On Tightening The Bounds of Multiplexer

November 2025

Abstract

Given its nature — selecting a bit from a list of data bits based on an address encoded in binary — the Multiplexing function (MUX) captures an essential form of computation: indirect addressing or memory lookup. Despite its simple nature, proving tight lower and upper bounds for its circuit complexity has remained open for decades. In this work, we revisit the complexity of the multiplexer function MUX and provide two main contributions.

1. We improve the $2N - 2$ lower bound (Paul, 1975) to $2N + \log N$ via a more intricate application of Gate Elimination (Section 3).
2. We revisit the $2N + O(\sqrt{N})$ upper bound (Klein & Patterson, 1980) and refine the construction by giving exact summation formulas for our recursive constructions and providing a clean and easy to analyze construction that helps provide better understanding on how MUX-function works (Appendix B).

1 Introduction

1.1 Overview

Understanding the circuit complexity of natural Boolean functions is a fundamental pursuit in theoretical computer science for over five decades. Among existing techniques, Gate Elimination — a robust method for proving circuit lower bounds by analyzing the effects of input substitution — has emerged as one of the most tractable and intuitive methods for proving circuit lower bounds. In particular, current known lower bounds in the DeMorgan, $\mathcal{U}_2$ [Red73; Sch74; Zwi91; LR01; IM02], and $\mathcal{B}_2$ [Sch74; Sto77; DK11; FGHK16; LY22] basis were proven via Gate Elimination. However, despite its appeal and the long-standing interest, progress in proving strong lower bounds remains limited. To date, no super-linear lower bounds are known for general Boolean circuit classes over different bases, and breaching this barrier via Gate Elimination alone remains elusive [Weg87]. In this work, we consider the DeMorgan Basis: fan-in 2 AND, fan-in 2 OR, and fan-in 1 NOT gates. We consider the complexity measure which counts *only* the binary gates. The table below summarizes the best known upper bound and lower bound of some explicit well-known Boolean functions under our measurement.

Function(s)	Lower Bound		Upper Bound	Source(s)
XOR	$\mathbf{3(n-1)}$	=	$\mathbf{3(n-1)}$	[Sch74]
Multiplexer	$2(n-1)$		$2n + O(\sqrt{n})$	[Pau75; KP80]
Sum Mod 4	$4n - O(1)$		$5n - O(1)$	[Zwi91]
Sum Mod 2^k	$4n - O(1)$		$7n - o(n)$	[Zwi91]
Well-Mixed	$5n - o(n)$		poly	[LR01; IM02]
Weighted Sum of Parities	$5n - o(n)$		$5n + o(n)$	[AT11]

Table 1: Explicit Functions with Circuit Lower Bounds in the DeMorgan Basis

In the table, only XOR-function has a proven tight-bound, so attempting to tight the bounds for the other candidates via Gate Elimination (or other known techniques) remains an interesting research direction. In this work, we make progress towards the Multiplexter (MUX) defined below.

Definition. *We define the multiplexer* $\mathsf{MUX}_n : \{0,1\}^n \times \{0,1\}^{2^n} \to \{0,1\}$ *as follows. For all* (a, x) *where* $a \in \{0,1\}^n, x \in \{0,1\}^{2^n}$*, we have*

$$\mathsf{MUX}_n(a, x) = x_{(a)}$$

where $x_{(a)}$ *is the* $(a+1)$*-th bit of* x *(1-indexed) when* a *is interpreted as a base-2 number.*

Throughout this paper, we will address $a_1, \dots a_n$ as the *address bits*, and $x_1, \dots, x_{2^n}$ as the *data bits*, index the function with the number of address bits n, and set $N = 2^n$ as the number of data bits,. For example, if $n = 2$, $a = 01$, and $x = x_1x_2x_3x_4 = 0101$, then $(a) = 1$, thus $\mathsf{MUX}_n(a, x) = x_2 = 1$. We show the following lower bound for MUX_n which an improvement on Paul's $2N - 2$ lower bound [Pau75].

Main Theorem. *Let* C *be an optimal circuit computing* MUX_n*, then under the DeMorgan basis, free negations, the size of* C *is at least* $2N + \log N$ *where* $N = 2^n$*.*

In Appendix B, we provide a detailed analysis the best know upper-bound construction of MUX first introduced by Klein and Patterson [KP80] that is asymptotically close to our lower bound and discuss the feasibility of it being the *optimal* construction. In general, we now obtain

$$2N + \log N \le CC(\mathsf{MUX}_n) \le 2N + O(\sqrt{N}), \text{ where } N = 2^n$$

1.2 Our Technique

We first discuss Paul's technique for showing the $2(N - 1)$ lower bound. The proof follows a standard gate elimination framework that was used to show the $3(n-1)$ lower bound for XOR_n, the XOR-function on n-bits. In particular, Schnorr analyzed the local structure at the bottom level of an optimal circuit computing XOR_n in which some variable x_i is fed into two different binary gates h, g, and without loss of generality, one can argue h cannot be the output gate which means h is fed into a third binary gate k. Thus, there exists a substitution of x_i such that h, g, k are eliminated. Since restricting x_i yields a circuit computing XOR_{n-1}, i.e. XOR-function is downward self-reducible, one can inductively argue the $3(n - 1)$ lower bound.

Paul's lower bound proof follows a similar idea, but the downward self-reducibility of MUX_n behaves differently. Namely, when we restrict an address bit to obtain a circuit computing MUX_{n-1}, we lose half of the current number of data bits. Thus, we cannot apply the downward self-reducibility of MUX-function directly via restricting an address bit using Schnorr's framework. Instead, we will restrict a data bit. However, the resulting circuit after restricting a data bit does not compute MUX_{n-1}, and rather, a function that behaves like MUX_n on every address except one that corresponds to the index of the data bit that was restricted. Thus, we get around this issue by defining a class of function $\mathcal{F} = \{f \mid f \text{ behaves like MUX on a set of addresses } S \subseteq \{0,1\}^n\}$. Clearly, when $S = \{0,1\}^n$, then $f \equiv \mathsf{MUX}_n$. Thus, if we can obtain a lower bound for the class $\mathcal{F}$, we obtain a lower bound for MUX_n. With this observation, Paul's argument says that at the bottom level of an optimal circuit for f, a restriction on a data bit eliminates at least 2 gates, and the lower bound $2(N-1)$ of MUX_n inductively follows. We recount Paul's argument in Appendix A.

We observe that to obtain a better lower bound for MUX_n, we need to restrict an address bit because the fact that half of the data bits become irrelevant to the circuit can do a lot more damage than just two binary gates at a time. Let C be an optimal circuit computing MUX_n, we first observe the local structure of C around an address bit a_i and the set of $N/2$ data bits that becomes irrelevant to the circuit after substituting $a_i \leftarrow b$. Namely, we argue that a_i and each of the $N/2$ data bits are fed to $N/2 + 1$ distinct binary gates, as otherwise, we can exhibit an appropriate substitution that makes at least a_i or one of the data bits degenerate which contradicts the fact that MUX_n depends on all its address and data bits. Thus, at this point, there exists a substitution that eliminates $N/2 + 1$ binary gates.

Then, we harness Gate Elimination on a "mass substitution", that is, we take advantage of the fact that if a circuit C has q constants with a cumulative of ℓ fan-outs after substitutions, then we can eliminate at least ℓ more binary gates deeper in the circuit (Lemma 4, a useful gate elimination tool established in [CDJ25]). Thus, with an appropriate substitution to the $N/2$ data bits that were made irrelevant by $a_i \leftarrow b$, we can further eliminate $N/2$ costly gates in a deeper level of the circuit which yields a total of $N + 1$ binary gates eliminated at this point.

Finally, restricting $a_i \leftarrow b$ results in a circuit computing MUX_{n-1}, so we can repeat the process above with a different address bit and construct a recurrence that lower bounds the number of gates eliminated after each substitution of an address bit.

1.3 Discussion

An obvious open problem is to further improve our lower bound, as close to the upper-bound as possible. A more ambitious direction is to show that $2N+O(\sqrt{N})$ is a tight bound and that the upper-bound construction is indeed optimal. However, we believe Gate Elimination alone might be too difficult to tight the bound further, let alone characterizing optimal circuits for MUX_n. In Appendix B (Section B.3), we provide some insight on why that is the case and thereby guiding us to investigate other techniques.

2 Preliminaries

2.1 Boolean Circuits

We study general Boolean circuits over the *DeMorgan basis* $\mathcal{B} = \{\wedge, \vee, \neg, 0, 1\}$ of Boolean functions: binary $\wedge$ and $\vee$, unary $\neg$ and zero-ary (constants) 1 and 0. Circuits take zero-ary variables in $X = \{x_1, x_2, \dots, x_n\}$ for some fixed n as inputs. The standard formulation of circuits consists of single-sink DAGs with nodes labeled by function symbols or variables and edges as "wires" between the gates. We write V_C and E_C to denote the set of nodes and the set of edges of a circuit C respectively, omitting the subscript when it is clear which circuit is being referenced. Boolean circuits compute Boolean functions through *substitution* followed by *evaluation*.

An *assignment* of input variables is a mapping from the set of inputs to $\{0, 1\}$. To substitute into a circuit according to an assignment, each input x_i is replaced by the constant. To evaluate a circuit, the values of interior nodes labeled by function symbols are computed in increasing topological order. For each interior node labeled by a function, it's value is obtained by applying it's function to the value of the node's incoming wires. The output of the circuit overall is the value of its sink.

Let $\mathcal{F}_n$ be the family of Boolean functions on n variables. We say a circuit G on n variables computes $g \in \mathcal{F}_n$ if for all $\alpha \in \{0,1\}^n$, $G(\alpha) = g(\alpha)$. The size of a circuit G, denoted $|G|$, is the number of $\wedge$ and $\vee$ gates in the circuit, and $CC(g)$, the circuit complexity of a Boolean function g, is the minimum size of any circuit computing g. Since only $\wedge$ and $\vee$ gates contribute to circuit size. We say a circuit G computing g is optimal if $|G| = CC(g)$.

2.2 Gate Elimination

Our lower bounds use the classical *gate elimination* method: we fix some inputs to constants, then repeatedly simplify the resulting circuit using local Boolean identities until no further simplifications apply. Intuitively, each simplification either replaces a gate by a constant or merges it into one of its inputs, thereby reducing the number of costly gates while preserving the computed function on the restricted input space. Table 2 lists the local rewrite rules we use, grouped intocfour types:

- *Fixing* rules: directly turn a gate into the constant 0 or 1 (e.g. $0 \wedge \gamma \to 0$).
- *Passing* rules: replace a gate by one of its inputs (e.g. $1 \wedge \gamma \to \gamma$), letting that input inherit the outgoing wires of the gate.
- *Resolving* rules: eliminate a literal and its negation (e.g. $\gamma \wedge \neg\gamma \to 0$).
- *Pruning* rules: remove duplicate inputs (e.g. $\gamma \wedge \gamma \to \gamma$) and double negations.

Fixing	Passing	Resolving	Pruning
$0 \wedge \gamma \to 0$	$1 \wedge \gamma \to \gamma$	$\gamma \wedge \neg\gamma \to 0$	$\gamma \wedge \gamma \to \gamma$
$\gamma \wedge 0 \to 0$	$\gamma \wedge 1 \to \gamma$	$\neg\gamma \wedge \gamma \to 0$	
$1 \vee \gamma \to 1$	$0 \vee \gamma \to \gamma$	$\gamma \vee \neg\gamma \to 1$	$\gamma \vee \gamma \to \gamma$
$\gamma \vee 1 \to 1$	$\gamma \vee 0 \to \gamma$	$\neg\gamma \vee \gamma \to 1$	
$\neg 0 \to 1$			$\neg\neg\gamma \to \gamma$
$\neg 1 \to 0$			

Table 2: Gate elimination rules

All these rules strictly decrease the number of costly gates and preserve the Boolean function computed by the circuit. We also use a trivial "garbage collection" step: any gate with fan-out 0 that is not the output gate may be deleted.Figure 1 illustrates a passing rule: the child gate γ takes over the outgoing wires of α and α can be deleted once it has fan-out 0.

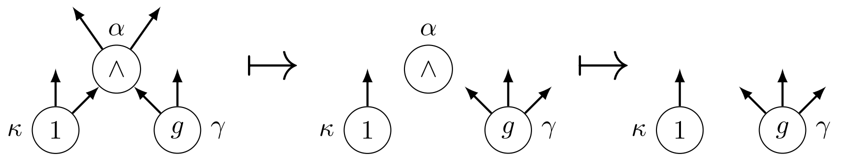

Figure 1: An example of applying the passing simplification $1 \wedge \gamma \to \gamma$. Notice that γ inherits the fanout from α, and α can then be garbage collected.

Note that the identifier of α **never** changes; only its type and incident wires. Depending on the initial fanout of each gate involved in the pattern, applying a rule could totally disconnect some gate(s) and leave the circuit in a state that needs garbage collection. We introduce notation for applying sequences of these steps to circuits, and define composed circuit manipulations that are "well behaved" with respect to specific circuits.

Definition 1. A *simplification* is a sequence of gate elimination and garbage collection steps. If λ is a simplification and C is a circuit, write $\lambda(C)$ to denote the new circuit obtained by applying each step of λ to C in order. If a single step of λ fails to apply in C, then fix $\lambda(C) = C$ so that invalid λ for C are idempotent by convention. A simplification λ is called

- *terminal* for C if, for every λ' that extends λ by one additional gate elimination or garbage collection step, $\lambda(C) = \lambda'(C)$, and/or
- *layered* for C if the depth of **binary** gates binding to α in each step is non-decreasing.

Simplifications formalize every sequence of circuit-manipulation steps except for substitution. They change the function computed by a circuit if and only if some input gate is garbage-collected. Simplifications suffice to impose convenient structural properties on arbitrary circuits.

Definition 2. A circuit C is *normalized* or *in normal form* if

1. C is constant-free **or** C is a single constant,
2. every gate in C has a path to the output, and
3. no sub-circuit of C matches the left hand side of a gate elimination rule in Table 2.

Any terminal simplification suffices to put a circuit C in normal form, and it is straightforward to see that the number of constants in C lower-bounds the number of eliminated gates. We can even push it further: every circuit C can be normalized by a *layered* simplification, and the number of eliminated gates is lower-bounded by the *cumulative fanout* of constants in C (Lemma 4).

Lastly, we emphasize that substitution *always* changes the function computed by C, restricting its domain to a subcube. Additionally, C remains well-formed after any substitution, because input gates have fan-in 0. However, C is *never* in normal form immediately after a substitution because it introduces a constant. So we generalize simplifications by allowing for substitution steps.

Definition 3. A *restriction* is a sequence of substitution, gate elimination, and garbage collection steps. Restrictions generalize simplifications, so we extend notation and conventions accordingly: $\rho(C)$ denotes the result of applying restriction ρ to circuit C, and invalid restrictions for C are idempotent by convention.

The categories of *terminal* and *layered* simplifications apply immediately to restrictions, because they explicitly reference only gate elimination or garbage collection steps. Thus, restrictions are terminal and/or layered only with respect to how they simplify and **not** how they substitute. When applying a restriction ρ to a Boolean function instead of a circuit, we simply ignore the additional simplification steps (if any).

3 A Better Lower Bound for MUX

For our lower bound for MUX_n, we take advantage of its downward self-reducibility after substituting one address bit. Such a substitution will make half of the current data bits irrelevant to the resulting circuit computing MUX_{n-1}. It is easy to see that MUX depends on all its address and data bits.

Furthermore, the observation that every circuit C can be normalized by a *layered* simplification, and the number of eliminated gates is lower-bounded by the *cumulative fan-out* of constants in C. This property is immensely useful for our MUX's lower bound. The full proof of this property can be found in [CDJ25]. Below, we provide a proof sketch.

Lemma 4 ([CDJ25])**.** *For any circuit C with q constants that have total fanout ℓ, there is a terminal and layered simplification λ such that $\lambda(C)$ is a single constant **or** $|\lambda(C)| \leq |C| - \ell$.*

Proof Sketch. We write ϕ_t as the total fan-out of all constants in C after applying t rule manipulations, and μ_t as the number of binary gates eliminated so far. Note that since a circuit is normalized only when it is a single constant or when $\phi_t = 0$, it is sufficient to exhibit a terminal layered simplification λ such that $\mu_t \geq \ell$ if $\phi_t = 0$. We initialize $\phi_t = \ell$ and $\mu_t = 0$.

Fix an order of the gates in C by depth (breaking ties arbitrarily). Apply gate elimination rules in non-decreasing depth order, interleaving garbage collection of gates with fan-out 0. This yields a layered simplification λ by construction.

We compute the changes to fan-out after each of the 4 types of elimination to show that: (i) each rule application decreases ϕ_t by 1, (ii) whenever ϕ_t decreases by 1, at least one binary gate is eliminated, so μ_t increases by at least 1, and (iii) garbage collection never increases ϕ_t and only increases μ_t. If a single gate remains after λ terminates, then it must be a constant, and we are done. Otherwise, there are no constants with positive fan-out in the simplified circuit, so $\phi_t = 0$. Since each unit dropped in ϕ_t forces an increase of at least 1 in μ_t, and we initialize $\phi_t = \ell$, it follows that $\mu_t \geq \ell$ which yields $|\lambda(C)| \leq |C| - \ell$ as desired. ☐

Theorem 5. *For the DeMorgan basis $\{\wedge, \vee, \neg\}$ with free $\neg$-gates, we have $CC(\mathsf{MUX}_n) \geq 2$ when $n = 1$ or $CC(\mathsf{MUX}_n) \geq 2N + \log N$ when $n > 1$ and $N = 2^n$.*

Proof. We exploit the downward self-reducibility of MUX_n: substituting an address bit a_i, $i \in [n]$, will eliminate half of the current data bits whose indices are not compatible with the substitution, and the restricted function is a smaller multiplexer. Thus, our strategy is to establish a lower bound on the number of gates eliminated under a specific setting of a_i as well as the data bits that are eliminated, and we repeat this strategy for the remaining address bits, one by one to establish a recurrence on the number of gates eliminated. To this end, let C be an optimal circuit computing MUX_n for $n > 1$, and set $N = 2^n$. Without the loss of generality, we focus on a_n, the last address bit. Let D be the set of data bits that becomes degenerate when a_n is set to some binary constant b. It is obvious that $|D| = N/2$, and for convenience, we index data bits in D as $\{x_j \mid j \in [N/2]\}$.

We first claim that there exists a set S of $N/2$ distinct binary gates such that each gate in S reads a distinct $x_j \in D$. Process each $x_j \in D$ one by one and select some unseen binary gate arbitrarily from x_j's out neighbors that has not been selected yet. That is, for x_1, choose any binary out-neighbor and put it into S; for x_2, choose any binary out-neighbor not already in S; and so on. If this process fails for some x_j, then all binary out-neighbors of x_j are already in S. By appropriately fixing the previously processed data bits $\{x_1, \ldots, x_{j-1}\}$, we can then apply fixing rules at each gate in S while leaving x_j unconstrained and in turns disconnecting x_j from C. This contradicts the fact that C depends on every data bit, and thus, such set S exists.

Now, we argue that a_n feeds to a binary gate that is outside of S. If every binary gate reading a_n is in S, then by fixing the data bits in D appropriately we could again apply fixing rules at all gates in S and thereby disconnect a_n from C, contradicting that MUX_n depends on every address bit. Furthermore, if we substitute $a_n \leftarrow b$ and then substitute each $x_j \in D$ to whichever allows its selected gate to be eliminated with a fixing rule, then we will eliminate at least $N/2+1$ binary gates. We claim that none of these $N/2+1$ gates is the output gate of C. Indeed, if one of them were the output, then by choosing the corresponding input value for a_n or some $x_j \in D$ appropriately, we could make the entire circuit constant.

Let C' denote the circuit obtained after these substitutions, and let $S' = S \cup g_{a_n}$, where g_{a_n} is a binary gate fed by a_n and $g_{a_n} \notin S$. Then, each gate in S' is now a constant with fan-out at least 1, so the constants in C' have total fan-out of at least $|S'| = N/2 + 1$. By Lemma 4, there is a terminal, layered simplification of C' that eliminates at least $N/2+1$ further binary gates. Altogether, fixing a_n and the data bits in D and then simplifying eliminates at least $N/2 + N/2 + 1 = N + 1$ binary gates.

After these substitutions, the simplified circuit computes MUX_{n-1} on the remaining address bits and the remaining half of the data bits. We can further repeat this process for the remaining address bits a_i and the corresponding data bits that they make degenerate and obtain circuit computing smaller multiplexers. In particular, let $CC(\mathsf{MUX}_n) = T(N)$, where $N = 2^n$, which denotes the minimum size of an optimal circuit for MUX_n, then we can lower-bound $T(N)$ with the following recurrence.

$$T(N) \geq \begin{cases} 2, & \text{if } N = 2 \ (n = 1) \\ T(N/2) + N + 1, & \text{if } N > 2 \ (n > 1) \end{cases}$$

The base case follows from Paul's lower bound [Pau75], and the recurring case follows from our analysis. We have $n = \log N$ number of address bits, so we can unroll the recurrence as follows

$$\begin{aligned} T(N) &\geq T(N/2) + N + 1 \\ &\geq T(N/4) + N + N/2 + 2 \\ &\geq T(N/8) + N + N/2 + N/4 + 3 \\ &\cdots \\ &\geq T(2) + \sum_{i=0}^{\log N - 1} \left(\frac{N}{2^i} + 1 \right) \end{aligned}$$

We have

$$\begin{aligned} \sum_{i=0}^{\log N-1} \frac{N}{2^i} &= N \sum_{i=0}^{n-1} \frac{1}{2^i} \\ &= N \left(\frac{1-(1/2)^n}{1-1/2} \right) \quad \text{(geometric series for } r = 1/2) \\ &= N(2 - 2^{1-n}) \\ &= 2N - 2 \qquad (2^{1-n} = 2/2^n = 2/N) \end{aligned}$$

Thus, $CC(\mathsf{MUX}_n) \geq 2 + 2N - 2 + \log N = 2N + \log N$ as desired. □

References

[AT11] Kazuyuki Amano and Jun Tarui. "A well-mixed function with circuit complexity 5n: Tightness of the Lachish–Raz-type bounds". In: *Theoretical computer science* 412.18 (2011), pp. 1646–1651.

[CDJ25] Marco Carmosino, Ngu Dang, and Tim Jackman. "Simple Circuit Extensions for XOR in PTIME". In: *arXiv preprint arXiv:2511.16903. To Appear in STACS 2026* (2025).

[CO25] Bruno Cavalar and Igor Oliveira. "Boolean Circuit Complexity and Two-Dimensional Cover Problems". In: *ACM Transactions on Computation Theory* 17.2 (2025), pp. 1–23.

[DK11] Evgeny Demenkov and Alexander S. Kulikov. "An Elementary Proof of a 3n - o(n) Lower Bound on the Circuit Complexity of Affine Dispersers". In: *Mathematical Foundations of Computer Science 2011 - 36th International Symposium, MFCS 2011, Warsaw, Poland, August 22-26, 2011. Proceedings*. Ed. by Filip Murlak and Piotr Sankowski. Vol. 6907. Lecture Notes in Computer Science. Springer, 2011, pp. 256–265. DOI: 10.1007/978-3-642-22993-0_25. URL: https://doi.org/10.1007/978-3-642-22993-0%5C_25.

[FGHK16] Magnus Gausdal Find, Alexander Golovnev, Edward A. Hirsch, and Alexander S. Kulikov. "A Better-Than-3n Lower Bound for the Circuit Complexity of an Explicit Function". In: *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*. 2016, pp. 89–98. DOI: 10.1109/FOCS.2016.19.

[IM02] Kazuo Iwama and Hiroki Morizumi. "An Explicit Lower Bound of 5n - o(n) for Boolean Circuits". In: *Mathematical Foundations of Computer Science 2002, 27th International Symposium, MFCS 2002, Warsaw, Poland, August 26-30, 2002, Proceedings*. Ed. by Krzysztof Diks and Wojciech Rytter. Vol. 2420. Lecture Notes in Computer Science. Springer, 2002, pp. 353–364. DOI: 10.1007/3-540-45687-2_29. URL: https://doi.org/10.1007/3-540-45687-2%5C_29.

[KP80] Klein and Paterson. "Asymptotically optimal circuit for a storage access function". In: *IEEE Transactions on Computers* 100.8 (1980), pp. 737–738.

[LR01] Oded Lachish and Ran Raz. "Explicit lower bound of 4.5n - o(n) for boolena circuits". In: *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*. STOC '01. Hersonissos, Greece: Association for Computing Machinery, 2001, pp. 399–408. ISBN: 1581133499. DOI: 10.1145/380752.380832. URL: https://doi.org/10.1145/380752.380832.

[LY22] Jiatu Li and Tianqi Yang. "3.1n - $o(n)$ circuit lower bounds for explicit functions". In: *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022*. Ed. by Stefano Leonardi and Anupam Gupta. ACM, 2022, pp. 1180–1193. DOI: 10.1145/3519935.3519976. URL: https://doi.org/10.1145/3519935.3519976.

[Pau75] Wolfgang J Paul. "A 2.5 n-lower bound on the combinational complexity of Boolean functions". In: *Proceedings of the seventh annual ACM symposium on Theory of computing*. 1975, pp. 27–36.

[Red73] NP Red'kin. "Proof of minimality of circuits consisting of functional elements". In: *Systems Theory Research: Problemy Kibernetiki* (1973), pp. 85–103.

[SC98] JE Savage and Models Of Computation. *Exploring the power of Computing*. 1998.

[Sch74] Claus-Peter Schnorr. "Zwei lineare untere Schranken für die Komplexität Boolescher Funktionen". In: *Computing* 13.2 (1974), pp. 155–171. DOI: 10.1007/BF02246615. URL: https://doi.org/10.1007/BF02246615.

[Sto77] Larry J. Stockmeyer. "On the Combinational Complexity of Certain Symmetric Boolean Functions". In: *Math. Syst. Theory* 10 (1977), pp. 323–336. DOI: 10.1007/BF01683282. URL: https://doi.org/10.1007/BF01683282.

[Weg87] Ingo Wegener. *The Complexity of Boolean Functions*. Wiley-Teubner, 1987.

[Wig93] Avi Wigderson. "The fusion method for lower bounds in circuit complexity". In: *Combinatorics, Paul Erdos is Eighty* 1.453-468 (1993), p. 68.

[Zwi91] Uri Zwick. "A 4n lower bound on the combinational complexity of certain symmetric boolean functions over the basis of unate dyadic Boolean functions". In: *SIAM Journal on Computing* 20.3 (1991), pp. 499–505.

A Recounting Paul's Lower Bound

In this section we will recount Paul's lower bound for the multiplexing function (MUX) [Pau75]. We note that Paul's basis was $\{\wedge, \oplus, \neg\}$ (with free $\neg$-gates and costly $\wedge, \oplus$-gates) which is different from our basis. Still, the core ideas for the argument remain the same, an argument via Gate Elimination. Here, we simply provide our refined presentation of Paul's lower bound that aligns with our basis.

To this end, we define the set of i index selector functions as follows

Definition 6. Let $f : \{0,1\}^n \times \{0,1\}^{2^n}$ be a Boolean function with two distinguished sets of input variables: $a_1, \dots a_n$ the set of *address bits* and $x_1, \dots x_{2^n-1}$ be the set of *data bits.*

We define $A(f) = \{\alpha \in \{0,1\}^n : f\restriction_{a=\alpha} \equiv x_{(\alpha)}\}$ to be the set of *selectable addresses* of f.

We define S_i, the set of i index selector functions, as:

$$S_i = \{f : \{0,1\}^n \times \{0,1\}^{2^n} \to \{0,1\} : |A(f)| \geq i\}$$

In words, the set S_i contains Boolean functions f on $(n + 2^n)$ input variables such that there are at least i addresses a satisfying $f(a, x) = x_{(a)}$. Thus, it is easy to see that when $i = 2^n$, we have $f \equiv \mathsf{MUX}_n$. With this observation, we obtain the following lower bound on MUX.

Theorem 7. *[Pau75] For the DeMorgan basis $\{\wedge, \vee, \neg\}$ where $\neg$-gates are for free, and for all $f \in S_i$, we have $CC(f) \geq 2i - 2$. Furthermore, $CC(\mathsf{MUX}_n) \geq 2N - 2$ where $N = 2^n$.*

Proof. Let C be an optimal circuit for some $f_i \in S_i$ (as in Definition 6) with $i \geq 1$, we proceed with proving the lower-bound by induction on i. The base case $i = 1$ is vacuous as $2 \cdot 1 - 2 = 0$ and the circuit complexity of all functions is non-negative.

For $i = 2$, we wish to show that $CC(f_2) \geq 2 \cdot 2 - 2 = 2$ for some $f_2 \in S_2$. By Definition 6, there are two selectable addresses α_1, α_2. By definition, f_2 is a Boolean function such that there exists at least two addresses $\alpha_1, \alpha_2 \in \{0,1\}^\ell$ such that $f_2(\alpha_1, x) = x_{(\alpha_1)}$ and $f_2(\alpha_2, x) = x_{(\alpha_2)}$. Since α_1 and α_2 corresponds to two different data bits, we need at least one $\wedge$-gate to select the appropriate appropriate data bit and one $\vee$-gate to combine the results of the selection (either one of them). Thus, $CC(f_2) \geq 2$.

For some $i = k \geq 1$, assume that $CC(f) \geq 2k - 2$ for any $f \in S_k$. Fix $f_{k+1} \in S_{k+1}$, we will show that $CC(f_{k+1}) \geq 2(k+1) - 2$. To this end, let C be an optimal circuit computing f_{k+1}. By definition of S_{k+1} we know $|A(f_{k+1})| \geq k + 1$. Consider any $\alpha \in A$ and it's corresponding input $x_{(\alpha)}$. We analyze the following cases on the fan-out of $x_{(\alpha)}$ and argue the desired lower-bound for $|C|$ on all cases. First, we note that $\text{fanout}(x_{(\alpha)})$ cannot be 0. Assume towards contradiction that $\text{fanout}(x_{(\alpha)}) = 0$. Since f_{k+1} depends on $x_{(\alpha)}$ then $x_{(\alpha)}$ must be the output of the circuit (i.e. $f_{k+1} \equiv x_{(\alpha)}$ regardless of the value of a). Since $k \geq 1$, $|A(f_{k+1})| \geq 2$ and thus there exists an $a' \in A$ that is distinct from A. However by definition, $f_{k+1}\restriction_{a=\alpha'} \equiv x_{(\alpha')} \not\equiv x_{(\alpha)}$, a contradiction. We now analyze the remaining cases.

- Case 1: $\text{fanout}(x_{(\alpha)}) > 1$. In other words, $x_{(\alpha)}$ is feeds into at least two other costly gates in C. Fix $b \in \{0,1\}$ and consider C', the circuit obtained by moving any wires originating from $x_{(\alpha)}$ in C to the constant b and then performing standard gate elimination. We note that $|C'| \leq |C| - 2$ since $x_{(\alpha)}$ had at least two costly neighbors which would have been eliminated after our constant substitution.

  Let $f' : \{0,1\}^\ell \times \{0,1\}^N$ be the function C' computes. Observe that $A(f') = A(f) \setminus \{\alpha\}$. By the inductive hypothesis, $2^k - 2 \leq CC(f') \leq |C'| \leq |C| - 2 \leq CC(f_{k+1}) - 2$. Rearranging yields $CC(f_{k+1}) \geq 2k - 2 + 2 = 2(k+1) - 2$.

- Case 2: $\text{fanout}(x_{(\alpha)}) = 1$. This means that $(\neg)x_{(\alpha')}$ is fed into exactly one costly gate in C. Let us call this costly gate g. We first show that it is not the output of the circuit. If it were, notice that there is some constant we can substitute in for $x_{(\alpha')}$ that would eliminate g via a fixing rule and leaving the circuit constant. In other words, there exists a constant $b \in \{0,1\}$ such that $f_{k+1}\restriction_{x_{(\alpha)}=b}$ is degenerate with respect to all $x_{(\alpha')}$ for any $\alpha' \neq \alpha$ and $\alpha \neq \alpha'$. However, since $|A(f_{k+1})| \geq 2$, there is at least one other α'' such that $f_{k+1}\restriction_{a=\alpha'', x_{(\alpha)}=b} \equiv x_{(\alpha'')} \neq b$.

Therefore, g is not the output gate and thus $(\neg)g$ feeds another costly gate $h \in \{\wedge, \vee\}$. Substituting the constant for $x_{(\alpha)}$ which eliminates g via a fixing rule will then also eliminate h as well. Thus, we obtain $CC(f_{k+1}) = |C| \geq 2(k+1) - 2$ via the same argument as in case 2.

Since $CC(f_{k+1}) \geq 2 \cdot 2^{k+1} - 2$ for both cases, the statement is true by the principle of mathematical induction. Set $i = 2^n = N$, we obtain $CC(\mathsf{MUX}_n) \geq 2N - 2$. □

B Best Known Upper Bound Construction

To better understand the construction for MUX-circuits, we define a binary-to-positional decoder, an important component in the circuit construction, as follows

Definition 8 (Positional Decoder)**.** A function $\mathsf{DEC}_n : \{0,1\}^n \to \{0,1\}^{2^n}$ is called a binary-to-positional decoder if the following holds:

$$\mathsf{DEC}_n(a) = (sel_0(a), \ldots, sel_{2^n-1}(a))$$

where $sel_i(a) = 1$ iff $i = (\mathbf{a})$

Thus we can write the definition of MUX as a DNF using the output of DEC_n as follows

$$\mathsf{MUX}_n(a, x) = \bigvee_{i=0}^{2^n-1} sel_i(a) \wedge x_i$$

.

Furthermore, to simplify analysis and avoid irregularities arising in generic even or odd input lengths, we define and analyze restricted variants of these functions—Exponent Decoder (eDEC) and Exponent Multiplexer (eMUX) where the address length is always a power of two. This constraint enables recursive constructions that are clean, easily analyzable, and asymptotically optimal while remaining representative of the general case. In particular, we provided a fully formal and expanded version of the upper bound construction from [KP80] by first analyzing the construction for eDEC (Theorem 11) and then lift that to eMUX (Theorem 14). Namely,

Definition 9 (Exponent Positional Decoder and Multiplexer)**.** Let $\ell \in \mathbb{N}$, we define the classes of functions of Exponent Positional Decoders $\mathsf{eDEC}_\ell : \{0,1\}^{2^\ell} \to \{0,1\}^{2^{2^\ell}}$ and Exponent Multiplexers $\mathsf{eMUX}_\ell : \{0,1\}^{2^\ell} \times \{0,1\}^{2^{2^\ell}} \to \{0,1\}$ as follows

$$\mathsf{eDEC}_\ell(a) = (sel_0(a), \ldots, sel_{2^{2^\ell}-1}(a))$$

where $sel_i(a) = 1$ iff $i = (\mathbf{a})$

$$\mathsf{eMUX}_\ell(a, x) = \bigvee_{i=0}^{2^{2^\ell}-1} sel_i(a) \wedge x_i$$

Here, notice the subscript of ℓ used in the definition of Binary Positional Decoder and Multiplexer which no longer indicates the number of input variables. Instead, it serves as an "index" for the function in this particular family. So for example, eDEC_0 is the first and "smallest" one in the eDEC-function family and it corresponds to DEC_1 in the standard decoder family. Similarly, eDEC_1 is the second one in the family and corresponds to DEC_2, eDEC_2 to DEC_4, and so on. The same logic applies for the eMUX-function family.

B.1 Decoder Upper-Bound Construction

To begin with, we will look at a naive construction for a circuit computing DEC_n. Observe that for n address bits, there are $2^n = N$ possible minterms, and each minterm corresponds to exactly one output of DEC_n. Thus, our circuit construction is to simply compute all possible minterms composed by the address bits $a_1, \ldots, a_n$. To construct a minterm, we need $n - 1 = \log N - 1$ $\wedge$-gates. Therefore, this construction yields an upper bound $CC(\mathsf{DEC}_n) \leq (\log N - 1)N$ which is $O(N \log N)$ costly gates.

However, observe that we can achieve a better upper-bound by constructing the circuit for DEC_n recursively. Namely, we notice that by definition, minterms have a recursive property, i.e. a minterm on n variables is an AND of 2 smaller minterms, one is on half of the variables, and the other one is on the other half of the variables. That said, a better way to construct a DEC_n-circuit is to use two subcircuits, $\mathsf{DEC}_{\lfloor n/2 \rfloor}$-circuit and $\mathsf{DEC}_{\lceil n/2 \rceil}$-circuit, and combine every minterm generated by the first $\mathsf{DEC}_{\lfloor n/2 \rfloor}$-circuit with every minterm generated by the second $\mathsf{DEC}_{\lceil n/2 \rceil}$-circuit with $\wedge$-gates. We present Figure 2 regarding this construction in our upper bound proof in Section B.1 below. Thus, this construction yields a circuit complexity that is at most twice the complexity of $\mathsf{DEC}_{n/2}$-circuit plus N $\wedge$-gates that are used to combine the minterms of the two DEC-subcircuits.

Lemma 10 ([KP80; SC98]). *$CC(\mathsf{DEC}_n) \leq N + CC(\mathsf{DEC}_{\lfloor n/2 \rfloor}) + CC(\mathsf{DEC}_{\lceil n/2 \rceil}) = N + O(\sqrt{N})$, where $N = 2^n$*

Notice that each output entry of DEC_n corresponds to exactly one minterm the variables $\{a_1, a_2, \ldots, a_n\}$. Each minterm computes a distinct non-constant function which requires at least one costly gate to compute, and there are 2^n possible minterms. Thus, an obvious lower bound for the circuit complexity of DEC_n is $CC(\mathsf{DEC}_n) \geq N$ [SC98]. However, this lower bound only gives us information regarding the output layer on the top of the circuit as n gets large which leaves rooms for improvement if one can witness a restriction that is guaranteed to eliminate gates in the intermediate levels of the circuit.

Below, we present a detailed analysis of the upper bound construction for eDEC. We note that our construction for the upper bound is based on that of [KP80; SC98], but we provide an exact count of gates for the case of eDEC.

Theorem 11. *Let $\ell \in \mathbb{N}$ and $N = 2^{2^\ell}$, then $CC(\mathsf{eDEC}_\ell) \leq N + \mathcal{S}(N)$ where $\mathcal{S}(N) = \sum_{i=1}^{\log\log N - 1} 2^i \cdot N^{1/2^i}$*

Proof. Let $r = s = \frac{2^\ell}{2} = 2^{\ell-1}$, then for some $a = b \circ c$, where $b \in \{0,1\}^r$, $c \in \{0,1\}^s$, and $i = 0, \ldots, 2^r - 1$, $j = 0, \ldots 2^s - 1$, we have

$$sel_{i \cdot 2^r + j}(a) = sel_{i \cdot 2^r + j}(b \circ c) = sel_i(b) \wedge sel_j(c)$$

which implies the following equation computing eDEC_ℓ defined in Definition 9

$$\mathsf{eDEC}_\ell(a) = \mathsf{eDEC}_\ell(b \circ c) = (sel_i(b) \wedge sel_0(c), \ldots, sel_i(b) \wedge sel_{2^j-1}(c) \text{ for } i = 0, \ldots, 2^r - 1)$$

In words, for every $i = 0, \ldots, 2^s - 1$, we compute $sel_i(b) \wedge sel_j(c)$ for every $j = 0, \ldots, 2^r - 1$, so $sel_0(a)$ corresponds to $sel_i(b) \wedge sel_j(c)$ where $(i,j) = (0,0)$, and $sel_1(a)$ corresponds to the term where $(i,j) = (0,1)$, and so on.

We observe the following recursive construction for D_ℓ based on the equation above. In particular, let D_ℓ be the circuit computing eDEC_ℓ, then the construction of D_ℓ uses two $D_{\ell-1}$ sub-circuits and the output of D_ℓ are given by connecting the outputs of the two $D_{\ell-1}$ in the following manner: for each output of one $D_{\ell-1}$, we connect it to each output of the other $D_{\ell-1}$ using an $\wedge$-gate which requires a total of N $\wedge$-gates. Thus, we obtain

$$CC(\mathsf{eDEC}_\ell) \leq 2 \cdot CC(\mathsf{eDEC}_{\ell-1}) + N$$

We can repeat this process to construct $D_{\ell-1}$ using two sub-circuits $D_{\ell-2}$, and so on which suggests a recurrence for the construction of eDEC_ℓ-circuits. In particular, observe that the base case of the recursive construction is when $\ell = 0$ which is a simple decoder that has $2^{2^0} = 2$ outputs. This decoder requires zero costly gate since we have only two possible minterms, i.e. either the variable itself or its negation, so $CC(\mathsf{eDEC}_0) = 0$. Thus, we can express the upper-bound of $CC(\mathsf{eDEC}_\ell)$ via the following recurrence with $N = 2^{2^\ell} \implies \sqrt{N} = 2^{2^\ell/2} = 2^{2^{\ell-1}}$ (meaning decreasing ℓ by 1 accounts for taking the square root on the number of outputs)

$$T(N) = \begin{cases} 0, & \text{if } N = 2 \\ 2 \cdot T(N^{1/2}) + N, & \text{if } N > 2 \end{cases}$$

Now, we compute the number of steps to reach the base case. Let this number be k, then we have

$1 = \frac{\ell}{2^k} \implies 2^k = \log N \implies k = \log\log N$. Next, we unroll the recurrence for k steps.

$$\begin{aligned}
T(N) &= 2 \cdot T(N^{1/2}) + N = 2(2 \cdot T(N^{1/4}) + N^{1/2}) + N \\
&= 4 \cdot T(N^{1/4}) + 2 \cdot N^{1/2} + N \\
&= 8 \cdot T(N^{1/8}) + 4 \cdot N^{1/4} + 2 \cdot N^{1/2} + N \\
&\dots \\
&= 2^k \cdot T(2) + \sum_{i=0}^{k-1} 2^i \cdot N^{1/2^i} \\
&= \log N \cdot 0 + N + \sum_{i=1}^{\log\log N - 1} 2^i \cdot N^{1/2^i} \\
&= N + \sum_{i=1}^{\log\log N - 1} 2^i \cdot N^{1/2^i}
\end{aligned}$$

Setting $\mathcal{S}(N) = \sum_{i=1}^{\log\log N-1} 2^i \cdot N^{1/2^i}$, we have $CC(\mathsf{eDEC}_\ell) \leq N + \mathcal{S}(N)$ as desired. □

Remark 12. *Our upper bound for eDEC_ℓ is consistent with the lower bound of DEC_n in Lemma 10. In particular, the first term in $\mathcal{S}(N)$, $2 \cdot N^{1/2}$, dominates the others, and thus, we believe one can show that $\mathcal{S}(N) = O(\sqrt{N})$. Furthermore, a useful thing to try is to derive the exact count for the general case of DEC_n where n is even or odd, each may differ by a few gates, but should be in $O(\sqrt{N})$ still.*

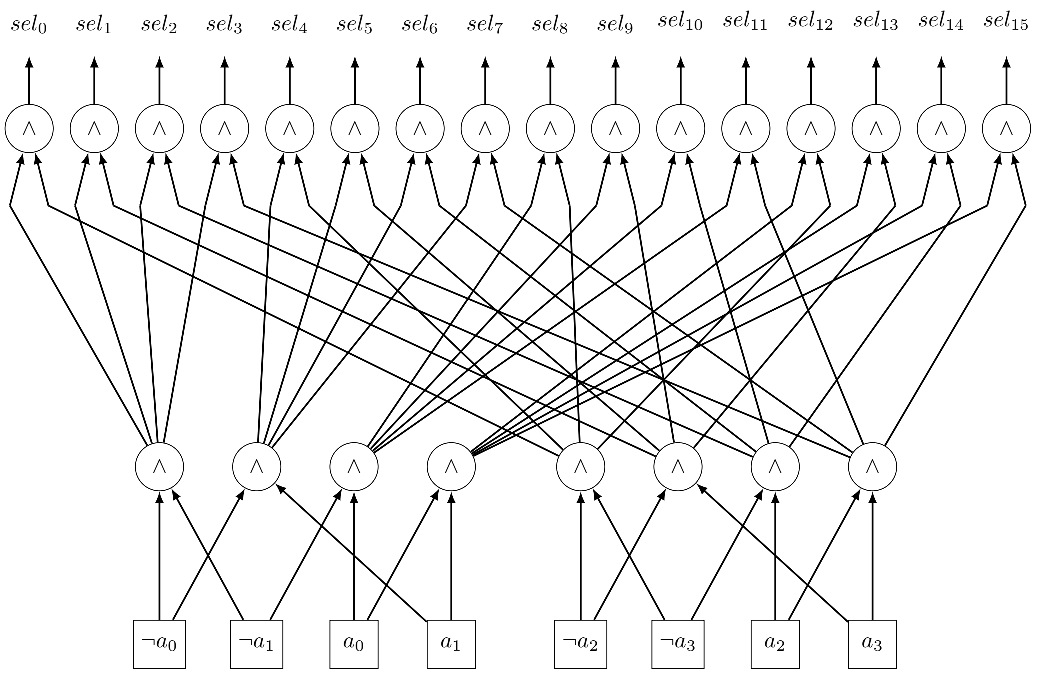

Figure 2: The figure shows the construction of DEC_4 that uses two DEC_2 subcircuits. The total complexity of this construction is $16 + 2 \cdot 16^{1/2^1} = 16 + 8 = 24$ costly gates. For readability, we simplify the negations of the input variables by treating each negation as its own input node.

B.2 Multiplexer Upper-Bound Construction

We begin with establishing an upper bound for MUX_n, and we start with the naive circuit construction. In particular, the construction follows from the alternative definition of MUX_n via Definition 8 of DEC_n. As a reminder,

$$\mathsf{MUX}_n(a, x) = \bigvee_{i=0}^{2^n-1} sel_i(a) \wedge x_i$$

In particular, it uses a total of $N - 1$ $\vee$-gates and N $\wedge$-gates, where $N = 2^n$ (see Figure 3). Thus, $CC(\mathsf{MUX}_n) \leq 2N - 1 + CC(\mathsf{DEC}_n) \leq 3N + O(\sqrt{N})$.

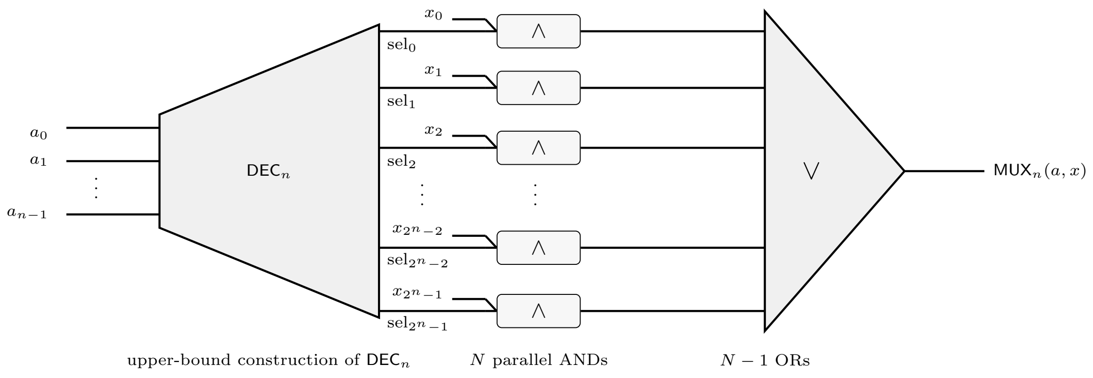

Figure 3: Naïve MUX_n-circuit construction (on input address bits $a_0, \ldots, a_{n-1}$ and data bits $x_0, \ldots, x_{2^n-1}$). The decoder DEC_n (on inputs $a_0, \ldots, a_{n-1}$) emits a one-hot vector $(sel_0, sel_1, \ldots, sel_{2^n-1})$. Each x_i and sel_i are fed into an AND-gate, and the results are combined by an OR-circuit to produce the final output.

The "overhead" of this construction is upper bounded by $2N - 1$, but it turns out that we can also apply a recursive construction for MUX_n to obtain a better upper bound for the overhead. A better construction proposed by Klein and Paterson [KP80] takes advantage of the downward self-reducibility of MUX. In particular, the "overhead" of N costly gates can be reduced by using two smaller decoders *in sequence*, selecting part of the address with respect to different blocks of arguments in turn (see Figure 4). Namely, we have the following upper bound

Lemma 13 ([KP80]). *$CC(\mathsf{MUX}_n) \leq 2N + O(\sqrt{N})$, where $N = 2^n$*

Proof. Let $r = \lfloor \frac{n}{2} \rfloor$, $s = \lceil \frac{n}{2} \rceil$, and let D_r, D_s denote the circuits computing DEC_r, DEC_s respectively. We have the following sequence of equations that uses D_r and D_s in sequence.

$$\begin{aligned}
\mathsf{MUX}_n(a, x) = \mathsf{MUX}_n(b \circ c, x) &= \bigvee_{i=0}^{2^r-1} \bigvee_{j=0}^{2^s-1} (sel_i(b) \wedge sel_j(c) \wedge x_{i \cdot 2^r + j}) \\
&= \bigvee_{i=0}^{2^r-1} \left(sel_i(b) \wedge \left(\bigvee_{j=0}^{2^s-1} (sel_j(c) \wedge x_{i \cdot 2^r + j}) \right) \right) \\
&= \mathsf{MUX}_r \left(b, \left(\bigvee_{j=0}^{2^s-1} (sel_j(c) \wedge x_{i \cdot 2^r + j}) \text{ , for } i = 0, \cdots, 2^r - 1 \right) \right)
\end{aligned}$$

The circuit construction in this case is as follows (which is suggested by the middle equation): we first use D_s to select the appropriate j given by its binary presentation c. Then, for this "fixed" j, the term $sel_j(c) \wedge x_{i \cdot 2^r + j}$ can "filter out" the input indices that do not correspond to j. In other words, at this stage, we are asking the question: for all possible prefixes $b \in \{0, 1\}^r$, what is the only correct suffix c? Thus, the output of this stage is the remaining $2^{\lfloor n/2 \rfloor}$ data variables x's whose suffix of the its index in binary is the string c. That said, we can ignore the inputs that do not meet this condition and proceed with selecting the other half b. Finally, we use D_r to select the other half b and the expression $b + 2^r \cdot c$ yields the correct index. Thus, we obtain a recursive computation of MUX_n in terms of MUX_r as suggested by the last equation.

We will now analyze the circuit complexity of this construction. In particular,

- the first stage of selecting the suffix c for the data bit's index in binary requires $2^r \cdot 2^s = 2^n$ $\wedge$-gates and $2^r(2^s - 1) = 2^n - 2^r$ $\vee$-gates. Specifically, there are 2^r possible prefixes $b \in \{0,1\}^r$, and for each b, we use 2^s $\wedge$-gates and $2^s - 1$ $\vee$-gates to select the correct c. Taking the complexity of D_s into account (i.e. $CC(\mathsf{DEC}_s)$), the total number of gates used in this stage is

$$CC(\mathsf{DEC}_s) + 2 \cdot 2^n - 2^r$$

- the second stage of selecting the prefix b for the data bit's index in binary, given a "fixed" suffix c requires 2^r $\wedge$-gates and $2^r - 1$ $\vee$-gates. Taking the complexity of D_r into account (i.e. $CC(\mathsf{DEC}_r)$), the total number of gates used in this stage is

$$CC(\mathsf{DEC}_r) + 2 \cdot 2^r - 1$$

  Note that the construction of this stage is exactly like the naive approach but for MUX_r.

Therefore, by Lemma 10 and substituting $N = 2^n$ and $r = \lfloor n/2 \rfloor$, $s = \lceil n/2 \rceil$, the total complexity of this construction is upper bounded by

$$CC(\mathsf{MUX}_n) \le CC(\mathsf{DEC}_r) + CC(\mathsf{DEC}_s) + 2 \cdot 2^n + 2^r - 1 = 2N + O(\sqrt{N})$$

□

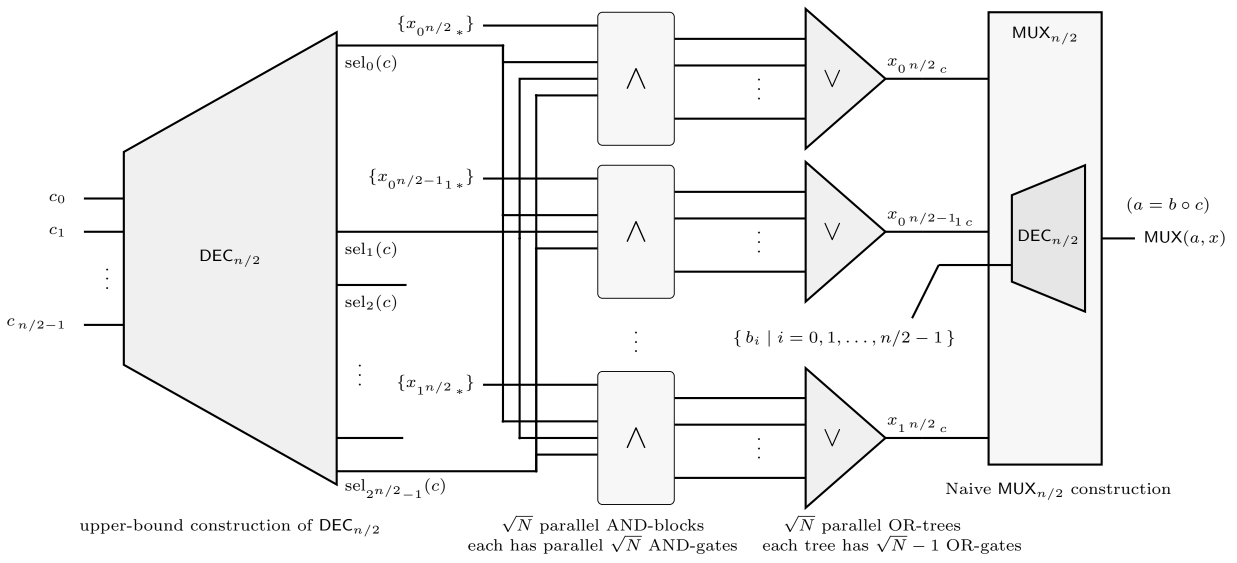

Figure 4: One-level expansion of MUX_n. The construction of $\mathsf{MUX}_{n/2}$-circuit on the right side follows the naive construction as in Figure 3.

The same analysis can be used to show an upper bound for eMUX_ℓ with an exact count rather than bigO. In particular, apply the upper bound in Theorem 11 for $\mathsf{eDEC}_{\ell-1}$, we obtain

$$\begin{aligned}
CC(\mathsf{eMUX}_\ell) &\le 2N + \sqrt{N} - 1 + 2 \cdot CC(\mathsf{eDEC}_{\ell-1}) \\
&= 2N + \sqrt{N} - 1 + 2(\sqrt{N} + \mathcal{S}(\sqrt{N})) \\
&= 2N + 3\sqrt{N} + 2 \cdot \mathcal{S}(\sqrt{N}) - 1, \quad \text{where } \mathcal{S}(\sqrt{N}) = \sum_{i=1}^{\log\log\sqrt{N}-1} 2^i \cdot (\sqrt{N})^{1/2^i}
\end{aligned}$$

which is consistent with [KP80]. However, Klein and Patterson pointed out that the analysis of Lemma 13 suggests a recursive construction which can give further improvement to the $O(\sqrt{N})$ term, and in particular, decrease the coefficient of the term $\sqrt{N}$ for eMUX_ℓ.

The analysis in Lemma 13 suggests a recursive construction for the circuit which can yield an improvement for the $3\sqrt{N}$ term. In particular, we notice that we can split the string b into two halves and continue "filtering" the remaining variables and repeat the process. With this observation, we establish the following upper bound for eMUX_ℓ.

Theorem 14. *Let $\ell \in \mathbb{N}$, $CC(\mathsf{eMUX}_\ell) \leq 2N + \mathcal{S}(N) + 3$, where $N = 2^{2^\ell}$, and $\mathcal{S}(N) = \sum_{i=1}^{\log\log N - 1} 2^i \cdot N^{1/2^i}$*

Proof. Let $r = s = 2^\ell/2$. We apply one more iteration of the recursive construction and observe the improvement. In particular, we expand the expression for eMUX_ℓ similar to Lemma 13 further by splitting b into b_1 and c_1 where each has length $r/2 = 2^\ell/4$, then use two subcircuits computing $\mathsf{eDEC}_{\ell-2}$, one to fix c_1 and the other to process b_1 accordingly.

$$\begin{aligned}
\mathsf{eMUX}_\ell(a,x) &= \mathsf{eMUX}_\ell(b_1 \circ c_1 \circ c_0, x)\\
&= \bigvee_{i=0}^{2^{r/2}-1} \bigvee_{j=0}^{2^{r/2}-1} \bigvee_{k=0}^{2^s-1} (sel_i(b_1) \wedge sel_j(c_1) \wedge sel_j(c_0) \wedge x_{i\cdot 2^r + j\cdot 2^{r/2}+k})\\
&= \bigvee_{i=0}^{2^{r/2}-1} \left(sel_i(b_1) \wedge \left(\bigvee_{j=0}^{2^{r/2}-1} \left(sel_j(c_1) \wedge \left(\bigvee_{k=0}^{2^s-1} (sel_k(c_0) \wedge x_{i\cdot 2^r + j\cdot 2^{r/2}+k}) \right)\right)\right)\right)\\
&= \mathsf{eMUX}_{\ell-2}\left(b_1, \left(\bigvee_{j=0}^{2^{r/2}-1} \left(sel_j(c_1) \wedge \left(\bigvee_{k=0}^{2^s-1} (sel_k(c_0) \wedge x_{i\cdot 2^r + j\cdot 2^{r/2}+k}) \text{ , for } i = 0, \ldots, 2^{r/2}-1 \right)\right)\right)\right)
\end{aligned}$$

We will now analyze the circuit complexity of this construction. In particular,

- The first stage of fixing c_0 is the same as the analysis in Lemma 13 which requires
$$CC(\mathsf{eDEC}_{\ell-1}) + 2 \cdot 2^{2^\ell} - 2^{2^\ell/2}$$
- The second stage of selecting the c_1-part for the index requires $2^{\ell/2}$ $\wedge$-gates and $2^{r/2}(2^{r/2}-1)$ $\vee$-gates since this stage goes through all $2^{\ell/2}$ indices to select the c_1-part and combine. Taking $CC(\mathsf{eDEC}_{\ell-2})$ into account, the total number of gates used in this stage is
$$CC(\mathsf{eDEC}_{\ell-2}) + 2^{2^{\ell/2}} + 2^{2^{\ell/4}}(2^{2^{\ell/4}} - 1) = CC(\mathsf{eDEC}_{\ell-2}) + 2 \cdot 2^{2^{\ell/2}} - 2^{2^{\ell/4}}$$
- The third stage of selecting the b_1-part for the index, given fixed c, c_1-parts requires $2^{\ell/4}$ $\wedge$-gates and $2^{\ell/4} - 1$ $\vee$-gates. Taking the complexity $CC(\mathsf{eDEC}_{\ell-2})$, the total number of gates used in this stage is
$$CC(\mathsf{eDEC}_{\ell-2}) + 2 \cdot 2^{2^{\ell/4}} - 1$$

Substituting $N = 2^{2^\ell}$ and apply Theorem 11 for $CC(\mathsf{eDEC}_{\ell-1})$ and $CC(\mathsf{eDEC}_{\ell-2})$, we obtain

$$\begin{aligned}
CC(\mathsf{eMUX}_\ell) &\leq CC(\mathsf{eDEC}_{\ell-1}) + 2\cdot 2^{2^\ell} - 2^{2^{\ell/2}} + CC(\mathsf{eDEC}_{\ell-2}) + 2 \cdot 2^{2^{\ell/2}} - 2^{2^{\ell/4}} + CC(\mathsf{eDEC}_{\ell-2}) + 2\cdot 2^{2^{\ell/4}} - 1\\
&= 2\cdot 2^{2^\ell} + 2^{2^{\ell/2}} + 2^{2^{\ell/4}} + CC(\mathsf{eDEC}_{\ell-1}) + 2\cdot CC(\mathsf{eDEC}_{\ell-2}) - 1\\
&\leq 2N + N^{1/2} + N^{1/4} + N^{1/2} + \mathcal{S}(N^{1/2}) + 2N^{1/4} + 2\cdot\mathcal{S}(N^{1/4}) - 1\\
&= 2N + 2N^{1/2} + 3N^{1/4} + \mathcal{S}(N^{1/2}) + 2\cdot\mathcal{S}(N^{1/4}) - 1
\end{aligned}$$

where $\mathcal{S}(M) = \sum_{i=1}^{\log\log M - 1} 2^i \cdot M^{1/2^i}$. We can repeat the process for $\ell = \log\log N$ times and express the upper bound of $CC(\mathsf{eMUX}_\ell)$ as the following recurrence

$$T(N) = \begin{cases} 3, & \text{if } N = 2\\ T(N^{1/2}) + 2N + \mathcal{S}(N^{1/2}), & \text{if } N > 2 \end{cases}$$

For the base case $N = 2$ (i.e. $\ell = 0$), we have eMUX_0 which is equivalent to multiplexing 2 data bits x_0, x_1. We can construct the circuit by using a eDEC_0-circuit which requires zero costly gate because it is simply the input variable itself (a_0) or its negation ($\overline{a_0}$). Then, we construct $\overline{a_0} \wedge x_0$, $a_0 \wedge x_1$ and combine their results with an $\vee$-gate which yields a total of 3 costly gates.

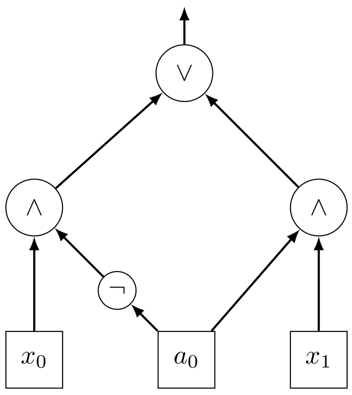

Figure 5: A circuit computing eMUX_0

Justification for the case of $N > 2$ is based on the analysis of performing one recursive step as in Lemma 13 as shown below

$$\begin{aligned}
CC(\mathsf{eMUX}_\ell) &\leq \underbrace{CC(\mathsf{eDEC}_{\ell-1}) + (2N - \sqrt{N})}_{\text{selecting data bits based on an address slice stage}} + CC(\mathsf{eMUX}_{\ell-1}) \\
&\leq \underbrace{\sqrt{N} + \mathcal{S}(\sqrt{N})}_{\text{apply Theorem 11 for } CC(\mathsf{eDEC}_{\ell-1})} + (2N - \sqrt{N}) + \underbrace{CC(\mathsf{eMUX}_{\ell-1})}_{T(N^{1/2})} \\
&= 2N + \mathcal{S}(N^{1/2}) + T(N^{1/2}) = T(N)
\end{aligned}$$

Now we unroll the recurrence for $\log\log N$ steps (until the data bits domain shrink to the base case)

$$\begin{aligned}
T(N) &= T(N^{1/2}) + 2N + \mathcal{S}(N^{1/2}) \\
&= T(N^{1/4}) + 2N^{1/2} + \mathcal{S}(N^{1/4}) + 2N + \mathcal{S}(N^{1/2}) \\
&= T(N^{1/8}) + 2N^{1/4} + \mathcal{S}(N^{1/8}) + 2N^{1/2} + \mathcal{S}(N^{1/4}) + 2N + \mathcal{S}(N^{1/2}) \\
&\dots \\
&= T(2) + \sum_{i=0}^{\log\log N - 1} (2N^{1/2^i} + \mathcal{S}(N^{1/2^{i+1}})) \\
&= 3 + \sum_{i=0}^{\log\log N - 1} (2N^{1/2^i} + \mathcal{S}(N^{1/2^{i+1}}))
\end{aligned}$$

Note that we can simplify the expression further by simplifying $\sum_{i=0}^{\log\log N-1}(2N^{1/2^i} + \mathcal{S}(N^{1/2^{i+1}}))$. In particular, for each i, we expand $\mathcal{S}(N^{1/2^{i+1}}) = \sum_{j=1}^{\log\log N - i - 2} 2^j N^{1/2^{i+1+j}}$. For readability, set $L = \log\log N$, then, we have

$$\sum_{i=0}^{L-1}(2N^{1/2^i} + \mathcal{S}(N^{1/2^{i+1}})) = \sum_{i=0}^{L-1} 2N^{1/2^i} + \sum_{i=0}^{L-1}\sum_{j=1}^{L-i-2} 2^j N^{1/2^{i+1+j}}$$

The index of the inner sum runs up to $L - 1 - 2$ because by definition $\mathcal{S}(M) = \sum_{i=1}^{\log\log M - 1} 2^i \cdot M^{1/2^i}$, then for $M = N^{1/2^{i+1}}$, we have $\log\log M - 1 = L - (i+1) - 1 = L - i - 2$.

Re-index the double sum with $u = i + j + 1$, then since $j \geq 1$, we have $u - 1 - i \geq 1 \iff i \leq u - 2$. Thus,

$$\sum_{i=0}^{L-1} \sum_{j=1}^{L-i-2} 2^j N^{1/2^{i+1+j}} = \sum_{u=1}^{L-1} \sum_{i=0}^{u-2} 2^{u-i-1} N^{1/2^u}$$

Rename the index of the single sum with u and factor out the first term of the sum, we have

$$\sum_{i=0}^{L-1} 2N^{1/2^i} = 2N + \sum_{u=1}^{L-1} 2N^{1/2^u}$$

Putting everything together, we have

$$\begin{aligned}\sum_{i=0}^{L-1} 2N^{1/2^i} + \sum_{i=1}^{L-1} \sum_{i=1}^{L-i-2} 2^j N^{1/2^{i+1+j}} &= 2N + \sum_{u=1}^{L-1} 2N^{1/2^u} + \sum_{u=1}^{L-1} \sum_{i=0}^{u-2} 2^{u-i-1} N^{1/2^u} \\ &= 2N + \sum_{u=1}^{L-1} \left(2 + \sum_{i=0}^{u-2} 2^{u-i-1}\right) N^{1/2^u}\end{aligned}$$

Finally, we evaluate $2 + \sum_{i=0}^{u-2} 2^{u-i-1}$ to determine the coefficient of $N^{1/2^u}$. To this end, re-index the sum with $k = u - i - 1$, and since $i = 0, 1, \ldots, u - 2$, we have $k = u - 1, u - 2, \ldots, 1$. In other words,

$$2 + \sum_{i=0}^{u-2} 2^{u-i-1} = 2 + \sum_{k=1}^{u-1} 2^k = 2 + \frac{2(2^{u-1} - 1)}{2 - 1} = 2^u \quad \text{(apply geometric series formula)}$$

Thus, substitute $L = \log \log N$ and $\mathcal{S}(N) = \sum_{i=1}^{\log \log N - 1} 2^i \cdot N^{1/2^i}$, we obtain the desired upper bound

$$T(N) = 3 + 2N + \sum_{u=1}^{\log \log N - 1} 2^u N^{1/2^u} = 2N + \mathcal{S}(N) + 3$$

□

If we unroll the first few terms in the sum $\mathcal{S}(N)$, we get $2N + 2\sqrt{N} + O(\sqrt[4]{N})$ which is a slight improvement over Lemma 13 which is $2N + 3\sqrt{N} + O(\sqrt[4]{N})$

Remark 15. *We think it can be beneficial for gaining a better understanding on how* MUX_n *works for general cases where* n *is either even or odd by getting an exact count of the upper bound for those cases. We think each may differ by a few gates, but the general form of the upper bound should still be* $2N + 2\sqrt{N} + O(\sqrt[4]{N})$.

B.3 Future Directions

An obvious interesting research direction is to show that the bounds of MUX are tight and that the recursive construction of MUX as in Theorem 14 is optimal. This will be beneficial to show that MUX is a potential candidate for proving ETH-hardness of MCSP as discussed in [CDJ25]. This work was successful in showing the optimal circuit structures compuing XOR_n which is a "tree" of $(n - 1)$ XOR_2-sub-circuits via Gate Elimination alone. The key idea is to argue a particular shape at the bottom level of the circuit via an intricate case analysis to show that any deviation from the target shape will violate the tight bound of XOR. The XOR-function is well-behaved enough that we can feasibly exhaust all the cases, but this approach is a major challenge for MUX because we will need to argue that the decoder DEC is a "must have" which remains elusive to us using Gate Elimination alone. This observation motivates us to look at other circuit lower bound techniques to gain more insight on MUX, one of which is the *Fusion Method* which yields a lower bound on the AND-complexity of circuits in Negation Normal Forms (NNF). Even though we consider generic circuits throughout this work, we note that the provided upper bound constructions for DEC and MUX are already in NNF, and gaining some knowledge about MUX in the NNF world is interesting and significant enough towards understanding the behavior of this function.

As surveyed by Wigderson [Wig93], the Fusion Method turns the computation into a static cover problem over a set of "fusing functionals." By choosing functionals tailored to the address/data separation in MUX_n, it may be possible to prove that small covers are impossible and thereby derive new bounds. Recently, Calavar and Oliveira recast Fusion Method using a modern set-theoretic language that works for various settings including non-monotone Boolean circuits [CO25]. They mention that reproving known circuit lower bounds via Gate Elimination, or better yet improving these lower bounds, for non-monotone functions using their framework remains an open problem.

Intuition on proving Lower Bound via Fusion Method. Informally speaking, the fusion method asks a simple question: if a circuit for f were really small, could we "mix and match" pieces of many true examples to accidentally accept a false one? Think of each input bit (or its negation) as a basic condition, and think of an AND as saying "you must satisfy all these conditions at once." A small circuit would impose only a few such "all-at-once" requirements. Fusion formalizes a tester that collects all conditions satisfied by a known true input and tries to keep combining them in the specific ways the small circuit would—hoping to land on a contradiction when we apply them to a false input. Our job, then, is to specify a short checklist of required combinations (or *a cover*) that forces any such attempt to end in a contradiction. The size of the smallest checklist tells you how many ANDs you truly need. In short, if every "mix and match" plan with too few AND-steps breaks, the circuit must use more AND gates—giving the lower bound. Thus, the minimal cover size exactly measures the AND cost needed to compute f. In this way, fusion cleanly yields DeMorgan circuit lower bounds by showing that any attempt to compute f with too few AND-gates is inevitably incorrect on certain inputs.

Circuits Model in Negation Normal Forms (NNF) as Set Theoretic. We briefly explain circuits in NNF and its cover complexity in a high level idea. Imagine every string $x \in \{0,1\}^n$ as a point in a universe of all length-n strings Γ. Each literal (a variable or its negation) denotes a "region" in that universe, i.e. x_i is the region of strings where the i^{th}-bit of is 1, and $\overline{x_i}$ is the complementary region, i.e. strings where the i^{th}-bit is 0. In negation normal form (NNF), every internal gate is either an AND-gate or an OR-gate, and we read those gates as pure set operations, i.e. an AND-gate corresponds to set intersection ($\cap$) and an OR-gate corresponds to set union ($\cup$). This means that an AND-gate keeps the overlap of two regions, whereas an OR-gate takes the combined regions.

We evaluate the circuit from the leaf-level upward, and as proceed, we construct the regions defined by the internal gates, step by step, and the region defined by the output gate is exactly the set A of length-n strings where the circuit evaluate to 1. Thus, counting gates now matches counting operations: each AND is one intersection, each OR is one union, and total circuit size is the total number of these set operations used to assemble A from the literals.

Then, for a high level idea, *a cover* preserves some properties and conditions on the computation of the circuit expressed as set-theoretic, and a *cover complexity* is a measurement on the size of the "checklist" of required combinations/conditions. Calaver and Oliveira proved that the number of intersections required is "sandwiched" between the cover complexity and its square.

Theorem 16. *(Informal of Theorem 22 and 24 of [CO25]) Let $D_\cap$ denote the intersection complexity of a Boolean circuit expressed in set theoretic, and let ρ denote its cover complexity, then $\rho \le D_\cap \le \rho^2$.*

Potential next steps towards this direction. Even though what we have learned so far about Fusion Method applies for DeMorgan circuits in NNF, rather than generic DeMorgan circuits, it is reasonable to think that this approach is helpful for gaining more knowledge on the lower bound of DEC and MUX for various reasons.

- Under the NNF restriction, we can observe that the construction shows in Figure 2 is the only option for the upper-bound construction for DEC in the NNF world since we cannot have negations in the middle of the circuit. This construction shows a layer of AND-gates only which is compatible with how Fusion Method is useful for determining the AND-complexity (i.e. the lower bound on the number of AND-gates). Thus, if we can extend the framework in [CO25] for multi-output functions, then it is reasonable to believe that DEC-circuits in NNF must use *many* AND-gates and thus a tighter bound.

- Along this line, we observe that the full recursive upper bound construction for MUX-circuits as proposed in Theorem 14 is already in NNF assuming the NNF construction of DEC-circuits. Looking further in the construction, we observe that it *mainly* uses AND-gates as when we shrink the overhead to just MUX_1 in this full recursive construction. Thus, it is also reasonable to believe that Fusion Method can help us show that the AND-complexity of MUX in NNF is high. It is plausible to believe that by analyzing and knowing the exact AND-complexity of MUX, we can gain more knowledge on how these AND-gates must be used structurally so that it does not violate the bounds.