Compact Circuits for Efficient Möbius Transform

Subhadeep Banik\(^1\) and Francesco Regazzoni\(^2\)\(^1\)

\(^1\) Università della Svizzera Italiana, Lugano, Switzerland, \{subhadeep.banik,francesco.regazzoni\}@usi.ch
\(^2\) University of Amsterdam, Amsterdam, Netherlands, f.regazzoni@uva.nl

Abstract. The Möbius transform is a linear circuit used to compute the evaluations of a Boolean function over all points on its input domain. The operation is very useful in finding the solution of a system of polynomial equations over GF(2) for obvious reasons. However the operation, although linear, needs exponential number of logic operations (around \(n \cdot 2^n - 1\) bit xors) for an \(n\)-variable Boolean function. As such, the only known hardware circuit to efficiently compute the Möbius Transform requires silicon area that is exponential in \(n\). For Boolean functions whose algebraic degree is bound by some parameter \(d\), recursive definitions of the Möbius Transform exist that requires only \(O(n^{d+1})\) space in software. However converting the mathematical definition of this space-efficient algorithm into a hardware architecture is a non-trivial task, primarily because the recursion calls notionally lead to a depth-first search in a transition graph that requires context switches at each recursion call for which straightforward mapping to hardware is difficult. In this paper we look to overcome these very challenges in an engineering sense. We propose a space efficient sequential hardware circuit for the Möbius Transform that requires only polynomial circuit area (i.e. \(O(n^{d+1})\)) provided the algebraic degree of the Boolean function is limited to \(d\). We show how this circuit can be used as a component to efficiently solve polynomial equations of degree at most \(d\) by using fast exhaustive search. We propose three different circuit architectures for this, each of which uses the Möbius Transform circuit as a core component. We show that asymptotically, all the solutions of a system of \(m\) polynomials in \(n\) unknowns and algebraic degree \(d\) over GF(2) can be found using a circuit of silicon area proportional to \(m \cdot n^{d+1}\) and circuit depth proportional to \(2 \cdot \log_2(n - d)\).

In the second part of the paper we introduce a fourth hardware solver that additionally aims to achieve energy efficiency. The main idea is to reduce the solution space to a small enough value by parallel application of Möbius Transform circuits over the first few equations of the system. This is done so that one can check individually whether the vectors of this reduced solution space satisfy each of the remaining equations of the system using lower power consumption. The new circuit has area also bound by \(m \cdot n^{d+1}\) and has circuit depth proportional to \(d \cdot \log_2 n\). We also show that further optimizations with respect to energy consumption may be obtained by using depth-bound Möbius circuits that exponentially decrease run time at the cost of additional logic area and depth.

Keywords: Boolean Functions, Möbius transform, Solution of Equation System.

1 Introduction

Several cryptanalytic problems can be reduced to instances of solving a system of multivariate polynomial equations over GF(2). For example, block ciphers with low multiplicative complexity like LowMC [ARS\(^*\)15] employ only 3-bit S-boxes of algebraic degree 2. It is known that given any single plaintext-ciphertext pair from an \(r\)-round instance of LowMC gives rise to a system of equations in the secret key-bits of algebraic degree

\[2 \cdot \log_2(n - d)\]
2^{n/2} [Din21]. The public key in the signature scheme PICNIC v3.0, consists of a single plaintext/ciphertext pair generated by the LowMC block cipher using the secret key as the block cipher key. The designers recommend using 4-round instances of the block cipher for this purpose. In this case, the secret key that corresponds to a given public key is fully described by a system of \( n \) Boolean equations of degree 4 in the \( n \) unknown bits of the key [Din21]. Thus finding the secret key amounts to cryptanalysis of the block cipher using the single plaintext/ciphertext pair available as the public key of the signature, which amounts to solving \( n \) degree 4 equations in \( n \) unknowns. (However, we would like to state that our work does not practically threaten the full specification of PICNIC.) It is also known that forging a signature in public key signature schemes like UOV can be done by solving a set of quadratic equations over \( GF(2) \) [KPG99]. Other than this there are specific problems in combinatorics like the graph-coloring problem (i.e. given a graph decide whether it can be colored using \( k \) colors with no two adjacent vertices assigned the same color) which can be reduced to an instance of solving multi-variate polynomials in \( GF(2) \) [Bar09, Appendix C].

The problem can be stated in the following way: given \( n \) indeterminates \( x_1, x_2, \ldots, x_n \), and \( m \) polynomials \( f_i \in \mathbb{F}[x_1, x_2, \ldots, x_n] \) (for \( i \in [1, m] \)), where \( \mathbb{F} \) is any finite field. The task is to find common solutions \( x^* \in \{0, 1\}^n \), such that \( f_i(x^*) = 0 \) for all \( i \). Over any finite field \( \mathbb{F} \), the problem is NP-complete already when the polynomials are quadratic. For a complete analysis of the significance of equation solvers in cryptography please see the discussion in [Bou22]. Hereafter, we will focus on the case of the Boolean field \( \mathbb{F} = GF(2) \).

### 1.1 Previous Work

To the best of our knowledge, there have been two previous works on hardware/software architectures for fast exhaustive search over \( GF(2) \). The main idea is as follows: the secret \( x^* \) we are looking for is obviously a point which evaluates to zero for all the \( f_i \). Thus at the index \( x^* \), the truth tables of all the Boolean polynomials \( f_i \) will have the constant 0. Hence, we are looking for the indices \( x^* \) at which the logical OR of all the truth tables of all the \( f_i \)'s is 0. In [BCC+10], the authors use the Gray code technique to evaluate the truth table of each polynomial \( f_i \). Gray codes are linear codes that have the property that successive codewords differ by only one bit. There are many methods of constructing such codes in literature, and one of the simplest way is to define the \( i \)-th code word as \( g_i = i \oplus (i \gg 1) \) (the \( \gg \) denotes the shift right operator). For example the eight 3-bit codewords listed sequentially are: 000, 001, 011, 010, 110, 111, 101, 100. Take any polynomial \( f_i \): we want to evaluate \( f_i \) over all \( 2^n \) points of its input domain. Then it is more efficient to do this evaluation in the order specified by the Gray code, i.e. first \( f_i(g_0) \), then \( f_i(g_1), f_i(g_2) \ldots \) etc. The reason for this is as follows: note \( f_i(g_0) = f_i(0) \) is just the constant term of \( f_i \), thereafter if \( t \) is the only bit-position where the successive codewords \( g_j \) and \( g_{j+1} \) differ in, and we already have the value of \( f_i(g_j) \) then we can use a Taylor-like expansion formula for Boolean functions to compute \( f_i(g_{j+1}) \):

\[
f_i(g_{j+1}) = f_i(g_j) \oplus \frac{\delta f_i}{\delta x_t}(g_j). \tag{1}
\]

Here \( \frac{\delta f_i}{\delta x_t} \) is the 1st order derivative of the function \( f_i \) at the point \( x_t \). For example if \( f_i = x_1 x_2 \oplus x_3 \oplus x_1 x_4 x_5 \), then \( \frac{\delta f_i}{\delta x_2} = x_2 \oplus x_4 x_5 \) and \( \frac{\delta f_i}{\delta x_3} = x_1, \frac{\delta f_i}{\delta x_4} = 1 \) etc. It is known that the derivative has algebraic degree at least one less than the original function, and so if the derivative is not a constant or a degree one function we recursively evaluate the derivative term in Equation (1) with another round of Taylor expansion. The method obviously works best if the function \( f_i \) is quadratic, but can also be applied to evaluate moderately higher degree functions too if some of the derivatives are precomputed. In a follow up work [BCC+13], the same authors proposed a hardware circuit for the problem, however, for only degree 2 functions (that needed negligible amount of pre-computations).
Another method to compute the truth table of a Boolean polynomial from its algebraic expression is via the Möbius Transform. This method does not require pre-computations. The transform can be simply evaluated as $\vec{v} = M_n \vec{u}$, where $\vec{u}$ is the $2^n \times 1$ algebraic normal form (ANF) vector of any $n$-variable Boolean function, $M_n$ is the $2^n \times 2^n$ binary Möbius matrix, and $\vec{v}$ is the truth-table of the function, with its $i$-th element being the function evaluation at the binary string representation of $i$. As we will soon see, a naive interpretation of this method requires time and space exponential in $n$ to compute. However, there exist more subtle methods to compute the matrix-vector product given above in polynomial space (bounded by $n^{d+1}$ where $d$ is the algebraic degree of the Boolean polynomial). Translating this to hardware is a non-trivial task as the underlying algorithm is significantly complex. In this paper, we will propose strategies to translate the Möbius Transform algorithm into a hardware circuit and we will demonstrate how to overcome the engineering challenges involved. We then show how multiple instances of the above Möbius Transform circuit can be efficiently used to solve or perform fast exhaustive search for roots of equation systems over $GF(2)$ whose degree is bound by some constant $d$. We show that asymptotically, with silicon footprint proportional to $m \cdot n^{d+1}$ we can describe a circuit that finds roots of a system of $m$ polynomial equations of degree $d$ in $n$ unknowns over $GF(2)$.

1.2 Impact and Comparison with the state of art

Till date we are not aware of any hardware architecture that solves equations over $GF(2)$ of degree larger than 2. This therefore presents the first instance of a solver in hardware for higher degree equation systems over $GF(2)$. Furthermore, building a truth table in hardware has many significant cryptographic applications (please see [Bou22, Section 1.3]). Very briefly, it is known that the pre-image attack [DS11] against the Hamsi-256 hash function requires the attacker to construct efficiently truth tables of degree 6 over 32 variables. The attack against the stream ciphers GEA-1/GEA-2 requires construction of degree 4 truth tables over 33 variables [BDL+21]. The cube attack on Trivium [HST+21] requires construction of such tables over up to 75 variables of degree 20. The attack against Pyjamask-96 [DRS20] also requires construction of truth tables of Boolean functions of degree 4 of around 128 variables.

1.2.1 Comparison with Linearization Algorithms

Linearization based algorithms like XL [CKPS00] and Elimlin [CB07] also attempt to find the solution of a system of Boolean equations through matrix manipulation techniques like Gaussian Elimination (GE). The idea is to rewrite every higher degree monomial in the equation system as a new linear variable. This converts a system of $m$ equations of any arbitrary algebraic degree $d$ to a system of $m$ linear equations in around $O(n^d)$ extended variables. Using hardware accelerators for GE like the SMITH framework [BMP+06], one could also describe a circuit that finds roots of the system using silicon area proportional to $m \cdot n^d$. However, as shown in [Bar09, Section 12.3], such an approach will generate basis vectors for a space containing an exponential number of false solutions, and it is not immediately clear how efficient circuit hardware architectures can be described to eliminate them. However, note that there are papers in software like [BDT22, JV17, BFSS13, LPT+17, Din21, BKW19] which achieve this in software in less than brute force time.

1.2.2 Area Time product

The Möbius Transform operation is similar to the Fast Fourier Transform (FFT) which is defined over larger rings. It is known that the if the layout of the circuit is restricted to 2
dimensions then the Area(A) and Time(T) product of the FFT is subject to a lower bound $AT^2 \geq N^3/2$ [Tho79]. Because the circuit has to hold the $N$ input data, this implies that $AT \geq N^{3/2}$. This lower-bound is based on communication complexity, i.e. the fact that “wires take space”. In this paper we begin with two circuits for Möbius Transform i.e. Expmob1/Expmob2 that take $AT^2 = N \cdot \log(N)$ and $N \cdot (\log(N))^2$ respectively.

Instead of optimizing $AT$ directly, consider the case when $A$ is bound polynomially by some $n^{d+2}$ and $T$ is the clock cycle count of the operation on the given circuit. It makes sense to consider this, since as $n$ increases, it is unreasonable to expect exponential amount of silicon resources to be available for manufacturing. We could ask the question: what is the minimum value of the $AT$ product given that $A \in O(n^{d+2})$ and the task is to solve $n$ equations of $n$ variables in degree $d$. In fact this is the metric we look at for the various circuit architectures that we propose in the paper. In Section 5.6, we show that the minimum $AT$ product for all the solvers that we have considered is around $4 \cdot n^{d+2} \cdot 2^{n-h}$ where $h^2 \cdot 2^h = n^{d+2}$.

1.3 Contribution and Organization

In this paper we present a novel hardware architecture for the Möbius transform for $n$-variable Boolean functions of degree $\leq d$ that requires silicon resources that are polynomially bounded by $n^{d+1}$. We use the recursive definition of the transform found in [Din21, Section 4.2], and identify and solve the engineering difficulties of translating such an algorithm into hardware. Parallel instances of this architecture can be combined to construct hardware solvers that find roots of an underlying equation system over $GF(2)$ by exhaustive search. We describe the architectures of three such solvers the last of which is able to find all roots of any system of $m$ Boolean equations in $n$ unknowns and algebraic degree $d$ in circuit area proportional to $m \cdot n^{d+1}$ and circuit depth proportional to $2 \cdot \log_2(n - d)$ units.

In the next part of the paper we address the issue of energy efficiency. Given an equation system with $m$ equations, performing $m$ Möbius computations leads to a lot of redundant computation and thus wastage of computational effort and energy. We introduce a solver architecture called Polysolve4, that aims to minimize these redundant computations. We first use the Möbius Transform circuit to extract the roots of some $\mu < m$ equations of the system. Assuming that this solution space is small enough we try to individually evaluate the remaining $m - \mu$ equations at the all the vectors of this reduced space. Then we try to construct Polysolve4 circuits with Möbius Transform circuits that are height bound. We explain the concept of height bound circuits. We then show that (for solving 20 quartic equations in 20 variables) these circuits can obtain around 100 times energy efficiency when compared to the Polysolve3 circuit.

The rest of the paper is organized in the following manner. Section 2 presents some preliminary lemmas and definitions in this field. In Section 3 we look at the recursive definition of Möbius transform and we explain in detail how the hardware circuit for the same is designed. In Section 4, we first show how to combine multiple instances of the Möbius Transform circuit that produces a solver that finds at least one root of the underlying equation system. We then list two variants of this architecture, the last of which is able to find all the roots of the equation system. Section 5 describes the solver Polysolve4 and analyzes of energy consumption of the circuit as a function of the internal parameter $\mu$ of the circuit. We further look at depth-bound Möbius Transform circuits that enable use to find better energy/time consumption figures. Section 6 concludes the paper.

1.3.1 Notations:

Note that henceforth in the paper $T$ will denote the number of clock cycles required to complete any operation. This value of $T$ is considered for all the $AT$ metric optimizations.
throughout the paper. Simultaneously we use the notations $T_{\text{min}}$, $T_{\text{cr}}$, $T_d$ at multiple points in the paper. All these symbols denote some physical time parameter associated with the circuit and will be made clear when they appear. As such they are measured in ns or similar denotations.

2 Definitions and Preliminaries

**Boolean function**: An $n$-variable Boolean function is a map from $\{0, 1\}^n \rightarrow \{0, 1\}$ and it can be uniquely represented by its algebraic expression, called algebraic normal form or ANF. The algebraic expression of such a function using the $(\oplus, \cdot)$ basis can be written as

$$f(\vec{x}) = f(x_0, x_1, \ldots, x_{n-1}) = \bigoplus_{i \in \{0, 1\}^n} a_i x^i$$

Here $i := i_0 i_1 \cdots i_{n-1}$ is the binary string of length $n$, with $i_j$ as the individual bits and $x^i$ is defined as $\prod x_j^{i_j}$. The ANF vector $\vec{a} = [a_0, a_1, \ldots, a_{2^n-1}]$ is defined as the $2^n$-length string of all the $a_i$’s.

**Example 1.** For example, consider the 3-variable function $f = 1 \oplus x_0 x_1 \oplus x_2 \oplus x_0 x_2$. We can write this as $x_0^3 x_1^3 x_2^3 \oplus x_0^3 x_1^3 x_2^3 \oplus x_0^3 x_1^3 x_2^3 \oplus x_0^3 x_1^3 x_2^3$. The function can be expressed as a length 8 bit-vector $\vec{u}$ with bits at locations given by the binary strings 000, 110, 001 and 101 i.e. 0, 6, 1 and 5 set to 1 and the rest of the bits 0, which is to say that $a_0 = a_1 = a_2 = a_6 = 1$ and the rest of the $a_i = 0$.

The algebraic degree of the function (the provided function is not identically null) is defined as the maximum hamming weight of the string $i$ such that $a_i = 1$. Thus in the previous example, the algebraic degree is 2. For functions having degree $d$, all the coefficients $a_i$ such that $\text{hw}(i) > d$ are naturally 0. Since there are exactly $\binom{n}{d}$ length $n$ strings of hamming weight $i$, we can see that the ANF of degree $d$ function can be expressed as $\binom{n}{d} := \sum_{i=0}^{d} \binom{n}{i} < n^d$ binary coefficients.

**Truth Table**: The vector of evaluations of a Boolean function at all its input points is called its Truth Table (therefore this is a $2^n$ length vector). The ANF and the Truth table vectors of any Boolean function are closely related by the Möbius transform. Let $\vec{v} = [v_0, v_1, \ldots, v_{2^n-1}]$ be the truth-table of the function $f$, with its $i$-th element being the function evaluation at the binary string representation of $i$, i.e. $v_i = f(i_0, i_1, \ldots, i_{n-1})$. Then it is well known that $\vec{v}, \vec{u}$ are related as $\vec{v} = M_n \cdot \vec{u}$, where $M_n$ is the Möbius matrix of size $2^n \times 2^n$. The $i, j$-th element of this matrix $m_{ij}$ is given as

$$m_{ij} = 1 \text{ if } j \preceq i \text{ and } 0 \text{ otherwise}.$$ 

The operator $\preceq$ is a partial order over all binary strings: we say that $j \preceq i$ if the binary string representing $j$ is less than or equal to the binary string representing $i$ in all indices. For example, $4 \preceq 5$, since 100 is less than 101 at all bit-locations, but $3 \not\preceq 4$ since 011 exceeds 100 in the last 2 bit-locations.

The Möbius matrix $M_n$ has been widely studied in literature: for example it is well known that is lower-triangular and involutive i.e. $M_n^{-1} = M_n$. Thus both $\vec{v} = M_n \cdot \vec{u}$ and $\vec{u} = M_n \cdot \vec{v}$ hold. An example of the $8 \times 8$ Möbius matrix $M_3$, i.e. for $n = 3$ is shown in Figure 1. This helps us see an alternative recursive definition of $M_n$. If we define $M_1 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$, then for all $n > 1$, we have $M_n = M_1 \otimes M_{n-1}$, where $\otimes$ is the matrix tensor product.

Multiplication of a vector by this matrix can be quickly executed by the butterfly-like operations shown in Figure 2. The butterfly operation shaded in blue is actually
Compact Circuits for Efficient Möbius Transform

\[
M_3 = \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
\]

*Figure 1:* An example of the Möbius matrix \(M\) for \(n = 3\)

Multiplication of the input 2-bit vector by the matrix \(M_1\). The figure tells us that for an \(n\)-variable function, the algorithm can be done in-place (without any additional memory) using around \(n \cdot 2^{n-1}\) xor operations and \(2^n\) space.

3 Implementing the Möbius Transform

Given Figure 2, we can think of many strategies to implement the basic transform if one has access to exponential silicon resources. The operation consists of \(n\) stages of sequential xor layers, with each layer having exactly \(2^{n-1}\) xor operations over bits. Given this, one can think of several circuit strategies to implement this:

**Expmob1** This architecture implements the circuit in Figure 2 as a single unrolled circuit, i.e. it implements all the \(n\) butterfly stages as dedicated circuits sequentially. Consider \(\text{one}_i(x) : \{0,1\}^{n-1} \rightarrow \{0,1\}^n\) to be the function that inserts a 1 in the \(i\)-th MSB position of \(x\), and \(\text{zero}_i(x)\) to be a function that inserts a 0 in the same position, i.e \(\text{one}_0(1001) = 11101\) and \(\text{zero}_0(1001) = 01001\) etc. Note that there are a total of \(2^{n-1}\) butterfly operations in each of the \(n\) stages. In the \(i\)-th stage (for \(0 \leq i \leq n-1\)), the \(j\)-th butterfly takes as input the bits in the position \(\text{zero}_i(j)\) and \(\text{one}_i(j)\) for all \(0 \leq j \leq 2^{n-1} - 1\). This requires a total of \(n \cdot 2^{n-1}\) number of 2-input xor gates in total. However such a circuit is able to compute the transformation in a single cycle.

**Expmob2** This configuration is slightly different from the previous circuit, in the sense that we have only a single stage butterfly which we operate over \(n\) clock cycles to compute the transform, i.e. similar to round based circuits of block ciphers in which a single round function circuit is iterated over a given number of cycles to compute the transform. Unlike the round function of a block cipher the successive stages

\[
x_0 \quad x_1 \quad x_2
\]

<table>
<thead>
<tr>
<th>(x_0)</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(\text{ANF}(f))</th>
<th>(\text{Truth Table}(f))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>1 1 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>1 1 0</td>
<td>0 0 1</td>
<td>0 0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0 0</td>
<td>0 1 0</td>
<td>0 1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0 0</td>
<td>0 1 1</td>
<td>0 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 0 0</td>
<td>1 0 1</td>
<td>1 0 1</td>
<td>1 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 0 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 0 0</td>
<td>1 1 1</td>
<td>1 1 1</td>
<td>1 1 1</td>
</tr>
</tbody>
</table>

\(f=1+x_0x_1+x_2+x_0x_2\)

*Figure 2:* Möbius transform on \(f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2\). The blue shaded component represents one butterfly unit.
of xor layers are not exactly similar. For example, consider the topmost butterfly circuit in each stage in Figure 2. The 1st stage takes bits at positions 0 and 4 as input, the second stage takes bits 0 and 2, the third stage takes bits 0 and 1 and so on. So to create a round based circuit, it would seem that one would need multiple $n$ to 1 multiplexers before each of the butterfly circuits. However this can be avoided using a simple observation. Consider $\pi_n$ to be the following permutation:

$$\pi_n(2x) = x, \text{ and } \pi_n(2x + 1) = 2^{n-1} + x \text{ for all } 0 \leq x < 2^{n-1}$$

The idea is that after the given stage of butterfly circuits, the bit at position $i$ be shifted to position $\pi_n(i)$. Such a permutation over the bits requires only re-routing of wires and thus no additional silicon area. This is essentially the entire round function circuit which has to be executed for a total of $n$ cycles for the transform to be computed. To see why this works, consider the following facts. Let $B_n$ be the block diagonal matrix defined as $B_n = M_1 \otimes I_{n-1}$, where $I_{n-1}$ is the identity matrix of size $2^{n-1} \times 2^{n-1}$. Note that $B_n$ is transformation defined by the first stage of butterfly layer in Figure 2. Let $P_n$ be the permutation matrix corresponding to $\pi_n$. Then it is easy to verify that the Möbius matrix $M_n = (P_n \cdot B_n)^n$.

### 3.1 Synthesis Results

In this section we will describe the flow of simulation followed for each of the circuits reported in the paper. The design was described at the RTL level using a hardware description language and functional correctness was first verified. Thereafter the circuit was synthesized using the Nangate 15nm Open Cell Library [MMR+15] using Synopsys Design Vision, mainly to ensure that the results obtained can be reproduced readily. One of the possible uses of the Möbius Transform, is in solving equation systems. In order to ensure that equations are solved as quickly as possible, the circuit compiler was instructed to specifically optimize the total critical path of the circuit. A timing analysis is then performed on the synthesized netlist using sufficient number of randomly generated test vectors, which outputs the switching statistic of every node in the circuit. This information is used by a power compiler software to estimate the average power consumed by the circuit. Energy is computed as the product of the average power and the total physical time taken for the circuit to execute a given operation.

In Figure 3, we present synthesis results for the circuits Expmob1 and Expmob2. It can be seen that Expmob1 performs better than Expmob2 in this regard, most probably due to the fact that additional hold/setup time constraints need to be met for Expmob2 for writing on to the register in each cycle. Similarly the additional energy required for the $n$ successive register writes makes Expmob2 less energy efficient as compared to Expmob1 as shown in Figure. Detailed results are shown in Table 7 in Appendix D.

However both these circuits require exponential amount of logic gates which starts to become a bottleneck as $n$ increases. We have already seen that a degree $d$ Boolean function can be represented with only $\binom{n}{d} < n^d$ binary coefficients, which means that for small values of $d$ the size of the ANF vector is polynomially bounded. Thus the size of the register that holds the ANF can be bounded by $n^d$. However, it is not possible to use Expmob1/Expmob2 circuit to compute the transform on this reduced size register, since, although the initial ANF vector is small, the output of each layer of butterflies are progressively larger till it reaches $2^n$ (which is the expected size of the truth table) after the last stage.

### 3.2 Recursive Algorithm for Möbius transform

There exists algorithms that perform the basic transform (on functions limited to degree $d$) using polynomial space only, i.e. bounded by $n^{d+1}$. We state the algorithm appearing in
The principal question is how do we circumvent the fact that even if we begin with a ANF vector of size that is polynomially bounded, each butterfly stage is likely to produce an output that is of size larger than the input.

**Example 2.** First let us make the following observation taking Figure 2 as a reference: consider the initial ANF vector $A_0 = [1100 0110]$ and the vector $A_1 = [1100 1010]$ just after the first layer. The initial vector corresponds to the function

$$f(x_0, x_1, x_2) = 1 \oplus x_0 x_1 \oplus x_2 \oplus x_0 x_2 = x_0 \cdot (x_1 \oplus x_2) \oplus (1 \oplus x_2)$$

Note that $[f(1, x_1, x_2) \oplus f(0, x_1, x_2)] := \frac{df}{dx_0}$ is simply the derivative of $f$ at the coordinate $x_0$. Both $\frac{df}{dx_0}$ and $f(0, x_1, x_2)$ have number of variables which is 1 less than the original function, and it is obvious that both their algebraic degrees can not be more than that of the original function.

Now consider the vectors in the top and bottom halves of $A_1$ i.e. $A_{\text{top}} = [1100]$ and $A_{\text{bottom}} = [1010]$. It is easy to observe/verify the following:

- **A:** $A_{\text{top}}$ is the ANF vector for $f(0, x_1, x_2)$ (in this case $1 \oplus x_2$) and $A_{\text{bottom}}$ is the ANF vector for $f(1, x_1, x_2)$ (in this case $1 \oplus x_1$).
- **B:** Both $A_{\text{top}}/A_{\text{bottom}}$ are outputs of the butterfly layer in which the input is $A_0$. Whereas $A_{\text{top}}$ is the arm of the butterfly that does not require xor computations, some xor computations are required for $A_{\text{bottom}}$. 

**Figure 3:** Synthesis results for **Expmob1** and **Expmob2** circuits. Energy measured at 1GHz

[Din21, Section 4.3]. The algorithm requires a notional depth-first traversal in a transition graph as shall be explained shortly.
The remaining steps from the 2nd stage onward can be seen as the parallel application of the Möbius Transform on the reduced variable Boolean functions \( f(0, x_1, x_2) \) and \( f(1, x_1, x_2) \).

Of course in the figure, both the transforms are computed parallelly, which requires \( 2^n - 1 \) space each and so the total space requirement is \( 2^n \) which is the same as the original. The idea behind the recursive transform is to do these 2 sub-transforms sequentially, i.e. one after the other so that the same space (i.e. register locations) can be used for both the transforms so that the cumulative space requirement does not add up. Let us state the algorithm now formally (Algorithm 1). The algorithm is parameterized by two quantities: number of variables \( n \), and the maximum algebraic degree \( d \) that the underlying function can have.

**Algorithm 1:** Recursive Möbius Transform

\[
\text{Möbius} \left( A_0, n, d \right)
\]

**Input:** \( A_0 \): The compressed ANF vector of a Boolean function \( f \)

**Output:** The Truth table of \( f \)

```java
/* Final recursion step, i.e. leaf nodes of recursion tree */
if \( n = d \) then
    Use the formula \( B = M_n \cdot A_0 \) to output partial truth table \( B \).
    /* Use either \texttt{Expmob1}/\texttt{Expmob2} to do this */
end
else
    Declare an array \( T \) of size \( \binom{n-1}{d} \) bits.
    /* Now we compute the 2 operations of the butterfly layer */
    1. Store 1st butterfly output i.e. \( A_{top} \) in \( T \) (requires no xors).
    Call Möbius \((T, n - 1, d)\)
    2. Store 2nd butterfly output i.e. \( A_{bottom} \) in \( T \) (requires some xors).
    Call Möbius \((T, n - 1, d)\)
end
```

For the sake of simplicity, we have excluded many operational details in the above algorithm to give the reader a better idea of the flow of the algorithm. The space requirement of this algorithm is easy to estimate from the algorithm description. We start out with \( \binom{n}{d} \) coefficients required to store \( A_0 \). Thereafter every successive \( i \)-th recursion stage requires \( \binom{n-i}{d} \) additional memory for all \( 1 \leq i \leq n - d \). The final stage can use \texttt{Expmob1}/\texttt{Expmob2} to perform Möbius Transform in-place and no additional memory is required. However in our experiments we preferred to use \texttt{Expmob1} because it is slightly faster. Hence the total space requirement of this procedure is given by (for a proof of the following please see Appendix B):

\[
S(n, d) = \sum_{i=0}^{n-d} \binom{n-i}{d} \in O(n^{d+1}). \tag{2}
\]

Notionally speaking the algorithm listed above describes a depth first recursion tree as shown in Figure 4, where each node in tree are connected to its two butterfly outputs. The depth first nature of the structure gives rise to complications even while implementing it in software. The problem with implementing such a routine, even in software, is the high number of context switches, that is needed to traverse one level down. In layman’s terms, before we can do a downward dive in the tree, the current state information, variables etc has to be stored in a separate memory location (usually denoted as “call-stack”). This costs time/energy and makes the algorithm less attractive from a practical point of view.
Compact Circuits for Efficient Möbius Transform

490

Figure 4: Recursion tree for the Möbius Transform algorithm. The blue shaded component roughly represents one arm of the butterfly unit. Note here \((x, \downarrow d) := \binom{x}{i_d}\).

3.3 Hardware circuit Polymob1

The goal obviously is to construct a circuit that does not take more than a total of \(S(n, d)\) bits of register space. As such we are looking at a circuit architecture similar to the one shown in Figure 5.

To understand the challenges in this circuit, note that one needs to follow the flow given by the orange line in the recursion tree in Figure 4. Now there is one top-level register of size \(\binom{n}{i_d}\) storing the initial ANF vector \(A_0\). There is only one \(\binom{n-1}{i_d}\) size register to store the second level coefficients \(A_{\text{top}}\) and \(A_{\text{bottom}}\). This implies that if in the first clock cycle the 2nd register stores \(A_{\text{top}}\), it must preserve this state till the entire left sub-tree rooted at this node is executed before it overwrites its state to \(A_{\text{bottom}}\). Similarly there is only one \(\binom{n-2}{i_d}\) register to store potentially four ANF vectors (two each from the butterfly operation on \(A_{\text{top}}\) and \(A_{\text{bottom}}\)). Thus the engineering challenge is to ensure that each register at the successive levels store and preserve appropriate state vectors till it is time to overwrite them, and so this in a manner that minimizes the total number of clock cycles required to execute the Möbius Transform.

Thus we arrive at the architecture in Figure 5. Each \(i\)-th level has a single register of size \(\binom{n-i}{i_d}\) (for \(0 \leq i \leq n - d\)), and from \(i = 1\) onwards each register is preceded by a 3:1 multiplexer of size \(\binom{n-1}{i_d}\). This is because each register must be able to accept 3 different inputs:

1. Its own output state, or in other words it must be able to preserve its state.
2. Either of the 2 outputs of the butterfly stage preceding it.

3.3.1 Architectural Details:

We begin by noting that a 3 : 1 multiplexer is not necessary for the above architecture unless we add other functionalities to the circuit like the one described in Section 4.3. For executing the basic Möbius Transform a 2 : 1 multiplexer will also serve the purpose. We explain this with the first couple of registers but the same principle holds for the registers in the lower levels too. Note that the second register in its lifetime can only store two vector values \(A_{\text{top}}\) and \(A_{\text{bottom}}\) depending on how far the execution has reached in the
The process of the traversal of the recursion tree. Both of these are obtained from the butterfly operation on $A_0$ which resides on the register at the level just above. Thus the idea is to have a single 2:1 multiplexer separating the two registers, which takes as input the two outputs of the butterfly operation. When the 2nd level register would need to preserve state ($A_{\text{top}}$ or $A_{\text{bottom}}$), it can be done by appropriately setting the select signal of the multiplexer: for example to preserve $A_{\text{bottom}}$ we just need to set the select signal of the preceding mux so that it accepts the $A_{\text{bottom}}$ signal from the previous butterfly stage.

It remains to be seen how one can effectively set the multiplexer signals. In order to do that let us try to observe a small example. We will make use of a more general notation for the successive ANF vectors instead of just $A_{\text{top}}/A_{\text{bottom}}$, since we have to accommodate ANFs at different levels. We use the notation $A[\ell]_b$ to denote the ANF vector at some level of the recursion tree: the $\ell$ term in the square braces denotes the level of the ANF vector in the recursion tree, and the term $b$ which can be seen as a binary string or integer contains information about the coordinates over which the derivatives have been computed to obtain the function.

**Example 3.** For example, take the case when $n = 5$, $d = 2$. The ANF of original function $f(x_0, x_1, x_2, x_3, x_4)$, we denote by the notation $A[0]_{000}$: note that the subscript is a binary string of length $n - d$ (which is 3 in this example). This is because there are $n - d$ levels in the recursion tree, each obtained by taking derivative over some co-ordinate variable. The level 1 ANFs corresponding to the functions $f(0, x_1, x_2, x_3, x_4)$ and $\frac{\delta f}{\delta x_0} = f(0, x_1, x_2, x_3, x_4) \oplus f(1, x_1, x_2, x_3, x_4)$ are denoted by $A[1]_{000}$ and $A[1]_{100}$ respectively (thus $A_{\text{top}}$ and $A_{\text{bottom}}$ defined earlier are equal to $A[1]_{000}$ and $A[1]_{100}$ respectively in this new notation). Similarly the two level 2 functions obtained by applying the butterfly layer on $A[1]_{000}$ (by taking derivative over $x_1$) are denoted as $A[2]_{000}$ and $A[2]_{010}$. Similarly butterfly over $A[1]_{100}$ yields the two vectors $A[2]_{100}$ and $A[2]_{110}$.

Generalizing this: if $A[\ell]_b$ is the ANF vector at some level $\ell$ of the tree, then after applying the butterfly over the coordinate $x_\ell$, the two output vectors are denoted as
would be initialized with $A[0]_{[0]}$ and $A[1]_{[0]}$, where $e_1$ is the unit vector of length $n - d$ with 1 at the $t$-th position, eg. $e_0 = 100 \ldots 0, e_1 = 010 \ldots 0$ etc. At this moment, let us turn towards the example in Figure 6, where we have manipulated the select signals of each multiplexer so that the entire Möbius Transform is computed in $2^{n-d} = 8$ cycles, i.e. in each of the 8 cycles we get one partial truth table of size $\binom{d}{\ell} = 2^d = 4$. Initially the top-level register would be initialized with $A[0]_{[0]}$ and the remaining registers would stay uninitialized. In the 2 cycles following this, the select signals of each multiplexer, is set to zero so that, after this each layer register contains $A[\ell]_{[0]}$. Figure 6 shows the flow of data in each of the 8 cycles succeeding this.

We introduce an additional notation: let $S[\ell]_t$ be the select signal of the multiplexer between the registers at levels $\ell$ and $\ell + 1$ at time $t$. Which is to say that if $S[\ell]_t = 0$ and the ANF vector at level $\ell$ at time $t$ is $A[\ell]_b$, then at time $t + 1$, the ANF vector at level $\ell + 1$ is $A[\ell + 1]_b$, and if $S[\ell]_t = 1$ then the corresponding vector is $A[\ell + 1]_{b + e_\ell}$ (this can also be written as $A[\ell + 1]_{b + a_t}$ since by design the coordinates of $b$ at positions larger than $\ell$ are all 0). The two expressions can obviously be combined to give the single compact expression $A[\ell + 1]_{b + S[\ell]_t e_\ell}$ that caters for both values of $S[\ell]_t$. In Figure 6, we have done a series of assignments to the variables $S[\ell]_t$ (for $0 \leq \ell < n - d$ and $0 \leq t < 2^{n-d} - 1$) so that the vector at the bottommost level of the register chain is always $A[n - d]_b$ for all $0 \leq t < 2^{n-d}$. Since Excmob1 is connected to the bottommost register, this ensures that all the partial truth tables are faithfully computed and the circuit indeed computes the Möbius Transform of any five variable Boolean function of degree upto 2. However we are more interested in engineering the multiplexer signals for general values of $n, d$. To do so, equivalently consider the subscripts of the ANF vectors as integers, and return to the example in Figure 6. Initially all the subscripts at all the levels are zeros: thereafter we have the following subscripts assuming that all the $S[\ell]_t$’s are unknowns.

### Table 1: An example table of the subscripts at all levels, with respect to time.

<table>
<thead>
<tr>
<th>$t$</th>
<th>$\ell = 0$</th>
<th>$\ell = 1$</th>
<th>$\ell = 2$</th>
<th>$\ell = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4 S[0]_0</td>
<td>2 S[1]_0</td>
<td>$S[2]_0$</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>4 S[0]_0</td>
<td>4 S[0]_0 + 2 S[1]_1</td>
<td>2 S[1]_0 + S[2]_1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4 S[0]_0</td>
<td>4 S[0]_0 + 2 S[1]_1</td>
<td>2 S[0]_0 + 2 S[1]_1 + S[2]_2</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4 S[0]_0</td>
<td>4 S[0]_0 + 2 S[1]_1</td>
<td>4 S[0]_0 + 2 S[1]_1 + S[2]_3</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>4 S[0]_0</td>
<td>4 S[0]_0 + 2 S[1]_1</td>
<td>4 S[0]_0 + 2 S[1]_1 + S[2]_4</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>4 S[0]_0</td>
<td>4 S[0]_0 + 2 S[1]_1</td>
<td>4 S[0]_0 + 2 S[1]_1 + S[2]_5</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>4 S[0]_0</td>
<td>4 S[0]_0 + 2 S[1]_1</td>
<td>4 S[0]_0 + 2 S[1]_1 + S[2]_6</td>
</tr>
</tbody>
</table>

Note that the above follows since $e_\ell = 2^{n-d-1-\ell}$ as an integer, and therefore $b + S[\ell]_t \cdot e_\ell = b + S[\ell]_t \cdot 2^{n-d-1-\ell}$. We have already seen that for this to serve our purpose, the integer values of the last column of the above table should be 0 to 7. In other words we need $S[\ell]_t$’s from the set $\{0, 1\}$ which are solutions of the following system of equations over the integers.

$$
S[2]_0 = 1
$$
$$
2 S[1]_0 + S[2]_1 = 2
$$
$$
4 S[0]_0 + 2 S[1]_1 + S[2]_2 = 3
$$
$$
4 S[0]_0 + 2 S[1]_1 + S[2]_3 = 4
$$
$$
4 S[0]_0 + 2 S[1]_1 + S[2]_4 = 5
$$
$$
4 S[0]_0 + 2 S[1]_1 + S[2]_5 = 6
$$
$$
4 S[0]_0 + 2 S[1]_1 + S[2]_6 = 7
$$
Figure 6: Dataflow for the first 8 cycles.
It can be verified that the assignments to the $S[\ell]$,’s in Figure 6 satisfy the above equation system. We address the issue of the general case with the following theorem.

**Theorem 1.** Given the circuit **Polymob1** in Figure 5, in which each of the registers have been initialized with the ANF vectors $A[\ell]$, for all $0 \leq \ell \leq n - d$ of an $n$-variable Boolean function $f$ of degree less than or equal to $d$. Then it is possible to design the multiplexer signals $S[\ell]$, for $0 \leq \ell \leq 2^n - d - 2$ using logic gates efficiently, so that the circuit computes the Möbius Transform of $f$ in exactly $2^{n-d}$ clock cycles.

**Proof.** We essentially have to prove that we can engineer the multiplexer signals $S[\ell]$, efficiently so that the subscripts of the ANF vectors at the bottommost i.e. level $n-d$, at $t = 0 \rightarrow 2^n - d - 1$ are each equal to $t$ itself. Generalizing the observations made above, we need $S[\ell]$’s from the set $\{0, 1\}$ which are solutions of the following system of equations over the integers. Let $u := n - d$. Let $i$ be a sequence variable and set $j := u - 1 - i$ for conciseness, then we have

\[
\begin{align*}
2^n - 1 \cdot S[0] &+ 2^{n-2} \cdot S[1] + \cdots + 2^2 \cdot S[j] + \cdots + S[u-1]_{u-1} = w \\
2^n - 1 \cdot S[0] &+ 2^{n-2} \cdot S[1] + \cdots + 2^2 \cdot S[j] + \cdots + S[u] = u + 1
\end{align*}
\]

To solve the above equation system, observe that the right side always has a $u$-bit integer i.e. between 1 and $2^u - 1$. Not only that, the left side of each equation resembles the decimal expansion of a $u$-bit binary string. For example the LHS of the last equation is the decimal expansion of the $u$-bit binary string $S[0]_{2^u-u-1}, S[1]_{2^u-u}, \cdots, S[u-1]_{2^u-2}$. Thus a trivial way to solve the above equation system is to assign to the unknowns the values obtained from the binary representation of the corresponding integer in the right side. For example, since the binary form of $2^u - 1$ is the $u$-bit string of all 1s we can assign $S[0]_{2^u-u-1} = S[1]_{2^u-u} = \cdots = S[u-1]_{2^u-2} = 1$.

Thus we can see that a solution to the above equation system exists: however we will further show that each of the signals $S[\ell]$, can be efficiently generated using a reasonable amount of logic circuits. Using the method outlined above, we can immediately see that $S[u-1] = t + 1 \mod 2$ for all $t$. With some misuse of notation the above can be written as **NOT** ($t \mod 2$), i.e. if we have a decimal up-counter implementing $t$, then the $S[u-1]$, signal can be implemented by inverting the least significant bit of $t$. Similarly the sequence $S[u-2], t = 0, 1, 2, \ldots$ is the second lsb of the sequence $2, 3, 4, \ldots$, i.e. the second lsb of $t + 2$. For the general case, let us look at the $i$-th column from the end of the above equation system which has been highlighted in green. It can be seen that the sequence $S[j] = S[u-1-i], t = 0, 1, 2, \ldots$ is the $i+1$-th lsb of the sequence $(i+1), (i+2), (i+3), \ldots$, i.e. the $(i+1)$-th lsb of $t + i + 1$. Thus to construct all the signals $S[\ell]$, all we need are the following circuit elements:

1. A $u$-bit decimal up-counter for the variable $t$.
2. A series of $u$ incrementers (i.e. add by 1 circuits) to generate $t + 1$, $t + 2$, $\ldots$, $t + u$.

This proves the theorem statement.
Theorem 2. Furthermore it is possible to design a control circuit that generates all the select signals of the multiplexers in the Polymob\textsuperscript{1} circuit, incurring a total delay of 2\log_2(n - d) gates.

Proof. As noted in the proof of Theorem 1, the control circuit consists of a u-bit decimal up-counter (where \( u := n - d \)) for the variable \( t \) and a series of \( u \) incrementers. However constructing the whole incrementer leads to a wastage of gates since we are only interested in generating the \((i + 1)\)-th lsb of \( t + i + 1 \) for \( i = 0, 1, \ldots, u - 1 \).

Consider any \( p \)-bit string \( \vec{w} = w_{p-1}, w_{p-2}, \ldots, w_1, w_0 \) (note that the indexing with starts from right side in this definition). Define the \( p \)-variable Boolean function \( g_{p,\vec{w}} \) as follows

\[
g_{p,\vec{w}} = \left\{ \begin{array}{ll}
\prod_{w_i=0} x_i \vee \bigvee_{j=i+1}^{p-1} x_j & \text{when } p = 0 \text{ or } \vec{w} = 1^p.
\end{array} \right.
\]

For example the function \( g_{8,00011000} = (x_0 \vee x_2 \vee x_4) \cdot (x_1 \vee x_2 \vee x_4) \cdot (x_3 \vee x_4) \cdot x_5 \cdot x_6 \cdot x_7 \) and \( g_{3,111} = 1 \). Each product term begins with a min index that has 0 in the sting \( \vec{w} \).

In the first example, in \( \vec{w} \) indices 0,1,3,5,6,7 have 0. Then each min index is ORed with indices larger than it that have 1 in \( \vec{w} \). Further, if the length of \( \vec{w} \) is more than \( p \), we truncate \( \vec{w} \) to its \( p \) least significant bits. We will prove that the \((i+1)\)-th lsb of \( x + i + 1 \) is given by the Boolean function \( x_i \oplus g_{i,bin(i)} \), where \( bin(i) \) is the binary encoding of \( i \) using \( i \) bits, i.e. prepended with leading zeros when necessary. For small \( i \), this is easy to verify. Denoting \( x_j \) as the Boolean variable for the \( j \)-th bit of \( x \), we know that for \( i = 0 \), the 1st lsb of \( x + 1 \) is given by \( x_0 \oplus 1 = x_0 \oplus 0.0. \) For \( i = 1 \), the 2nd lsb of \( x + 2 \) can be computed thus: when we add with 2, i.e. the string “10” the 1st lsb location generates no carry. The result of addition in the 2nd lsb location is therefore \( x_1 \oplus 1 \oplus 0 = 1 \oplus 1 = x_1 \oplus 1 \).

For general values of \( i \), we proceed as follows. Let \( bin_i(i) = c_{i-1}, c_{i-2}, \ldots, c_0 \), where each \( c_j \in \{0,1\} \). Of these let the locations \( 0 \leq n_1 < n_2 < \cdots < n_s \leq i - 1 \) be such that \( c_{n_k} = 1 \) for \( k = 1 \) to \( s \), and the remaining \( c_j \)'s be 0. When adding two strings \( a, b \), the carry out bit in the \( j \)-th position can be written as \( maj(a_j, b_j, carry_{j-1}) \) (where \( maj \) is the majority function). We use two properties of this function: (1) \( maj(x, y, 0) = xy \) and (2) \( maj(x, y, 1) = x \lor y \). Figure 7 visually represents the process of addition by the constant \( i + 1 \). Using the above property of the majority function, the figure becomes self explanatory: however we still have to explain the symbols \( z_j \) for \( j = 1 \) to \( s \), which are the carry-outs for the position \( n_j \). By using the second property, we have

\[
z_1 = x_{n_1} \lor \prod_{k=0}^{n_1-1} x_k = \prod_{k=0}^{n_1-1} (x_{n_1} \lor x_k) = g_{n_1,i}(x)
\]

The above follows because of the Boolean identity \( A \lor BC = (A \lor B)(A \lor C) \), and \( i \) in the subscript of \( g \) is the truncation of \( bin_i(i) \) to the appropriate number of bits. Following the same logic we now have

\[
z_2 = x_{n_2} \lor \left( z_1 \cdot \prod_{k=n_1+1}^{n_2-1} x_k \right)
\]

\[
= (x_{n_2} \lor z_1) \cdot \prod_{k=n_1+1}^{n_2-1} (x_{n_2} \lor x_k) = \left( x_{n_2} \lor \prod_{k=0}^{n_1-1} (x_{n_1} \lor x_k) \right) \cdot \prod_{k=n_1+1}^{n_2-1} (x_{n_2} \lor x_k)
\]

\[
= \prod_{k=0}^{n_1-1} (x_{n_2} \lor x_{n_1} \lor x_k) \cdot \prod_{k=n_1+1}^{n_2-1} (x_{n_2} \lor x_k) = g_{n_2,i}(x)
\]

Following this chain of arguments, it is straightforward to show that \( z_s = g_{n_s,i}(x) \) and that the carry out of the \((i - 1)\)-th location is \( z_s \cdot \prod_{k=n_1+1}^{i-1} x_k = g_{i,bin_i(i)}(x) \). Thus it follows that the sum we are looking for is \( x_i \oplus g_{i,bin_i(i)}(x) \).
The expression for $g_{i, \text{bin}(i)}(x)$ naturally has the longest circuit depth for $i = u - 1 = n - d - 1$. The number of product terms in the expression is bounded by $n - d$. Therefore the depth required to construct the product terms, if the and gates were arranged in a binary tree like manner is around $\log_2(n - d)$. Furthermore, each collection of bracket containing terms that are OR-ed together can also have a maximum of $n - d$ terms, which implies each such term can also be constructed using $\log_2(n - d)$ depth. Putting this together we arrive at $2 \cdot \log_2(n - d)$. We also have to account for the decimal up-counter $t$ which counts from $0 \rightarrow 2^{n-d} - 1$ in steps of $1$. But it is well known in circuit theory that the maximum depth required in this up-counter is only $\log_2(n - d)$ (i.e. for the update bit of the msb flip-flop which is $t_{n-d-1} \circ \prod_{k=0}^{n-d-2} t_k$).

3.3.2 Representation of the ANF vector

So far we have avoided some of the finer operational details of the circuit to concentrate on the macro-level issues of dataflow through the circuit. One of the important topics we have not dealt with so far, is the issue of representing any degree-limited ANF coefficient set as a bit vector. The uncompressed ANF vector of an $n$-variable Boolean function has $2^n$ entries and mapping each coefficient into an array can be done canonically as explained earlier in Section 2 and further shown in Figure 2. For example the $x_0x_2$ term has coefficient 1: since the term can be written as $x_{101} := x_0^1 \cdot x_1^0 \cdot x_2^1$, the exponent vector 101 (in decimal) denotes the position where a one is inserted in the array. However when we are dealing with functions of a small degree $d$, coefficients of all terms of degree larger than $d$ are zero and so in order to accommodate the potentially $\binom{n}{d}$ non-zero coefficients we must be able to map them into an array of equal length, i.e. we need to decide which array location a given coefficient is going to reside in. This is important to decide for the following reasons:

1. We have left the issue of the ordering in Lines 1,2 in Algorithm 1 open. The ANF vector should be so represented so that constructing the vectors $A_{\text{top}}/A_{\text{bottom}}$ from $A_0$ should be efficient at all levels of recursion.

2. The ANF representation should be such that we can efficiently use $\text{Expmob1}$ at the leaf nodes of the recursion tree.

3. The circuit constructed for some $n = n^*$, $d = d^*$, should produce correct result when used for all $n < n^*$ and $d < d^*$, i.e. the circuit should work seamlessly for all smaller and lower degree Boolean functions.

Let $H(n, d)$ be the set of all binary strings of length $n$ whose hamming weight is less than or equal to $d$, where we will treat the elements of this set as both binary strings and integers. The goal is to construct a mapping $\chi_{n,d} : H(n, d) \rightarrow \left[0, \binom{n}{d} - 1\right]$, so that the coefficient of the $x^D$ term for any $D \in H(n, d)$ is placed at location $\chi_{n,d}(D)$ in the
compressed ANF vector. From the description of Expmob1 in Section 3, the following things can be seen

a) if we are using a butterfly circuit to construct the derivative with respect to any variable \( x_\ell \), then the two inputs to the circuit are the coefficients at \( x^D \) and \( x^{D+\ell} \). Without loss of generality, let us assume that the \( \ell \)-th bit of \( D \) is zero i.e. \( D \cdot \ell = 0 \).

b) The coefficient of \( x^D \) is copied as is from \( A_0 \) to \( A_{\text{top}} \) (or if we follow the terminology developed later: from \( A[\ell]_b \) to \( A[\ell+1]_b \)). The coefficients of \( x^D \) and \( x^{D+\ell} \) are added and copied to \( A[\ell+1]_{b+e_\ell} \).

c) If \( D \) be such that \( hw(D \oplus e_\ell) > d \), then this last addition is not necessary since the coefficient of \( x^{D+\ell} \) is 0 by assumption.

However note that the size of the vectors \( A[\ell]_b \) and \( A[\ell+1]_b \) are \((n-\ell)^i\) and \((n-\ell-1)^i\) respectively. So if any \( D \in H(n-\ell,d) \), then we ought to not only decide what \( \chi_{n-\ell,d}(D) \) would be but also in which locations of \( A[\ell+1]_b/A[\ell+1]_{b+e_\ell} \), the butterfly outputs would go to. It seems we need to determine a series of mappings \( \chi_{n-\ell,d} \), however we will see how only unified mapping will take care of our requirements. Let \( \chi_{n,d}(D) = y \) if \( D \) be the \( y \)-th largest integer with hamming weight less than or equal to \( d \). For example when \( n = 8, \ d = 2 \), we have \( \chi_{8,2}(u) = u \) for \( 0 \leq u \leq 6 \), and \( \chi_{8,2}(8) = 7 \), \( \chi_{8,2}(9) = 8 \), \( \chi_{8,2}(10) = 9 \), \( \chi_{8,2}(12) = 10 \) etc. The example makes it clear that the map \( \chi_{n,d} \) induces a co-lexicographical (colex) ordering among all \( d \) and lesser hamming weighted binary strings of length \( n \). In fact \( \chi_{n,d} \) acts as a rank function that assigns strings ranks in accordance with the colex ordering.

Note that we have \( f(x_0, x_1, x_2, \ldots) = x_0 \cdot \frac{\delta f}{\delta x_0}(x_1, x_2, \ldots) \oplus f(0, x_1, x_2, \ldots) \). Rewrite this as \( x_0 \cdot f_1(x_1, x_2, \ldots) \oplus f_2(x_1, x_2, \ldots) \), where \( f_1 = \frac{\delta f}{\delta x_0} \) and \( f_2 = (0, x_1, x_2, \ldots) \). Note that \( A[1]_{00\ldots} \) gets the ANF vector of \( f_2 \) and \( A[1]_{10\ldots} \) gets the ANF vector of \( f_1 \oplus f_2 \) after the butterfly operation. Therefore we have the following transitions

1. Algebraically \( f_2 \) is simply all terms of \( f \) with the terms containing \( x_0 \) removed. In terms of ANF, \( f_2 \) is therefore simply the terms contained at the indices of type \( 0 \parallel s \) (where \( s \) is any \((n-1)-\)bit string) in the uncompressed ANF vector. These are simply copied to the index \( s \) in \( f_2 \). In the compressed world, therefore, all entries at location \( \chi_{n,d}(0 \parallel s) \) of \( A[0]_{00\ldots} \) should go to location \( \chi_{n-1,d}(s) \) of \( A[1]_{00\ldots} \). However note that when expanded as integers, \( 0 \parallel s \) and \( s \) give rise to the same integer. Thus for ease of use \( \chi_{n-1,d}(s) \) can simply be denoted as \( \chi_{n,d}(0 \parallel s) \), and if we view the arguments of these functions as integers we do not need to define any \( \chi_{n-i,d} \) separately.

2. Similarly for \( f_1 \oplus f_2 \), in the uncompressed form, all terms at indices \( 0 \parallel s \) are added with terms at \( 1 \parallel s \) and copied to \( 0 \parallel s \). Thus in the compressed form we should add terms at locations \( \chi_{n,d}(0 \parallel s), \chi_{n,d}(1 \parallel s) \) (if \( 1 \parallel s \) has hamming weight less than or equal to \( d \)) of \( A[0]_{00\ldots} \) and copy it to location \( \chi_{n,d}(s) \) of \( A[1]_{00\ldots} \).

3. The same idea applies to all the levels of the recursion tree.

4. Note that in \( \chi_{n,d} \) all integers of hamming weight less than or equal to \( d \) are mapped to itself. Thus at the lowest leaves of the recursion tree, we can apply the canonical version of Möbius Transform as used in Expmob1.

We are yet to determine if the mapping \( \chi_{n,d} \) can be computed efficiently. The following lemma addresses this computational issue.

**Lemma 1.** For positive integers \( n, d \) with \( d \leq n \), and \( s \in H(n, d) \), let \( s = 2^{i_0} + 2^{i_1} + \cdots \), be the binary expansion of the integer \( s \), where \( i_0 > i_1 > \cdots \geq 0 \). Then we have

\[
\chi_{n,d}(s) = \left( \frac{i_0}{d} \right) + \left( \frac{i_1}{d-1} \right) + \cdots
\]
where we extend the definition of \( \binom{x}{y} \) as follows:
\[
\binom{x}{y} = \begin{cases} 
\sum_{i=0}^{y} \binom{i}{y} & \text{if } x \geq y, \\
2^x & \text{otherwise}.
\end{cases}
\]

**Proof.** As per the definition of \( \chi_{n,d} \), given \( s \) we have to count how many integers strictly less than \( s \) have hamming weight bound by \( d \). It is obvious that this number for any \( 2^m \) is simply \( \binom{m}{d} \), i.e. number of \( m \)-bit strings of hamming weight less than or equal to \( d \).

Hence the number of such strings in the range \([0, 2^n]\) is \( \binom{n}{d} \). The number of such integers in the range \([2^n, 2^n + 2^n]\) are strings which have 1 in the \( i_0 \)-th position and of hamming weight less than or equal to \( d - 1 \) in the last \( i_1 \) bits, and therefore equal to \( \binom{i_0}{i_1} \). Taking this argument forward for the successive \( i_2, i_3, \ldots \), we arrive at the required result.

\[\square\]

### 3.4 Helping Circuit Compiler synthesize faster

The above lemma shows that the map \( \chi_{n,d}(\cdot) \) can be efficiently computed. However for ease of synthesis, one may want to precompute and store a few of the above values to help the circuit compiler construct an optimal circuit especially when \( n \) becomes larger.

One could store all values of \( \chi_{n,d}(s) \), \( \forall s \in H(n, d) \) for this purpose, but note that the arguments “\( s \)” of this function are not exactly contiguous integers and thus we would not be able to store the function table in any continuous memory structure like an array. We could employ a hash table for this purpose, however designing a good collision free hash function for this purpose is an open problem.

Another method we could employ is to store the adjacency matrix of a graph that we describe below. Note that at the \( \ell \)-th recursion step, we need access to locations \( \chi_{n,d}(0 \parallel s) \), \( \chi_{n,d}(1 \parallel s) \) of the current register, where \( s \) is an \( n - \ell - 1 \) bit string. Imagine the graph \( G = (V, E) \), in which the elements of \( \{0, \binom{n}{d} - 1\} \) are nodes and each node \( \alpha \) in this set is connected with at most \( n - d \) types of edges to at most \( n - d \) neighbors. An edge of type \( \ell \), (for \( 0 \leq \ell < n - d \)) connects \( \alpha \) to \( \beta := \chi_{n,d} \left( \chi_{n,d}^{-1}[\alpha] \oplus \epsilon_\ell \right) \) if \( \text{hw}(\beta) \leq d \) and unconnected otherwise. This is helpful because at step \( \ell \) of the recursion tree, if \( \alpha = \chi_{n,d}(0 \parallel s) \) then the two inputs to the butterfly circuit can be equivalently seen as the wires at locations \( \alpha \) and \( \beta \), as it can be easily deduced that \( \beta = \chi_{n,d}(1 \parallel s) \).

One can now define the reduced adjacency matrix \( AM \) of size \( \binom{n}{d} \times (n - d) \) such that
\[
AM[\alpha, \ell] = \begin{cases} 
\chi_{n,d} \left( \chi_{n,d}^{-1}[\alpha] \oplus \epsilon_\ell \right), & \text{if } \text{hw}(\chi_{n,d}^{-1}[\alpha] \oplus \epsilon_\ell) \leq d \\
0, & \text{otherwise}
\end{cases}
\]

Thus the \( \ell \)-th recursion step can be re-written from:

- For all \( n - \ell - 1 \) bit strings \( s \) with \( \text{hw} \leq d \)
  1. \( A[\ell + 1][b](\chi_{n,d}(s)) \leftarrow A[\ell][b](\chi_{n,d}(0 \parallel s)) \)
  2. If \( \text{hw}(\chi_{n,d}(1 \parallel s)) \leq d \):
     \( A[\ell + 1][b+\epsilon_\ell](\chi_{n,d}(s)) \leftarrow A[\ell][b](\chi_{n,d}(0 \parallel s)) \oplus A[\ell][b](\chi_{n,d}(1 \parallel s)) \)
  3. Else \( A[\ell + 1][b+\epsilon_\ell](\chi_{n,d}(s)) \leftarrow A[\ell][b](\chi_{n,d}(0 \parallel s)) \)

To the following equivalent form that uses the \( AM \) matrix:

- For \( \alpha = 0 \) to \( \binom{n-\ell-1}{i_d} - 1 \)
1. \( A[\ell + 1]_{b}(\alpha) \leftarrow A[\ell]_{b}(\alpha) \)

2. If \( AM[\alpha, \ell] \neq 0 \)
   \( A[\ell + 1]_{b+\varepsilon}(\alpha) \leftarrow A[\ell]_{b}(\alpha) \oplus A[\ell]_{b}(AM[\alpha, \ell]) \)

3. Else \( A[\ell + 1]_{b+\varepsilon}(\alpha) \leftarrow A[\ell]_{b}(\alpha) \)

Using the 2nd description is much easier to write an RTL code for describing the Möbius Transform circuit in any hardware description language. Additionally, the circuit compiler also outputs the optimized netlist faster. In Appendix A, we outline an algorithm to generate \( AM \) efficiently in polynomial time.

3.5 Further Utilities

Finding truth table when some variables are fixed to constants: Often one is interested to find solutions to a system of equations in which a fraction of variables has been fixed to some given constant. Since our strategy in solving a system of polynomial equations is to compute the OR the respective truth tables (see Section 1.1), we would therefore be interested to find the truth table of a polynomial when some variables are fixed. To do this, we could either first simplify the given Boolean polynomial by fixing some individual variables to constants and then using the corresponding reduced ANF vector as input to the circuit, after appropriately zero padding it. However this naturally requires additional computations, i.e. to simplify the original polynomial in the first place.

However if the \( t \) variables to be fixed are lexicographically the first \( t \) variables of the system for any \( t \leq n - d \) then we can do better. We see from Figure 6, that at the \( i \)-th stage the ANF vector at the bottom most register is \( A[n - d]_{bin_{n-d}(i)} \). As a result the truth table output after the Expmob1 circuit is \( f(bin_{n-d}(i), \ldots) \), i.e. in which the first \( n - d \) bits of \( f \) is already set to \( bin_{n-d}(i) \). Thus one can use this method to extract the truth tables when the number of variables to be fixed are less than \( n - d \). Note that one may think that one would need to wait exactly \( i \) cycles to obtain the tables, which can be counterproductive if \( i \) is large. Note that Figure 6 already starts with \( A[3]_{100} \) in the bottom most register at \( t = 0 \), as in the previous 3 cycles, i.e. \( t = -3, -2, -1 \), the corresponding \( S[i]_{s} \)'s were all set to zeros. Instead if all of these were set to 1, then at \( t = 0 \), the signal in the bottom most register would be \( A[3]_{111} \), and we would get the truth table of \( f(1, 1, 1, \ldots) \) from the Expmob1 circuit. Similarly by adjusting the initial select signals we can get the truth table where the first \( (n - d) \) variables are fixed to any arbitrary constant in the first cycle itself.
Table 2: Results for $d = 2, 3, 4$ for the Polymob1 circuit. Power reported at 1 GHz.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$d = 4$</th>
<th>$d = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Area (GE)</td>
<td>$T_{cr}$ (ps)</td>
</tr>
<tr>
<td>8</td>
<td>2,570</td>
<td>1.07</td>
</tr>
<tr>
<td>9</td>
<td>3,535</td>
<td>44.87</td>
</tr>
<tr>
<td>10</td>
<td>5,780</td>
<td>46.77</td>
</tr>
<tr>
<td>11</td>
<td>9,040</td>
<td>66.95</td>
</tr>
<tr>
<td>12</td>
<td>13,525</td>
<td>75.95</td>
</tr>
<tr>
<td>13</td>
<td>20,024</td>
<td>61.74</td>
</tr>
<tr>
<td>14</td>
<td>28,754</td>
<td>73.61</td>
</tr>
<tr>
<td>15</td>
<td>40,706</td>
<td>73.18</td>
</tr>
<tr>
<td>16</td>
<td>56,683</td>
<td>79.71</td>
</tr>
<tr>
<td>17</td>
<td>77,402</td>
<td>78.07</td>
</tr>
<tr>
<td>18</td>
<td>102,916</td>
<td>90.26</td>
</tr>
<tr>
<td>19</td>
<td>134,835</td>
<td>93.46</td>
</tr>
<tr>
<td>20</td>
<td>174,268</td>
<td>101.48</td>
</tr>
</tbody>
</table>

3.6 Synthesis Results

For the actual synthesis, we can do some optimizations as follows: In Figure 6, we can see that the topmost register of size $n^d$ essentially holds a constant value throughout the lifetime of the Möbius Transform operation, and as such it can be removed from the circuit if the ANF signal is assumed as available on the input wires to the circuit. Using this tweak, we again synthesized the Polymob1 circuit using the Nangate 15 nm open cell library for various values of $n \in [8, 20]$ and $d \in [2, 4]$. Note that the values of $d$ chosen apply to a number of instances of cryptanalytic problems known in literature as mentioned in the introduction.

The results are presented in Table 2. As stated earlier, the circuits were synthesized to minimize the total critical path, which allows us to clock them using higher frequencies. The minimum time $T_{min}$ taken to compute the transform is calculated as $[2n-d + (n-d)] \cdot T_{cr}$ where $T_{cr}$ is the critical path of the circuit. Since the depth of the circuit (and therefore to some extent also $T_{cr}$) increases logarithmically and the number of cycles increases exponentially with respect to $(n-d)$, some interesting tradeoffs can be observed: for example to compute the Möbius Transform of quadratic Boolean functions one may either use the circuit for $d = 2, 3$ or 4. Because of the exponential dependence on $n-d$, the total physical time taken to compute the transform undoubtedly decreases with increase in $d$: however it has to be paid for with larger circuit area and energy consumption. Furthermore, it can also be seen that for the range $8 \leq n \leq 14$ for which we have experimental data for the Expmob1, Expmob2, Polymob1 circuits, the energy consumed by the Polymob1 circuits is larger than the corresponding Expmob1/Expmob2 circuits. This is to be expected primarily because Polymob1 is essentially a serialized circuit that performs the transform using exponential amount of time in $(n-d)$ whereas Expmob1/Expmob2 either take constant or linear time to execute.
3.7 Energy Analysis of the Polymob1 Circuit

We will try to construct an analytical model of the energy consumed in the circuit. It is important to recall that two components are primarily responsible for the amount of energy dissipated in CMOS circuits:

- Dynamic power dissipation due to the charging and discharging of load capacitances and the short-circuit current. Each 0 → 1 / 1 → 0 transition contributes to the dynamic dissipation, and hence this component varies directly as the clock frequency.

- Static power dissipation due to leakage current and other current drawn continuously from the power supply. This type of power is generally not dependent on the frequency of the clock driving the circuit.

Thus the total energy dissipation can be written as \( E_{\text{total}} = E_{\text{dynamic}} + E_{\text{static}} \). Since static power is independent of clock frequency we can write \( E_{\text{static}} = P_{\text{static}} \cdot T_d \), where \( P_{\text{static}} \) is the static power consumption and \( T_d \) is the total physical time taken to compute the Möbius Transform.

It is not altogether unreasonable to assume that \( P_{\text{static}} \) is related to the circuit area. In this respect let us look at the combinatorial and sequential elements of the circuit separately. The total number of flip-flops in the circuit can be estimated easily to be equal to \( F(n, d) = \sum_{i=1}^{n-d} \binom{n-i}{d} \). The total static power due to this component can be estimated to be around \( P_{\text{static,seq}} = F(n, d) \cdot \alpha_s \), where \( \alpha_s \) is the leakage power due to a single flip-flop. To estimate the static power due to the combinatorial portion as an expression is trickier since as \( n \) increases the compiler does various optimizations (which involves including in final netlist, cells with higher number of inputs and drive strength) to reduce the combinatorial circuit area. As a result, as \( n \) increases, the ratio between (a) the actual combinatorial circuit area as reported by the compiler and (b) the area estimated by counting the number of two input multiplexers and xor gates, continually decreases. While the combinatorial static power \( P_{\text{static,comb}} \) can still be estimated as the product of some constant \( \alpha_c \) and the combinatorial area \( A_c \), there is no good way of estimating \( A_c \) in terms of \( n, d \). As a loose upper bound we can estimate \( A_c \) by counting the number of multiplexers and xor gates, but this overestimates the static power as \( n \) increases.

Regarding the dynamic power, the lion’s share of the consumption is due to the register writes. In our simulation results for \( (n, d) \in [8, 20] \times [2, 4] \) the dynamic power contribution due to the combinatorial portion of the circuit has been less than 10% of the total dynamic power. Hence \( P_{\text{dynamic}} \approx P_{\text{dynamic,seq}} = F(n, d) \cdot \beta_s \), for some constant \( \beta_s \). Combining the above three expressions the energy consumed in the circuit can be written as:

\[
E_{\text{Polymob1}} = T_d \cdot (F(n, d) \cdot \alpha_s + A_c \cdot \alpha_c + F(n, d) \cdot \beta_s)
\]  

(3)

We can estimate \( A_c = F(n, d) \cdot A_{\text{mux}} + (F(n, d) + d \cdot 2^{d-1}) \cdot A_{\text{ xor}} \), where \( A_{\text{mux}} / A_{\text{ xor}} \) are the silicon areas of the 2 input multiplexer/xor gate respectively. The values of \( \alpha_c, \alpha_s, \beta_s \) can be estimated using power simulation for \( n = 8, d = 2 \), or any other data point.

At higher frequencies: The left side plots in Figure 8 show the comparison of the actual energy consumption as reported by the power compiler and that estimated by the algebraic expression in Equation (3), when the clock frequency is 10 MHz. The purple plot which represents the figures obtained by Equation 3, clearly overestimates the energy. However consider the case when the clock frequency is increased to 1 GHz: a) since the static power is independent of the clock frequency, it remains the same, and b) the value of \( T_d \) decreases by a factor of 100. Both these factors ensure that the contribution of the static power to the total energy consumption decreases 100-fold. The contribution of the dynamic power however remains the same, since this is proportional to the clock frequency, i.e. \( P_{\text{dynamic}} \) itself increases 100-fold so that \( P_{\text{dynamic}} \cdot T_d \) remains constant. Then the total energy is almost entirely dynamic in nature, and so the numbers estimated by Equation (3) more closely matches the actual consumption as shown by the right side plots in Figure 8.
Figure 8: Energy plots for the Polymob1 circuit at clock frequency 10 MHz, 1 GHz. Note that “actual consumption” refers to the energy obtained after timing simulation on the synthesized circuit.
4 Solving Polynomial equations of degree $\leq d$

One of the primary uses of the Möbius Transform circuit is in finding solutions of a system of Boolean polynomials bounded by some algebraic degree $d$. Recapitulating, if $f_1$, $f_2$, $f_3$, $\ldots$, $f_m$ are the $m$ polynomials whose common root we are aiming to find, then the root $r \in \{0, 1\}^n$ is an $n$-bit vector which simultaneously satisfies $f_1(r) = f_2(r) = \cdots = f_m(r) = 0$. We can combine the above in a single equation:

$$\bigvee_{i=1}^{m} f_i(r) = 0$$

In other words, we take the truth tables of each $f_i$ and cumulatively compute the logical OR of them. The common root(s) will be indices at which the vector of cumulative OR of the tables have 0 in them. However note that the Möbius Transform circuit we have constructed outputs the truth table in parts. We have seen that when at the lowest register the ANF vector is $A[n - d]b$, then the circuit outputs the truth table of the function $f(b, x_{n-d}, \ldots, x_{n-1})$, i.e. when the first $n - d$ bits have been set to the constant $b$. With this information let us begin to see circuit configurations that compute the root of a system of equations.

4.1 Polysolve1

Let’s say we have $m$ Boolean equations $f_i$ we need to solve. We begin by having $m$ copies of the Möbius Transform circuit in parallel, each for one of the polynomials $f_i$. At the $b$-th step, we have the truth tables $f_i(b, \ldots)$ output from each of the circuits. We then have a layer of OR gates to compute the logical disjunction of all the truth tables. After we have done this we have the combined truth table vector of length $2^d$ bits, whose zeroes give us the roots of the equation. The following cases may occur

- The vector is the all 1 bit-string. This indicates that there is no root of the underlying system in which the first $n - d$ bits are set to the constant $b$.
- The vector contains a single 0 at some position $t$, whose binary encoding is given by $bin_d(t)$. In this case the root of the system of equations is $b \parallel bin_d(t)$.
- The vector contains a multiple 0s, which indicates that there are multiple roots of the system beginning with $b$. We may wish to find all such roots, or any one of them. For the moment let us concentrate on finding any one of them.

If the task is to find only one such root, the most efficient way to find this would be a priority encoder, that will encode to binary the first occurrence of zero in the $2^d$-bit string. The circuit is described pictorially in Figure 9a. We describe some micro-level details of the circuit below:

**OR Network**: In order to compute the disjunction of $m$ vectors of length $2^d$, each, it is obvious that we need $(m - 1) \cdot 2^d$ number of 2-input OR gates. However we can ensure that the network has a total latency bounded by $\lceil \log_2 m \rceil$ OR gates by arranging the gates in inverted binary tree like manner, in which each level would contain around half the gates contained in the previous level. For example if $m = 8$, the first level would have a total of 4 OR gates of width $2^d$ bits, the next level 2, and the final level a single gate. This makes the total latency of the network equal that incurred in three OR gates.

**Priority Encoder**: For a $2^d \rightarrow d$ bit priority encoder, the functionality can be simply described as a look-up table if $d$ is small enough. For larger $d$, we can also use recursive
description of the encoder functionality. In both cases, the critical path in the encoder is known to be proportional to \(d\) gates.

**Circuit Area:** If we do away with the first level register in the Polymob1 circuit the total number of scan flip-flops required for the successive registers in the \(m\) Polymob1 instances is 
\[
m \cdot \sum_{i=1}^{n-d} (\frac{n}{i^2}) < m \cdot (n - 1)^{d+1} < m \cdot n^{d+1}.
\]
It is not difficult to work out that the total number of xor gates required is 
\[
m \cdot \sum_{i=1}^{n-d} (\frac{n}{i^d}) - 1.
\]
Since the area of a 2-input xor gate is much less than that of a scan flip-flop the total area can be loosely upper bound by 
\[
m \cdot n^{d+1}.
\]
The remaining circuit elements contribute \(md \cdot 2^{d-1}\) xor gates (for the Exmob1 circuits), \((m-1) \cdot 2^d\) OR gates (for the OR network) and the area required for the priority encoder (this will be proportional to \(2^d\)). For small \(d\), this can be ignored with respect to \(m \cdot n^{d+1}\).

**Total Critical Path:** As shown in Figure 9a, the total critical path in this architecture is due to the combination of Exmob1, the OR network and the encoder (call this \(\tau_A\)). Since we have seen that each Möbius Transform takes around \(2^{n-d}\) clock cycles to output all the truth tables, this implies that the circuit will take at least \(\tau_A \cdot 2^{n-d}\) amount of physical time to solve the system of equations.

**AT product:** We are looking to find \(AT\) when \(AT\) when \(A\) is polynomially upper-bounded in \(n\). For solving \(n\) equations in \(n\) variables of degree \(d\), we have \(A \in O(n^{d+2})\) as required. Since \(T = 2^{n-d}\), we have 
\[
AT = n^{d+2} \cdot 2^{n-d}.
\]

### 4.2 Polysolve2

In order to break up the long chain of combinatorial circuitry after the Möbius Transform computations one could install pipeline stages in between them as shown in Figure 9b. The introduction of the pipeline stages requires only \(m+1\) registers of size \(2^d\) bits each (\(m\) for Reg1 and one more for Reg2 as shown in Figure 9b) and reduces the delay caused due to the long chain of combinatorial elements, and increases the computation time by only two cycles. However the breaking up of this combinatorial path means that the critical path in the circuit will now most likely be due to the series of \(u = n - d\) incremenetor circuits required for generating the select signals for the multiplexers in the Möbius Transform circuit or the optimized version of it described in Theorem 2.

Note that (as we shall see shortly) for smaller values of \(d\) that we report in this paper (i.e less than 4), there is not much difference between the critical paths of Polysolve1 and Polysolve2. For smaller values of \(d\), the total critical path \(\tau_A\) is not very high and the circuit compiler effectively balances out various parts of the netlist so that the total critical path of the Polysolve1 circuit is comparable with the Polysolve2 circuit. However as \(d\) increases, Polysolve2 performs much better with respect to total circuit latency.

### 4.3 Polysolve3

So far Polysolve1/Polysolve2 have been focused to find only a single root in the series of partial truth tables generated from each \(f_i\). However some applications may need the underlying hardware accelerator to find all the roots of a given equation system. There are a few solutions to the above problem we could consider. First, the circuit may choose to communicate the disjunction of partial truth tables back to the processor, without applying the encoder. The root would then be extracted by the processor using its own instruction set architecture. Second, instead of a priority encoder, the 2nd register (Reg 2) in Figure 9b, could be additionally equipped with bitwise shift functionality. After the disjunction of the \(m\) truth tables is loaded on to it, the bits would be shifted out serially with another counter maintaining the index of the bit shifted out. Now if one of the shifted out bits is zero, then the index counter can be used to construct the root. However this
would require freezing the operations of the Möbius Transform circuit for exactly $2^d$ cycles, i.e. the Möbius Transform circuit does not produce another partial truth table till the processing of the current table is completed. This implies that the underlying registers of the Polymob1 circuit would now actually need a 3:1 multiplexer preceding it to help in freezing the dataflow. However this increases the number of cycles required to execute the operation by a factor of $2^d$ i.e. from $2^{n-d}$ to $2^d \cdot 2^{n-d} = 2^n$ cycles.

However the solution we propose here will require exactly $R + 2^{n-d}$ cycles, where $R$ is the total number of roots of the underlying equation system. The main issue arises when the disjunction of truth tables contains multiple zeros. In that case a priority encoder only fishes out the location of the zero which is numerically smallest. However consider the event when this actually happens: using the inverse of an encoder i.e. a decoder, one can convert the encoded vector $V$ back to a $2^d$ vector of hamming weight one, with one at the $V$-th location. We then OR this vector with the current vector in Reg 2 and update it in the next cycle. The updated vector has one less 0 than the original vector in Reg 2. If this is now the all one vector then there are no more roots to fish out, else we repeat the process to decrease the number of zeros in Reg 2 by one, till it has the all one vector.

**Example 4.** If $d = 4$, and the OR of the truth tables is $T_0 = 1011 1111 1111 0111$, then the priority encoder in the first cycle outputs 0001 which is the index of the first 0. The decoder outputs $D_0 = 0100 0000 0000 0000$, which after OR with $T_0$ becomes $T_1 = T_0 \lor D_0 = 1111 1111 1111 0111$, and has one less zero than $T_0$, and is written back to Reg2. In the next cycle we get the next root 1100 from the priority encoder which decodes to $D_1 = 0000 0000 0000 1000$. Therefore we have $T_2 = T_1 \lor D_1 = 1111 1111 1111 1111$ which is now the all one string.

During this time the Polymob1 pipeline will have to be frozen (and thus a 3:1 mux functionality is needed in the Polymob1 circuit), and it is not difficult to see that if each of the $i$ disjunction of the partial truth tables (for $i \in [0, 2^{n-d} - 1]$) has $r_i$ roots (with $\sum r_i = R$) then the $i$-th step will execute for exactly $r_i + 1$ cycles. To see why, note that there are two scenarios: (a) when the disjunction of the partial truth tables is the all one string, it means that that the pipeline immediately moves on to the next partial truth table and thus only spends one cycle here, and (b) when the string has one or more than one zero the mechanism reduces the number of zeros in the string by one every cycle, as explained above, till the all one string is reached. This needs $1 + r_i$ cycles. Therefore the total number of clock cycles required is $\sum_{i=0}^{2^{n-d}-1} (1 + r_i) = 2^{n-d} + \sum_{i=0}^{2^{n-d}-1} r_i = R + 2^{n-d}$. The circuit is described diagrammatically in Figure 9c.

**Circuit Area:** The only significant addition to the Polysolve1 circuit is the additional
2:1 muxes before each of the scan flip-flops in the Polymob1 circuit (thus achieving 3:1 multiplexer functionality). Thus the circuit area is now bound by \( m \cdot n^{d+1} \) scan flip-flop and 2:1 muxes.

**Total Critical Path:** As \( n \) increases, the critical path is expected to be due to the select signal generation of the Polymob1 circuit which has been shown to be proportional to \( 2 \log_2 (n-d) \). If the underlying clock signal has this period then the total physical time taken to solve the system will be proportional to \( 2 \log_2 (n-d) \cdot (2^n - d + R) \approx 2 \log_2 (n-d) \cdot 2^n \).

**AT product:** For solving \( n \) equations of degree \( d \), we have that \( A \in O(n^d + 2) \) is again upper-bounded. So we have \( AT = n^{d+2} \cdot (R + 2^{n-d}) \).

### 4.4 Synthesis Results

In Figure 10, we present synthesis figures for the three solvers for the range of values of \( n \in [8, 20] \) and \( d \in [2, 4] \). We have let \( m = n \), so that the circuits directly output the roots of the underlying equation system. The figure shows the silicon area, theoretically the lowest physical time taken (which is the product of the critical path and the number of clock cycles required, which would occur if the circuit were to be clocked at the critical path) and the total energy consumed. The red and the blue curves for each plot (denoting the Polysolve1 and Polysolve2 circuit respectively) are almost coincident, which implies that for the range of values of \( d \) we have chosen, the circuits have similar performance metrics. The figures of time and energy for the Polysolve3 circuit has been computed assuming that \( R = n \). It is evident that the Polysolve3 circuit is much larger than the corresponding Polysolve1/Polysolve2 circuits since we need to use additional 2:1 multiplexer in each of the internal registers of the Polymob1 circuit to periodically freeze the dataflow. Detailed results are available in Tables 8, 9 in Appendix E.

### 5 Solving for energy efficiency: Polysolve4

In the previous section we used \( n \) instances of the Möbius Transform circuit when there are \( n \) equations to solve. While this will certainly extract the root faithfully, it will do a lot of redundant calculations. Imagine a scenario where \( n = 50 \), and there is a system of 50 equations such that the common solution-space of the first 10 equations is only around 90-100 vectors in \( \{0, 1\}^{50} \). In that case computationally it makes much more sense to do the following: a) Take each root \( r \in \) the common solution-space of the first 10 equations, and b) evaluate \( f_i(r) \) for \( 11 \leq i \leq 50 \): if all these \( f_i(r) \)'s evaluate to 0 then \( r \) is a root of th equation system. This amounts to evaluating around 40 Boolean functions over 90-100 points. So if we have only 10 Möbius Transform circuits that extract the common roots of the first 10 equations then we can do the functional evaluations of the remaining 40 equations over this common solution-space in software. This is exactly the approach followed in [BCC10,Din21]. On the other hand if we used the Möbius Transform circuit 50 times, mathematically it is equivalent to evaluating the remaining 40 equations over all the \( 2^{50} \) points of its input space. Naturally this amounts to a lot of wastage in computational power and energy.

In [BCC13], the authors had shown that computationally the optimum solution is to use the hardware backend to solve the first \( \mu \approx \log_2 n \) equations. This reduces the cardinality of the solution-space to small enough, so that the remaining \( n - \mu \) Boolean functions can be evaluated over the points of this solution-space in software and hardware. In [Din21, Appendix B], the author suggested that such evaluation can be done using a Horner-like method in batches. Both the above are software based solutions: in this paper we present a purely hardware based solution that achieves this.
A root expander RE$(n,d)$ is a circuit component that takes any $n$-bit vector and generates the values of all the degree $d$ monomials in $n$-variables. Thus essentially it is a map from $\{0,1\}^n \rightarrow \{0,1\}^{\binom{n}{d}}$. For example RE$(4,3)$ over the vector $(x_0, x_1, x_2, x_3) = (1, 0, 1, 1)$ will generate the monomial values: const value $= 1$, $x_0x_1 = 0$, $x_0x_2 = 1$, $x_0x_3 = 1$, $x_1x_2 = 0$, $x_1x_3 = 0$, $x_2x_3 = 1$, $x_0x_1x_2 = 0$, $x_0x_1x_3 = 0$, $x_0x_2x_3 = 1$, $x_1x_2x_3 = 0$. It can be deduced that the value of each such monomial takes one AND gate to evaluate (eg. $x_0x_1x_2$ can be calculated by AND-ing $x_0, x_1$ and $x_2$ etc), and so the total hardware overhead is $\binom{n}{d} - n$ AND gates. The outcome of such a circuit is thus an expanded root $r \in \{0,1\}^{\binom{n}{d}}$.

5.2 Dot-Product

A dot-product essentially takes the ANF vector of a Boolean function and evaluates it over an expanded root $r$. It can be shown that a simple dot-product over $GF(2)$ will achieve...
this. For example when \( n = 4, d = 2 \), consider the function \( f = 1 \oplus x_0 \oplus x_2 \oplus x_3 \). Using the mapping \( \chi_{4,2} \), the corresponding vector description for this function is \( \mathbf{v} = 1011 \ 0001 \ 001 \). For the point \( \mathbf{r} = (x_0, x_1, x_2, x_3) = (1, 0, 1, 1) \), we have, using the \( \chi_{4,2} \) map the corresponding description of the expanded root \( \mathbf{r} = 1111 \ 0001 \ 110 \). The dot-product \( \mathbf{r} \cdot \mathbf{v} = 0 \), gives the evaluation of \( f \) at the point \( \mathbf{r} \). Thus each dot-product will need \( \binom{n}{d} \) AND gates and \( \binom{n}{d} - 1 \) XOR gates to compute (in practice the product is usually computed using \( \binom{n}{d} \) NAND gates and a similar number of XNOR/XOR gates). While the AND/NAND are done in parallel, the XNOR/XOR gates can of course be arranged in a tree like structure giving a depth of \( \log_2 \binom{n}{d} \approx d \log_2 n \) gates.

### 5.3 Circuit Architecture for Polysolve4

The circuit architecture for Polysolve4 is shown in Figure 11. The core of the circuit is a Polysolve3 instance with only \( \mu < m \) instead of \( m \) equations. This core only outputs the common solutions \( \mathbf{r} \) of the first \( \mu \) equations \( f_1, f_2, \ldots, f_{\mu} \). Whenever such a root is output it is introduced to the next pipeline: the expander first expands it to \( \mathbf{r} \) of size \( \binom{n}{d} \). Then a simultaneous dot-product is done with the \( m - \mu \) equations \( f_{\mu+1}, f_{\mu+2}, \ldots, f_m \), which as we have seen, evaluates the \( f_i \)'s at the root \( \mathbf{r} \). If all these evaluations are 0, then so will the OR of them which is calculated thereafter. If this OR of the evaluations is 0, the root \( \mathbf{r} \) is output by the circuit as a valid root of the entire equation system.

**Circuit Area:** Over and above Polysolve3 the circuit which has area bound by \( \mu \cdot n^{d+1} \) flip-flops/XOR gates we have overheads due to \( m - \mu \) dot-product circuits, and expander circuit and an additional \( m - \mu \) gates for the second OR-tree. Since each dot-product circuit also takes \( \binom{n}{d} \in O(n^d) \) gates, the area requirement is still \( O(m \cdot n^{d+1}) \).

**Total Critical Path:** A long combinatorial path is added in the circuit due to the combination of the dot-product and OR-tree which we have seen is around \( \log_2 \binom{n}{d} \approx d \log_2 n \) XOR gates + \( \log_2 (m - \mu) \) OR gates.

**AT product:** Since \( A \in O(n^{d+2}) \) is again upper-bounded. We will see in Lemma 2 that the number of common roots \( R \) of the first \( \mu \) Möbius Transform circuits is around \( 2^{n-\mu} \). Therefore we have \( AT = n^{d+2} \cdot (2^{n-\mu} + 2^{n-d}) \).

### 5.4 Energy Analysis

If we conduct all experiments at high enough frequencies, the static portion of the energy consumption becomes less of an issue, and so throughout the experiments we have kept...
the clock frequency at 1 GHz. Before we proceed let us look at the following lemma:

**Lemma 2.** Let $f_1, f_2, \ldots, f_\mu$ be identically and independently distributed balanced Boolean functions of $n$ variables each. Then the expected cardinality of the solution space of the system of equations $f_1 = f_2 = \cdots = f_\mu = 0$ is $2^{n-\mu}$.

**Proof.** Given any $r \in \{0, 1\}^n$, the probability that it is a root of any $f_i$ is $\frac{1}{2}$, since all the polynomials are balanced. Assuming independence, the probability that $r$ is a common root of all the $f_i$'s is $2^{-\mu}$. Using linearity of expectation, the expected cardinality of the solution space is thus $2^{n-\mu}$.

In the above lemma we have assumed that Boolean functions are on average balanced. However note that a randomly sampled Boolean function need not be balanced with high probability. In fact this probability can be shown to be equal to approximately $\sqrt{\frac{1}{(\pi \cdot 2^{n-1})}}$. However using Stirling’s approximations it can be shown that $\Pr[|wt(f) - 2^{n-1}| \leq \sqrt{2^{n-1}}] > 0.8$ (for $f$ sampled uniformly randomly from the set of $n$-variable Boolean functions), which means that Boolean functions are close to balanced with high probability. Note that it is more probable that a randomly selected lower degree function is balanced. For example, [CB10, Theorem 2.3] tells us that a randomly selected quadratic function is balanced with probability more than 0.4, both when $n$ is even and odd.

Let us again note the construction of the Polysolve4 architecture. There is a Polysolve3 core that caters to only $\mu$ equations. The number of clock cycles that the Polysolve3 core would take to output its solutions space (i.e. of the first $\mu$ equations) is $R + 2^{n-\mu} \approx 2^{n-\mu} + 2^{n-d}$ by Lemma 2. Since each root is filtered by the dot-product immediately, this is also the time required by the Polysolve4 circuit to complete its operations.

Let $P_{\text{mob}}$, $P_{\text{dp}}$ be the dynamic power consumed by each Polymob1 and dot-product circuit respectively, then we can conclude that at high enough clock frequencies, the energy consumption is proportional to

$$E(m, \mu) = (\mu \cdot P_{\text{mob}} + (m - \mu) \cdot P_{\text{dp}} + C) \cdot (2^{n-\mu} + 2^{n-d}) \cdot T_{\text{clk}},$$

(4)

where $C$ is the power consumed by the other circuit components, and $T_{\text{clk}}$ is the clock period. The most energy efficient solution would be the value of $\mu$ that minimizes the above expression.

Note that Equation (4) presents a high-level overview of the energy consumption. In practice as $\mu$ increases or decreases other parts of the circuit like the OR/XOR tree may need to be scaled up or down resulting in the power consumption included in the term $C$ to be significant. However for a basic understanding we can use this expression. From our simulation results it was very clear that $P_{\text{mob}} \gg P_{\text{dp}}$, (eg. at 1 GHz, for $n = m = 20$, $d = 4$ and $\mu = 6$, we had $P_{\text{dp}} \approx 1.5$ mW and $P_{\text{mob}} \approx 70$ mW). So increasing $\mu$ will increase the power consumption and area of the circuit. However it brings down the time taken for the circuit to extract the roots significantly (since it depends on $2^{n-\mu}$). If $\mu < d$, then increasing $\mu$ brings down the time taken by a large enough factor, so that the total energy consumed decreases. However as soon as $\mu \geq d$, $2^{n-d}$ becomes the more dominant term in the time expression. And then any decrease in time due to increase in $\mu$ becomes insignificant and cannot offset the power increase incurred due to the inclusion of more Polymob1 circuits. Thus the optimum energy consumed is at some value of $\mu = d+$ some small constant. This has been verified by simulations shown in Figure 12.

The above analysis means that the optimal value of energy is attained at $\mu = d + \epsilon$ where $E(m, d + \epsilon) < E(m, d + \epsilon - 1)$ and $E(m, d + \epsilon) < E(m, d + \epsilon + 1)$. For $m = n = 20$, $d = 4$, using the values of $P_{\text{mob}}, P_{\text{dp}}$ solving the two inequalities gives $\epsilon = 2$. In our simulations we found slightly less energy consumption at $\epsilon = 4$. Assuming that $E(m, \mu)$ is minimized at this value of $\mu$, the energy consumption required to solve a set of $n$ equations of degree $d$ can be expressed asymptotically as proportional to $d \cdot P_{\text{mob}} \cdot 2^{n-d}$. Since $P_{\text{mob}}$...
itself is proportional to $S(n,d) \in O(n^{d+1})$, we can speculate that the optimal energy is in $O(n^{d+1} \cdot 2^{n-d})$. For detailed report of synthesis results please refer to Tables 3, 4, 5 in Appendix C.

5.5 Height-bound trees for time/energy trade-offs: Polymob2 $[h]$

Note that a full-depth Polymob1 tree has height $(n - d)$ and takes around $2^{n-d}$ clock cycles. If instead, we bound the height of the tree to $n - h$ (i.e. by limiting the number of vertically stacked registers in the circuit in Figure 5 to $n - h$), for some $h > d$, it follows that we obtain a new Möbius Transform circuit which completes in $2^{n-h}$ clock cycles which a factor $2^{h-d}$ faster than the plain Polymob1 circuit. This circuit (let’s call it Polymob2 $[h]$) faithfully computes the Möbius Transform of the original $n$-variable Boolean function if the following conditions are met:

- Note that the ANF vectors in the lowest registers are $A[n-h]_i, \forall i \in [0, 2^{n-h} - 1]$. These are naturally the ANFs of Boolean functions of $h$ variables. Thus the Expmob1 circuit connected to the bottom-most register must be of $h$-variables instead of $d$-variables needed in Polymob1.

- The Expmob1 circuit expects inputs of size $2^h$ bits canonically arranged as explained in Section 2. However, the lowest level vector $A[n-h]_i$ is of size $\binom{n}{h}$ and the bits of the vector are arranged according to the mapping $\chi_{n,d}$. Thus one needs to apply the inverse mapping $\chi_{n,d}^{-1}$ to the vector before it is input to the Expmob1 circuit. Since $\chi_{n,d}^{-1} : \left[0, \binom{n}{d} - 1\right] \rightarrow H(n,d)$, naturally all the $h$-bit locations with hamming weight greater than $d$ are initialized to 0.

Although it may appear that increasing $h$ would result in substantial decrease in time/energy of computation, however note that we can not decrease the number of clock cycles by arbitrarily increasing $h$. This is because as $h$ increases the critical path of the Expmob1 circuit becomes dominant. In that case the critical path of the circuit is essentially given by the maximum of $2 \log_2(n - d)$ and $h$ (which is the number of butterfly layers required in the Expmob1 circuit). Once this happens we can clock the circuit at lesser frequencies, which would increase the physical time of computation. In fact one can easily deduce that Polymob2 $[d]$ is equivalent to the Polymob1 circuit and Polymob2 $[n]$ is essentially equivalent to the fully combinatorial Expmob1 circuit for $n$ variables. So
increasing the value of \( h \) beyond a point always proves counter-productive. Note that when we use the Polymob2 \([h]\) circuits as components of equation solvers, as will be described in the next section, many circuit components that follow the Möbius unit, scale exponentially with \( h \). This also ensures that scaling the value of \( h \) after a certain point is infeasible.

### 5.6 Polysolve4 solvers with Polymob2 \([h]\) circuits

One can now think of Polysolve4 solvers that uses these depth bound Möbius Transform circuits as the core circuit replacing the plain Polymob1. Following the arguments in the previous sub-section it can be seen that the total number of cycles taken by this architecture is around \( 2^{n-h} + 2^{n-\mu} \). However as we have alluded to before, as \( h \) increases the sizes of the Expmob1 and Encoder-Decoder circuits that follow it also increase exponentially. This increases the critical path of the circuit and the power consumption and so after a certain point the energy and the total physical time required to solve the system increases. One can list the salient features of the circuit thus:

**Circuit Area:** The circuit now requires \( \mu \cdot \sum_{i=1}^{n-h} \binom{n-i}{\mu_d} \) flip-flops/XOR gates which is loosely upper-bounded by \( \mu \cdot n^{d+1} \). Combining with the dot-product circuits, the area requirement for the dot-product and Polymob1 circuits is still \( O(m \cdot n^{d+1}) \). As \( h \) increases we also need to be mindful of the additional overhead of the \( \mu \) Expmob1 circuits, and encoder and decoder circuit all of which grow exponentially with \( h \) and is of the order \( \mu \cdot h \cdot 2^h \).

**Total Critical Path:** The encoder/decoder and the Expmob1 both have circuit depth proportional to \( h \) gates. Thus the critical path which grows due to the combination of these 2 circuit components will be the maximum of the delay due to these 2 circuit parts and that of the plain Polysolve4 solver.

**AT product:** Note that \( A \) grows as \( h \) increases. For \( m = n \), we have \( A \approx n \cdot n^{d+1} + \mu \cdot h \cdot 2^h \). Since the time taken \( 2^{n-h} + 2^{n-\mu} \) is dependent on two parameters \( h, \mu \), in order to balance out the time contributions resulting form the 2 choices of parameter, we set \( h = \mu \). This leads us to the product \( AT = (n^{d+2} + h^2 \cdot 2^h) \cdot 2^{n-h+1} \). If we choose the value of \( h = \tilde{h} \) such that \( \tilde{h}^2 \cdot 2^\tilde{h} = n^{d+2} \), then \( A \in O(n^{d+2}) \) as required, and we have \( AT = 4 \cdot n^{d+2} \cdot 2^{n-h} \). This seems to be the minimum \( AT \) product for all the solver circuits we have considered.

### 5.7 Energy Analysis

For the next set of experiments we kept \( n = m = 20 \) and \( h = \mu \) and varied the value of \( h \). The goal was to find the value of \( h \) at which the energy consumed will be optimal. The results of the next set of experiments are presented in Table 6, and the results presented graphically in Figure 13.

To analyze the increase of energy consumption with respect to \( n, h \) we have to be mindful of the fact that as \( h \) increases the contribution of the Expmob1 circuit to the total energy consumption increases exponentially as \( h \). We already know that the size of Expmob1 circuit is \( O(h \cdot 2^h) \). We conjecture that the power consumed varies as \( h^2 \cdot 2^h \). The reasoning is similar to the one followed in [BBR15]. If we assume that each xor gate in the first layer of the butterfly consumes power proportional to 1 unit (see Figure 2), then due to the propagation of glitches from one layer to the other, each xor-gate in the second, third, fourth layers consume power proportional to \( 1 + \delta, 1 + 2\delta, 1 + 3\delta \) etc, where \( \delta \) is some positive constant. The sum \( \sum_{i=1}^{h} 1 + (i - 1)\delta \) is a well known arithmetic series and is of the order \( h^2 \delta \). Since there are \( 2^{n-h} \) xor gates in each layer we can conclude that the power consumption of the Expmob1 varies as \( h^2 \cdot 2^h \). The remaining analysis is similar to the plain Polysolve4 circuit. Let \( P_{mob}, P_{exmob}, P_{dp} \) be the dynamic power consumed by each Polymob1 (without Expmob1), the Expmob1 and dot-product circuit respectively,
where $C$ is the power consumed by the other circuit components, and $T_{clk}$ is the clock period. For $m = n$ and $h = \mu$, we obtain

$$E'(n, h) \approx (h \cdot (P_{mob} + P_{expmob}) + (n - h) \cdot P_{dp}) \cdot 2^{n-h+1} \cdot T_{clk},$$

(5)

Asymptotically since both $P_{mob}, P_{expmob} \gg P_{dp}$, and since $P_{mob} \propto \sum_{i=1}^{n-h} \binom{n-i-1}{d-1} < C_1 \cdot n^{d+1}$ and $P_{expmob} \propto h^3 \cdot 2^h \approx C_2 \cdot h^3 \cdot 2^h$, we have the asymptotical expression $E'(n, h) \approx (h \cdot C_1 \cdot n^{d+1} + h^3 \cdot C_2 \cdot 2^h) \cdot 2^{n-h+1} \cdot T_{clk}$, where $C_1, C_2$ are constants of proportionality. The value of $h$ at which a minimum is achieved depends on how much $C_1$ is larger than $C_2$. Generally in our experience, we find that $C_1$ is much larger than $C_2$ for not very large values of $h$, and increasing $h$ will continue to decrease $E'$ as long as the contribution due to $P_{expmob}$ is below that of $P_{mob}$. For example, for $m = n = 20$, $d = 4$, at 1 GHz for the Nangate 15 nm process, we find that $P_{mob} \approx 70 - 75 \mu W$ as $h$ varies between 5 to 12, which makes $C_1 \approx 3.5 \mu W$. $P_{expmob}$ more readily varies with $h$, and we found that for this set of parameters we can approximate it as $82 \cdot h^2 \cdot 2^h \mu W$ (this was estimated by studying the power simulation reports of the individual circuits for $h$ varying between 5-12, and is reasonably accurate). This gives us a minimum at $h = 14$, at which $E'(20,14)$ is estimated to be around 0.6 $\mu J$ which is much less than the 32.6 $\mu J$ we obtained for the full-depth Polysolve4 circuit, and around 100 times less than the 68.4 $\mu J$ obtained for the Polysolve3 circuit. 

Optimal energy consumption There are some practical issues in directly applying the asymptotic expression. As $h$ increases, the Expmob1 circuit dictates the critical path of the circuit. If we want to clock the circuit at frequency $f$, we need to ensure that $h$ is small enough so that the critical path does not exceed $1/f$. With this in mind let us re-examine the asymptotic expression of $E'(n, h)$. To minimize this expression we should set the smallest value of $h = h_0$ such that $h_0^3 \cdot 2^{h_0} \cdot C_2 > 2 \cdot h_0 \cdot C_1 \cdot n^{d+1}$, and which additionally keeps the critical path under $T_{clk}$. Then the optimal value of $E' < 3h_0^3 \cdot C_2 \cdot 2^n \cdot T_{clk}$.

Figure 13: Energy decrease with increasing $h$ for Polysolve4 solvers for $n = 20$, $h = \mu$. The colored horizontal lines indicate the optimal energy consumption for the full depth Polysolve4 circuit for the same equation system.

---

Due to time constraints we did the actual simulation only up to $h = 12$ as seen in Figure 13. For $h \geq 13$ the synthesis takes close to 24 hours.
6 Conclusion

In this paper, we propose, design and evaluate hardware architectures to perform Möbius Transform of Boolean functions using only polynomial amount of silicon area. In a nutshell, this is a serialized implementation of the basic transform and uses around $2^n - d$ clock cycles to generate the entire truth table of the Boolean function. The immediate application of the circuits is to use it to solve an underlying system of low degree equations over GF(2), a problem which occurs in many cryptanalytic attacks on real world cryptosystems. We further describe architectures for such equation solvers which keeps the critical path of the circuit to a minimum. One of the first conclusions of the paper is the demonstration that a system of $m$ Boolean equations in $n$ variables and algebraic degree up to $d$ can be solved using silicon area proportional to $m \cdot n^{d+1}$ gates and using physical time proportional to $2^{n-d} \cdot \log_2(n - d)$.

In the second part of the paper we introduce the Polysolve4 circuit that additionally aims to achieve energy efficiency by checking the common solutions of a reduced number of equations using specialized dot-product circuits. The new circuit has area also bound by $m \cdot n^{d+1}$ and has circuit depth proportional to $d \cdot \log_2 n$. We also show that further optimizations with respect to energy may be obtained by using depth-bound Möbius circuits that exponentially decrease run time at the cost of additional logic area and depth. For $n = m = 20$, $d = 4$, we show that the final circuit is around 100 times more energy efficient than the Polysolve3 circuit. Regarding future work, the next step could be the acceleration of the proposed architecture using an high performance computation environment [PBB+21].

Acknowledgments

This work is partially supported by the EU Horizon 2020 Programme under grant agreement No. 957269 (EVEREST). The authors thank the anonymous reviewers of TCHES for the productive comments that helped to improve the presentation of this work. Special thanks to Charles Bouillaguet for enabling us to make the paper mathematically more precise.

References


Appendix A: Fast generation of the AM graph

The following generates the adjacency matrix for the AM graph in polynomial time.

**Algorithm 2:** Generation of AM

<table>
<thead>
<tr>
<th>Generate AM $(n, d)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> $n$: Number of variables, $d$: Algebraic degree</td>
</tr>
<tr>
<td><strong>Output:</strong> The adjacency matrix AM of size $(n \downarrow d) \times (n - d)$</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>for $s \leftarrow$ the $k$-th string in $H(n, d)$ do</td>
</tr>
<tr>
<td>for $\ell \leftarrow 1 \rightarrow n - d$ do</td>
</tr>
<tr>
<td>Compute $\alpha := \chi_{n, d}(s) = \binom{n}{\downarrow d} + \binom{s_i}{\downarrow d-1} + \cdots$ / Assuming $s = 2^{i_0} + 2^{i_1} + \cdots$ /</td>
</tr>
<tr>
<td>if $hw(s') \leq d$ then</td>
</tr>
<tr>
<td>Compute $\beta := \chi_{n, d}(s')$</td>
</tr>
<tr>
<td>Assign $AM[\alpha, \ell] \leftarrow \beta$</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>Assign $AM[\alpha, \ell] \leftarrow 0$</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
<tr>
<td>end</td>
</tr>
</tbody>
</table>

Appendix B: Proof of Equation (2)

We need to prove the following:

$$S(n, d) = \sum_{i=0}^{n-d} \binom{n-i}{\downarrow d} \in O(n^{d+1}).$$

To prove this we make use of the hockey-stick identity [Jon96] which states that $\sum_{m=d}^{n} \binom{m}{d} = \binom{n+1}{d+1}$. Note that expanding out $S(n, d)$ we get

$$S(n, d) = \binom{n}{d} + \binom{n-1}{d-1} + \cdots + \binom{0}{d} + \binom{n-1}{d} + \binom{n-2}{d-1} + \cdots + \binom{0}{d} + \cdots + \binom{d}{d}$$

Applying the hockey-stick identity on each column we get

$$S(n, d) < \binom{n+1}{d+1} + \binom{n+1}{d} + \cdots + \binom{n+1}{1}$$

Using mathematical induction it is easy to prove the hypothesis $P(d) : \sum_{i=0}^{d} \binom{n}{i} < n^d$, for all $d \geq 2$, $n > d$. The base case for $d = 2$, amounts to $n(n-1)/2 + n + 1 < n^2 \Rightarrow n^2 > n + 2$,
which holds for all \( n > 2 \). Taking \( P(d) \) to be true we have

\[
P(d + 1) = \sum_{i=0}^{d+1} \binom{n}{i} < n^d + \frac{n^{d+1}}{(d+1)!} < n^d + \frac{n^{d+1}}{(d+1)!}
\]

Therefore we have \( S(n, d) < (n + 1)^{d+1} \), from which we can conclude it is \( O(n^{d+1}) \).

### Appendix C: Synthesis results for Polysolve4 circuit

#### Table 3: Synthesis results for the Polysolve4 circuit for \( d = 4 \). Power reported at 1 GHz.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \mu )</th>
<th>Area (KGE)</th>
<th>( T_{cr} ) (ns)</th>
<th>( T_{min} ) (ns)</th>
<th>Power (mW)</th>
<th>Energy (J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>5.946</td>
<td>3.421</td>
<td>7.150</td>
<td>0.0002</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12.361</td>
<td>5.369</td>
<td>2.781</td>
<td>5.577</td>
<td>0.0002</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15.547</td>
<td>6.172</td>
<td>3.960</td>
<td>8.856</td>
<td>0.0002</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>17.343</td>
<td>5.825</td>
<td>4.501</td>
<td>5.571</td>
<td>0.0001</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20.133</td>
<td>6.182</td>
<td>5.271</td>
<td>6.172</td>
<td>0.0001</td>
<td></td>
</tr>
</tbody>
</table>

#### Table 4: Synthesis results for the Polysolve4 circuit for \( d = 3 \). Power reported at 1 GHz.

<table>
<thead>
<tr>
<th>( n )</th>
<th>( \mu )</th>
<th>Area (KGE)</th>
<th>( T_{cr} ) (ns)</th>
<th>( T_{min} ) (ns)</th>
<th>Power (mW)</th>
<th>Energy (J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>6.744</td>
<td>3.497</td>
<td>7.367</td>
<td>0.0002</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>12.431</td>
<td>5.987</td>
<td>2.821</td>
<td>5.577</td>
<td>0.0002</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>15.547</td>
<td>6.172</td>
<td>3.960</td>
<td>8.856</td>
<td>0.0002</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>17.343</td>
<td>5.825</td>
<td>4.501</td>
<td>5.571</td>
<td>0.0001</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>20.133</td>
<td>6.182</td>
<td>5.271</td>
<td>6.172</td>
<td>0.0001</td>
<td></td>
</tr>
</tbody>
</table>

### Compact Circuits for Efficient Möbius Transform

518 Compact Circuits for Efficient Möbius Transform
Table 5: Synthesis results for the Polysolve4 circuit for $d = 2$. Power reported at 1 GHz.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$\mu$</th>
<th>Area (KGE)</th>
<th>$T_{ov}$ (ps)</th>
<th>$T_{min}$ (ns)</th>
<th>Power (mW)</th>
<th>Energy ($\mu$J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>3,369</td>
<td>47.68</td>
<td>6.10</td>
<td>1.022</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>4,205</td>
<td>48.07</td>
<td>4.61</td>
<td>1.332</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>5,442</td>
<td>52.52</td>
<td>4.20</td>
<td>1.638</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>6,378</td>
<td>54.62</td>
<td>3.93</td>
<td>1.839</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>7,489</td>
<td>57.24</td>
<td>3.26</td>
<td>2.077</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8,862</td>
<td>57.92</td>
<td>2.64</td>
<td>2.232</td>
<td>0.0001</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>5,947</td>
<td>57.22</td>
<td>10.99</td>
<td>1.862</td>
<td>0.0004</td>
</tr>
<tr>
<td></td>
<td>9</td>
<td>6,436</td>
<td>57.74</td>
<td>10.12</td>
<td>2.069</td>
<td>0.0009</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>12,463</td>
<td>56.96</td>
<td>16.40</td>
<td>3.011</td>
<td>0.0009</td>
</tr>
<tr>
<td></td>
<td>11</td>
<td>7,860</td>
<td>55.69</td>
<td>57.03</td>
<td>2.418</td>
<td>0.0025</td>
</tr>
<tr>
<td></td>
<td>12</td>
<td>10,348</td>
<td>61.94</td>
<td>47.57</td>
<td>3.008</td>
<td>0.0023</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>12,774</td>
<td>60.75</td>
<td>38.88</td>
<td>3.576</td>
<td>0.0023</td>
</tr>
<tr>
<td></td>
<td>14</td>
<td>15,237</td>
<td>59.50</td>
<td>34.27</td>
<td>3.925</td>
<td>0.0023</td>
</tr>
<tr>
<td></td>
<td>15</td>
<td>17,672</td>
<td>58.23</td>
<td>29.40</td>
<td>4.331</td>
<td>0.0024</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>20,976</td>
<td>57.06</td>
<td>24.50</td>
<td>4.736</td>
<td>0.0024</td>
</tr>
<tr>
<td></td>
<td>17</td>
<td>9,952</td>
<td>62.97</td>
<td>114.86</td>
<td>3.314</td>
<td>0.0058</td>
</tr>
<tr>
<td></td>
<td>18</td>
<td>13,162</td>
<td>63.42</td>
<td>97.41</td>
<td>3.701</td>
<td>0.0057</td>
</tr>
<tr>
<td></td>
<td>19</td>
<td>16,174</td>
<td>67.64</td>
<td>65.86</td>
<td>4.272</td>
<td>0.0055</td>
</tr>
<tr>
<td></td>
<td>20</td>
<td>19,302</td>
<td>73.25</td>
<td>83.35</td>
<td>4.803</td>
<td>0.0054</td>
</tr>
<tr>
<td></td>
<td>21</td>
<td>22,434</td>
<td>77.59</td>
<td>101.49</td>
<td>5.305</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>22</td>
<td>25,483</td>
<td>81.89</td>
<td>119.56</td>
<td>5.904</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>23</td>
<td>28,599</td>
<td>85.89</td>
<td>137.55</td>
<td>6.513</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>24</td>
<td>31,765</td>
<td>89.89</td>
<td>155.54</td>
<td>7.122</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>25</td>
<td>34,974</td>
<td>93.89</td>
<td>173.53</td>
<td>7.732</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>26</td>
<td>38,233</td>
<td>97.89</td>
<td>191.52</td>
<td>8.342</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>41,548</td>
<td>101.89</td>
<td>209.51</td>
<td>8.953</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>28</td>
<td>44,918</td>
<td>105.89</td>
<td>227.50</td>
<td>9.564</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>29</td>
<td>48,347</td>
<td>109.89</td>
<td>245.49</td>
<td>10.175</td>
<td>0.0053</td>
</tr>
<tr>
<td></td>
<td>30</td>
<td>51,844</td>
<td>113.89</td>
<td>263.48</td>
<td>10.786</td>
<td>0.0053</td>
</tr>
</tbody>
</table>

Table 6: Synthesis results for the Polysolve4 circuit with height bound Möbius Transform circuits and $h = \mu$ and for $n = m = 20$. Power reported at 1 GHz.

<table>
<thead>
<tr>
<th>$n$</th>
<th>$d = 2$</th>
<th>Area (KGE)</th>
<th>$T_{ov}$ (ps)</th>
<th>$T_{min}$ (ns)</th>
<th>Power (mW)</th>
<th>Energy ($\mu$J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5,667</td>
<td>124.97</td>
<td>12968.39</td>
<td>26.177</td>
<td>5.2822</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>7,302</td>
<td>119.63</td>
<td>15689.14</td>
<td>22.852</td>
<td>2.9953</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>8,357</td>
<td>122.94</td>
<td>8057.00</td>
<td>25.667</td>
<td>1.682</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10,102</td>
<td>124.37</td>
<td>4075.36</td>
<td>28.102</td>
<td>0.9208</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>12,215</td>
<td>130.94</td>
<td>2145.32</td>
<td>33.520</td>
<td>0.4592</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>15,002</td>
<td>142.25</td>
<td>1157.44</td>
<td>43.894</td>
<td>0.3591</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>206,305</td>
<td>143.17</td>
<td>586.42</td>
<td>79.914</td>
<td>0.5405</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>531,863</td>
<td>163.87</td>
<td>167.81</td>
<td>208.395</td>
<td>0.2314</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>996,395</td>
<td>171.78</td>
<td>87.95</td>
<td>435.441</td>
<td>0.2229</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$n$</th>
<th>$d = 3$</th>
<th>Area (KGE)</th>
<th>$T_{ov}$ (ps)</th>
<th>$T_{min}$ (ns)</th>
<th>Power (mW)</th>
<th>Energy ($\mu$J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>124,265</td>
<td>208.87</td>
<td>15688.50</td>
<td>398.157</td>
<td>26.993</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>186,587</td>
<td>213.82</td>
<td>7080.45</td>
<td>474.863</td>
<td>15.583</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>204,906</td>
<td>225.81</td>
<td>3689.67</td>
<td>571.070</td>
<td>9.3545</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>237,227</td>
<td>235.72</td>
<td>1995.59</td>
<td>652.621</td>
<td>5.4461</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>260,779</td>
<td>247.49</td>
<td>1013.15</td>
<td>765.022</td>
<td>3.1335</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>299,840</td>
<td>262.67</td>
<td>537.95</td>
<td>933.736</td>
<td>1.9213</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>377,842</td>
<td>264.61</td>
<td>270.96</td>
<td>1192.6</td>
<td>1.2212</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>490,615</td>
<td>281.91</td>
<td>144.34</td>
<td>1831.8</td>
<td>0.9378</td>
<td></td>
</tr>
</tbody>
</table>

Appendix D: Synthesis Results for Expmob1,Expmob2
Table 7: Synthesis results for the Expmob1/Expmob2 circuits. Power reported at 1 GHz.

<table>
<thead>
<tr>
<th>n</th>
<th>Circuit</th>
<th>Area (µm²)</th>
<th>T_{off} (ns)</th>
<th>T_{on} (ps)</th>
<th>Power (mW)</th>
<th>Energy (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Expmob1</td>
<td>157.04</td>
<td>38.50</td>
<td>84.80</td>
<td>10.30</td>
<td>0.001</td>
</tr>
<tr>
<td>7</td>
<td>Expmob1</td>
<td>24.12</td>
<td>3.12</td>
<td>3.30</td>
<td>1.30</td>
<td>0.002</td>
</tr>
<tr>
<td>8</td>
<td>Expmob1</td>
<td>24.50</td>
<td>3.45</td>
<td>3.50</td>
<td>1.45</td>
<td>0.003</td>
</tr>
<tr>
<td>9</td>
<td>Expmob1</td>
<td>24.75</td>
<td>3.50</td>
<td>3.50</td>
<td>1.50</td>
<td>0.004</td>
</tr>
<tr>
<td>10</td>
<td>Expmob1</td>
<td>24.75</td>
<td>3.50</td>
<td>3.50</td>
<td>1.50</td>
<td>0.004</td>
</tr>
</tbody>
</table>

Appendix E: Synthesis Results for Polysolve1/2/3 circuits

Table 8: Synthesis results for d = 3, 4 for the Polysolve1/Polysolve2/Polysolve3 circuits.

<table>
<thead>
<tr>
<th>n</th>
<th>Circuit</th>
<th>Area (µm²)</th>
<th>T_{off} (ns)</th>
<th>T_{on} (ps)</th>
<th>Power (mW)</th>
<th>Energy (pJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Polysolve1</td>
<td>157.04</td>
<td>38.50</td>
<td>84.80</td>
<td>10.30</td>
<td>0.001</td>
</tr>
<tr>
<td>9</td>
<td>Polysolve1</td>
<td>24.12</td>
<td>3.12</td>
<td>3.30</td>
<td>1.30</td>
<td>0.002</td>
</tr>
<tr>
<td>10</td>
<td>Polysolve1</td>
<td>24.50</td>
<td>3.45</td>
<td>3.50</td>
<td>1.45</td>
<td>0.003</td>
</tr>
<tr>
<td>11</td>
<td>Polysolve1</td>
<td>24.75</td>
<td>3.50</td>
<td>3.50</td>
<td>1.50</td>
<td>0.004</td>
</tr>
</tbody>
</table>

Compact Circuits for Efficient Möbius Transform
Table 9: Synthesis results for $d = 2$ for the Polysolve1/Polysolve2/Polysolve3 circuits.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Area (kGE)</th>
<th>$T_{cr}$ (ps)</th>
<th>$T_{min}$ (ns)</th>
<th>Power (mW)</th>
<th>Energy ($\mu$J)</th>
<th>Area (kGE)</th>
<th>$T_{cr}$ (ps)</th>
<th>$T_{min}$ (ns)</th>
<th>Power (mW)</th>
<th>Energy ($\mu$J)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>Polysolve1</td>
<td>6.825</td>
<td>54.60</td>
<td>3.82</td>
<td>2.390</td>
<td>0.0002</td>
<td>15</td>
<td>Polysolve1</td>
<td>6.825</td>
<td>54.60</td>
</tr>
<tr>
<td>Polysolve2</td>
<td>7.129</td>
<td>54.49</td>
<td>3.92</td>
<td>2.539</td>
<td>0.0002</td>
<td>Polysolve3</td>
<td>9.561</td>
<td>75.43</td>
<td>5.88</td>
<td>2.766</td>
</tr>
<tr>
<td>9</td>
<td>Polysolve1</td>
<td>10.399</td>
<td>61.78</td>
<td>8.33</td>
<td>4.497</td>
<td>0.0005</td>
<td>16</td>
<td>Polysolve1</td>
<td>10.399</td>
<td>61.78</td>
</tr>
<tr>
<td>Polysolve2</td>
<td>10.946</td>
<td>63.38</td>
<td>8.68</td>
<td>3.633</td>
<td>0.0005</td>
<td>Polysolve3</td>
<td>14.599</td>
<td>87.10</td>
<td>12.54</td>
<td>4.039</td>
</tr>
<tr>
<td>10</td>
<td>Polysolve1</td>
<td>14.918</td>
<td>56.44</td>
<td>14.90</td>
<td>4.887</td>
<td>0.0013</td>
<td>17</td>
<td>Polysolve1</td>
<td>14.918</td>
<td>56.44</td>
</tr>
<tr>
<td>Polysolve2</td>
<td>15.293</td>
<td>53.45</td>
<td>14.22</td>
<td>5.072</td>
<td>0.0013</td>
<td>Polysolve3</td>
<td>19.899</td>
<td>72.16</td>
<td>25.25</td>
<td>5.226</td>
</tr>
<tr>
<td>11</td>
<td>Polysolve1</td>
<td>21.818</td>
<td>64.94</td>
<td>33.83</td>
<td>7.037</td>
<td>0.0037</td>
<td>18</td>
<td>Polysolve1</td>
<td>21.818</td>
<td>64.94</td>
</tr>
<tr>
<td>Polysolve2</td>
<td>22.227</td>
<td>63.42</td>
<td>33.17</td>
<td>7.187</td>
<td>0.0037</td>
<td>Polysolve3</td>
<td>26.870</td>
<td>88.56</td>
<td>71.01</td>
<td>9.966</td>
</tr>
<tr>
<td>12</td>
<td>Polysolve1</td>
<td>28.812</td>
<td>68.57</td>
<td>71.31</td>
<td>9.678</td>
<td>0.0100</td>
<td>19</td>
<td>Polysolve1</td>
<td>28.812</td>
<td>68.57</td>
</tr>
<tr>
<td>Polysolve2</td>
<td>30.370</td>
<td>68.56</td>
<td>71.01</td>
<td>9.966</td>
<td>0.0103</td>
<td>Polysolve3</td>
<td>34.265</td>
<td>98.14</td>
<td>102.65</td>
<td>9.760</td>
</tr>
<tr>
<td>13</td>
<td>Polysolve1</td>
<td>40.159</td>
<td>68.50</td>
<td>141.04</td>
<td>12.886</td>
<td>0.0265</td>
<td>20</td>
<td>Polysolve1</td>
<td>40.159</td>
<td>68.50</td>
</tr>
<tr>
<td>Polysolve2</td>
<td>40.713</td>
<td>67.87</td>
<td>139.88</td>
<td>13.116</td>
<td>0.0270</td>
<td>Polysolve3</td>
<td>57.725</td>
<td>106.39</td>
<td>220.44</td>
<td>13.906</td>
</tr>
<tr>
<td>14</td>
<td>Polysolve1</td>
<td>52.478</td>
<td>77.81</td>
<td>319.64</td>
<td>17.036</td>
<td>0.0700</td>
<td>Polysolve2</td>
<td>52.478</td>
<td>77.81</td>
<td>319.64</td>
</tr>
<tr>
<td>Polysolve3</td>
<td>70.379</td>
<td>111.85</td>
<td>461.05</td>
<td>17.169</td>
<td>0.0705</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>