FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Veikko Vähäsöyrinki

STUDY ON BIT PARALLEL AND SERIAL ARITHMETIC LOGIC APPROACHES

Bachelor's Thesis
Degree Programme in Computer Science and Engineering
March 2023

Vähäsöyrinki V. (2023) Study on Bit Parallel and Serial Arithmetic Logic Approaches. University of Oulu, Degree Programme in Computer Science and Engineering, 21 p.

ABSTRACT

This paper provides general overview of how computers process numbers and how computers do arithmetic. Different ways to implement digital arithmetic logic are presented. Bit-serial designs can save chip real estate, but require more clock cycles for arithmetic operations such as additions and multiplications. Bit-parallel designs produce results with fewer clock cycles, but require more gates, e.g., due to carry-look-ahead generators. This may translate into higher power dissipation.

This BSc thesis presents an exploration of bit-serial-parallel and bit-parallel arithmetic logic designs. The intention is to gain understanding of their basic design characteristics.

Keywords: VHDL, multiplication, addition, clocking

Vähäsöyrinki V. (2023) Rinnan ja peräkkäin bitit käsittelevien logiikkojen tarkastelu. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 21 s.

TIIVISTELMÄ

Tämä tutkimus antaa yleiskatsauksen tietokoneiden tapaan prosessoida numeroida ja suorittaa laskutoimituksia. Tässä esitellään erilaisia tapoja implementoida digitaalista laskulogiikkaa. Bitti kerrallaan - toteutus voi säästää sirutilaa, mutta se vaatii enemmän kellojaksoja operaatioihin kuten yhteen- ja kertolasku. Bitit rinnakkain-toteutus vähentää kellojaksoja, mutta vaatii enenmmän logiikkaportteja. Tämä voi johtaa virtahävikkiin.

Tämä kandidaatin tutkielma paneutuu bitti rinnakkain-kerrallaan ja bitti rinnakkain- arithmetiikkalogiikkatoteutukseen. Tarkoitus on saada ymmärrystä niiden toteutuksien perusteista.

Avainsanat: VHDL, kertolasku, yhteenlasku, kellotus

TABLE OF CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

Bit	Smallest unit of information in binary number system
Bit-serial	Processing bits 1 bit at a time
Serial-parallel	Processing some bits in serial and some bits in parallel
Bit-parallel	Processing all bits at the same time
Ripple-carry adder	Bit-parallel adder where the carry propagates
Binary number system	Base-2 numeral system
Decimal number system	Base-10 numeral system
Digital system	System which processes discrete numbers
Carry bit	A bit that has information of arithmetic overflow
Sign bit	A Bit that has the sign information of a number
MSB	Most significant bit - the bit that has the largest value
LSB	Least significant bit - the bit that has the smallest value
Register	A Digital circuit that stores bits
D Flip-flop	A Digital circuit that stores 1 bit
Clock frequency	Duration of a clock cycle in hertz
Combinational logic	Digital logic which does not rely on clock signal
Sequential logic	Digital logic that depends on clock cycles to operate
RTL	Register Transfer Level - A layer of abstraction in digital logic
Logic synthesis	A process where RTL code is generated into logic gates
Critical path	The path from input to output with a maximum delay

1. INTRODUCTION

Design and implementation of digital arithmetic logic has always depended on the available technologies. All early computers were serial computers, meaning they process data 1 bit at a time. This is because in the days of early computers logic was very expensive. Bit-serial system is cheaper to implement due to its simplicity over parallel system. Bit-serial processing requires less logic, but it is much slower than bit-parallel processing.

Figure 1 shows a basic 4-bit parallel ripple-carry adder. Ripple-carry adder is not very optimal, but it shows well the nature of bit-parallel adder. Ripple-carry adder has usages in shorter word lengths. In longer word lengths carry look-ahead logic is needed to make bit-parallel adders efficient with short critical path. Carry look-ahead logic block can be seen in Figure 2. The carry look-ahead block takes both operands as an input and generates carry to the full adders. This avoids the long critical path of ripple-carry adder. In the bit-parallel design, all inputs are loaded at the same time, and the adder processes all bits at the same time. Combinational logic increases when increasing the word size in bit-parallel adders.

Figure 3 presents a basic 4-bit serial adder. Serial arithmetic requires the use of shift registers. Bit-serial block only has 1 full adder, where the bit-parallel adder has 4. In contrast, the bit-serial design only processes 1 bit from the inputs at a time. Serial-parallel design usually processes one input in serial and the other in parallel. Bit-serial adders are very flexible when increasing word lengths - it only needs to increase the size of the registers. When designing massively parallel computers, bit-serial and bit-parallel arithmetic blocks are at the opposite solutions where a serial-parallel solution is somewhere in between.

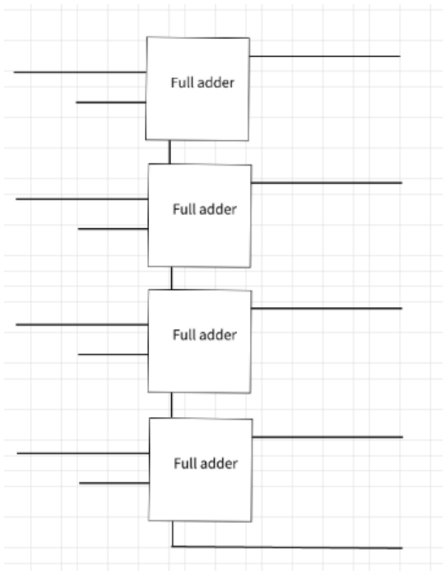

Figure 1. 4-bit bit-parallel adder.

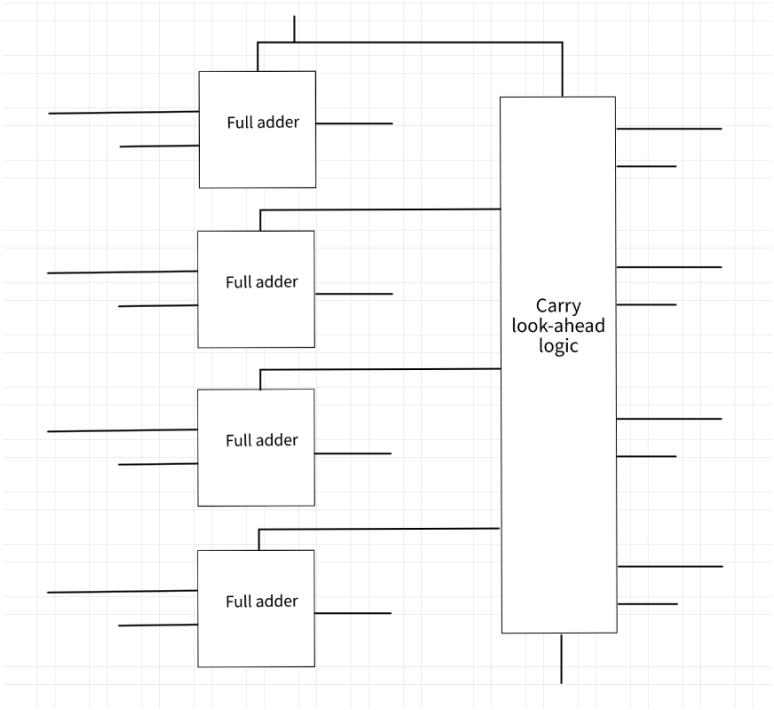

Figure 2. 4-bit bit-parallel adder with carry look-ahead.

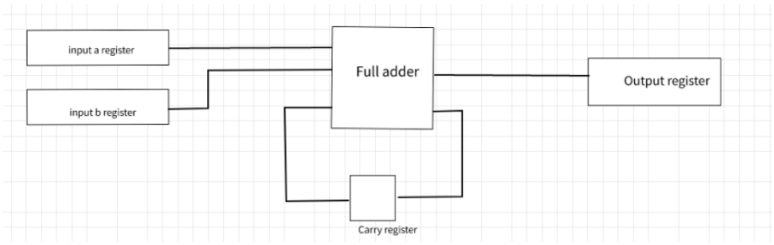

Figure 3. 4-bit bit-serial adder.

Some estimations about the gate counts can be made between these adders. In a study [1] 64-bit ripple carry adder and carry look-ahead adders are compared: Ripple carry adder has 320 logic gates and carry look-ahead adder has 256 logic gates in 64-bit configuration. Note that 64-bit ripple carry-adder would not be feasible due to the long critical path. If a D flip-flop has 6 gates then 3 64-bit registers would have $64 * 6 = 384 * 3 = 1152$ gates. Adding the 5 gates from the full adder and the carry register would result in 1162 gates. No estimations about energy consumption can be made in the absence of clocking data.

This paper provides insight into the differences of serial-parallel and bit-parallel arithmetic blocks. Basic design and implementation of bit-serial and serial-pararallel 4-bit multiplication blocks is done in VHDL and Design Compiler is used to synthesize designs. A comparison of area, power and timing is done with both designs.

2. DIGITAL ARITHMETIC

In digital computers, numbers are represented in binary numeral system. A binary number of length N is a sequence of digits. A single digit in a binary number is called a bit. Binary number system has a radix of 2, meaning that a bit can have a value 0 or 1.

$$(x_{N-1}, x_{N-2}, x_{N-3}, \cdots, x_1, x_0)$$

A binary number by default represents an unsigned(positive) integer number. A binary N-bit unsigned integer number system produces numbers in the range 0 to $2^N - 1$. The Table 1 shows all possible numbers unsigned binary numbers when $N = 2$.

Table 1. All possible decimal values of 2-bit unsigned binary numbers

Unsigned binary	Decimal value
00	0
01	1
10	2
11	3

2.1. Different Binary Number Notations

Naturally, computers also have to do arithmetic with binary numbers. Obvious drawback of unsigned binary representation is that for example the result of $10_2 - 11_2$ or $01_2/11_2$ cannot be represented. To overcome this problem, different binary number representations have been developed so that negative and fractional numbers can be accessed.

2.1.1. One's Complement

One's complement system gives access to negative numbers. Negative numbers are accessed by using unary bitwise negation operator $(\sim)$. For example a binary -1 is taken from binary 1:

$$\sim 01_2 = 10_2$$

N-bit one's complement system's range is $-(2^{N-1} - 1)$ to $(2^{N-1} - 1)$. The Table 2 shows a drawback of the one's complement number system, zero has 2 representations. This has to be taken into account when designing arithmetic hardware that uses one's complement system.

The MSB (Most Significant Bit, the leftmost bit) in signed notations is the sign bit, meaning that 1 represents a negative and 0 represents a positive [2].

Table 2. Values of integers in unsigned binary and one's complement system

Decimal value	Unsigned binary	One's complement
-3	-	100
-2	-	101
-1	-	110
0	000	000 & 111
1	011	001
2	010	010
3	011	011

The one's complement representation was used by early computers such as the UNIVAC 1107 [3]. Another problem of the one's complement system is end around borrow and carry. In Figure 4 addition of -1 and 2 is done. Normal addition will not produce the correct answer and the carry bit needs to be added back to the result to produce the correct answer. This results in extra addition.

```
  110
+ 010
-----
1 000
+ 001    ← add the carry
-----
  001
```

Figure 4. Example of end-around carry in one's complement addition.

2.1.2. Two's Complement

Two's complement of a binary number is taken with a negation operator $(\sim)$, and $+1$ is added to the result. For example: -1 in two's complement:

$$\sim 01_2 = 10_2 + 1_2 = 11_2$$

The range of N-bit two's complement system is $-(2^{N-1})$ to $(2^{N-1} - 1)$. The Table 3 shows that the two's complement system does not suffer from the double representation of zero unlike one's complement system. Two's complement's range is also one integer greater towards negative than one's complement's range. It is also more straightforward to use with negative numbers because same arithmetic algorithms can be used with unsigned binary numbers and 2's complement numbers.

Moreover, with one's complement extra carry bit has to be added in order to get the right result from a arithmetic operation. Figure 5 shows the same addition of -1 and 2 that was done in one's complement in Figure 4. It can be observed that in two's complement representation no extra addition is needed to get the correct

result. This is why 2's complement is more common over 1's complement in modern computers.

Table 3. All possible 3-bit two's complement binary number values

Decimal value	Unsigned binary	Two's complement
-4	-	100
-3	-	101
-2	-	110
-1	-	111
0	000	000
1	001	001
2	010	010
3	011	011

```
  111
+ 010
-----
  001
```

Figure 5. Example of addition in two's complement.

2.1.3. Sign Magnitude Representation

Sign magnitude representation is perhaps the most straightforward way to represent negative numbers in binary. Negative counterpart from a unsigned binary number is obtained by assigning some bit as a sign-bit. Usually the sign-bit is the MSB.

For example: Decimal 15 in unsigned binary is 1111_2 but to get the sign-magnitude representation the number is extended one bit and assigned as the sign-bit: 01111_2. -15_{10} is obtained by flipping the sign: 11111_2. Sign magnitude representation suffers from the same double representation of zero: Both 1000_2 and 0000_2 for 0_{10}. Sign magnitude is used in floating point representation.

2.1.4. Fixed Point

Fixed-point and floating point representations enables binary to represent real numbers. Fixed-point representation is done by assuming the decimal point somewhere in the between the bits of the binary number. Bits to the right from the decimal point represent the fractions and bits to the left represent the whole.

The Table 4 shows a representation of a number $7{,}625_{10}$ with $N = 6$ in fixed-point binary, assuming the point between 3rd and 4th bit: $111{,}101$. The point would not be stored in the memory because arithmetic hardware is designed to handle the fractional bits.

Table 4. Fixed-point representation of a number $7{,}625$

2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
4	2	1	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$
1	1	1	1	0	1

Fixed-point representation can be combined with 2's complement, 1's complement or sign-magnitude representations to have access to real binary numbers. The range is a trade-off between the fraction's precision and the width of the integer range.

2.1.5. Floating Point

Floating point representation is the most common way to represent real numbers and it is defined in IEEE 754 [4] standard. Consider the scientific notation for real numbers:

$$m * 10^n$$

Where a real number m is the mantissa and an integer n is the exponent. The floating point representation is the binary equivalent for this scientific notation. IEEE 754 standard for single precision 32-bit floating point binary number is seen in Figure 6:

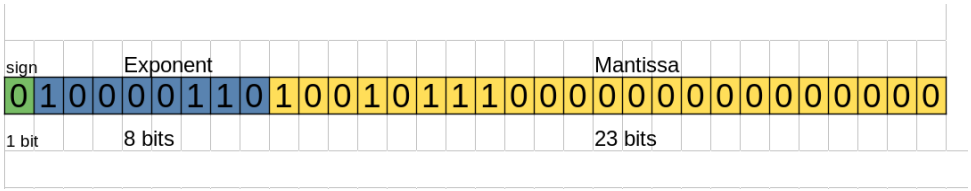

Figure 6. IEEE 32-bit single precision floating number representation of $203{,}5_{10}$.

The mantissa is a fixed-point representation of a real positive number. Mantissa's first bit is hidden and it is always 1. The decimal point is located between the hidden first mantissa bit and the second fraction bit. The mantissa's fraction bits have a precision down to 2^{-23}. Steps to convert this number:

- Determine the sign: $+$
- Determine the exponent: $10000110_2 = 134_{10}$
- Subtract the exponent bias: $134 - 127 = 7$
- Convert the mantissa: $2^{-1} + 2^{-4} + 2^{-6} + 2^{-7} + 2^{-8} = 0{,}58984375$
- Add the hidden bit and include the sign: $1 + 0{,}58984375 = 1{,}58984375$
- Calculate the result = $1{,}58984375 * 2^7 = 203{,}5$

The IEEE 754 has also defined standards for 16-bit half precision, 64-bit double precision, 128-bit quadruple precision and 256-bit octuple precision binary numbers.

2.2. Basic Arithmetic Hardware and Arithmetic Architectures

2.2.1. Half Adder and Full Adder

Adders are one of the most basic digital circuits that perform arithmetic. Half adder adds two 1-bit binary numbers together. Half adder has 2 inputs A and B. Two outputs S representing sum and C representing overflow, also known as carry. Figure 7 presents the logic circuit of a half adder.

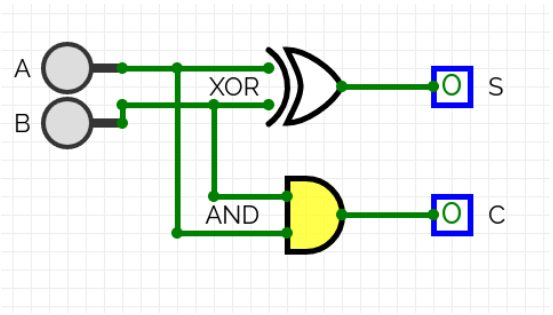

Figure 7. Logic circuit for a half adder.

Table 5 shows the truth table for half adder. As the Figure 8 presents, half adder is used in full adder as a component. A full adder performs the same calculation as a half adder but the difference is a carry input from a previous full adder. is more useful because it has a carry input so multiple full adders can be chained together [5] to create parallel adder designs.

Table 5. Truth table for a half adder

IN		OUT	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

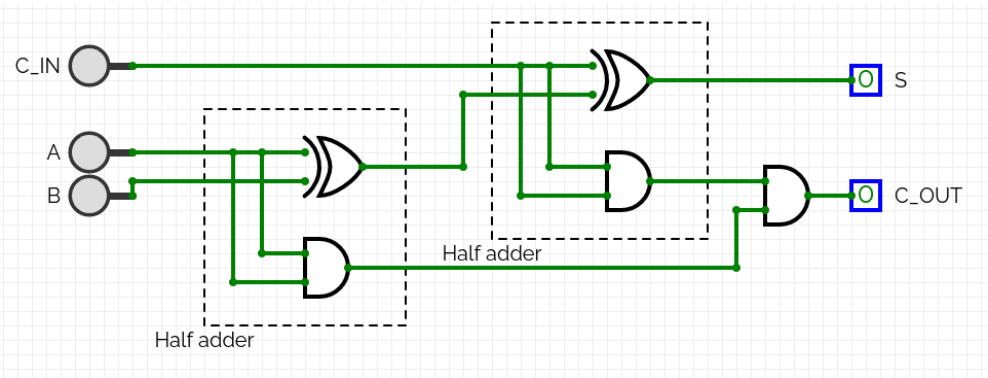

Figure 8. Logic circuit for a full adder.

2.2.2. Bit-Parallel, Serial-Parallel and Bit-Serial

Bit-parallel, serial-parallel and bit-serial are arithmetic architectures. In bit-parallel arithmetic corresponding bits of the input are processed at the same clock cycle. In Figure 9 can be seen the parallel inputs for corresponding bits.

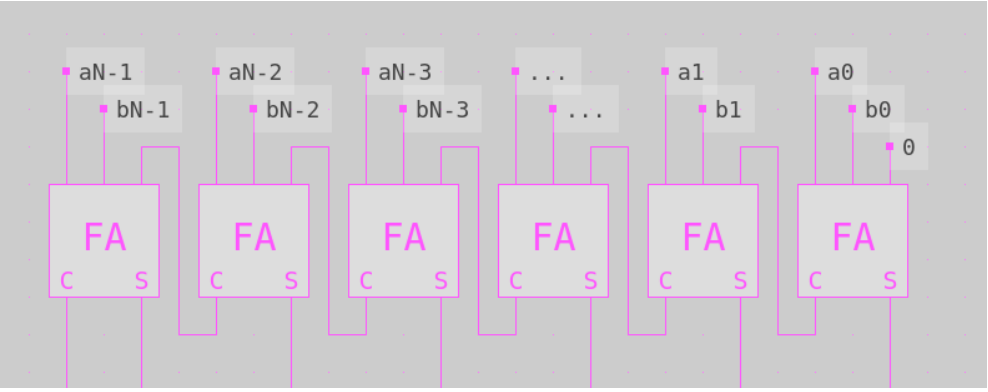

Figure 9. N-bit parallel ripple-carry adder.

In serial arithmetic one or all of the inputs is taken in serially which means one bit at a time. In bit-serial systems all inputs are in serial form but it is possible to have a mixed system where one input input is parallel form while other is in serial. This is known as serial-parallel. Serial systems utilize registers so the same arithmetic circuitry can be used in calculation over multiple clock cycles. Serial arithmetic is useful when the interconnections and area (price) of the arithmetic block has to be small but delay is acceptable [6]. This means serial-parallel arithmetic has initially more delay compared to the bit-parallel arithmetic.

Figure 10 presents N-bit serial adder. In comparison to the Figure 9 which has N amount of full adders, the former has only 1. Both inputs are loaded in parallel to the registers and sum accumulates to the A shift register. Carry is saved to the D flip-flop and is available in the next clock-cycle calculation. Shift control creates shift signal N times and after that the sum is complete.

There are number of multiplication algorithms for bit-parallel e.g. Booth multiplication, Wallace multiplication and Braun multiplication. Braun multiplier has the simplest algorithm of all bit-parallel multipliers [7]. Wallace and Braun are shift-and add multipliers meaning they utilize stages of full adder circuits which produce and add partial products of the multiplication. These kind of multipliers require more area compared to the conventional adders.

In contrast to the number of existing bit-parallel multiplication algorithms, there are not that many bit-serial multiplication algorithms. Bit-serial multipliers are area and power efficient and are useful in low speed applications [8].

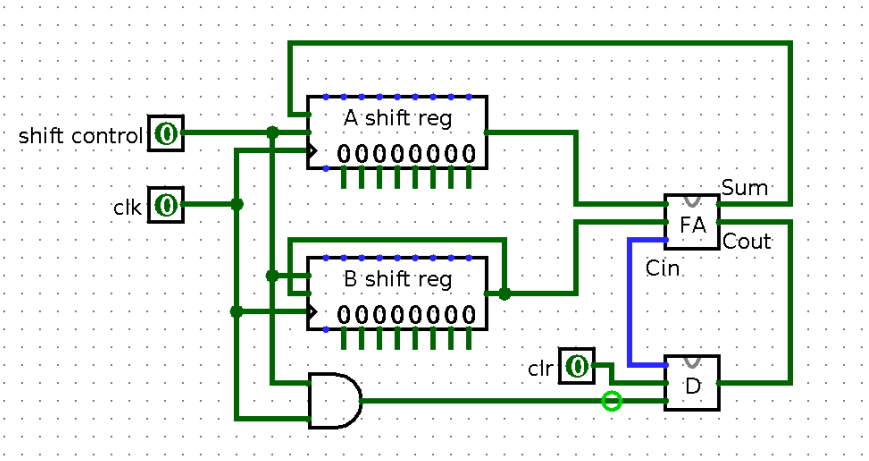

Figure 10. N-bit serial adder.

3. IMPLEMENTING DIGITAL ARITHMETIC HARDWARE

There are endless ways to implement multiplication to hardware. The designer balances between speed of the circuit, area and power consumption. The design is optimized for a specific use case. There is not a simple, most optimal way to implement a fast and efficient multiplication circuit. Designs are evolving and are more optimized over previous implementations. A presentation of existing comparison studies is done in this chapter.

It would seem obvious that serial-parallel multiplication algorithm would initially do less operations in a given time frame than bit-parallel multiplication algorithm due to some-bit parallel multiplication circuits being purely combinational circuits. In space-constrained applications bit-parallel multiplication method could be too large, even tough it would complete the calculation in 1 clock cycle [9]. Serial-parallel multiplication sees usage in at least digital communication systems, signal processing, embedded computing [10]. The article also proposes optimized way to implement two's complement multiplication of binary numbers from 2 to 32 bits. The paper presents implementation of serial-parallel multiplication circuit with up to 30-percent smaller area without speed penalty compared to the serial-parallel multiplication method presented in [11].

3.0.1. Advantages of SIMD

Singe-instruction multiple-data (SIMD) allows the computer to perform multiple multiplication operations over multiple value pairs with only one instruction. This is useful in matrix multiplication where the usage of SIMD optimizes the speed of the matrix multiplication. SIMD enables whole blocks of data to be used in multiplication operation instead of one value-pair at a time. Effectiveness of SIMD is presented in [12].

3.1. Synthesis of Bit-Serial and Serial-Parallel Multiplicator Designs

Both implementations are 4-bit multiplication circuits. Both circuits take two 4-bit inputs A and B, and have one 8-bit $RESULT$ output.
The bit-parallel implementation is a purely combinational circuit which means it does not have any sequential logic. The design utilizes a booth's multiplication algorithm. Figure 11 presents the schematic from synthesis.

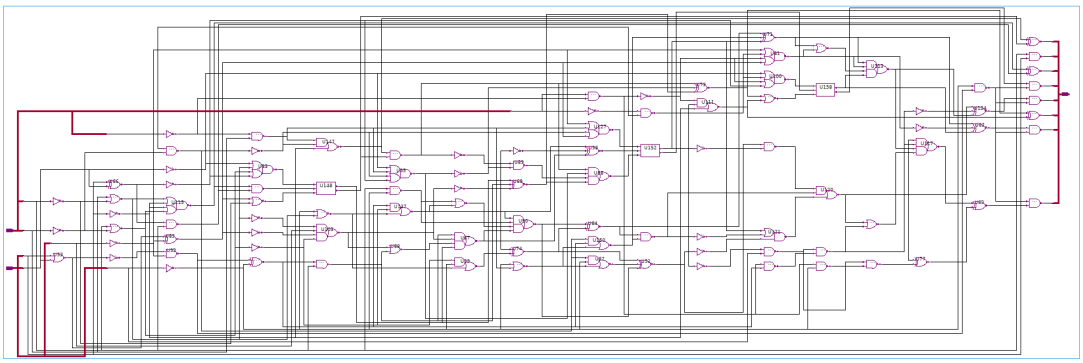

Figure 11. Schematic of the bit-parallel multiplicator.

The serial-parallel implementation utilizes registers to be able to serialize one input. Both inputs are loaded as parallel, and the circuit also has control signals $load$, $enable$, rst and clk. Both inputs and the result are stored to internal registers and 8-bit result signal shows the output along with 1-bit $ready$ indicating the end of the calculation. Figure 12 shows the synthesized serial-parallel circuit.

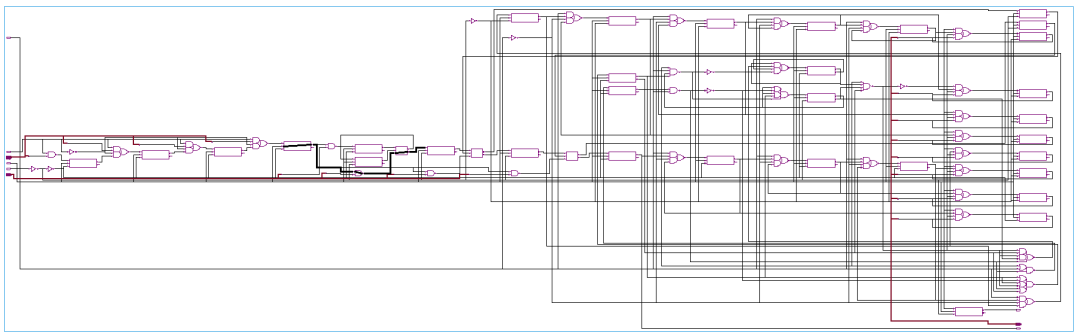

Figure 12. Schematic of the serial-parallel multiplicator.

From the Figure 13 can be seen that she sequential serial-parallel multiplicator produces the output with 9 clock cycles of delay. Adding the clock cycles for parallel load and output, the complete calculation takes 11 clock cycles.

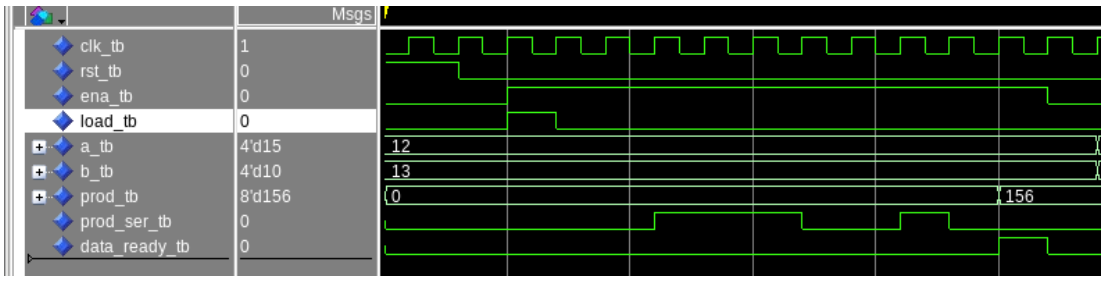

Figure 13. Simulation waves of the serial-parallel multiplication algorithm.

4. COMPARISON

Figure 14 shows the power consumptions of the circuits when optimized for given clock frequencies. Both 4-bit multiplication circuits are optimized towards minimal slack. The slack decreases towards higher clock frequencies and is greater towards the lower clock frequencies. Both designs were synthesized for a given clock cycle duration. Table 6 presents the clock durations used for synthesis.

There are great differences in the power consumption between the two designs. In the case of the serial-parallel design, power consumption keeps decreasing with the clock frequency, but the power consumption is greater at higher end frequencies.

Bit-parallel design sees only small increase towards 667 megahertz frequency. Interestingly, there is also small increase of power consumption in the low end of the frequencies. The design is the most optimal between 9 - 1.5 ns clock period.

The bit-parallel design could not be synthesized to 1 nanosecond clock period without violating the slack. This means that the bit-parallel design could not produce the result inside 1 clock period. Combinational logic circuits can take more than 1 clock period to complete, but this needs to be accounted for.

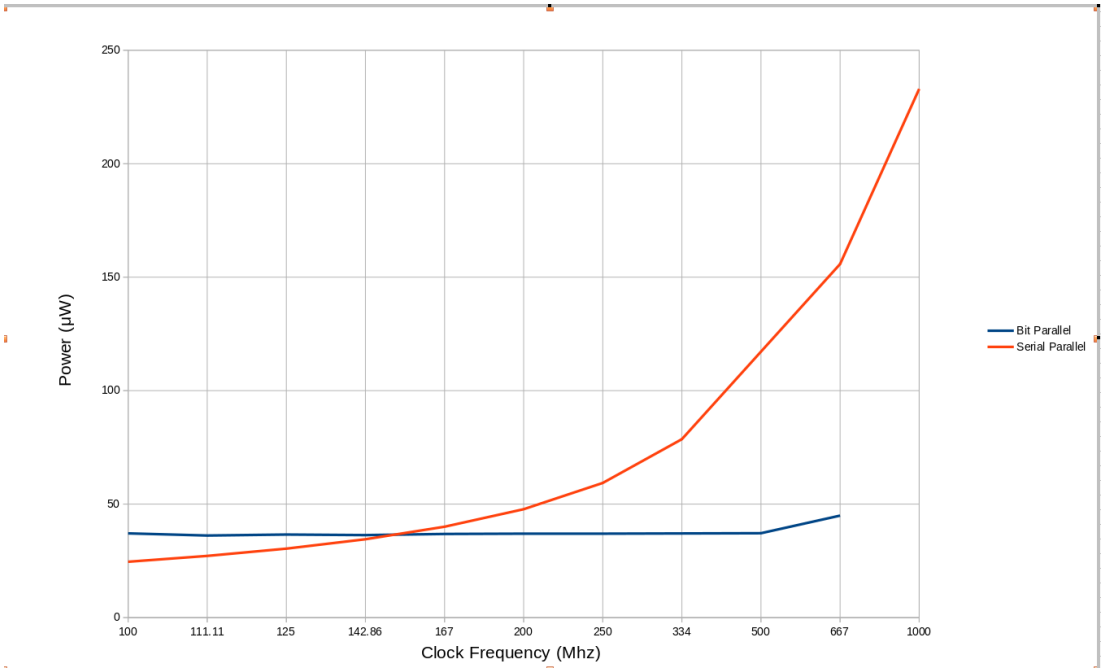

Figure 14. Chart of both multiplication algorithms.

The bit-parallel multiplication algorithm completes the calculation in 1 clock cycle. Interesting results can be seen when analyzing the cell area to clock frequency chart. Figure 15 shows the results. In contrast to the Figure 14, the serial-parallel design does not grow in size when the power consumption grows. The area stays constant.

Table 6. Conversion table for clock period duration to clock frequency

Duration of clk period (ns)	Clk frequency (Mhz)
10	100
9	111.11
8	124.99
7	142.86
6	166.66
5	200
4	259.99
3	333.33
2	499.99
1.5	666.66
1	999.99

The bit-parallel multiplier circuit starts to grow in cell area when the circuit is synthesized for 3 nanosecond clock period or less. Before 300 megahertz clock frequency it sees minimal area growth. Registers take the most of the cell area in the serial-parallel multiplier. Registers consume a lot of power and that can be seen as the power consumption growth in Figure 14.

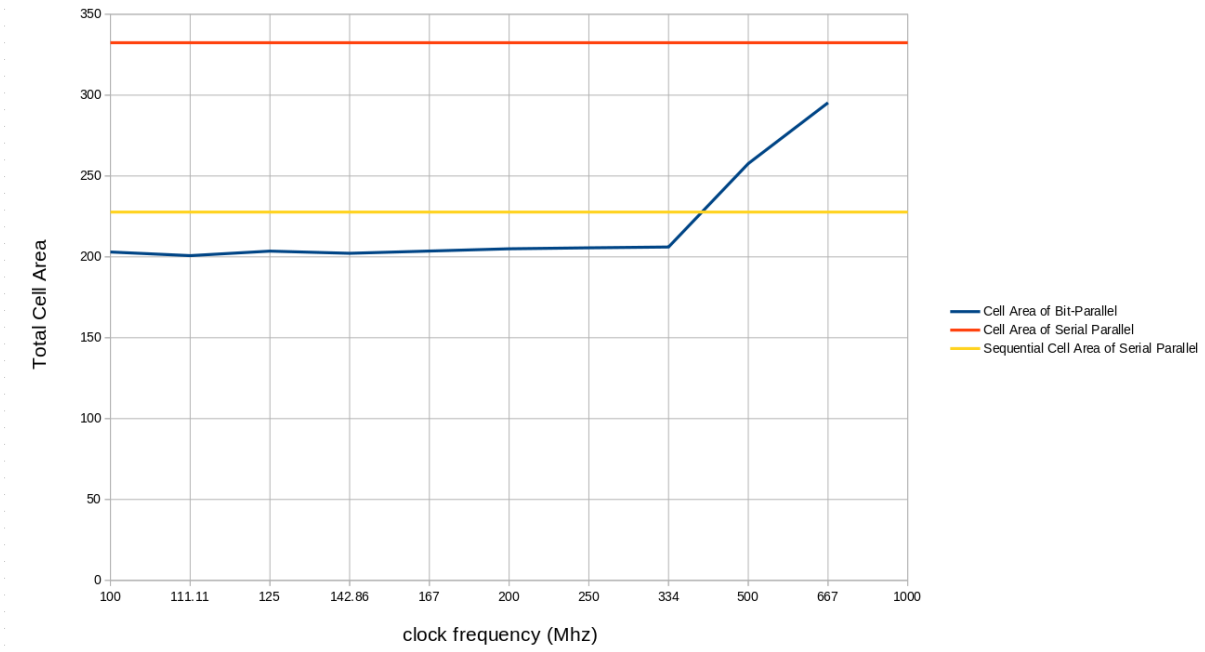

Figure 15. Chart of both multiplication algorithms.

5. DISCUSSION

Not much research has been done about this kind of subject where bit-parallel, serial-parallel and bit-serial arithmetic architectures are compared. The current results provide an insight into the differences between the schemes. The expectation was that the break-even point of the power consumption in Figure 14 would be towards higher clock frequency. The obvious expectation also was that the power consumption would rise with the clock frequency. That can be observed in the Figure 14.

The synthesis results depend greatly from the chosen algorithms. Especially the serial-parallel circuit could greatly be optimized and the synthesis results would drastically change. The algorithms were chosen to be easy to implement. Synthesis results also depend upon the chosen technology cell library. For this research synthesis was done using high threshold voltage library. That corresponds to lower speed and lower power consumption.

According to [13], the dependence between delay and area of combinational circuits usually follows the graph presented in Figure 16. It concludes that if the operation of the circuit is sped up by reducing the levels of logic, its area will increase. This kind of trend can somewhat be observed in Figure 15 when approaching high end of frequencies.

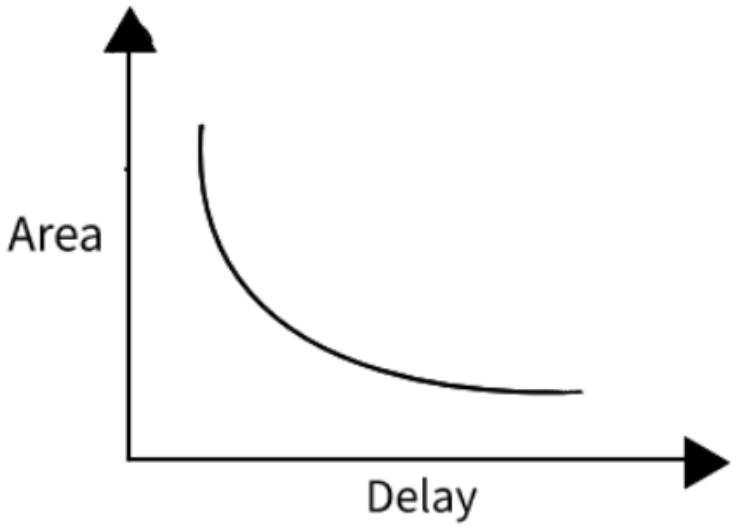

Figure 16. Delay and area graph of combinational circuits.

6. SUMMARY

This paper provided an overview of digital arithmetic and how computers do calculations. In choosing solutions for applications where highly parallel computations can be employed, meticulous analysis between the energy dissipation and latency is needed. Different binary number system notations are presented and Comparison of bit-serial and serial-parallel 4-bit multiplier circuits is also done. Synthesis done for both circuits and results are compared. A break-even point in power consumption is found.

7. REFERENCES

[1] Saini J., Agarwal S. & Kansal A. (2015) Performance, analysis and comparison of digital adders. In: 2015 International Conference on Advances in Computer Engineering and Applications, pp. 80–83.

[2] Knuth D. (1997) The Art Of Computer Programming. Addison-Wesley.

[3] Univac 1107 central computer technical bulletin. URL: http://www.bitsavers.org/pdf/univac/1107/UT-2463_CPU_Nov61.pdf. Accessed 20.12.2022.

[4] Ieee std 754-2008 standard for floating-point arithmetic. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4610935. Accessed 2.1.2023.

[5] Koyada B., Meghana N., Jaleel M.O. & Jeripotula P.R. (2017) Comparative study on adders. In: IEEE WiSPNET 2017 conference, IEEE, pp. 2226–2228.

[6] Lang T. & Ercegovac M. (2004) Digital Arithmetic. Morgan Kaufmann.

[7] P. S. & Khan A.A. (2018) Comparison of braun multiplier and wallace multiplier techniques in vlsi. In: 2018 4th International Conference on Devices, Circuits and Systems (ICDCS), pp. 48–53.

[8] Akhter S. & Chaturvedi S. (2014) Hdl based implementation of n×n bit-serial multiplier. In: 2014 International Conference on Signal Processing and Integrated Networks (SPIN), pp. 470–474.

[9] Korti R.H. & Vijaya C. (2017) Hardware implementation and performance comparison of normal basis multiplication methods over gf (2m). In: 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), pp. 2241–2247.

[10] Moh S. (2007) Vlsi-oriented architecture for two's complement serial-parallel multiplication without speed penalty. In: 2007 International Conference on Computational Science and its Applications (ICCSA 2007), pp. 9–13.

[11] Moh S. & Yoon S. (1995) Serial-parallel multiplier for two's complement numbers. In: IEE Electronics Letters, vol. 31, vol. 31, pp. 703–704.

[12] Pradyumna S. (2017) Performance comparison of matrix multiplication algorithms. In: 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA), pp. 461–466.

[13] Combinational logic design. Accessed 12.2.2023.