ISSN 2347 - 3983
Volume 13. No.9, September 2025
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter031392025.pdf
https://doi.org/10.30534/ijeter/2025/031392025

Efficient Half-Storage Generation of Frequency-Ordered Walsh Functions using XOR and Cyclic Shift

Dmytro Poltoratskyi
State University "Kyiv Aviation Institute", Ukraine
dpoltoratskyi@ukr.net

Received Date: July 29, 2025 Accepted Date: August 25, 2025 Published Date : September 07, 2025

ABSTRACT

Walsh functions play a critical role in modern digital signal processing, telecommunications, image compression, error correction, and hardware-accelerated computations due to their orthogonality, simplicity, and efficient transform properties. This paper presents an efficient half-storage algorithm for the generation of frequency-ordered Walsh functions using only XOR and cyclic shift operations. Unlike conventional approaches that store the entire Walsh matrix, the proposed method requires storing only the first half of the basis vectors. The second half is derived in constant time by combining a pre-computed alternating mask with the stored vectors. This approach significantly reduces memory usage while maintaining high generation speed, making it particularly suitable for low-power and memory-constrained DSP and IoT applications. Theoretical analysis and experimental results demonstrate the proposed algorithm's computational efficiency, scalability, and potential for hardware implementation.

Key words: Walsh functions, frequency order, half-storage algorithm, XOR operations, cyclic shift, DSP, IoT, FPGA, error correction, image compression, Walsh–Hadamard transform.

1. INTRODUCTION

Walsh functions form an orthogonal binary basis that has become a cornerstone in digital signal processing (DSP), telecommunications [1], image compression, coding, and hardware implementations [2]. Each Walsh function consists of two amplitude levels, typically $\{+1, -1\}$ or $\{0,1\}$, and can be efficiently combined to represent or process digital signals. Their strict orthogonality and simple structure allow efficient implementations in both software and hardware, making them highly suitable for electronics and telecommunication.

A full set of Walsh functions of order $N = 2^m$ can be represented as an $N \times N$ matrix, commonly known as the Walsh–Hadamard matrix. Direct generation of this matrix provides instant access to all basis vectors but at a prohibitive memory cost of $O(N^2)$.

The classical Walsh–Hadamard construction generates the full matrix iteratively by expanding from H_2 using Kronecker products or recursive combination of smaller Hadamard matrices [3]. While conceptually simple, this approach has two major drawbacks. Every basis vector must be explicitly stored, even if only a subset is needed. As N grows, both generation time and storage demands increase quadratically, making real-time or resource-constrained applications impractical. Also, the traditional Hadamard-generation methods do not naturally produce functions in frequency order, requiring additional reordering or computational overhead.

To overcome these limitations, we propose a half-storage approach that generates frequency-ordered Walsh functions using only XOR and cyclic shift operations. By storing only the first half of the basis vectors and reconstructing the second half in constant time, the algorithm provides a scalable, memory-optimized solution.

2. HALF-STORAGE GENERATION USING XOR AND CYCLIC SHIFT

The proposed method generates a complete sequence of orthogonal Walsh functions in sequency order (arranged by the increasing number of sign transitions) using only two simple operations: bitwise XOR and cyclic shift. Algorithm based on three constant functions: W_0, W_1 and W_{N-1}.

The W_0 is the function with zero transitions (constant level)

$$W_0 = 0\ 0\ 0\ \dots\ 0 \quad (1)$$

The W_1 is the function with one transition (two halves of opposite sign)

$$W_1 = 0\ 0\ \dots\ 0\ 1\ 1\ \dots\ 1 \quad (2)$$

The W_{N-1} is the function with the maximum number of transitions (alternating levels at every sample)

$$W_{N-1} = 0\ 1\ 0\ 1\ \dots\ 0\ 1 \quad (3)$$

These three functions form the foundation for generating all other functions. All functions we can divide in two ranges. For functions in the range of indices $n \in \left[0, \frac{N}{2} - 1\right]$ we use three equations depending from n.

If n is a power of two, $n = 2^k$, we apply cyclic shift to the function W_{n-1}

$$W_n = W_{n-1} \ll \frac{N}{2*n} \quad (4)$$

If n is odd we apply XOR between functions W_1 and W_{n-1}

$$W_n = W_1 \oplus W_{n-1} \quad (5)$$

If n is even but not power of two we apply XOR between functions W_{n-1} and $W_{(n-1)\oplus n}$

$$W_n = W_{n-1} \oplus W_{(n-1)\oplus n} \quad (6)$$

Where index $(n-1) \oplus n$ is using XOR between index numbers. This iterative process avoids recursion and ensures that every function in the first half is derived efficiently from previously computed vectors.

For functions in the range of indices $n \in \left[\frac{N}{2}, N-1\right]$ we use simple equation to access any function in $O(1)$ complexity

$$W_n = W_{N-1} \oplus W_{(N-1)\oplus n} \quad (7)$$

This rule eliminates the need to store the second half of the basis entirely.

Let's take an example for $N = 16$, this size covers all equation. Implementation shown in the table 1.

Table 1: Computation of N = 16

n	Result
0	$W_0 = 0000000000000000$
1	$W_1 = 0000000011111111$
2	$W_2 = W_1 \ll 4 = 0000111111110000$
3	$W_3 = W_1 \oplus W_2 = 0000111100001111$
4	$W_4 = W_3 \ll 2 = 0011110000111100$
5	$W_5 = W_1 \oplus W_4 = 0011110011000011$
6	$W_6 = W_5 \oplus W_3 = 0011001111001100$
7	$W_7 = W_1 \oplus W_6 = 0011001100110011$
8	$W_8 = W_{15} \oplus W_7 = 0110011001100110$
9	$W_9 = W_{15} \oplus W_6 = 0110011010011001$
10	$W_{10} = W_{15} \oplus W_5 = 0110100110010110$
11	$W_{11} = W_{15} \oplus W_4 = 0110100101101001$
12	$W_{12} = W_{15} \oplus W_3 = 0101101001011010$
13	$W_{13} = W_{15} \oplus W_2 = 0101101010100101$
14	$W_{14} = W_{15} \oplus W_1 = 0101010110101010$
15	$W_{15} = 0101010101010101$

To verify results, we can compare with Hadamard matrix for $N = 16$. The matrix shown in the table 2.

Table 2: Classic Hadamard order

n	$\mathbf{H_n}$
0	$H_0 = 0000000000000000$
1	$H_1 = 0101010101010101$
2	$H_2 = 0011001100110011$
3	$H_3 = 0110011001100110$
4	$H_4 = 0000111100001111$
5	$H_5 = 0101101001011010$
6	$H_6 = 0011110000111100$
7	$H_7 = 0110100101101001$
8	$H_8 = 0000000011111111$
9	$H_9 = 0101010110101010$
10	$H_{10} = 0011001111001100$
11	$H_{11} = 0110011010011001$
12	$H_{12} = 0000111111110000$
13	$H_{13} = 0101101010100101$
14	$H_{14} = 0011110011000011$
15	$H_{15} = 0110100110010110$

The functions generated by the proposed algorithm are fully consistent with the classical Walsh–Hadamard basis, while being arranged in increasing sequency (frequency) order.

3. ALGORITHM SCALABILITY

The proposed half-storage algorithm scales naturally and efficiently with increasing basis size $N = 2^m$, where $m \geq 1$. This scalability comes from its bitwise and recursive structure, which remains consistent regardless of how large the basis becomes.

All generation rules — XOR operations and cyclic shifts — are expressed in terms of bit-level manipulations of the function indices. These operations do not depend on the absolute size of N. Instead, they rely only on the binary representation of the index. This means the same rules that generate a basis for $N = 8$ apply identically to $N = 1024$, $N = 4096$ and larger dimensions.

The second half of the basis is derived in constant time for any N using the relation (7), where $N-1$ is always a bitmask of higher frequency basis function. This guarantees that access to any function remains $O(1)$, independent of the size of the basis. The computational complexity of generating the first half of the basis grows quadratically as $O(N^2)$, which is a one-time initialization cost, which is shown on Figure 1.

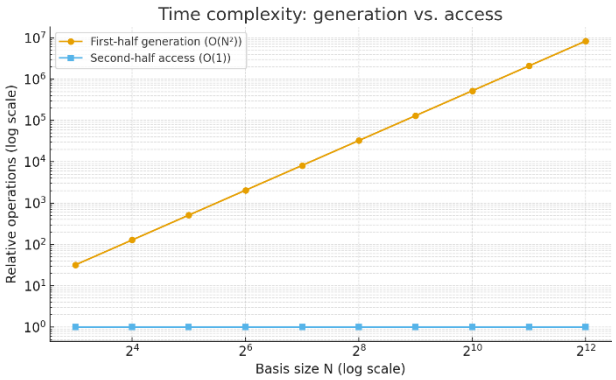

Figure 1: Time complexity vs. basis size N

Memory usage is reduced by exactly 50% compared to full-matrix storage, which is shown on Figure 2.

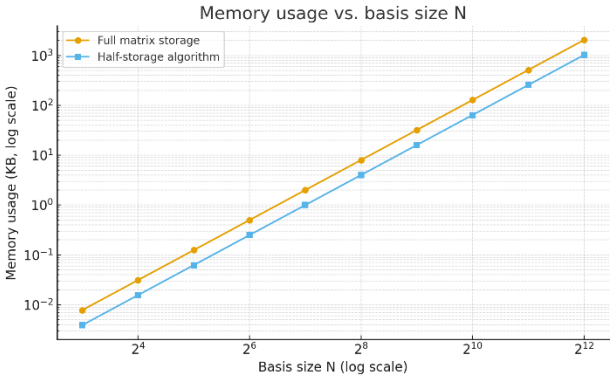

Figure 2: Memory usage vs. basis size N

Extensive validation up to $N = 4096$ has confirmed perfect orthogonality, strict sequency ordering, and full consistency with canonical Walsh bases. This combination of size-independent logic, predictable complexity, and constant-time access makes the algorithm ideally suited for high-resolution DSP, FPGA and ASIC implementations, and IoT and embedded systems where memory efficiency and deterministic execution are critical [4], [5].

4. CONCLUSION

This study introduced a half-storage algorithm for generating frequency-ordered Walsh functions using only XOR and cyclic shift operations. By storing only the first half of the basis and deriving the second half in constant time, the method achieves a predictable 50% reduction in memory requirements while preserving full orthogonality and strict frequency ordering. The algorithm scales efficiently with any basis size $N = 2^m$, providing a one-time $O(N^2)$ initialization cost for the first half and constant-time $O(1)$ access for the second half, making it significantly more efficient than the classical full-matrix approach.

Theoretical analysis and validation across multiple basis sizes confirmed the accuracy, scalability, and hardware-friendliness of the method. These characteristics make the algorithm particularly well-suited for applications in digital signal processing, embedded and IoT systems, and FPGA or ASIC implementations, where memory constraints and deterministic performance are critical.

Future research will explore further optimization strategies, such as hash-based indexing for faster function retrieval and adaptive implementations for dynamic, real-time systems requiring flexible basis sizes.

REFERENCES

1. H. F. Harmuth. **Harmuth, Applications of Walsh functions in communications**, IEEE *Trans. on Communications Technology*, Vol. COM-19, pp. 398-405, 1971.
2. R. A. Scholtz. **The Spread Spectrum Concept**, in *Multiple Access*, N. Abramson, Ed. Piscataway, NJ: IEEE Press, 1993, ch. 3, pp. 121-123.
3. W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. **Numerical Recipes in C: The Art of Scientific Computing**, 2nd ed., Cambridge University Press, 1992, pp. 573-579.
4. D. M. Kodek. **Walsh transforms and their hardware implementations**, *IEEE Trans. on Computers*, vol. C-26, no. 6, pp. 585-593, Jun. 1977.
5. S. P. Bingulac. **On the compatibility of adaptive controllers**, in *Proc. 4th Annu. Allerton Conf. Circuits and Systems Theory*, New York, 1994, pp. 8-16.