic Sūtras improves the computational skill in a wide-ranging problems and is rigorously centered on lucidity & common-sense with great swiftness as well as accuracy.