Complexity of basic boolean operators for digital circuit design*

Igor S. Sergeev[†]

Abstract

This article provides a survey of circuit complexity bounds for basic boolean transforms exploited in digital circuit design and efficient methods for synthesizing such circuits. The exposition covers structurally simple functions and operators, such as counters, adders, encoders, and multiplexors, and excludes more complex algebraic operations with numbers, polynomials, and matrices. Several applications to implementing more specific operations are also discussed.

Introduction

This paper attempts to collect information on the complexity of the simplest and most fundamental boolean transforms, which are widely used in both practical circuit design and theoretical circuit synthesis problems. These include encoders, multiplexors, comparators, and other operators that are structurally no more complex than addition. These transforms often lack a clear independent meaning, but rather serve as "building blocks" for solving substantive problems. These transforms are typically the starting point for studying the fundamentals of digital circuit design in popular textbooks; see, e.g., [6, 25]. Therefore, in this survey, we (informally) refer to them as *basic*.

The theory of algorithms for multiplying numbers or matrices, or discrete Fourier transforms, is so developed that it requires multi-volume editions to cover. Meanwhile, the "workhorses" of electronics, such as the shift operator, the priority encoder, or the unary converter, are usually overlooked. Popular monographs on computational complexity, such as [10, 11, 28], provide only fragmentary information on the basic operators. The goal of this survey is to present as complete a picture as possible.

A significant portion of the results and methods considered below should be relegated to folklore due to their simplicity and familiarity. Therefore, they are presented here without citing any references. In other cases, where results requiring considerable effort are discussed, especially for lower bounds, references to works known to the author are provided, as is customary. The author had to fill in some gaps himself, but without resorting to nontrivial constructions.

We consider the implementation of boolean transforms in the model of circuits of functional elements, i.e. boolean or logical circuits, see, e.g., [4, 28]. Of the standard mathematical models, it most closely corresponds to real electronic circuits. Recall that a *circuit over a basis* (a set of functions) $\mathcal{B}$ is an acyclic directed graph in which vertices that have

---

no incoming edges are marked as inputs, and some vertices are marked as outputs. The inputs are labeled by symbols of variables or constants of the basis $\mathcal{B}$, while the remaining vertices (these vertices are called functional elements or gates) by symbols of functions from the basis $\mathcal{B}$. The functioning of the circuit is defined naturally, from inputs to outputs: at each vertex, the associated function is computed, the arguments of which are the functions arriving along the edges entering the vertex. A circuit implements an operator (a system of functions) F if all components of the operator are computed at the circuit's outputs. The *complexity* of a circuit is defined as the number of vertices in its graph, excluding inputs. The complexity of an operator F when implemented by circuits over a basis $\mathcal{B}$ is defined as the complexity of the minimal circuit implementing it. The *depth* of a circuit is the length (measured in edges or functional elements) of the longest directed input-output path. Similarly, the depth of an operator F is defined as the minimal depth of a circuit implementing it. Actually, the complexity roughly corresponds to the area of a circuit, and the depth to the propagation delay of a signal from the inputs to the outputs.

We restrict our consideration to the basis of all binary boolean functions. We denote the complexity of a circuit Φ over this basis by $\mathsf{C}(\Phi)$, and the complexity of an operator F by $\mathsf{C}(F)$. We also introduce the functional $\mathsf{C}_{\log}(F_n)$, which denotes the complexity of implementing a sequence of operators F_n by circuits of depth $O(\log n)$, where n is the number of input variables. Informally, this is the complexity of parallel computation of the operator. In practical circuit design, parallel circuits are preferred. It is worth noting that the operators considered below are quite simple and can be implemented by parallel circuits.

When designing electronic circuits, wider bases are typically available, which may include multi-input gates and even gates with multiple outputs. However, efficient synthesis methods are generally quite universal and can be adopted to any basis. Recall that the order of complexity/depth of a function over any complete finite basis is the same.

The complexity bounds here characterize the computational complexity in the asymptotic sense, i.e., for the number of inputs $n \to \infty$. However, it is well known that asymptotically efficient synthesis methods do not necessarily yield good results for practically significant values n. However, the simple methods discussed below usually perform well even for very small values n. Moreover, for such values, parallel synthesis methods lead to circuits with compromised depth and complexity characteristics.

Further, we present information on the complexity and efficient implementation methods of basic transforms. Basic operations include: prefix and suffix sums, numeric increment/decrement, up-down counter, Gray counter, carry computation, addition and comparison of two numbers, maximum/minimum of two numbers, decoder, multiplexor, direct and cyclic shift, encoder, extraction of the first one and its position number, bit summation and comparison of the sum with a threshold, computation of the width of a block of ones, conversion between binary and unary encodings, truncating, and sorting array of bits. Complexity bounds for these operations are summarized in Table 1.

The survey is supplemented with examples of application of basic operations and some useful design ideas to constructing parallel circuits for a two-selector, a weight-preserving counter, multiple selection, and permutation of a pair of bits.

The following notations are used throughout the presentation:

$\mathbb{B} = \{0, 1\}$;

$\mathbb{B}^n$ — the set of boolean strings or vectors of length n; by default, the bits in a string are numbered from zero, left to right;

$[\![n]\!]$ — the set of integers from 0 to $n-1$, specified in binary notation of length $\lceil \log n \rceil$,

the bits are numbered from zero, right to left;

$|s|$ — the length of a boolean vector or string;

$\nu(X)$ — the binary weight (the number of ones) of a boolean string or number X;

$\overline{x}$, $x \vee y$, $x \wedge y$ or $x \cdot y$, $x \oplus y$, $x \sim y$ — boolean operations of negation, disjunction, conjunction, addition modulo 2, and equivalence;

$X^\alpha = \bigwedge x_i^{\alpha_i}$ — elementary conjunction of a vector of variables $X = [x_1, x_2, \ldots]$, where $\alpha = [\alpha_1, \alpha_2, \ldots] \in \mathbb{B}^{|X|}$; by definition, $x^1 = x$ and $x^0 = \overline{x}$;

"$\|$" — string concatenation operation;

$[\sigma]^m$ — a vector or string of length m consisting of boolean values σ.

All logarithms below are base 2.

Complexity of basic operators

Prefix sums. The operator $\texttt{PREF}_n^* : \mathbb{S}^n \to \mathbb{S}^n$ computes a family of prefix sums of n variables from the semigroup $(\mathbb{S}, *)$:

$$p_i = x_1 * x_2 * \ldots * x_i, \quad 1 \le i \le n. \tag{1}$$

Further applications, with the exception of constructions of carry circuits, will be restricted to the case $\mathbb{S} = \mathbb{B}$ and $* \in \{\vee, \wedge, \oplus\}$. Circuits that compute the system (1) over the basis $\{*\}$ are called prefix circuits.

There is an extensive literature devoted to prefix circuits, see, e.g., [2, 20],[28, §3.1]. We restrict ourselves to only brief information. Obviously, the operator $\texttt{PREF}_n^*$ may be implemented by a circuit of $n-1$ gates $*$, in which the prefix sums are calculated sequentially. The complexity C and the depth D of a circuit of operations $*$ that calculates prefix sums of n variables are related as $C + D \ge 2n - 2$. Therefore, the complexity of a parallel circuit is at least $2n - \Theta(\log n)$. This bound is achieved, for instance, by circuits proposed by Yu. P. Ofman [16] (but in the literature they are usually called Brent—Kung circuits). In the circuit with $n = 2^k$ inputs, the highest sum p_n is computed by a complete binary tree T of depth k. Any missing sum p_i is computed as $p_{\triangledown i} * p'$, where $\triangledown i$ denotes the number obtained from i by setting the least significant one to zero, and p' is an appropriate subsum computed in the tree T. It is easy to verify that such circuit has depth $2k - 2$. The 8-input circuit is shown in Fig. 1. The circuit with an arbitrary number $n < 2^k$ inputs is obtained by truncating the 2^k- input circuit.
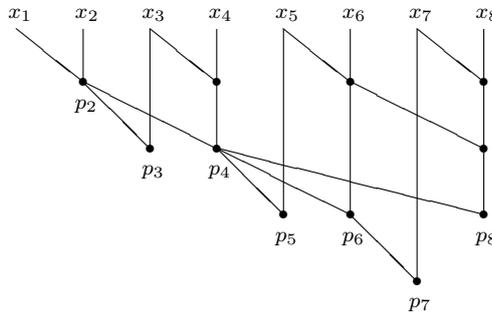

Figure 1: Ofman's (Brent—Kung) prefix circuit

In the case of boolean operations $*$, all the mentioned bounds hold for circuits over the binary boolean basis, precisely, $\mathsf{C}(\texttt{PREF}_n^*) = n - 1$ and $\mathsf{C}_{\log}(\texttt{PREF}_n^*) = 2n - \Theta(\log n)$.

3

There is also the problem of jointly computing prefix and suffix sums of n variables. The latter are defined as

$$s_i = x_i * x_{i+1} * \ldots * x_n, \quad 1 \leq i \leq n.$$

The complexity of the corresponding operator $\mathtt{PS}_n^* : \mathbb{S}^n \to \mathbb{S}^{2n-1}$ is clearly $2n - 3$. Minimal parallel prefix-suffix circuits have complexity $3n - \Theta(\log n)$. Such a circuit can, for example, be constructed by combining Ofman's prefix and suffix circuits: the tree computing the sum $p_n = s_1$ of all variables is the common part for these two circuits.

Incrementor. The operator $\mathtt{INC}_n : [\![2^n]\!] \to [\![2^n]\!]$ increments an n-bit number by one modulo 2^n.

The upper bound $\mathsf{C}(\mathtt{INC}_n) \leq 2n - 2$ for $n \geq 2$ can be easily obtained. If we denote the input by $X = [x_{n-1}, \ldots, x_0] \in [\![2^n]\!]$, the digits of the sum $X + 1 \bmod 2^n = [z_{n-1}, \ldots, z_0]$ are determined by formulas[1]

$$z_k = x_k \oplus x_{k-1} \cdot \ldots \cdot x_0. \tag{2}$$

Compute the products in these formulas via the operator $\mathtt{PREF}_{n-1}^\wedge$, and then perform bitwise modulo-2 addition of vectors of length n. Employing a parallel prefix circuit leads to the bound $\mathsf{C}_{\log}(\mathtt{INC}_n) \leq 3n - \Theta(\log n)$.

The complementary lower bound $\mathsf{C}(\mathtt{INC}_n) \geq 2n - 2$ is probably easier to prove by considering the operator $\mathtt{INC}_n'$, which computes the sum $X + 1$ entirely. It is easy to verify that $\mathsf{C}(\mathtt{INC}_n') = \mathsf{C}(\mathtt{INC}_n) + 1$ for $n \geq 2$. Further, $\mathsf{C}(\mathtt{INC}_n') = 2n - 1$, since substituting $x_{n-1} = 0$ into the circuit for $\mathtt{INC}_n'$ reduces the complexity by at least 2, and the resulting circuit computes the operator $\mathtt{INC}_{n-1}'$.

Decrement circuits that compute the difference $X - 1 \bmod 2^n = [z_{n-1}, \ldots, z_0]$ may be constructed dually: the digits of the difference are determined by formulas

$$z_k = x_k \oplus \overline{x_{k-1}} \cdot \ldots \cdot \overline{x_0}. \tag{3}$$

Up-down counter. The operator $\mathtt{UDC}_n : [\![2^n]\!] \times \mathbb{B} \to [\![2^n]\!]$, depending on the control input $\sigma \in \mathbb{B}$, either increments (for $\sigma = 1$) or decrements (for $\sigma = 0$) an n-bit number modulo 2^n.

The bounds $\mathsf{C}(\mathtt{UDC}_n) \leq 3n - 3$ for $n \geq 2$ and $\mathsf{C}_{\log}(\mathtt{UDC}_n) \leq 4n - \Theta(\log n)$ are obtained by combining the incrementor and decrementor circuits from the previous paragraph. The result's digits are computed by formulas combining (2) and (3):

$$z_k = x_k \oplus x_{k-1}^\sigma \cdot \ldots \cdot x_0^\sigma.$$

Boolean powers x_k^σ are computed simply as $\sigma \sim x_k$, $k = 0, \ldots, n - 2$. Then, the operator $\mathtt{PREF}_{n-1}^\wedge$ is applied, followed by bitwise addition of n-bit vectors.

Gray counter. The operator $\mathtt{GRC}_n : \mathbb{B}^n \to \mathbb{B}^n$, given a Boolean string of length n, computes the next string according to the standard Gray encoding in cyclic order. In other words, it is an incrementor in Gray encoding. In this paragraph, we number digits of strings from right to left.

Recall that a sequence G_n of n-bit Gray strings can be defined recursively: $G_1 = (0, 1)$ and then $G_{k+1} = (0 \,\|\, G_k, 1 \,\|\, G_k^R)$, where G_k^R is the sequence G_k written in reverse order; the concatenation operation is applied string-wise. The main property of Gray sequences is that the next string differs from the previous one in exactly one position. For more details, see, e.g., [27, Chapter 13].

---

[1] The formula also holds for $k = 0$, if we adopt the convention that the empty conjunction is equal to 1.

The upper bounds $\mathsf{C}(\mathtt{GRC}_n) \leq 4n - 7$ and $\mathsf{C}_{\log}(\mathtt{GRC}_n) \leq 6n - \Theta(\log n)$ are obtained by conversion to and from the binary encoding. Let $\mathrm{bin}_n(X)$ denote the binary representation of the index of a string X in the sequence G_n. Then $\mathtt{GRC}_n(X) = \mathrm{bin}_n^{-1}(\mathtt{INC}_n(\mathrm{bin}_n(X)))$. By denoting $[y_{n-1}, \ldots, y_0] = \mathrm{bin}_n(x_{n-1}, \ldots, x_0)$, it is easy to verify that $y_{n-1} = x_{n-1}$ and for any $i \leq n - 2$,
$$y_i = x_i \oplus x_{i+1} \oplus \ldots \oplus x_{n-1}, \qquad x_i = y_{i+1} \oplus y_i.$$
Therefore, the conversion bin_n may be performed by the operator $\mathtt{PREF}_n^{\oplus}$, and bin_n^{-1} may be reduced to bitwise addition of strings of length $n - 1$. It is slightly more convenient to compute the complement bits $\overline{y_i}$ instead of y_i. When implementing the operator $\mathtt{INC}_n(X)$ in sequential manner, the two least significant bits do not need to be computed, since they are equal to x_0 and $\overline{y_0}$. The remaining part of the increment circuit has complexity $2n - 4$. Another operation may be saved in the last step of the conversion to Gray encoding: the least significant bit of the result is simply $\overline{y_1}$.

Carry operator. The operator $\mathtt{CAR}_n : (\mathbb{B}^n)^2 \to \mathbb{B}^n$ computes the system of carry functions
$$c_1 = x_0, \qquad c_{i+1} = x_i \oplus y_i c_i, \quad i = 1, \ldots, n - 1, \tag{4}$$
of boolean variables $x_0, x_1, \ldots, x_{n-1}$ and $y_0, y_1, \ldots, y_{n-1}$.

It follows directly from the definition (4) that $\mathsf{C}(\mathtt{CAR}_n) = 2n - 2$. The upper bound $\mathsf{C}_{\log}(\mathtt{CAR}_n) \leq 5n - \Theta(\log n)$ is proved by reducing to the computation of a system of prefix sums. Define the binary operation $\star$ on the set of boolean length-2 vectors as $[a_1, b_1] \star [a_2, b_2] = [a_2 \oplus b_2 a_1, b_2 b_1]$. It is easy to verify that the introduced operation is associative, hence $(\mathbb{B}^2, \star)$ is a semigroup. With the notation $p_i = y_{i-1} \cdot \ldots \cdot y_1$, the system (4) is transformed into a system of prefix sums:
$$[c_1, p_1] = [x_0, 1], \qquad [c_{i+1}, p_{i+1}] = [c_i, p_i] \star [x_i, y_i], \quad i = 1, \ldots, n - 1.$$
Note that the products p_i themselves do not need to be computed when implementing carries. Then, for the complexity of parallel carry circuits, we have the bound $\mathsf{C}_{\log}(\mathtt{CAR}_n) \leq \mathsf{C}_{\log}(\mathtt{PREF}_n^{\star}) - (n - 1)$. It remains to observe that the complexity of the operation $\star$ is 3.

The idea of computing carries via parallel prefix circuits arose no later than the 1960s, for example, in [23, 16].

Adder. The operator $\mathtt{ADD}_n : [\![2^n]\!]^2 \to [\![2^{n+1}]\!]$ calculates the sum of two n-bit integers A and B.

The upper bounds $\mathsf{C}(\mathtt{ADD}_n) \leq 5n - 3$ and $\mathsf{C}_{\log}(\mathtt{ADD}_n) \leq 8n - \Theta(\log n)$ are obtained by attaching two layers of $3n - 1$ gates to the carry circuits from the previous paragraph[2]. Denote $A = [a_{n-1}, \ldots, a_0]$, $B = [b_{n-1}, \ldots, b_0]$, and $A + B = [z_n, \ldots, z_0]$. Set $x_i = a_i \wedge b_i$, $y_i = a_i \oplus b_i$. Let c_i be defined according to (4). Then $z_0 = y_0$, $z_n = c_n$, and $z_i = y_i \oplus c_i$ for other i's.

N. P. Red'kin [19] showed that the former of the two bounds is tight: in fact, $\mathsf{C}(\mathtt{ADD}_n) = 5n - 3$.

We do not consider the integer subtraction separately, since negative numbers are usually written in the complement code, in which subtraction and addition are performed uniformly using adders, see, e.g., [10, Sec. 4.1], [28, §3.1].

Comparator. The function $\mathtt{CMP}_n : [\![2^n]\!]^2 \to \mathbb{B}$ compares n-bit numbers A, B, i.e., it evaluates the predicate $A > B$.

---

[2]In [12, 28], complexity estimates for parallel adders are given in the form $\mathsf{C}_{\log}(\mathtt{ADD}_n) \leq 8n + O(1)$.

The upper bound $\mathsf{C}(\mathtt{CMP}_n) \leq 4n-3$ is obtained straightforwardly. Let $A = [a_{n-1}, \ldots, a_0]$, $B = [b_{n-1}, \ldots, b_0]$, and $x_i = a_i \wedge \overline{b_i}$, $y_i = a_i \sim b_i$. Then $\mathtt{CMP}_n(A, B) = c_n$, where c_n is defined according to (4). Therefore, $\mathsf{C}(\mathtt{CMP}_n) \leq 2n - 1 + \mathsf{C}(\mathtt{CAR}_n)$, given that there is no need to compute y_0.

The bound $\mathsf{C}_{\log}(\mathtt{CMP}_n) \leq 2n - 1 + \mathsf{C}_{\log}(c_n) \leq 5n - \Theta(\log n)$ is established similarly. Compute c_n by a depth-d parallel prefix tree composed of $n-1$ gates $\star$. Note that d gates computing the second components of the prefix sums (products of the input variables) can be eliminated.

Non-strict comparison is implemented similarly, since the predicate $A \geq B$ is the negation of the predicate $B > A$.

The extended operator $\mathtt{CMP}_n^* : [\![2^n]\!]^2 \to \mathbb{B}^2$ additionally computes the predicate $A = B$, i.e., the product $y_0 \cdot y_1 \cdot \ldots \cdot y_{n-1}$. Its complexity is estimated as $\mathsf{C}_{\log}(\mathtt{CMP}_n^*) \leq 5n - 3$. Just add the calculation of y_0 and all the second components of the prefix sums into the parallel comparator circuit described above. The desired product is finally obtained from these components.

Maximum of two numbers. The operator $\mathtt{MAX}_n : [\![2^n]\!]^2 \to [\![2^n]\!]$ computes the maximum of two n-bit numbers.

The upper bounds $\mathsf{C}(\mathtt{MAX}_n) \leq 6n - 3$ and $\mathsf{C}_{\log}(\mathtt{MAX}_n) \leq 7n - \Theta(\log n)$ are obtained by attaching a layer of $2n$ gates to the comparator circuits from the previous section.

Denote $c = \mathtt{CMP}_n(A, B)$, then $\max(A, B) = \overline{(A \sim B)} \cdot [c]^n \oplus B$, where the operations in the last formula are bitwise. Recall that the vector $A \sim B$, except for its least significant bit, has already been computed by the comparator circuit. Therefore, to determine any digit of the result, it is sufficient to perform two additional operations. Finally, note that the most significant bit of the maximum is simply $a_{n-1} \vee b_{n-1}$.

To compute the minimum along with the maximum, it is sufficient to attach another n gates to the circuit, since $\min(A, B) = \overline{(A \sim B)} \cdot [c]^n \oplus A$.

Decoder. The operator $\mathtt{DEC}_n : [\![n]\!] \to \mathbb{B}^n$ computes a boolean string of length n with a single one at a given position. The components of the operator are elementary conjunctions X^α of variables X, where the vector α runs over the set of binary representations of numbers from 0 to $n-1$.

The upper bound $\mathsf{C}_{\log}(\mathtt{DEC}_n) \leq n + \Theta(\sqrt{n})$ is obtained trivially by dividing the set of variables in half: if $X = [X_2, X_1]$, then $X^{\alpha_2 \| \alpha_1} = X_2^{\alpha_2} \wedge X_1^{\alpha_1}$. Hence, $\mathsf{C}_{\log}(\mathtt{DEC}_n) \leq n + \mathsf{C}_{\log}(\mathtt{DEC}_{2^k}) + \mathsf{C}_{\log}(\mathtt{DEC}_{\lceil n2^{-k} \rceil})$, where $k = |X_1|$. It remains to choose $k \approx \log n / 2$.

A lower bound of the form $\mathsf{C}(\mathtt{DEC}_n) \geq n + \Theta(\sqrt{n})$ is easily established by observing that the set of circuit elements immediately preceding the outputs has cardinality at least $\sqrt{n}$.

The extended operator $\mathtt{DEC}_n^* : [\![n]\!] \times \mathbb{B} \to \mathbb{B}^n$ has an additional information input $y \in \mathbb{B}$ and computes a vector with bit y in a given position and the rest set to zeros. Such an operator is often called a demultiplexor. Its complexity differs from that of the decoder by no more than 2: in the decoder circuit, it suffices to replace the least significant variable x and its negation with xy and $\overline{x} \cdot y$, respectively.

Multiplexors. The function $\mathtt{MUX}_n : [\![n]\!] \times \mathbb{B}^n \to \mathbb{B}$ selects one of n information boolean variables by its index (otherwise called an address). This function is called an $(n, 1)$-multiplexor, as well as a selector, a switching function, or a memory access function.

The best known upper bound $\mathsf{C}_{\log}(\mathtt{MUX}_n) \leq 2n + O(\sqrt{n})$ was obtained by P. Klein and M. Paterson in [8]. Represent the address input as $X = [X_2, X_1]$, where $|X_1| = q$. Divide the string of information variables into blocks of length 2^q: $Y = Y_0 \| Y_1 \| \ldots \| Y_{p-1}$,

$p = \lceil n/2^q \rceil$ (the last block can be shorter). Decompose the function MUX_n over variables X_2:

$$\text{MUX}_n(X;Y) = \bigvee_{\alpha=0}^{p-1} X_2^\alpha \cdot \text{MUX}_{|Y_\alpha|}(X_1;Y_\alpha). \tag{5}$$

Inner multiplexor functions can be calculated by formulas

$$\text{MUX}_{|Y_\alpha|}(X_1;Y_\alpha) = \bigvee_{\beta=0}^{|Y_\alpha|-1} y_{\alpha,\beta} \cdot X_1^\beta, \tag{6}$$

where the introduction of a two-index numbering on the set of variables $Y = \{y_{\alpha,\beta}\}$ is implied.

All elementary conjunctions of groups of variables X_1 and X_2 are computed with complexity of order $2^q + p$ using the decoders DEC_{2^q} and DEC_p. Another $2n + p$ operations are sufficient to complete the computations by formulas (5), (6). It remains to choose $q \approx \log n/2$.

This bound is asymptotically tight: W. Paul [18] established that $\mathsf{C}(\text{MUX}_n) \geq 2n - 2$.

A more general problem often arises: implementing an (n, k)-multiplexor MUX_n^k : $[\![n]\!] \times (\mathbb{B}^k)^n \to \mathbb{B}^k$, whose information inputs are k-bit numbers or boolean vectors. The circuit for the operator MUX_n^k is obtained by parallel combining the $(n, 1)$-multiplexor circuits described above for each of the k bits. These circuits have a common part $\text{DEC}_{2^q}(X_1)$ and $\text{DEC}_p(X_2)$. Choosing $q = \lceil \min(\log n, \log(kn)/2) \rceil$ yields the bound $\mathsf{C}_{\log}(\text{MUX}_n^k) \leq 2kn + O(\sqrt{kn})$.

Shift operators. The operator $\text{CYC}_{k,n} : [\![k]\!] \times \mathbb{B}^n \to \mathbb{B}^n$ performs a cyclic shift of a boolean vector of length n by $X \in [\![k]\!]$ positions[3].

The upper bound $\mathsf{C}_{\log}(\text{CYC}_{k,n}) \leq 3\lceil \log k \rceil n$ is trivial. The corresponding circuit has the form of an l-fold composition $\text{MUX}_2^n \circ \ldots \circ \text{MUX}_2^n$, where $l = \lceil \log k \rceil$. Write $X = [x_{l-1}, \ldots, x_1, x_0]$. The first subcircuit performs a cyclic shift by x_0, the second by $2x_1$, the third by $4x_2$, and so on.

The regular shift operator $\text{SFT}_{k,n} : [\![k]\!] \times \mathbb{B}^n \to \mathbb{B}^{n+k-1}$ is implemented similarly. The bound $\mathsf{C}_{\log}(\text{SFT}_{k,n}) \leq 3\lceil \log k \rceil n - 2(k-1)$ holds, taking into account that $k - 1$ of $(2, 1)$-multiplexors in the above circuit may be reduced to a single operation.

Encoder. The operator $\text{ENC}_n : \mathbb{B}^n \to [\![n]\!]$ is a linear boolean operator with a matrix U_n of size $\lceil \log n \rceil \times n$, whose columns contain sequentially the numbers from 0 to $n-1$ in binary representation[4]. For an input vector of weight 1, the encoder determines the position of a single one. Therefore, on the set of vectors of weight 1, the encoder ENC_n is the inverse transform of DEC_n.

It is known that $\mathsf{C}(\text{ENC}_n) = \mathsf{C}_{\log}(\text{ENC}_n) = 2(n - \lceil \log n \rceil - 1)$. The lower bound was proved by A. V. Chashkin [3]. The upper bound can be obtained trivially via the well-known relation between the complexity of linear operators with transposed matrices (the transposition principle [14]): the complexity of an operator with a matrix U_n^T is simply $n - \lceil \log n \rceil - 1$. However, it is not hard to explicitly describe the construction of the corresponding circuits. We define a family of circuits Φ_n computing ENC_n recursively, together with circuits Φ_n' that implement transforms ENC_n' with matrices U_n padded with rows of all ones. Let $n = 2^k + p \leq 2^{k+1}$. Split a vector X of length n into two parts X_1, X_2 of length

---

[3]The direction of the shift is irrelevant for complexity estimates.

[4]For $n = 2^k$, this matrix is the parity-check matrix of the Hamming code up to the presence of a zero column.

2^k and p. The circuit $\Phi_n(X)$ is obtained from $\Phi_{2^k}(X_1)$ and $\Phi'_p(X_2)$ by attaching additional $\lceil \log p \rceil$ gates $\oplus$, and $\Phi'_n(X)$ is obtained from $\Phi'_{2^k}(X_1)$ and $\Phi'_p(X_2)$ by adding $\lceil \log p \rceil + 1$ gates. Recurrence relations

$$\mathsf{C}(\Phi_n) = \mathsf{C}(\Phi_{2^k}) + \mathsf{C}(\Phi'_p) + \lceil \log p \rceil, \qquad \mathsf{C}(\Phi'_n) = \mathsf{C}(\Phi'_{2^k}) + \mathsf{C}(\Phi'_p) + \lceil \log p \rceil + 1$$

are resolved as $\mathsf{C}(\Phi_n) = 2(n - \lceil \log n \rceil - 1)$ and $\mathsf{C}(\Phi'_n) = 2n - \lceil \log n \rceil - 2$ taking into account the initial conditions $\mathsf{C}(\Phi_1) = \mathsf{C}(\Phi'_1) = 0$. The depth of Φ_n and Φ'_n circuits is $\lceil \log n \rceil - 1$ and $\lceil \log n \rceil$ respectively. See [4] for more details.

In an alternative definition, the encoder $\mathtt{ENC}^*_n$ is a partial boolean operator defined only on the set of weight-1 vectors and coinciding with $\mathtt{ENC}_n$ on this set. Following the proof of the lower bound in [4], it is easy to verify that weakening the definition does not yield a significant gain: $\mathsf{C}(\mathtt{ENC}^*_n) = 2n - \Theta(\log n)$.

Unary encoding. The operator $\mathtt{UN}_n : [\![n+1]\!] \to \mathbb{B}^n$ converts a number from standard binary notation to unary notation in which the number k is written as a string starting with k ones and followed by zeros.

The upper bound $\mathsf{C}(\mathtt{UN}_n) \le 2n + O(\sqrt{n})$ is easily achieved by a circuit of type $\mathtt{PREF}^\vee \circ \mathtt{DEC}$. The operator $\mathtt{DEC}_{n+1}(x)$ computes a vector with a single one at position x. Discarding the least significant component of the vector, compute the suffix sums of the remaining components.

The given bound may be improved to $\mathsf{C}_{\log}(\mathtt{UN}_n) \le 2n + O(\sqrt{n})$. First, inductively construct circuits for $\mathtt{UN}_{2^k-1}$ of complexity $2(2^k - k - 1)$ and depth $k - 1$. Let a boolean string $S = [s_1, \ldots, s_{2^k-1}]$ be obtained as $S = \mathtt{UN}_{2^k-1}(X)$, where X is a boolean vector representing a k-bit number. Then for $y \in \mathbb{B}$,

$$\mathtt{UN}_{2^{k+1}-1}(y, X) = \begin{cases} S \,\|\, [0]^{2^k}, & y = 0 \\ [1]^{2^k} \,\|\, S, & y = 1 \end{cases} = [s_1 \vee y, \ \ldots, \ s_{2^k-1} \vee y, \ y, \ s_1 \cdot y, \ \ldots, \ s_{2^k-1} \cdot y].$$

Thus, the circuit for $\mathtt{UN}_{2^{k+1}-1}$ is constructed from the circuit for $\mathtt{UN}_{2^k-1}$ by adding a layer of $2(2^k - 1)$ conjunction and disjunction gates. Hence, given $\mathsf{C}(\mathtt{UN}_1) = 0$, the required estimates follow.

Next, we construct a circuit for $\mathtt{UN}_n$ using the block method. Let $X = [X_2, X_1]$, where $|X_1| = k \approx \log n / 2$. Denote $p = \lceil (n+1)/2^k \rceil$. The result $\mathtt{UN}_n(X)$ can be written in block form as $B_0 \,\|\, B_1 \,\|\, \ldots \,\|\, B_{p-1}$, where all substrings B_i have length 2^k, except for the last that is incomplete[5]. Compute $\mathtt{DEC}_p(X_2) = [a_0, a_1, \ldots, a_{p-1}]$. This is a vector indicating the position of the first not all-ones block. Such a block has the form $S = \mathtt{UN}_{2^k-1}(X_1) \,\|\, 0$ (shortened in the case of the last block). To the left of it the blocks are all-ones, to the right are all-zeros. Via the operator $\mathtt{PREF}^\vee_p$ compute the prefix sums $b_i = a_0 \vee a_1 \vee \ldots \vee a_i$. Then we have

$$B_i = \begin{cases} [1]^{2^k}, & a_i = b_i = 0 \\ S, & a_i = b_i = 1 \\ [0]^{2^k}, & a_i = 0, b_i = 1 \end{cases} = S \cdot [a_i]^{2^k} \vee [\overline{b_i}]^{2^k}, \tag{7}$$

where on the right-hand side, operations with boolean strings are performed bitwise; for $i = p - 1$, the block is truncated. The resulting circuit is composed of subcircuits $\mathtt{DEC}_p$, $\mathtt{PREF}_p$, $\mathtt{UN}_{2^k-1}$ of complexity $O(\sqrt{n})$ and a layer of $2n$ operations following (7).

---

[5]Its length is from 0 to $2^k - 1$.

The inverse transform $\text{UN}_n^{-1} : \mathbb{B}^n \to [\![n+1]\!]$ from unary to binary encoding is easy: $\mathsf{C}_{\log}(\text{UN}_n^{-1}) = n - 1$. The digits of the number $[y_k, \ldots, y_0] = \text{UN}_n^{-1}(s_1, \ldots, s_n)$ are determined by formulas

$$y_j = \bigoplus_{i \geq 1} s_{i \cdot 2^j}.$$

All the necessary sums can be computed in a tree of $n - 1$ gates $\oplus$ with depth $\lceil \log n \rceil$.

Truncation. The operator $\text{TRN}_n : [\![n+1]\!] \times \mathbb{B}^n \to \mathbb{B}^n$ in a boolean string of length n preserves the first k bits and fills the rest of the string with zeros.

The upper bound $\mathsf{C}_{\log}(\text{TRN}_n) \leq 3n + O(\sqrt{n})$ is straightforward since $\text{TRN}_n(k; X) = \text{UN}_n(k) \wedge X$ (the conjunction is performed bitwise).

First-one indicator. The operator $\text{FOI}_n : \mathbb{B}^n \to \mathbb{B}^n \times \mathbb{B}$ leaves the very first one in a boolean string of length n, replacing the rest with zeros, and additionally computes the indicator of the presence of a one in the string.

It is easy to see that the operator transforms the string $X = [x_0, x_1, \ldots, x_{n-1}]$ into $Y = [y_0, y_1, \ldots, y_{n-1}; z]$, where $y_k = \overline{(x_0 \vee \ldots \vee x_{k-1})} \cdot x_k$ and $z = x_0 \vee \ldots \vee x_{n-1}$. To compute it, it suffices to add a layer of $n - 1$ gates of type $\overline{a} \wedge b$ to a circuit that computes $\text{PREF}_n^{\vee}(X)$. Consequently, $\mathsf{C}(\text{FOI}_n) \leq 2n - 2$ and $\mathsf{C}_{\log}(\text{FOI}_n) \leq 3n - \Theta(\log n)$.

The first bound is tight: it is easy to check that the circuit implementing FOI_n, after substituting $x_{n-1} = 0$, computes FOI_{n-1} and at the same time at least two gates may be eliminated.

Priority encoder. The operator $\text{PENC}_n : \mathbb{B}^n \to [\![n]\!] \times \mathbb{B}$ determines the position of the first one in a boolean string of length n and additionally computes the indicator of the presence of ones in the string[6].

Upper bounds $\mathsf{C}(\text{PENC}_n) \leq 2n - 2$ and $\mathsf{C}_{\log}(\text{PENC}_n) \leq 3n - \Theta(\log n)$ are achieved by circuits of the form $\text{UN}^{-1} \circ \text{PREF}$. Indeed, the first-one position of a string X can be found as $\text{UN}_n^{-1}(\overline{\text{PREF}_n^{\vee}(X)})$, where the negation is applied bitwise. By the way, the internal operator $\text{PREF}_n^{\vee}$ computes the presence of ones. For $n \geq 3$, even $\mathsf{C}(\text{PENC}_n) \leq 2n - 3$ is true: the first-one position is correctly computed as $\text{UN}_{n-1}^{-1}(\overline{\text{PREF}_{n-1}^{\vee}(X')})$, where X' is the string X with the last bit removed.

The lower bound can be stated as $\mathsf{C}(\text{PENC}_n) \geq \mathsf{C}(\text{ENC}_n^*) = 2n - \Theta(\log n)$, since the operator PENC_n is an extension of ENC_n^*.

Summation of bits. The operator $\text{SUM}_n : \mathbb{B}^n \to [\![n+1]\!]$ computes the arithmetic sum of n boolean variables.

The complexity of bit summation is fairly well studied. Efficient bit summation circuits are built from compressor subcircuits. A compressor transforms several bit inputs into a smaller number of outputs while preserving the sum (taking into account significance of bits). An example of summation circuit composed of $(3, 2)$-compressors $\Sigma_{3,2}$, which compute the sum of three bits, is shown in Fig. 2.

In [5], a circuit of complexity $4.5n - \Theta(\log n)$ was constructed from special $(5, 3)$-compressors. At the cost of increasing the complexity to $4.5n + o(n)$, the circuit can be made parallel [21]. For this, for example, one can divide the inputs into groups of $\log n$ pieces, calculate the sums in the groups by the method [5], then sum the group sums via any parallel compressor circuit. The first stage has complexity $4.5n - o(n)$, and the second $o(n)$. Thus, $\mathsf{C}(\text{SUM}_n) \leq 4.5n - \Theta(\log n)$ and $\mathsf{C}_{\log}(\text{SUM}_n) \leq 4.5n + o(n)$. Constructing parallel circuits of compressors is discussed in detail in [17], see also [28, §3.2].

---

[6] This is a partial boolean operator: the position is undefined for zero input.

Figure 2: Standard circuit for summing n bits

The lower bound can be stated as $\mathsf{C}(\mathtt{SUM}_n) \geq 2.5n + \Theta(\log n)$. It is obtained by combining the bound $2.5n - O(1)$ proved by L. Stockmeyer [24], which holds for every component of the operator $\mathtt{SUM}_n$ except perhaps the lowest and several highest ones, and the simple relation [13] for the complexity of a system of functions:

$$\mathsf{C}(f_1, \ldots, f_k) \geq \min_i \mathsf{C}(f_i) + k - 1. \tag{8}$$

Counting the width of a block. The operator $\mathtt{BW}_n : \mathbb{B}^n \to [\![n + 1]\!]$ for a boolean string of length n that contains a single block of ones, computes the width of this block.

Upper bounds $\mathsf{C}(\mathtt{BW}_n) \leq 4n - \Theta(\log n)$ and $\mathsf{C}_{\log}(\mathtt{BW}_n) \leq 4n + o(n)$ are proved via the method of compressors. The basic circuit is constructed as in Fig. 2 from compressor subcircuits $\Sigma_{3,2} : (a, b, c) \to [u, v]$, summing triplets of bits: $a + b + c = 2u + v$. Provided that the ones at the input are located in one continuous block, such subcircuits can be implemented with complexity 4: $u = b(a \vee c)$, $v = a \oplus b \oplus c$. It is easy to see that if a string containing a single one-block is fed to the input of the circuit in Fig. 2, then the ones at the input of each internal compressor will be adjacent, in other words, the input vector $(1, 0, 1)$ will never occur. The total complexity of the described circuit is $4n - \Theta(\log n)$. A parallel circuit may be obtained as explained in the previous paragraph.

Threshold symmetric functions. The operator $\mathtt{THR}_n^k : \mathbb{B}^n \to \mathbb{B}$ compares the number of ones in a boolean string X of length n with a threshold k and thus computes the value of the predicate $\nu(X) \geq k$.

Upper bounds $\mathsf{C}(\mathtt{THR}_n^k) \leq 4.5n + O(\log n)$ and $\mathsf{C}_{\log}(\mathtt{THR}_n^k) \leq 4.5n + o(n)$ follow directly from the corresponding complexity bounds for the bit summation operator. It suffices to attach a circuit comparing the sum with the threshold to the circuit implementing $\mathtt{SUM}_n$. L. Stockmeyer [24] proved the lower bound $\mathsf{C}(\mathtt{THR}_n^k) \geq 2n + \min\{k - 2, n - k - 1\} - 3$.

Circuits for other symmetric functions, such as periodic ones, can be constructed similarly.

For a constant threshold k, special methods of synthesis allow to obtain better complexity bounds. In particular, $\mathsf{C}(\mathtt{THR}_n^2) = \mathsf{C}_{\log}(\mathtt{THR}_n^2) = 2n + \Theta(\sqrt{n})$, $\mathsf{C}_{\log}(\mathtt{THR}_n^3) \leq 3n + O(\log n)$, and in general for $2^{p-1} < k \leq 2^p$ we have $\mathsf{C}_{\log}(\mathtt{THR}_n^k) \leq (4.5 - 2^{2-p})n + \theta_k(n)$, where $\theta_k(n) = O(\sqrt{n})$ or $\theta_k(n) = O(\log n)$, see [22].

Sorting. The operator $\mathtt{SORT}_n : \mathbb{B}^n \to \mathbb{B}^n$ sorts a bit string in ascending order (zeros on the left, ones on the right).

Upper bounds $\mathsf{C}(\mathtt{SORT}_n) \leq 6.5n + O(\sqrt{n})$ and $\mathsf{C}_{\log}(\mathtt{SORT}_n) \leq 6.5n + o(n)$ are achieved

by circuits of the type $\mathtt{UN} \circ \mathtt{SUM}$. First, the operator $\mathtt{SUM}_n$ calculates the number of ones in the string, then $\mathtt{UN}_n$ computes the unary representation of this number.

The lower bound $\mathsf{C}_{\log}(\mathtt{SORT}_n) \geq 3n - 6$ is obtained by combining the lower bound $2n - 3$ due to B. M. Kloss [9], which holds for any component of the operator except the lowest and highest (minimum and maximum), and the bound (8).

Table 1: Complexity bounds for basic operators

operator F	lower bound $\mathsf{C}(F)$	upper bound $\mathsf{C}(F)$	upper bound $\mathsf{C}_{\log}(F)$
$\mathtt{PREF}_n$	$n - 1$		$2n - \Theta(\log n)$ [16]
$\mathtt{PS}_n$	$2n - 3$		$3n - \Theta(\log n)$
$\mathtt{INC}_n$	$2n - 2$		$3n - \Theta(\log n)$
$\mathtt{UDC}_n$	—	$3n - 3$	$4n - \Theta(\log n)$
$\mathtt{GRC}_n$	—	$4n - 7$	$6n - \Theta(\log n)$
$\mathtt{CAR}_n$	$2n - 2$		$5n - \Theta(\log n)$
$\mathtt{ADD}_n$	$5n - 3$ [19]		$8n - \Theta(\log n)$
$\mathtt{CMP}_n$	—	$4n - 3$	$5n - \Theta(\log n)$
$\mathtt{MAX}_n$	—	$6n - 3$	$7n - \Theta(\log n)$
$\mathtt{DEC}_n$	$n + \Theta(\sqrt{n})$		
$\mathtt{MUX}_n$	$2n - 2$ [18]	$2n + O(\sqrt{n})$ [8]	
$\mathtt{MUX}_n^k$	—	$2kn + O(\sqrt{kn})$	
$\mathtt{CYC}_{k,n}$	—	$3\lceil \log k \rceil n$	
$\mathtt{SFT}_{k,n}$	—	$3\lceil \log k \rceil n - \Theta(k)$	
$\mathtt{ENC}_n$	$2(n - \lceil \log n \rceil - 1)$ [3]		
$\mathtt{UN}_n$	—	$2n + O(\sqrt{n})$	
$\mathtt{UN}_n^{-1}$	$n - 1$		
$\mathtt{TRN}_n$	—	$3n + O(\sqrt{n})$	
$\mathtt{FOI}_n$	$2n - 2$		$3n - \Theta(\log n)$
$\mathtt{PENC}_n$	$2n - \Theta(\log n)$	$2n - 3$	$3n - \Theta(\log n)$
$\mathtt{SUM}_n$	$2.5n + \Theta(\log n)$ [24, 13]	$4.5n - \Theta(\log n)$ [5]	$4.5n + o(n)$ [21]
$\mathtt{THR}_n^k$	$2n + \min\{k, n - k\} - 5$ [24]	$4.5n + O(\log n)$	$4.5n + o(n)$
$\mathtt{BW}_n$	—	$4n - \Theta(\log n)$	$4n + o(n)$
$\mathtt{SORT}_n$	$3n - 6$ [9, 13]	$6.5n + O(\sqrt{n})$	$6.5n + o(n)$

Some applications

In this section, we discuss several applications of the constructions mentioned above to building some specific parallel circuits. The first example illustrates the use of parallel prefix-suffix circuits. The second example relies on an efficient implementation of unary-binary transforms. The third example illustrates the principle of mass production and leads to the fourth example.

Two-selector. The operator $\mathtt{TOI}_n : \mathbb{B}^n \to \mathbb{B}^n \times \mathbb{B}$ preserves two ones in a boolean string of length n, replacing the rest with zeros, and additionally computes the indicator of the presence of ones in the string. This operator is an extension of the operator $\mathtt{FOI}_n$

and is used to select two active channels marked with ones in a set of n data channels, see, e.g., [1].

The bound $\mathsf{C}_{\log}(\mathtt{TOI}_n) \leq 5n - \Theta(\log n)$ is achieved by a circuit that selects two extreme ones on different sides. Given the original string $[x_1, \ldots, x_n]$, compute the string of prefix sums $[p_1, \ldots, p_n] = [0]^k[1]^{n-k}$ and the string of suffix sums $[s_1, \ldots, s_n] = [1]^l[0]^{n-l}$ applying the operator $\mathtt{PS}_n^{\vee}$, where $k+1$ and l are the positions of the first and last ones (if there are ones). In this case, $p_n = s_1$ serves as an indicator of the presence of ones. The resulting string $[z_1, \ldots, z_n]$ may be computed bitwise as $z_i = x_i \cdot (\overline{p_{i-1} \vee s_{i+1}})$.

Weight-preserving counter. The operator $\mathtt{NCK}_n : \llbracket 2^n \rrbracket \to \llbracket 2^n \rrbracket$ computes the next number in ascending order after a given one with the same binary weight. If there is no greater number, the output is the same as the input. This operator was discussed, in particular, in [15] in connection with applications in image processing. There it is called $\binom{n}{k}$-counter. The complexity of a parallel implementation of the operator $\mathtt{NCK}_n$ in [15] is stated as $O(n)$, but a bound of about $16n$ may be extracted from the circuit description.

The upper complexity bound can be refined to $\mathsf{C}_{\log}(\mathtt{NCK}_n) \leq 13n + o(n)$. By appending an extra zero to the left of a non-zero n-bit number, this number can be uniquely represented as $S \,\|\, 0\,[1]^j[0]^i$, where $i \geq 0$, $j \geq 1$, and S is a non-empty substring in the case when the number is not maximal among numbers of the same weight. Then the operator $\mathtt{NCK}_n$ transforms this non-maximal number as $S \,\|\, 0\,[1]^j[0]^i \to S \,\|\, 1\,[0]^{i+1}[1]^{j-1}$. The sequence of computations leading to the required result is shown in Fig. 3, where $*$ denotes irrelevant parts of strings, $\mathtt{bit[]}$ denotes the corresponding bitwise operation. The arguments and results of some operations are shifted by 1. Extra bits are shown for generality of notation; they are not used in the calculations. The circuit consists of subcircuits implementing the operators $\mathtt{PREF}_n^{\vee}$, $\mathtt{UN}_n$, $\mathtt{UN}_n^{-1}$ (twice), seven bitwise operations with n-bit vectors, and subtraction of $\log n$-bit numbers. The indicator c of the maximality of an input number can be written as $(i + j = n)$. It is computed in the second step as the second from the left (i.e., the most significant) bit of the number $[0]^{n+1-i-j}[1]^{i+j}$. In the final step of the algorithm, when $c = 1$, the input number is selected; when $c = 0$, the result is composed of the fragment S of the input number and the computed fragment $1\,[0]^{i+1}[1]^{j-1}$. The circuit also works correctly with zero input.
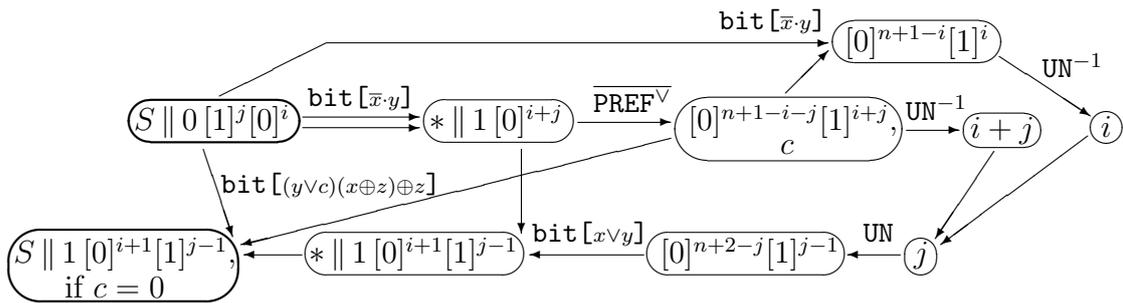

Figure 3: Computational scheme of the weight-preserving counting circuit

In practical circuit design, the condition $\bar{c}$ can be used separately as an indicator of the result's validity (for storing it in a register), then the final operation can be simplified to $\mathtt{bit}[xy \vee z\bar{y}]$. In this case, the circuit complexity decreases to $12n + o(n)$.

Multiple selection. Let us consider another generalization of the multiplexor: the operator $\mathtt{SEL}_n^k : \llbracket n \rrbracket^k \times \mathbb{B}^n \to \mathbb{B}^k$ selects k of n boolean information variables by their addresses. D. Uhlig's method (see [26, 28]) allows to construct smaller circuits for multise-

lection compared to a straightforward collection of k independent multiplexor circuits. To demonstrate the idea, we restrict ourselves to the case $k = 2$.

Let us show that $\mathsf{C}_{\log}(\mathtt{SEL}_n^2) \le 3n + O(n^{2/3})$. Denote $m = \lceil \log n \rceil$. Split a string $Y \in \mathbb{B}^n$ into 2^r substrings Y_i, $0 \le i < 2^r$, where the string $Y_i = [y_i, y_{i+2^r}, y_{i+2\cdot2^r}, \ldots]$ includes variables with indices equal to i modulo 2^r. By construction, $|Y_i| = q := \lceil n/2^r \rceil$ (we pad shorter strings to length q arbitrarily). Compute the strings

$$\widetilde{Y}_0 = Y_0, \ \ \widetilde{Y}_1 = Y_0 \oplus Y_1, \ \ldots, \ \ \widetilde{Y}_i = Y_{i-1} \oplus Y_i, \ \ldots, \ \ \widetilde{Y}_{2^r-1} = Y_{2^r-2} \oplus Y_{2^r-1}, \ \ \widetilde{Y}_{2^r} = Y_{2^r-1}, \ \ (9)$$

where addition operations are bitwise. Note that for any i,

$$\widetilde{Y}_0 \oplus \ldots \oplus \widetilde{Y}_i = Y_i = \widetilde{Y}_{i+1} \oplus \ldots \oplus \widetilde{Y}_{2^r}.$$

Let $a = [a_1, a_0]$ and $b = [b_1, b_0]$ denote the address inputs of the circuit – there we distinguish groups of the least significant r bits: $|a_0| = |b_0| = r$.

Thus, if $a_0 \le b_0$, then we can determine

$$\mathtt{SEL}_n^2(a, b; Y) = [\mathtt{MUX}_n(a; Y), \mathtt{MUX}_n(b; Y)] = [\mathtt{MUX}_q(a_1; Y_{a_0}), \mathtt{MUX}_q(b_1; Y_{b_0})]$$

implementing operators $\mathtt{MUX}_q(z_i; \widetilde{Y}_i)$, while setting $z_i = a_1$ for $0 \le i \le a_0$ and $z_i = b_1$ for $b_0 < i \le 2^r$. In the case $a_0 > b_0$, the roles of a and b are swapped. In accordance with this rule, we introduce indicator functions

$$\eta_i^a = (i \le a_0 \le b_0) \vee (i > a_0 > b_0), \quad \eta_i^b = (i > b_0 \ge a_0) \vee (i \le b_0 < a_0), \qquad (10)$$

which choose whether to use the subcircuit $\mathtt{MUX}_q(z_i; \widetilde{Y}_i)$ to compute $\mathtt{MUX}_n(a; Y)$ or to compute $\mathtt{MUX}_n(b; Y)$. The final result is determined by formulas

$$\mathtt{MUX}_n(a; Y) = \bigoplus_{i=0}^{2^r} \eta_i^a \, \mathtt{MUX}_q(z_i; \widetilde{Y}_i), \quad \mathtt{MUX}_n(b; Y) = \bigoplus_{i=0}^{2^r} \eta_i^b \, \mathtt{MUX}_q(z_i; \widetilde{Y}_i),$$

$$z_i = [\eta_i^a]^{m-r} \cdot a_1 \vee [\eta_i^b]^{m-r} \cdot b_1, \quad (11)$$

where the operations in the last formula are bitwise. The circuit complexity results from the computations implied by (9), (10), (11), and is estimated as

$$\mathsf{C}_{\log}(\mathtt{SEL}_n^2) \le (2^r - 1)q + (2^r + 1)\mathsf{C}_{\log}(\mathtt{MUX}_q) + O(2^r m) \le 3 \cdot 2^r q + O(2^r(m + \sqrt{q}) + q).$$

The required bound is obtained by choosing $r \approx \log n/3$. The method is already practical for $r = 1$.

In fact, a linear complexity upper bound $\mathsf{C}_{\log}(\mathtt{SEL}_n^k) = O(n)$ holds for all $k \le n/\log^3 n$, as shown in [7] by a rather nontrivial method.

Permutation of a pair of bits. The operator $\mathtt{EXC}_n : [\![n]\!]^2 \times \mathbb{B}^n \to \mathbb{B}^n$ permutes two bits with given addresses in a boolean string of length n. This fairly popular operation (especially in programming) is discussed, for example, in [11, §7.1.3].

Based on the result of the previous paragraph, it is easy to derive the bound $\mathsf{C}_{\log}(\mathtt{EXC}_n) \le 7n + O(n^{2/3})$. The solution also exploits the well-known trick of exchanging the contents of two registers, see [27, Chapter 2]. First, compute the required pair of bits $[u, v] = \mathtt{SEL}_n^2(a, b; Y)$. Then, via two demultiplexors, produce the strings $\mathtt{DEC}_n^*(a; u \oplus v)$ and $\mathtt{DEC}_n^*(b; u \oplus v)$ containing bits $u \oplus v$ in each of the two given positions a, b. At last, add these strings bitwise to each other and to the input string Y (modulo 2).

References

[1] Ahn J.-S., Jeong D.-K., Yang M. *Fast three-dimensional programmable two-selector.* Electronics Letters. 2004. **40**(18), 1098–1100.

[2] Blelloch G. E. *Prefix sums and their applications.* Synthesis of parallel algorithms. San Francisco: Morgan Kaufmann, 1993, 35–60.

[3] Chashkin A. V. *On the complexity of Boolean matrices, graphs and their corresponding Boolean functions.* Discrete Math. and Appl. 1994, **4**(3), 229–257.

[4] Chashkin A. V. *Discrete mathematics.* Moscow: Akademija, 2012. (in Russian)

[5] Demenkov E., Kojevnikov A., Kulikov A., Yaroslavtsev G. *New upper bounds on the Boolean circuit complexity of symmetric functions.* Inform. Proc. Letters. 2010. **110**(7), 264–267.

[6] Harris D. M., Harris S. L. *Digital design and computer architecture.* San Francisco: Morgan Kaufmann, 2012.

[7] Holmgren J., Rothblum R. *Linear-size Boolean circuits for multiselection.* Proc. CCC (Ann Arbor, USA, 2024). LIPIcs. 2024. **300**. Art. 11.

[8] Klein P., Paterson M. S. *Asymptotically optimal circuit for a storage access function.* IEEE Trans. on Computers. 1980. C-**29**(8), 737–738.

[9] Kloss B. M., Malyshev V. A. *Complexity bounds for some classes of functions.* Vestnik Mosk. Univ. Ser. 1. Matematika. Mekhanika. 1965. (4), 44–51. (in Russian).

[10] Knuth D. E. *The art of computer programming. Vol. 2. Seminumerical algorithms.* Reading: Addison-Wesley, 1997.

[11] Knuth D. E. *The art of computer programming. Vol. 4A. Combinatorial algorithms, part 1.* Upper Saddle River: Addison-Wesley, 2011.

[12] Ladner R. E., Fischer M. J. *Parallel prefix computation.* J. ACM. 1980. **27**(4), 831–838.

[13] Lamagna E. A., Savage J. E. *On the logical complexity of symmetric switching functions in monotone and complete bases.* Tech. report. Brown Univ., 1973.

[14] Mitiagin B. S., Sadovskii B. N. *On linear Boolean operators.* Doklady AN SSSR. 1965. **165**(4), 773–776. (in Russian)

[15] Nakano K., Yamagishi Y. *Hardware n choose k counters with applications to the partial exhaustive search.* IEICE Trans. on Information & Systems. 2005. E**88**-D(7), 1350–1359.

[16] Ofman Yu. P. *On the algorithmic complexity of discrete functions.* Soviet Physics Doklady. 1963. **7**, 589–591.

[17] Paterson M. S., Pippenger N., Zwick U. *Optimal carry save networks.* LMS Lecture Notes Series. Boolean function complexity. Vol. 169. Cambridge Univ. Press, 1992, 174–201.

[18] Paul W. J. *A 2.5n-lower bound on the combinational complexity of Boolean functions.* SIAM J. Comput. 1977. **6**(3), 427–443.

[19] Red'kin N. P. *On the minimal implementation of a binary adder.* In: Problemy Kibernetiki. Vol. 38. Moscow: Nauka, 1981, 181–216. (in Russian)

[20] Sergeev I. S. *On the complexity of parallel prefix circuits.* ECCC report TR13–041. 2013.

[21] Sergeev I. S. *Upper bounds on the depth of symmetric Boolean functions.* Moscow Univ. Comput. Math. and Cybern. 2013. **37**(4), 195–201.

[22] Sergeev I. S. *On the complexity of monotone circuits for threshold symmetric Boolean functions.* Discrete Math. and Appl. 2021. **31**(5), 345–366.

[23] Sklansky J. *Conditional-sum addition logic.* IRE Trans. Electr. Comput. 1960. EC-**9**, 226–231.

[24] Stockmeyer L. J. *On the combinational complexity of certain symmetric Boolean functions.* Math. Systems Theory. 1977. **10**, 323–336.

[25] Ugryumov E. P. *Digital circuit design.* Saint Petersburg: BHV-Peterburg, 2010. (in Russian)

[26] Uhlig D. *Networks computing Boolean functions for multiple input values.* LMS Lecture Notes Series. Boolean function complexity. Vol. 169. Cambridge Univ. Press, 1992, 165–173.

[27] Warren H. S., Jr. *Hacker's delight.* Upper Saddle River: Addison-Wesley, 2002.

[28] Wegener I. *The complexity of Boolean functions.* Stuttgart: Wiley–Teubner, 1987.