IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–40. DOI:XXXXXXXX

Compact Circuits for Efficient Möbius Transform

Subhadeep Banik[1], and Francesco Regazzoni[2,1]

[1] Universita della Svizzera Italiana, Lugano, Switzerland, subhadeep.banik@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 $2^{r/2}$ [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 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 $f_i(g_0) = f_i(\vec{0})$ 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 $f_i = x_1x_2 \oplus x_3 \oplus x_1x_4x_5$, then $\frac{\delta f_i}{\delta x_1} = x_2 \oplus x_4x_5$ and $\frac{\delta f_i}{\delta x_2} = x_1$, $\frac{\delta f_i}{\delta x_3} = 1$ 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 and stored in memory. 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 upto 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 $AT^2 \geq N^2$ (where $N = 2^n$) [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 $4 \cdot n^{d+2} \cdot 2^{n-\hat{h}}$ where $\hat{h}^2 \cdot 2^{\hat{h}} = n^{d+2}$.

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 μ 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_{min}, T_{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 denominations.

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 $\vec{u} = [a_0, a_1, \ldots, a_{2^n-1}]$ is defined as the 2^n-length string of all the a_i's.

Example 1. For example, consider the 3-variable function $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. We can write this as $x_0^0x_1^0x_2^0 \oplus x_0^1x_1^1x_2^0 \oplus x_0^0x_1^0x_2^1 \oplus x_0^1x_1^0x_2^1$. 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 $a_0 = a_1 = a_5 = a_6 = 1$ and the rest of the $a_i = 0$.

The algebraic degree of the function (provided the 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 $hw(i) > d$ are naturally 0. Since there are exactly $\binom{n}{i}$ length n strings of hamming weight i, we can see that the ANF of degree d function can be expressed using $\binom{n}{\downarrow d} := \sum_{i=0}^{d}\binom{n}{i} < n^d$ 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 8×8 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

$$M_3 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

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 $\textbf{one}_i(x) : \{0,1\}^{n-1} \rightarrow \{0,1\}^n$ to be the function that inserts a 1 in the i-th MSB position of x, and $\textbf{zero}_i(x)$ to be a function that inserts a 0 in the same position, i.e $\textbf{one}_0(1001) = 1\ 1101$ and $\textbf{zero}_0(1001) = 0\ 1001$ 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 $\textbf{zero}_i(j)$ and $\textbf{one}_i(j)$ 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

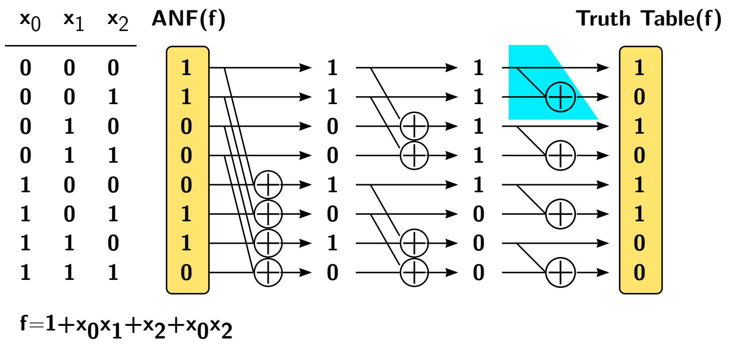


Figure 2: Möbius transform on $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. The blue shaded component represents one butterfly unit.

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 π_n to be the following permutation:

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

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 π_n. Then it is easy to verify that the Möbius matrix $M_n = (P_n \cdot B_n)^n$.

3.1 Synthesis Results

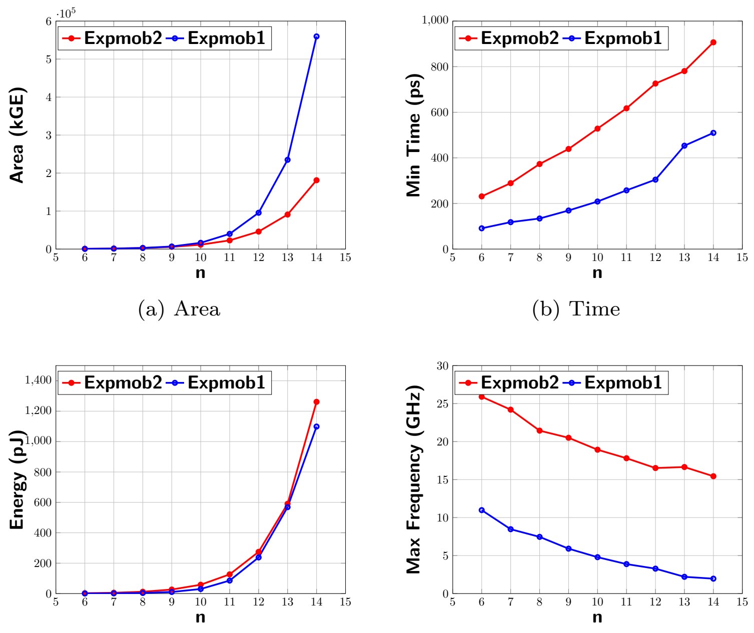
In this section we will describe the flow of simulation followed for each of the circuits reported in the paper. The design was described at the RTL level using a hardware description language and functional correctness was first verified. Thereafter the circuit was synthesized using the Nangate 15nm Open Cell Library [MMR+15] using *Synyopsys 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 $\binom{n}{\downarrow d} < n^d$ 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



(a) Area (b) Time

(c) Energy (d) Maximum Frequency

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 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

$$\begin{aligned} f(x_0, x_1, x_2) &= 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2 = x_0 \cdot (x_1 \oplus x_2) \oplus (1 \oplus x_2) \\ &= x_0 \cdot \big[f(1, x_1, x_2) \oplus f(0, x_1, x_2)\big] \oplus f(0, x_1, x_2) \end{aligned}$$

Note that $\big[f(1, x_1, x_2) \oplus f(0, x_1, x_2)\big] := \frac{\delta f}{\delta x_0}$ is simply the derivative of f at the coordinate x_0. Both $\frac{\delta f}{\delta x_0}$ 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. $A_{\mathsf{top}} = [1100]$ and $A_{\mathsf{bottom}} = [1010]$. It is easy to observe/verify the following:

A: A_{top} is the ANF vector for $f(0, x_1, x_2)$ (in this case $1 \oplus x_2$) and A_{bottom} is the ANF vector for $f(1, x_1, x_2)$ (in this case $1 \oplus x_1$).

B: Both $A_{\mathsf{top}}/A_{\mathsf{bottom}}$ are outputs of the butterfly layer in which the input is A_0. Whereas A_{top} is the arm of the butterfly that does not require xor computations, some xor computations are required for A_{bottom}.

C: 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

```
Möbius $(A_0, n, d)$
Input: $A_0$: The compressed ANF vector of a Boolean function $f$
Input: $n$: Number of variables, $d$: Algebraic degree
Output: The Truth table of $f$

/* 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 Expmob1/Expmob2 to do this */
end
else
    Declare an array $T$ of size $\binom{n-1}{\downarrow d}$ bits.
    /* Now we compute the 2 operations of the butterfly layer */
1   Store 1st butterfly output i.e. $A_{\mathsf{top}}$ in $T$ (requires no xors).
    Call Möbius $(T, n-1, d)$
2   Store 2nd butterfly output i.e. $A_{\mathsf{bottom}}$ 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 $\binom{n}{\downarrow d}$ coefficients required to store A_0. Thereafter every successive i-th recursion stage requires $\binom{n-i}{\downarrow d}$ additional memory for all $1 \leq i \leq n-d$. The final stage can use **Expmob1**/**Expmob2** to perform Möbius Transform in-place and no additional memory is required. However in our experiments we preferred to use **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):

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

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.

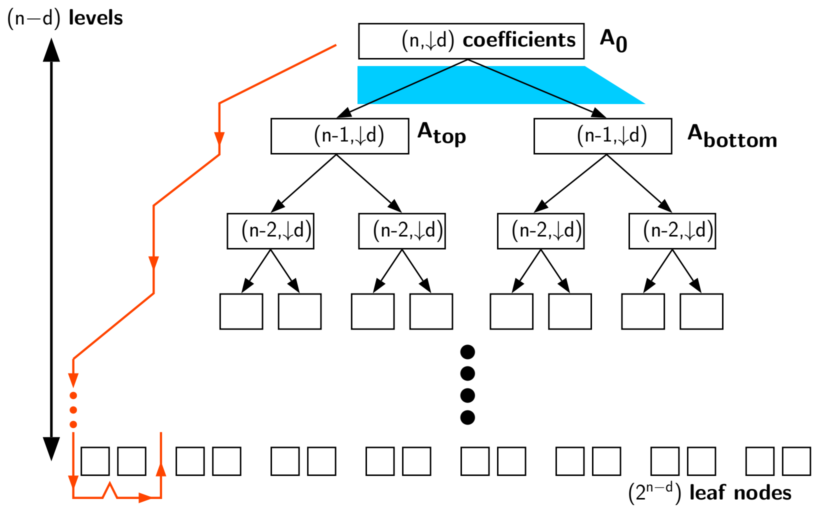


Figure 4: Recursion tree for the Möbius Transform algorithm. The blue shaded component roughly represents one arm of the butterfly unit. Note here $(x, \downarrow d) := \binom{x}{\downarrow d}$.

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 $\binom{n}{\downarrow d}$ storing the initial ANF vector A_0. There is only one $\binom{n-1}{\downarrow d}$ size register to store the second level coefficients A_{top} and A_{bottom}. This implies that if in the first clock cycle the 2nd register stores A_{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_{bottom}. Similarly there is only one $\binom{n-2}{\downarrow d}$ register to store potentially four ANF vectors (two each from the butterfly operation on A_{top} and A_{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 $\binom{n-i}{\downarrow d}$ (for $0 \le i \le n-d$), and from $i = 1$ onwards each register is preceded by a 3:1 multiplexer of size $\binom{n-i}{\downarrow d}$. 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_{top} and A_{bottom} depending on how far the execution has reached in the

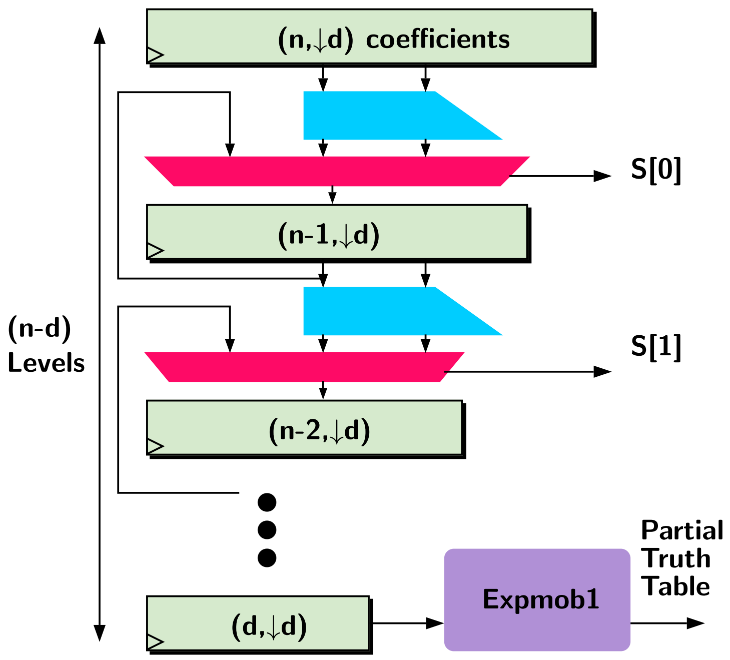


Figure 5: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit. Note here $(x, \downarrow d) := \binom{x}{\downarrow d}$.

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_{top} or A_{bottom}), it can be done by appropriately setting the select signal of the multiplexer: for example to preserve A_{bottom} we just need to set the select signal of the preceding mux so that it accepts the A_{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 $A_{\mathsf{top}}/A_{\mathsf{bottom}}$, 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 ℓ 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_{top} and A_{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 ℓ of the tree, then after applying the butterfly over the coordinate x_ℓ, the two output vectors are denoted as

$A[\ell+1]_b$ and $A[\ell+1]_{b\oplus e_\ell}$, where e_t is the unit vector of length $n-d$ with 1 at the t-th position, eg. $e_0 = 100\ldots0, e_1 = 010\ldots0$ 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 $\binom{d}{\downarrow d} = 2^d = 4$. Initially the top-level register would be initialized with $A[0]_{000}$ 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 level ℓ register contains $A[\ell]_{000}$. Figure 6 shows us the flow of data in each of the 8 cycles succeeding this.

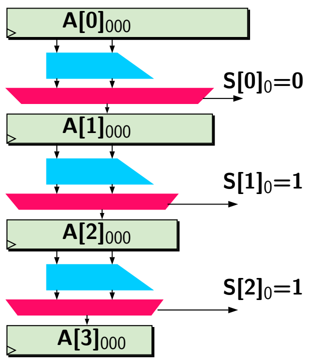
We introduce an additional notation: let $S[\ell]_t$ be the select signal of the multiplexer between the registers at levels ℓ and $\ell+1$ at time t. Which is to say that if $S[\ell]_t = 0$ and the ANF vector at level ℓ at time t is $A[\ell]_b$, then at time $t+1$, the ANF vector at level $\ell+1$ is $A[\ell+1]_b$, and if $S[\ell]_t = 1$ then the corresponding vector is $A[\ell+1]_{b\oplus e_\ell}$ (this can also be written as $A[\ell+1]_{b+e_\ell}$ since by design the coordinates of b at positions larger than ℓ are all 0). The two expressions can obviously be combined to give the single compact expression $A[\ell+1]_{b+S[\ell]_t\cdot e_\ell}$ 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 $0 \le \ell < n-d$ and $0 \le t < 2^{n-d}-1$) so that the vector at the bottommost level of the register chain is always $A[n-d]_t$ for all $0 \le t < 2^{n-d}$. Since **Expmob1** is connected to the bottommost register, this ensures that the 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 al levels, with respect to time.

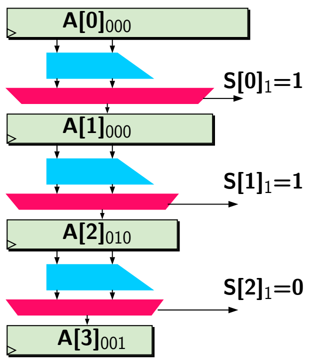
t	$\ell=0$	$\ell=1$	$\ell=2$	$\ell=3$
0	0	0	0	0
1	0	$4\cdot S[0]_0$	$2\cdot S[1]_0$	$S[2]_0$
2	0	$4\cdot S[0]_1$	$4\cdot S[0]_0+2\cdot S[1]_1$	$2\cdot S[1]_0+S[2]_1$
3	0	$4\cdot S[0]_2$	$4\cdot S[0]_1+2\cdot S[1]_2$	$4\cdot S[0]_0+2\cdot S[1]_1+S[2]_2$
4	0	$4\cdot S[0]_3$	$4\cdot S[0]_2+2\cdot S[1]_3$	$4\cdot S[0]_1+2\cdot S[1]_2+S[2]_3$
5	0	$4\cdot S[0]_4$	$4\cdot S[0]_3+2\cdot S[1]_4$	$4\cdot S[0]_2+2\cdot S[1]_3+S[2]_4$
6	0	$4\cdot S[0]_5$	$4\cdot S[0]_4+2\cdot S[1]_5$	$4\cdot S[0]_3+2\cdot S[1]_4+S[2]_5$
7	0	$4\cdot S[0]_6$	$4\cdot S[0]_5+2\cdot S[1]_6$	$4\cdot S[0]_4+2\cdot S[1]_5+S[2]_6$

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.

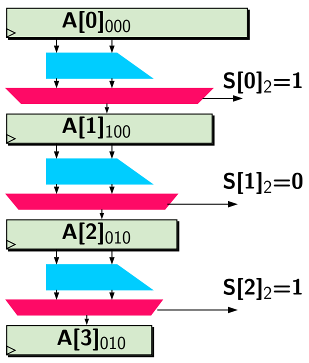
$$\begin{aligned}
S[2]_0 &= 1\\
2\cdot S[1]_0 + S[2]_1 &= 2\\
4\cdot S[0]_0 + 2\cdot S[1]_1 + S[2]_2 &= 3\\
4\cdot S[0]_1 + 2\cdot S[1]_2 + S[2]_3 &= 4\\
4\cdot S[0]_2 + 2\cdot S[1]_3 + S[2]_4 &= 5\\
4\cdot S[0]_3 + 2\cdot S[1]_4 + S[2]_5 &= 6\\
4\cdot S[0]_4 + 2\cdot S[1]_5 + S[2]_6 &= 7
\end{aligned}$$



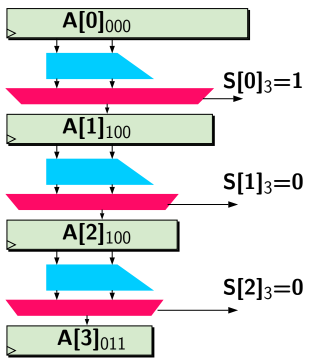
(a) t=0



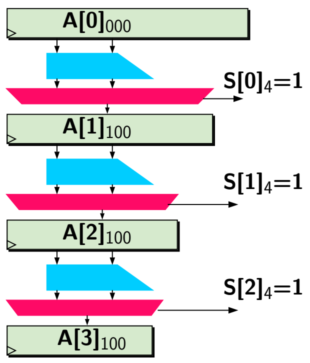
(b) t=1



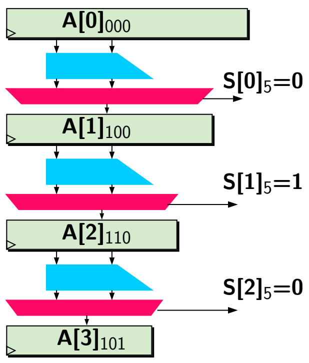
(c) t=2



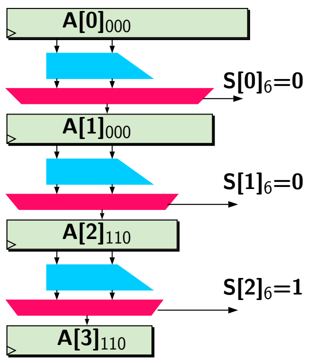
(d) t=3



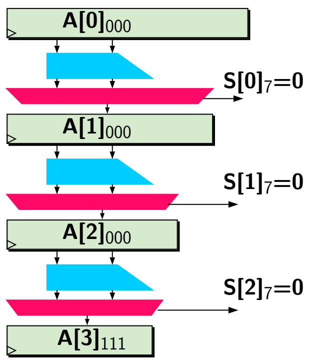
(e) t=4



(f) t=5



(g) t=6



(h) t=7

Figure 6: Dataflow for the first 8 cycles.

It can be verified that the assignments to the $S[\ell]_t$'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 $A[\ell]_{0^{n-d}}$ for all $0 \le \ell \le n-d$ of an n-variable Boolean function f of degree less than or equal to d. Then it is possible to design the multiplexer signals $S[\ell]_t$ for $0 \le t \le 2^{n-d}-2$ 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 $S[\ell]_t$ efficiently so that the subscripts of the ANF vectors at the bottommost i.e. level $n-d$, at $t = 0 \to 2^{n-d}-1$ are each equal to t itself. Generalizing the observations made above, we need $S[\ell]_t$'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

$$\begin{array}{lllllll}
 & & & & 2\cdot S[u-2]_0 & S[u-1]_0 & = 1 \\
 & & & & & + S[u-1]_1 & = 2 \\
 & & & & & & \vdots \\
 & & & 2^i \cdot S[j]_0 & + \cdots & + S[u-1]_i & = i+1 \\
 & & & & & & \vdots \\
2^{u-1}\cdot S[0]_0 & + 2^{u-2}\cdot S[1]_1 & + \cdots + & 2^i \cdot S[j]_j & + \cdots & + S[u-1]_{u-1} & = u \\
2^{u-1}\cdot S[0]_1 & + 2^{u-2}\cdot S[1]_2 & + \cdots + & 2^i \cdot S[j]_{j+1} & + \cdots & + S[u-1]_u & = u+1 \\
 & & & & & & \vdots \\
2^{u-1}\cdot S[0]_{2^u-u-1} & + 2^{u-2}\cdot S[1]_{2^u-u} & + \cdots + & 2^i\cdot S[j]_{-i+2^u-2} & + \cdots & + S[u-1]_{2^u-2} & = 2^u-1
\end{array}$$

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 $S[\ell]_t$ can be efficiently generated using a reasonable amount of logic circuits. Using the method outlined above, we can immediately see that $S[u-1]_t = t+1 \bmod 2$ for all t. With some misuse of notation the above can be written as **NOT** ($t \bmod 2$), i.e. if we have a decimal up-counter implementing t, then the $S[u-1]_t$ signal can be implemented by inverting the least significant bit of t. Similarly the sequence $S[u-2]_t,\ t = 0,1,2,\ldots$ 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 $S[j]_t = S[u-1-i]_t,\ t = 0,1,2,\ldots$ 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 $S[\ell]_t$ 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* ***Polymob1*** *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

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

For example the function $g_{8,\ 0001\ 0100} = (x_0 \vee x_2 \vee x_4)\cdot(x_1 \vee x_2 \vee x_4)\cdot(x_3 \vee x_4)\cdot x_5 \cdot x_6 \cdot x_7$ 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 **OR**ed 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 $x_i \oplus g_{i,bin_i(i)}$, where $bin_i(i)$ 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 $x_0 \oplus 1 = x_0 \oplus g_{0,0}$. 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 $x_1 \oplus 1 \oplus 0 = 1 \oplus x_1 = x_1 \oplus g_{1,1}$.

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 $0 \le n_1 < n_2 < \cdots < n_s \le i-1$ 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 $\textbf{maj}(a_j, b_j, carry_{j-1})$ (where **maj** is the majority function). We use two properties of this function: **(1)** $\textbf{maj}(x,y,0) = xy$ and **(2)** $\textbf{maj}(x,y,1) = x \vee y$. 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

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

The above follows because of the Boolean identity $A \vee BC = (A \vee B)(A \vee C)$, 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

$$\begin{aligned} z_2 &= x_{n_2} \vee \left(z_1 \cdot \prod_{k=n_1+1}^{n_2-1} x_k\right) \\ &= (x_{n_2} \vee z_1) \cdot \prod_{k=n_1+1}^{n_2-1} (x_{n_2} \vee x_k) = \left(x_{n_2} \vee \prod_{k=0}^{n_1-1} (x_{n_1} \vee x_k)\right) \cdot \prod_{k=n_1+1}^{n_2-1} (x_{n_2} \vee x_k) \\ &= \prod_{k=0}^{n_1-1} (x_{n_2} \vee x_{n_1} \vee x_k) \cdot \prod_{k=n_1+1}^{n_2-1} (x_{n_2} \vee x_k) = g_{n_2,i}(x) \end{aligned}$$

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 $z_s \cdot \prod_{k=n_s+1}^{i-1} x_k = g_{i,bin_i(i)}(x)$. Thus it follows that the sum we are looking for is $x_i \oplus g_{i,bin(i)}(x)$.

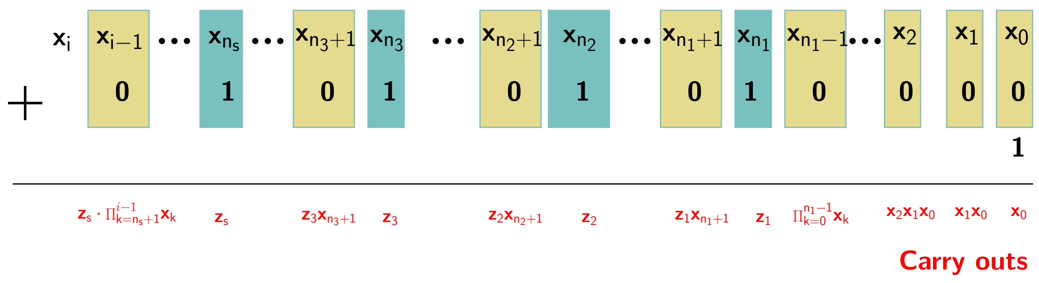


Figure 7: Visual representation of the addition

The expression for $g_{i,bin(i)}(x)$ 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 $0 \to 2^{n-d} - 1$ 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 $t_{n-d-1} \oplus \prod_{k=0}^{n-d-2} t_k$). $\square$

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 (5 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 $\binom{n}{\downarrow d}$ 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 $A_{\mathsf{top}}/A_{\mathsf{bottom}}$ from A_0 should be efficient at all levels of recursion.

2. The ANF representation should be such that we can efficiently use **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 $\chi_{n,d} : H(n, d) \to \left[0, \binom{n}{\downarrow d} - 1\right]$, 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_ℓ, then the two inputs to the circuit are the coefficients at x^D and $x^{D\oplus e_\ell}$. Without loss of generality, let us assume that the ℓ-th bit of D is zero i.e. $D \cdot e_\ell = 0$.

b) The coefficient of x^D is copied as is from A_0 to A_{top} (or if we follow the terminology developed later: from $A[\ell]_b$ to $A[\ell+1]_b$). The coefficients of x^D and $x^{D\oplus e_\ell}$ 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 $x^{D\oplus e_\ell}$ is 0 by assumption.

However note that the size of the vectors $A[\ell]_b$ and $A[\ell+1]_b$ are $\binom{n-\ell}{\downarrow d}$ and $\binom{n-\ell-1}{\downarrow d}$ 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 $0 \le u \le 6$, 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 $A[1]_{10\ldots}$.

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 $d \le n$, and $s \in H(n, d)$, let $s = 2^{i_0} + 2^{i_1} + \cdots$, be the binary expansion of the integer s, where $i_0 > i_1 > \cdots \ge 0$. Then we have*

$$\chi_{n,d}(s) = \binom{i_0}{\downarrow d} + \binom{i_1}{\downarrow d-1} + \cdots$$

where we extend the definition of $\binom{x}{\downarrow y}$ as follows:

$$\binom{x}{\downarrow y} = \begin{cases} \sum_{i=0}^{y} \binom{x}{i} & \text{if } x \geq y, \\ 2^x & \text{otherwise.} \end{cases}$$

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 $\binom{m}{\downarrow d}$, 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 $[0, 2^{i_0})$ is $\binom{i_0}{\downarrow d}$. The number of such integers in the range $[2^{i_0}, 2^{i_0} + 2^{i_1})$ 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 $\binom{i_1}{\downarrow d-1}$. Taking this argument forward for the successive i_2, $i_3, \ldots$, we arrive at the required result. □

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 ℓ-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 $\left[0, \binom{n}{\downarrow d} - 1\right]$ are nodes and each node α in this set is connected with at most $n-d$ types of edges to at most $n-d$ neighbors. An edge of type ℓ, (for $0 \leq \ell < n-d$) connects α to $\beta := \chi_{n,d}\left[\chi_{n,d}^{-1}[\alpha] \oplus e_\ell\right]$ if $hw(\beta) \leq d$ and unconnected otherwise. This is helpful because at step ℓ 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 α and β, 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 $\binom{n}{\downarrow d} \times (n-d)$ such that

$$AM[\alpha, \ell] = \begin{cases} \chi_{n,d}\left[\chi_{n,d}^{-1}[\alpha] \oplus e_\ell\right], & \text{if } hw(\chi_{n,d}^{-1}[\alpha] \oplus e_\ell) \leq d \\ 0 & \text{otherwise.} \end{cases}$$

Thus the ℓ-th recursion step can be re-written from:

```
• For all $n-\ell-1$ bit strings $s$ with hw $\leq d$
    1 $A[\ell+1]_b(\chi_{n,d}(s)) \leftarrow A[\ell]_b(\chi_{n,d}(0 \parallel s))$
    2 If $hw(\chi_{n,d}(1 \parallel s)) \leq d$:
        $A[\ell+1]_{b+e_\ell}(\chi_{n,d}(s)) \leftarrow A[\ell]_b(\chi_{n,d}(0 \parallel s)) \oplus A[\ell]_b(\chi_{n,d}(1 \parallel s))$
    3 Else $A[\ell+1]_{b+e_\ell}(\chi_{n,d}(s)) \leftarrow A[\ell]_b(\chi_{n,d}(0 \parallel s))$
```

to the following equivalent form that uses the AM matrix:

```
• For $\alpha = 0$ to $\binom{n-\ell-1}{\downarrow d} - 1$
    1 $A[\ell+1]_b(\alpha) \leftarrow A[\ell]_b(\alpha)$
```

```
2 If $AM[\alpha, \ell] \neq 0$
      $A[\ell+1]_{b+e_\ell}(\alpha) \leftarrow A[\ell]_b(\alpha) \oplus A[\ell]_b(AM[\alpha, \ell])$
3 Else $A[\ell+1]_{b+e_\ell}(\alpha) \leftarrow A[\ell]_b(\alpha)$
```

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

Using the circuit for smaller functions: The circuit once constructed for some upper limit (n, d) also caters for Boolean functions for any number of variables $n_0 < n$. Since any n_0-variable Boolean function (for $n_0 < n$) is also an n-variable Boolean function, i.e. with the additional variables set to zero, the only thing we need to do is to embed the ANF of the n_0-variable Boolean function as a the ANF of an n-variable Boolean function with appropriate zero padding. This is aided by the fact that $\chi_{n,d}$ has been defined in a manner so that $\chi_{n-1,d}(s)$ is the same as $\chi_{n,d}(0 \parallel s)$ for any $(n-1)$-bit string s. Thus in order to embed any $(n-1)$-variable Boolean function we simply add the coefficient corresponding to s in $\chi_{n,d}(0 \parallel s)$ and place 0 in $\chi_{n,d}(1 \parallel s)$. Since the function $\chi_{n,d}$ is monotonous this would amount to filling up locations $\left[0, \binom{n-1}{\downarrow d} - 1\right]$ with coefficients of the smaller Boolean function and padding the remaining i.e. $\left[\binom{n-1}{\downarrow d}, \binom{n}{\downarrow d} - 1\right]$ locations with zeros. By induction on i, the same applies to any arbitrary $(n-i)$-variable function.

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 $A[3]_{000}$ in the bottom most register at $t = 0$, as in the previous 3 cycles, i.e. $t = -3, -2, -1$, the corresponding $S[i]_t$'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.

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 $\binom{n}{\downarrow d}$ essentially holds a constant value throughout

Table 2: Results for $d = 2, 3, 4$ for the **Polymob1** circuit. Power reported at 1 GHz.

	$d = 4$					$d = 3$				
n	Area GE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (nJ)	Area GE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (nJ)
8	2078	40.57	0.811	0.707	0.014	1468	41.11	1.521	0.479	0.018
9	3535	44.87	1.660	1.121	0.041	2265	41.47	2.903	0.721	0.050
10	5780	46.77	3.274	1.749	0.122	3338	45.48	6.134	1.043	0.141
11	9040	66.95	9.038	2.715	0.366	4682	60.25	15.906	1.461	0.386
12	13525	75.95	20.051	4.020	1.061	6563	48.69	25.367	2.029	1.057
13	20024	61.74	32.167	5.800	3.022	8900	74.46	76.992	2.740	2.834
14	28754	73.61	76.113	8.375	8.661	11792	72.00	148.248	3.620	7.453
15	40706	73.18	150.678	11.854	24.409	15228	74.82	307.361	4.729	19.428
16	56683	79.71	327.449	16.385	67.311	19457	68.95	565.735	6.091	49.976
17	77402	78.07	640.564	22.305	183.016	24253	83.31	1366.117	7.748	127.051
18	102916	90.26	1480.084	29.821	489.015	30839	81.32	2665.014	9.718	318.573
19	134835	93.46	3063.899	39.289	1288.025	37781	76.57	5019.317	11.988	785.846
20	174268	101.48	6652.217	51.141	3352.416	45653	83.87	10994.434	14.695	1926.389

	$d = 2$				
n	Area GE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (nJ)
8	856	49.24	3.445	0.284	0.020
9	1160	51.82	6.996	0.394	0.053
10	1506	47.20	12.461	0.490	0.129
11	1984	58.09	30.265	0.630	0.328
12	2552	69.16	71.511	0.802	0.829
13	3123	65.45	134.761	0.986	2.030
14	3817	70.22	288.464	1.183	4.860
15	4584	70.96	582.227	1.429	11.727
16	5445	71.81	1177.540	1.720	28.208
17	6495	74.33	2346.761	2.057	67.440
18	7575	75.33	4938.032	2.420	158.639
19	8766	86.07	11282.830	2.819	369.585
20	10168	81.94	21481.554	3.286	861.349

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 $[2^{n-d} + (n-d)] \cdot T_{cr}$ 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 $0 \to 1\,/\,1 \to 0$ 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 $E_{total} = E_{dynamic} + E_{static}$. Since static power is independent of clock frequency we can write $E_{static} = P_{static} \cdot T_d$, where P_{static} is the static power consumption and T_d is the total physical time taken to compute the Möbius Transform.

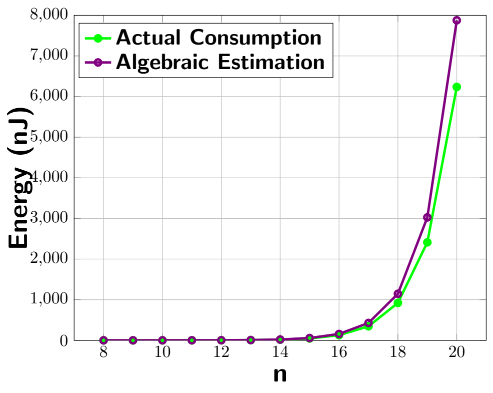
It is not altogether unreasonable to assume that P_{static} is related to the circuit area. In this respect let us look at the combinatorial and sequential elements of the circuit separately. The total number of flip-flops in the circuit can be estimated easily to be equal to $F(n,d) = \sum_{i=1}^{n-d} \binom{n-i}{\downarrow d}$. The total static power due to this component can be estimated to be around $P_{static,seq} = F(n,d) \cdot \alpha_s$, where α_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 $P_{static,comb}$ can still be estimated as the product of some constant α_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 $P_{dynamic} \approx P_{dynamic,seq} = F(n,d) \cdot \beta_s$, for some constant β_s. Combining the above three expressions the energy consumed in the circuit can be written as:

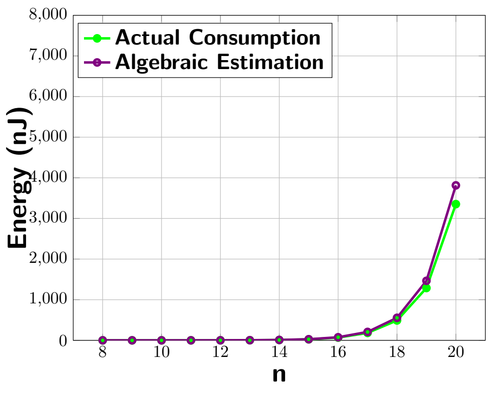
$$E_{\textsf{Polymob1}} = T_d \cdot \big(F(n,d) \cdot \alpha_s + A_c \cdot \alpha_c + F(n,d) \cdot \beta_s\big) \tag{3}$$

We can estimate $A_c = F(n,d) \cdot A_{mux} + (F(n,d) + d \cdot 2^{d-1}) \cdot A_{xor}$, where A_{mux}/A_{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.

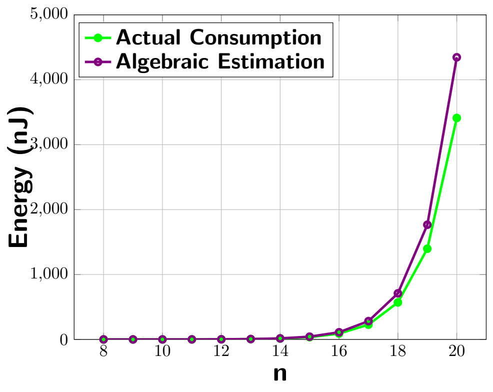
At higher frequencies: The left side plots in Figure 8 show the comparison of the actual energy consumption as reported by the power compiler and that estimated by the algebraic expression in Equation (3), when the clock frequency is 10 MHz. The purple plot which represents the figures obtained by Equation 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 $P_{dynamic}$ itself increases 100-fold so that $P_{dynamic} \cdot T_d$ 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.



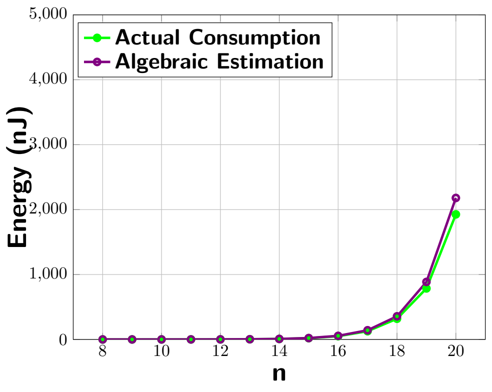
(a) Degree=4, Freq=10 MHz



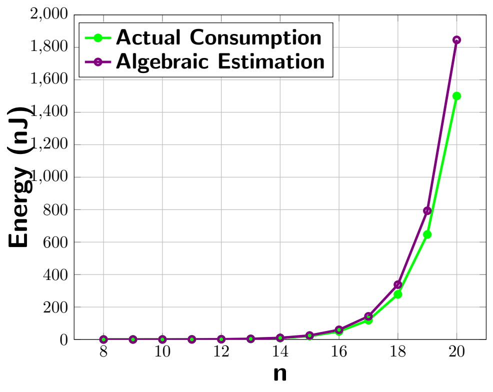
(b) Degree=4, Freq=1 GHz



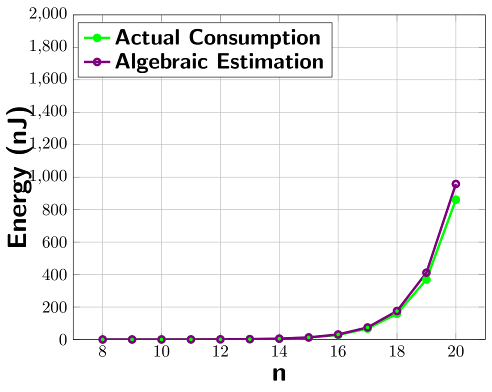
(c) Degree=3, Freq=10 MHz



(d) Degree=3, Freq=1 GHz



(e) Degree=2, Freq=10 MHz



(f) Degree=2, Freq=1 GHz

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 $2^d \to d$ 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 $m \cdot \sum_{i=1}^{n-d} \binom{n-i}{\downarrow d} < m \cdot (n-1)^{d+1} < m \cdot n^{d+1}$. It is not difficult to work out that the total number of xor gates required is $m \cdot \sum_{i=1}^{n-d} \binom{n-i-1}{\downarrow d}$. 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 **Expmob1** 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 **Expmob1**, the **OR** network and the encoder (call this τ_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 $AT = n^{d+2} \cdot 2^{n-d}$.

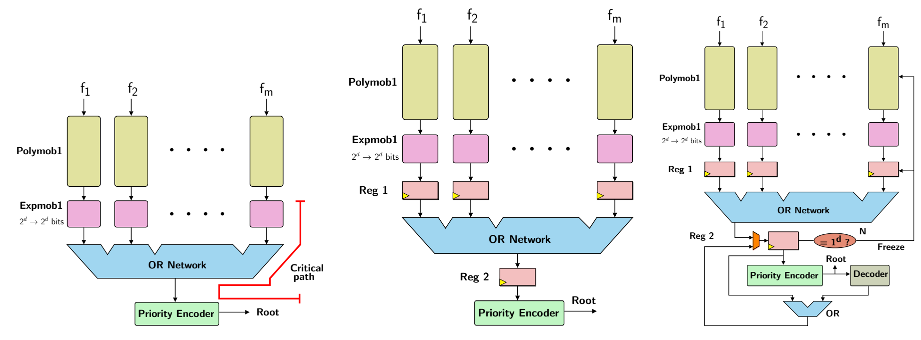
4.2 Polysolve2

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 $u = n - d$ incrementer 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 τ_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



(a) **Polysolve1** (b) **Polysolve2** (c) **Polysolve3**

Figure 9: Circuits for solving m equations

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 $T_1 = T_0 \vee D_0 = 1111\ 1111\ 1111\ 0111$, 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 $T_2 = T_1 \vee D_1 = 1111\ 1111\ 1111\ 1111$ 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 $2 \cdot \log_2(n-d)$. If the underlying clock signal has this period then the total physical time taken to solve the system will be proportional to $2 \cdot \log_2(n-d) \cdot (2^{n-d} + R) \approx 2 \cdot \log_2(n-d) \cdot 2^{n-d}$.

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 [BCC+10, 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 [BCC+13], 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.

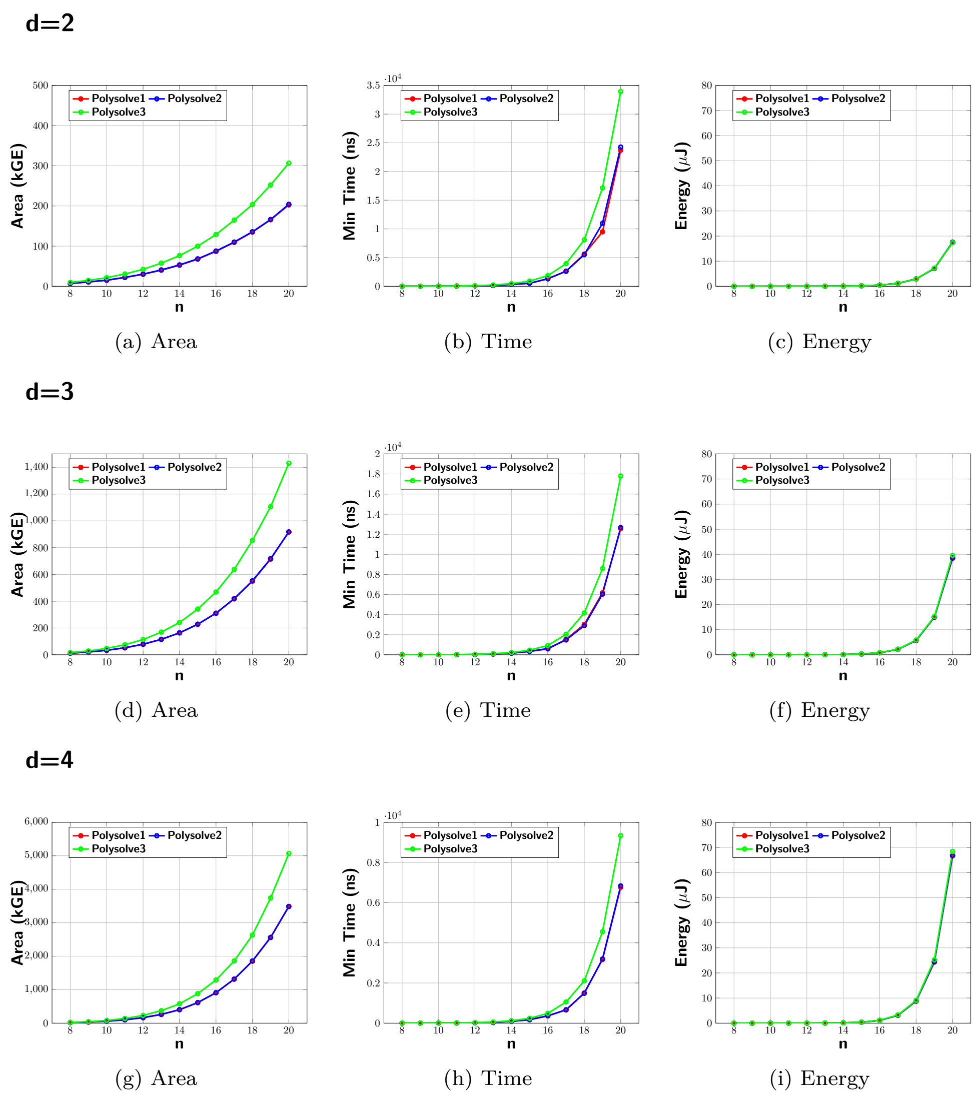


Figure 10: Synthesis results for **Polysolve1**/**Polysolve2**/**Polysolve3** circuits

5.1 Root Expander

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 $\{0,1\}^n \to \{0,1\}^{\binom{n}{\downarrow d}}$. 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_0x_1 and x_2 etc), and so the total hardware overhead is $\binom{n}{\downarrow d} - n$ **AND** gates. The outcome of such a circuit is thus an expanded root $\mathbf{r} \in \{0,1\}^{\binom{n}{\downarrow d}}$.

5.2 Dot-Product

A dot-product essentially takes the ANF vector of a Boolean function and evaluates it over an expanded root $\mathbf{r}$. It can be shown that a simple dot-product over $GF(2)$ will achieve

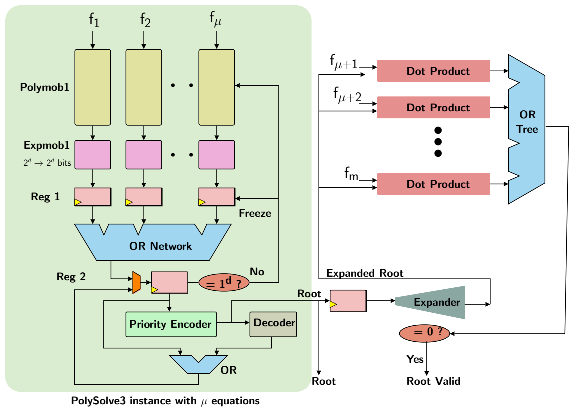


Figure 11: **Polysolve4** Architecture

this. For example when $n = 4, d = 2$, consider the function $f = 1 \oplus x_0 \oplus x_2 \oplus x_0x_1 \oplus x_2x_3$. Using the mapping $\chi_{4,2}$, the corresponding vector description for this function is **v**=1011 0001 001. For the point $r = (x_0, x_1, x_2, x_3) = (1, 0, 1, 1)$, we have, using the $\chi_{4,2}$ map the corresponding description of the expanded root **r**=1111 0001 110. The dot-product $\mathbf{r} \cdot \mathbf{v} = 0$, gives the evaluation of f at the point r. Thus each dot-product will need $\binom{n}{\downarrow d}$ **AND** gates and $\binom{n}{\downarrow d} - 1$ **XOR** gates to compute (in practice the product is usually computed using $\binom{n}{\downarrow d}$ **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 $\log_2 \binom{n}{\downarrow d} \approx d \log_2 n$ 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 r of the first μ 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 **r** of size $\binom{n}{\downarrow d}$. 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 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 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 $\binom{n}{\downarrow d} \in O(n^d)$ 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 $\log_2 \binom{n}{\downarrow d} \approx d \log_2 n$ **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 μ 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 $\sqrt{1/(\pi \cdot 2^{n-1})}$. However using Stirling's approximations it can be shown that $\mathbf{Pr}[|wt(f) - 2^{n-1}| \leq \sqrt{2^{n-1}}] > 0.8$ (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 μ equations. The number of clock cycles that the **Polysolve3** core would take to output its solutions space (i.e. of the first μ equations) is $R + 2^{n-d} \approx 2^{n-\mu} + 2^{n-d}$ 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_{mob}, P_{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

$$E(m,\mu) = (\mu \cdot P_{mob} + (m-\mu) \cdot P_{dp} + C) \cdot (2^{n-\mu} + 2^{n-d}) \cdot T_{clk}, \tag{4}$$

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

Note that Equation (4) presents a high-level overview of the energy consumption. In practice as μ 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 $P_{mob} \gg P_{dp}$, (eg. at 1 GHz, for $n = m = 20$, $d = 4$ and $\mu = 6$, we had $P_{dp} \approx 1.5$ mW and $P_{mob} \approx 70$ mW). So increasing μ 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 μ 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 μ 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_{mob}, P_{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 μ, the energy consumption required to solve a set of n equations of degree d can be expressed asymptotically as proportional to $d \cdot P_{mob} \cdot 2^{n-d}$. Since P_{mob}

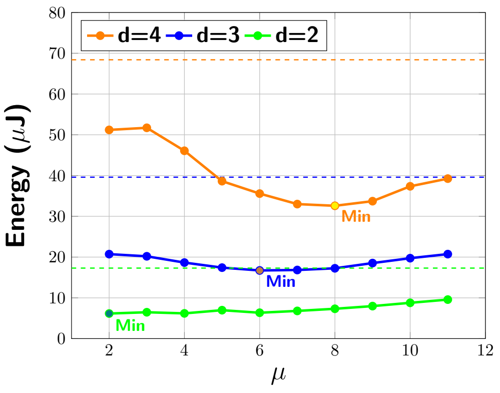


Figure 12: Energy consumption for varying μ for $n = m = 20$. The colored dashed lines show the energy consumed by **Polysolve3** for the corresponding equation systems.

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 $\binom{h}{\downarrow d}$ 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 $\chi_{n,d}^{-1} : \left[0, \binom{n}{\downarrow d} - 1\right] \to H(n, d)$, 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 $\mu \cdot \sum_{i=1}^{n-h} \binom{n-i}{\downarrow d}$ 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 μ **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, μ, 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 $h = \hat{h}$ such that $\hat{h}^2 \cdot 2^{\hat{h}} = n^{d+2}$, then $A \in O(n^{d+2})$ as required, and we have $AT = 4 \cdot n^{d+2} \cdot 2^{n-\hat{h}}$. 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 δ 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^{h-1} 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_{expmob}, P_{dp} be the dynamic power consumed by each **Polymob1** (without **Expmob1**), the **Expmob1** and dot-product circuit respectively,

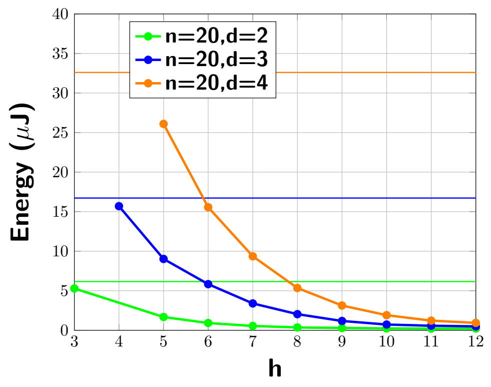


Figure 13: Energy decrease with increasing h for **Polysolve4** solvers for $n = 20$, $h = \mu$. The colored horizontal lines indicate the optimal energy consumption for the full depth **Polysolve4** circuit for the same equation system.

then we can conclude that at high enough clock frequencies, the energy consumption is proportional to

$$E'(m, \mu) = (\mu \cdot P_{mob} + \mu P_{expmob} + (m - \mu) \cdot P_{dp} + C) \cdot (2^{n-\mu} + 2^{n-h}) \cdot T_{clk}, \qquad (5)$$

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

Asymptotically since both $P_{mob}, P_{expmob} \gg P_{dp}$, and since $P_{mob} \propto \sum_{i=1}^{n-h} \binom{n-i-1}{\downarrow d} < C_1 \cdot n^{d+1}$ and $P_{expmob} \propto h^2 \cdot 2^h \approx C_2 \cdot h^2 \cdot 2^h$, 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 $P_{mob} \approx 70 - 75\ mW$ 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 $82 \cdot h^2 \cdot 2^h\ nW$ (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 μJ which is much less than the 32.6 μJ we obtained for the full-depth **Polysolve4** circuit, and around 100 times less than the 68.4 μJ obtained for the **Polysolve3** circuit. [1]

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 $E' < 3h_0^3 \cdot C_2 \cdot 2^n \cdot T_{clk}$.

[1]Due to time constraints we did the actual simulation only upto $h = 12$ 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 upto 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.

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

```
Generate $AM$ $(n,d)$
Input: $n$: Number of variables, $d$: Algebraic degree
Output: The adjacency matrix $AM$ of size $\binom{n}{\downarrow d} \times (n-d)$

for $s \leftarrow$ the k-th string in $H(n,d)$ do
    Compute $\alpha := \chi_{n,d}(s) = \binom{i_0}{\downarrow d} + \binom{i_1}{\downarrow d-1} + \cdots$ /*Assuming $s = 2^{i_0} + 2^{i_1} + \cdots$ */
    for $\ell \leftarrow 1 \rightarrow n-d$ do
        Compute $s' \leftarrow s \oplus e_\ell$
        if $hw(s') \leq d$ then
            Compute $\beta := \chi_{n,d}(s')$
            Assign $AM[\alpha, \ell] \leftarrow \beta$
        end
        else
            Assign $AM[\alpha, \ell] \leftarrow 0$
        end
    end
end
```

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

$$S(n,d) = \begin{array}{ccccccccc} \binom{n}{d} & + & \binom{n}{d-1} & + & \cdots & + & \binom{n}{0} & + \\ \binom{n-1}{d} & + & \binom{n-1}{d-1} & + & \cdots & + & \binom{n-1}{0} & + \\ \binom{n-2}{d} & + & \binom{n-2}{d-1} & + & \cdots & + & \binom{n-2}{0} & + \\ \vdots & \vdots & \ddots & & \ddots & & \vdots & \\ \binom{d}{d} & + & \binom{d}{d-1} & + & \cdots & + & \binom{d}{0} & \end{array}$$

Applying the hockey-stick identity on each column we get

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

Using mathematical induction it is easy to prove the hypothesis $\mathcal{P}(d) : \sum_{i=0}^{d} \binom{n}{i} < n^d$, 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 $\mathcal{P}(d)$ to be true we have

$$\begin{aligned} \mathcal{P}(d+1) : \sum_{i=0}^{d+1} \binom{n}{i} &< n^d + \binom{n}{d+1} \\ &< n^d + \frac{n^{d+1}}{(d+1)!} = n^d \left(1 + \frac{n}{(d+1)!}\right) < n^{d+1} \end{aligned}$$

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.

n	μ	Area KGE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)	n	μ	Area KGE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)
8	2	9.980	54.62	4.37	2.357	0.0002	15	2	201.483	80.36	822.89	43.166	0.4419
	3	12.361	56.39	2.71	3.557	0.0002		3	253.167	81.95	503.50	64.634	0.3971
	4	15.547	61.36	1.96	4.858	0.0002		4	301.172	79.60	326.04	83.143	0.3406
	5	17.343	58.35	1.40	5.517	0.0001		5	353.891	81.47	250.28	95.265	0.2927
9	2	17.345	58.26	9.32	4.007	0.0006	16	2	273.810	85.20	1744.90	57.472	1.1770
	3	21.623	63.35	6.08	5.921	0.0006		3	344.113	88.21	1083.92	92.117	1.1319
	4	26.325	63.70	4.08	7.714	0.0005		4	415.411	86.36	707.46	118.521	0.9709
	5	30.881	61.82	2.97	9.172	0.0004		5	486.599	87.23	535.94	132.367	0.8133
10	2	28.411	59.73	19.11	6.408	0.0021	17	2	372.018	82.40	3375.10	72.115	2.9538
	3	35.737	62.97	12.09	9.557	0.0018		3	466.637	80.37	1975.17	115.223	2.8318
	4	42.833	66.79	8.55	12.398	0.0016		4	563.460	85.59	1402.31	155.530	2.5482
	5	49.947	66.62	6.40	14.450	0.0014		5	665.159	89.91	1104.81	179.444	2.2050
11	2	44.009	67.80	43.39	9.665	0.0062	18	2	497.044	89.69	7347.40	96.226	7.8828
	3	56.271	65.20	25.04	14.402	0.0055		3	619.838	94.31	4635.53	153.796	7.5594
	4	67.118	68.08	17.43	19.338	0.0050		4	749.498	89.44	2930.77	200.594	6.5731
	5	78.388	67.52	12.96	22.084	0.0042		5	870.225	94.38	2319.48	231.616	5.6922
12	2	67.289	71.61	91.66	14.138	0.0181	19	2	651.059	95.90	15712.26	123.955	20.3087
	3	85.850	66.75	51.26	22.190	0.0170		3	814.677	97.02	9537.45	205.160	20.1680
	4	102.337	74.37	38.08	29.480	0.0151		4	980.324	100.14	6562.78	268.407	17.5903
	5	119.146	74.35	28.55	33.264	0.0128		5	1144.174	102.38	5032.18	311.335	15.3027
13	2	100.638	68.34	174.95	20.869	0.0534	20	2	838.381	97.32	31889.82	156.245	51.1984
	3	126.899	68.65	105.45	32.594	0.0501		3	1053.007	106.76	20989.87	263.029	51.7137
	4	152.480	71.14	72.85	42.800	0.0438		4	1266.550	102.77	13470.27	351.538	46.0767
	5	178.574	74.47	57.19	48.137	0.0370		5	1480.936	116.42	11444.55	392.951	38.6286
14	2	142.414	85.40	437.25	30.458	0.1559		6	1696.708	108.59	8895.69	434.253	35.5740
	3	178.953	81.96	251.78	47.662	0.1464		7	1905.735	111.40	8213.30	447.604	33.0009
	4	215.429	89.16	182.60	63.146	0.1293		8	2124.088	108.03	7541.15	467.919	32.5821
	5	252.591	89.36	137.26	72.549	0.1114		9	2337.793	117.02	7908.68	498.876	33.7160

Table 4: Synthesis results for the **Polysolve4** circuit for $d = 3$. Power reported at 1 GHz.

n	μ	Area KGE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)	n	μ	Area KGE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)
8	2	6.514	51.93	4.99	1.855	0.0002	15	2	70.259	75.30	925.29	17.405	0.2139
	3	8.590	53.36	3.42	2.691	0.0002		3	89.858	78.79	645.45	26.429	0.2165
	4	10.401	54.08	2.60	3.164	0.0002		4	110.598	78.62	483.04	31.874	0.1958
	5	12.297	56.68	2.27	3.606	0.0001		5	130.313	79.51	407.09	34.420	0.1762
9	2	10.148	52.36	10.05	2.821	0.0005	16	2	90.243	68.96	1694.76	21.048	0.5173
	3	13.140	58.10	7.44	3.940	0.0005		3	116.456	70.74	1159.00	31.734	0.5199
	4	16.003	59.40	5.70	4.736	0.0005		4	142.741	70.98	872.20	39.131	0.4808
	5	18.860	58.35	4.67	5.271	0.0004		5	168.370	75.69	775.07	42.968	0.4400
10	2	15.017	58.95	22.64	4.112	0.0016	17	2	114.245	84.46	4151.38	27.180	1.3360
	3	19.110	58.39	14.95	5.742	0.0015		3	147.433	88.71	2906.85	42.078	1.3788
	4	23.420	60.09	11.54	6.814	0.0013		4	180.560	90.85	2232.73	51.946	1.2766
	5	27.734	61.07	9.77	7.512	0.0012		5	214.646	96.40	1974.27	59.980	1.2284
11	2	21.483	58.25	44.74	5.622	0.0043	18	2	143.817	85.24	8379.43	34.845	3.4254
	3	27.491	60.21	30.83	8.020	0.0041		3	184.922	85.93	5631.51	51.837	3.3972
	4	33.699	62.23	23.90	9.528	0.0037		4	226.011	84.89	4172.51	62.658	3.0798
	5	39.903	60.77	19.45	10.405	0.0033		5	267.020	86.74	3552.87	68.206	2.7937
12	2	29.538	64.43	98.96	7.850	0.0121	19	2	176.951	90.52	17796.96	41.905	8.2389
	3	38.282	63.23	64.75	11.283	0.0116		3	229.179	85.19	11166.02	63.657	8.3436
	4	46.166	70.66	54.27	13.399	0.0103		4	282.592	95.25	9363.46	81.613	8.0229
	5	55.035	66.86	42.79	14.612	0.0094		5	333.350	96.94	7941.32	84.799	6.9467
13	2	40.042	70.22	215.72	10.106	0.0310	20	2	217.541	97.47	38326.76	52.712	20.7272
	3	51.436	76.39	156.45	15.365	0.0315		3	281.114	92.49	24245.70	77.029	20.1928
	4	62.886	70.52	108.32	18.282	0.0281		4	343.450	87.47	17197.30	94.877	18.6536
	5	74.448	75.14	96.18	20.144	0.0258		5	409.301	94.84	15538.59	106.279	17.4128
14	2	53.475	74.45	457.42	13.535	0.0832		6	473.097	92.87	13694.24	113.402	16.7218
	3	69.084	75.20	308.02	20.829	0.0853		7	536.420	101.86	14185.31	120.851	16.8302
	4	84.936	74.81	229.82	24.060	0.0739		8	595.860	96.24	13008.57	127.639	17.2527
	5	99.137	77.52	198.45	26.821	0.0687		9	660.410	101.60	13524.99	139.206	18.5311

Table 5: Synthesis results for the **Polysolve4** circuit for $d = 2$. Power reported at 1 GHz.

n	μ	Area KGE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)	n	μ	Area KGE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)
8	2	3.369	47.68	6.10	1.022	0.0001	15	2	18.753	61.64	1009.91	4.403	0.0721
	3	4.265	48.07	4.61	1.332	0.0001		3	24.750	64.46	792.08	6.654	0.0818
	4	5.442	52.52	4.20	1.638	0.0001		4	30.766	64.61	661.61	8.242	0.0844
	5	6.378	54.62	3.93	1.839	0.0001		5	36.348	63.87	588.63	9.187	0.0847
9	2	4.562	54.14	13.86	1.408	0.0004	16	2	22.463	65.24	2137.78	6.300	0.2064
	3	5.947	57.22	10.99	1.862	0.0004		3	29.591	70.96	1743.91	8.487	0.2086
	4	7.289	56.65	9.06	2.077	0.0003		4	36.805	68.78	1408.61	9.906	0.2029
	5	8.682	57.29	8.25	2.332	0.0003		5	43.979	76.41	1408.39	11.355	0.2093
10	2	5.985	67.20	34.41	1.783	0.0009	17	2	26.627	74.01	4850.32	7.420	0.4863
	3	7.942	67.10	25.77	2.397	0.0009		3	35.310	79.17	3891.36	10.314	0.5070
	4	9.697	59.74	19.12	2.690	0.0009		4	43.789	79.27	3246.90	11.072	0.4535
	5	11.696	56.96	16.40	3.011	0.0009		5	52.148	82.52	3042.02	12.116	0.4467
11	2	7.860	55.69	57.03	2.418	0.0025	18	2	31.431	75.79	9933.95	8.582	1.1249
	3	10.348	61.94	47.57	3.008	0.0023		3	41.495	75.28	7400.33	11.307	1.1115
	4	12.774	60.75	38.88	3.576	0.0023		4	51.532	82.75	6778.88	13.387	1.0966
	5	15.237	59.50	34.27	3.925	0.0023		5	61.721	79.93	5893.08	15.086	1.1123
12	2	9.941	69.27	141.86	3.373	0.0069	19	2	36.697	81.72	21422.41	10.309	2.7024
	3	13.162	63.42	97.41	3.701	0.0057		3	48.708	67.57	13284.80	13.541	2.6623
	4	16.174	67.64	86.58	4.272	0.0055		4	59.890	88.79	14547.35	15.764	2.5828
	5	19.302	72.35	83.35	4.803	0.0055		5	72.199	71.23	10503.29	17.770	2.6203
13	2	12.494	64.99	266.20	3.733	0.0153	20	2	42.653	84.39	44244.66	11.736	6.1533
	3	16.467	64.89	199.34	4.867	0.0150		3	56.022	82.54	32456.05	16.453	6.4695
	4	20.344	67.36	172.44	5.498	0.0141		4	70.047	82.56	27053.26	18.896	6.1918
	5	24.418	76.39	176.00	6.289	0.0145		5	83.638	87.08	25680.94	23.650	6.9748
14	2	15.455	77.08	631.44	4.499	0.0369		6	97.631	89.53	24936.61	22.800	6.3504
	3	20.268	63.79	391.93	6.015	0.0370		7	111.060	92.73	25068.26	25.137	6.7956
	4	25.063	71.40	365.57	6.795	0.0348		8	125.232	90.86	24190.57	27.490	7.3189
	5	29.819	70.14	323.21	7.448	0.0343		9	139.441	90.40	23882.96	30.188	7.9754

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.

	$d = 2$						$d = 3$				
h	Area KGE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)	h	Area KGE	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)
3	56.677	124.67	32681.49	20.177	5.2892						
4	70.332	119.63	15680.14	22.852	2.9953	4	348.237	152.21	19950.47	119.732	15.6935
5	85.475	122.94	8057.00	25.667	1.6821	5	413.541	153.15	10036.84	137.641	9.0204
6	101.202	124.37	4075.36	28.102	0.9208	6	474.366	158.04	5178.65	178.054	5.8345
7	122.115	130.94	2145.32	33.520	0.5492	7	543.620	164.43	2694.02	207.352	3.3973
8	153.022	141.29	1157.45	43.836	0.3591	8	622.745	174.61	1430.41	249.145	2.0410
9	206.305	143.17	586.42	70.914	0.2905	9	729.193	184.04	753.83	287.649	1.1782
10	312.200	156.39	320.29	116.714	0.2390	10	899.647	191.15	391.48	355.497	0.7281
11	531.863	163.87	167.80	208.395	0.2134	11	1209.696	195.50	200.19	560.684	0.5741
12	996.395	171.78	87.95	435.441	0.2229	12	1859.154	197.57	101.16	969.071	0.4962
	$d = 4$										
5	1621.265	208.87	13688.50	398.157	26.0936						
6	1866.587	213.82	7006.45	474.863	15.5603						
7	2084.936	225.81	3699.67	571.070	9.3564						
8	2372.227	238.72	1955.59	652.621	5.3463						
9	2659.779	247.40	1013.35	765.022	3.1335						
10	2996.840	262.67	537.95	933.736	1.9123						
11	3677.842	264.61	270.96	1192.6	1.2212						
12	4908.635	281.91	144.34	1831.8	0.9378						

Appendix D: Synthesis Results for Expmob1,Expmob2

Table 7: Synthesis results for the **Expmob1**/**Expmob2** circuits. Power reported at 1 GHz.

n	Circuit	Area (μm^2)	Area (kGE)	T_{cr} (ps)	T_{min} (ps)	Power (mW)	Energy (pJ)
6	**Expmob1**	95.600	486.247	91.05	91.05	0.488	0.488
	Expmob2	157.041	798.750	38.53	231.18	0.412	2.472
7	**Expmob1**	233.128	1185.750	118.05	118.05	1.403	1.403
	Expmob2	300.958	1530.750	41.29	289.03	0.773	5.410
8	**Expmob1**	572.473	2911.750	134.02	134.02	4.085	4.085
	Expmob2	573.702	2918.000	46.63	373.04	1.496	11.970
9	**Expmob1**	1333.936	6784.750	169.12	169.12	10.547	10.547
	Expmob2	1138.754	5792.000	48.78	439.02	2.947	26.525
10	**Expmob1**	3227.910	16418.000	208.57	208.57	30.092	30.092
	Expmob2	2255.831	11473.750	52.82	528.20	5.818	58.181
11	**Expmob1**	7855.473	39955.000	257.55	257.55	85.621	85.621
	Expmob2	4486.152	22817.749	56.12	617.32	11.494	126.432
12	**Expmob1**	18784.420	95542.500	304.49	304.49	237.821	237.821
	Expmob2	9046.868	46014.749	60.50	726.00	22.865	274.384
13	**Expmob1**	46181.007	234888.750	453.37	453.37	568.696	568.696
	Expmob2	17860.706	90844.247	60.05	780.65	45.437	590.687
14	**Expmob1**	110026.212	559622.251	509.58	509.58	1098.835	1098.835
	Expmob2	35611.803	181130.994	64.77	906.78	90.101	1261.414

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

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

		$d = 4$					$d = 3$				
n	Circuit	Area (kGE)	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)	Area (kGE)	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)
8	**Polysolve1**	16.576	49.18	0.98	5.824	0.0001	11.805	47.33	1.75	3.907	0.0001
	Polysolve2	17.619	48.30	1.06	6.523	0.0001	12.334	46.33	1.81	4.217	0.0002
	Polysolve3	23.398	70.25	1.97	7.694	0.0002	16.901	66.15	2.98	5.013	0.0002
9	**Polysolve1**	31.515	61.62	2.28	10.306	0.0004	20.359	54.33	3.80	6.608	0.0005
	Polysolve2	32.896	62.05	2.42	11.082	0.0004	21.066	53.33	3.84	6.953	0.0005
	Polysolve3	43.996	86.10	3.96	14.245	0.0005	28.364	78.26	6.18	8.085	0.0006
10	**Polysolve1**	57.685	69.68	4.88	18.351	0.0013	33.175	52.49	7.09	10.625	0.0014
	Polysolve2	58.516	69.53	5.01	18.703	0.0013	33.761	53.58	7.34	10.962	0.0015
	Polysolve3	79.632	91.00	7.28	21.804	0.0015	47.450	87.42	12.68	12.356	0.0017
11	**Polysolve1**	99.034	55.81	7.53	29.951	0.0040	51.578	59.87	15.81	15.963	0.0042
	Polysolve2	99.996	61.50	8.43	30.383	0.0041	52.477	57.69	15.35	16.344	0.0043
	Polysolve3	139.714	99.33	14.50	35.960	0.0049	74.780	92.76	25.51	17.685	0.0047
12	**Polysolve1**	163.072	63.52	16.77	48.516	0.0128	78.050	58.69	30.58	24.201	0.0126
	Polysolve2	164.813	62.42	16.60	49.358	0.0130	79.222	54.43	28.47	24.591	0.0128
	Polysolve3	231.275	102.78	28.37	52.569	0.0139	113.831	95.53	50.92	26.127	0.0136
13	**Polysolve1**	260.169	65.06	33.90	77.411	0.0403	114.578	73.69	76.20	35.416	0.0366
	Polysolve2	261.486	63.93	33.44	78.132	0.0407	115.318	75.48	78.20	35.908	0.0371
	Polysolve3	372.485	107.27	57.28	83.274	0.0434	169.211	105.02	109.96	37.626	0.0389
14	**Polysolve1**	401.092	86.48	89.42	119.916	0.1240	163.204	77.76	160.11	50.536	0.1041
	Polysolve2	402.774	86.68	89.80	120.761	0.1249	163.953	77.76	160.26	51.022	0.1051
	Polysolve3	578.104	115.28	120.81	126.689	0.1310	241.382	105.22	218.12	53.265	0.1097
15	**Polysolve1**	613.099	81.59	167.99	178.653	0.3678	227.500	83.39	342.57	70.985	0.2916
	Polysolve2	614.177	84.58	174.32	179.428	0.3694	228.338	80.84	332.25	71.420	0.2934
	Polysolve3	881.542	115.42	239.38	186.955	0.3849	341.370	110.99	457.61	75.146	0.3087
16	**Polysolve1**	906.754	88.38	363.07	264.163	1.0852	310.420	70.43	577.88	96.845	0.7946
	Polysolve2	910.589	88.52	363.82	265.064	1.0889	310.099	76.10	624.55	97.275	0.7981
	Polysolve3	1288.795	118.22	487.54	273.345	1.1229	468.551	112.82	927.49	100.740	0.8266
17	**Polysolve1**	1315.172	79.99	656.32	379.680	3.1153	417.223	93.62	1535.18	132.329	2.1699
	Polysolve2	1315.516	81.29	667.15	380.764	3.1242	419.061	90.15	1478.46	132.354	2.1703
	Polysolve3	1859.152	128.32	1055.05	397.604	3.2623	637.230	124.26	2039.73	134.859	2.2114
18	**Polysolve1**	1852.910	91.64	1502.71	535.199	8.7762	549.628	92.24	3023.90	173.735	5.6955
	Polysolve2	1854.821	90.74	1488.14	536.014	8.7896	552.144	88.42	2898.85	174.157	5.7094
	Polysolve3	2630.265	128.36	2107.16	552.747	9.0640	854.195	126.92	4163.10	179.275	5.8772
19	**Polysolve1**	2559.844	96.69	3169.79	742.491	24.3411	714.216	94.08	6167.13	227.982	14.9447
	Polysolve2	2561.273	97.32	3190.64	743.608	24.3777	717.534	92.17	6042.11	228.667	14.9896
	Polysolve3	3738.741	138.64	4547.67	769.867	25.2385	1105.275	130.95	8586.52	231.834	15.1972
20	**Polysolve1**	3483.777	103.19	6764.31	1016.707	66.6472	917.092	95.78	12555.70	293.383	38.4593
	Polysolve2	3485.519	104.23	6832.69	1017.967	66.7298	917.558	96.59	12662.08	293.985	38.5382
	Polysolve3	5067.633	142.41	9338.11	1043.130	68.3793	1430.485	135.74	17796.74	302.021	39.5917

Table 9: Synthesis results for $d = 2$ for the **Polysolve1**/**Polysolve2**/**Polysolve3** circuits.

n	Circuit	Area (kGE)	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)	n	Area (kGE)	T_{cr} (ps)	T_{min} (ns)	Power (mW)	Energy (μJ)
8	**Polysolve1**	6.825	54.60	3.82	2.390	0.0002	15	67.639	63.50	521.02	21.608	0.1773
	Polysolve2	7.129	54.49	3.92	2.539	0.0002		68.253	63.81	523.69	21.934	0.1800
	Polysolve3	9.561	75.43	5.88	2.766	0.0002		99.913	110.51	908.39	22.013	0.1806
9	**Polysolve1**	10.592	61.78	8.34	3.499	0.0005	16	87.013	81.31	1333.32	28.442	0.4664
	Polysolve2	10.946	63.38	8.68	3.633	0.0005		87.722	79.22	1299.21	28.788	0.4721
	Polysolve3	14.599	87.10	12.54	4.039	0.0005		128.802	113.30	1859.71	28.375	0.4653
10	**Polysolve1**	14.918	56.44	14.90	4.887	0.0013	17	109.243	80.49	2638.70	35.017	1.1480
	Polysolve2	15.293	53.45	14.22	5.072	0.0013		110.079	79.91	2619.85	35.494	1.1636
	Polysolve3	21.499	92.16	25.25	5.226	0.0014		164.822	120.03	3936.98	35.135	1.1518
11	**Polysolve1**	21.818	64.94	33.83	7.037	0.0037	18	134.939	85.62	5612.56	44.435	2.9128
	Polysolve2	22.227	63.42	33.17	7.187	0.0037		135.727	83.92	5501.29	44.790	2.9361
	Polysolve3	30.582	91.18	48.51	7.376	0.0038		203.528	123.36	8088.72	44.059	2.8881
12	**Polysolve1**	29.812	68.97	71.31	9.678	0.0100	19	165.645	72.36	9485.60	53.935	7.0703
	Polysolve2	30.370	68.56	71.03	9.966	0.0103		166.119	83.69	10971.01	54.284	7.1161
	Polysolve3	42.265	98.14	102.65	9.760	0.0101		252.294	130.64	17127.95	55.161	7.2310
13	**Polysolve1**	40.159	68.50	141.04	12.886	0.0265	20	202.519	90.44	23709.93	66.604	17.4611
	Polysolve2	40.713	67.87	139.88	13.116	0.0270		204.073	92.54	24260.66	67.036	17.5742
	Polysolve3	57.725	106.39	220.44	13.906	0.0286		306.535	129.47	33944.70	66.084	17.3248
14	**Polysolve1**	52.478	77.81	319.64	17.036	0.0700						
	Polysolve2	53.252	79.29	325.88	17.281	0.0710						
	Polysolve3	76.379	111.85	461.05	17.169	0.0705						

References

[ARS$^+$15] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory* *and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 430–454, 2015.

[Bar09] Gregory Bard, editor. *Algebraic Cryptanalysis*. Springer, 2009.

[BBR15] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Exploring energy efficiency of lightweight block ciphers. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015*, volume 9566 of *Lecture Notes in Computer Science*, pages 178–194. Springer, 2015.

[BCC$^+$10] Charles Bouillaguet, Hsieh-Chung Chen, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, Adi Shamir, and Bo-Yin Yang. Fast exhaustive search for polynomial systems in F_2. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010, 12th International Workshop, Santa Barbara, CA, USA, August 17-20, 2010. Proceedings*, volume 6225 of *Lecture Notes in Computer Science*, pages 203–218. Springer, 2010.

[BCC$^+$13] Charles Bouillaguet, Chen-Mou Cheng, Tung Chou, Ruben Niederhagen, and Bo-Yin Yang. Fast exhaustive search for quadratic systems in F_2 on FPGAs. In Tanja Lange, Kristin E. Lauter, and Petr Lisonek, editors, *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers*, volume 8282 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2013.

[BDL$^+$21] Christof Beierle, Patrick Derbez, Gregor Leander, Gaëtan Leurent, Håvard Raddum, Yann Rotella, David Rupprecht, and Lukas Stennes. Cryptanalysis of the GPRS encryption algorithms GEA-1 and GEA-2. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 155–183. Springer, 2021.

[BDT22] Charles Bouillaguet, Claire Delaplace, and Monika Trimoska. A simple deterministic algorithm for systems of quadratic polynomials over f_2. In Karl Bringmann and Timothy M. Chan, editors, *5th Symposium on Simplicity in Algorithms, SOSA@SODA 2022, Virtual Conference, January 10-11, 2022*, pages 285–296. SIAM, 2022.

[BFSS13] Magali Bardet, Jean-Charles Faugère, Bruno Salvy, and Pierre-Jean Spaenlehauer. On the complexity of solving quadratic boolean systems. *J. Complex.*, 29(1):53–75, 2013.

[BKW19] Andreas Björklund, Petteri Kaski, and Ryan Williams. Solving systems of polynomial equations over GF(2) by a parity-counting self-reduction. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019, July 9-12, 2019, Patras, Greece*, volume 132 of *LIPIcs*, pages 26:1–26:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[BMP$^+$06] Andrey Bogdanov, M. C. Mertens, Christof Paar, Jan Pelzl, and Andy Rupp. A parallel hardware architecture for fast gaussian elimination over GF(2). In *14th IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM 2006), 24-26 April 2006, Napa, CA, USA, Proceedings*, pages 237–248. IEEE Computer Society, 2006.

[Bou22] Charles Bouillaguet. Boolean polynomial evaluation for the masses. *IACR Cryptol. ePrint Arch.*, page 1412, 2022.

[CB07] Nicolas T. Courtois and Gregory V. Bard. Algebraic cryptanalysis of the data encryption standard. In Steven D. Galbraith, editor, *Cryptography and Coding, 11th IMA International Conference, Cirencester, UK, December 18-20, 2007, Proceedings*, volume 4887 of *Lecture Notes in Computer Science*, pages 152–169. Springer, 2007.

[CB10] Thomas W. Cusick and Yuri L. Borissov. A refinement of cusick–cheon bound for the second order binary reed–muller code. *Discrete Mathematics*, 310(24):3537–3543, 2010.

[CKPS00] Nicolas T. Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 392–407. Springer, 2000.

[Din21] Itai Dinur. Cryptanalytic applications of the polynomial method for solving multivariate equation systems over GF(2). In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 374–403. Springer, 2021.

[DRS20] Christoph Dobraunig, Yann Rotella, and Jan Schoone. Algebraic and higher-order differential cryptanalysis of pyjamask-96. *IACR Trans. Symmetric Cryptol.*, 2020(1):289–312, 2020.

[DS11] Itai Dinur and Adi Shamir. An improved algebraic attack on hamsi-256. In Antoine Joux, editor, *Fast Software Encryption - 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011*, volume 6733 of *Lecture Notes in Computer Science*, pages 88–106. Springer, 2011.

[HST$^+$21] Kai Hu, Siwei Sun, Yosuke Todo, Meiqin Wang, and Qingju Wang. Massive superpoly recovery with nested monomial predictions. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 392–421. Springer, 2021.

[Jon96] CH Jones. Generalized hockey stick identities and n-dimensional block walking. *Fibonacci Quarterly*, 34(3):280–288, 1996.

[JV17] Antoine Joux and Vanessa Vitse. A crossbred algorithm for solving boolean polynomial systems. In Jerzy Kaczorowski, Josef Pieprzyk, and Jacek Pomykala, editors, *Number-Theoretic Methods in Cryptology - First International Conference, NuTMiC 2017, Warsaw, Poland, September 11-13, 2017, Revised Selected Papers*, volume 10737 of *Lecture Notes in Computer Science*, pages 3–21. Springer, 2017.

[KPG99] Aviad Kipnis, Jacques Patarin, and Louis Goubin. Unbalanced oil and vinegar signature schemes. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 206–222. Springer, 1999.

[LPT$^+$17] Daniel Lokshtanov, Ramamohan Paturi, Suguru Tamaki, R. Ryan Williams, and Huacheng Yu. Beating brute force for systems of polynomial equations over finite fields. In Philip N. Klein, editor, *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, January 16-19*, pages 2190–2202. SIAM, 2017.

[MMR$^+$15] Mayler Martins, Jody Maick Matos, Renato P. Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. Open cell library in 15nm freepdk technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15, page 171–178, New York, NY, USA, 2015. Association for Computing Machinery.

[PBB$^+$21] Christian Pilato, Stanislav Böhm, Fabien Brocheton, Jerónimo Castrillón, Riccardo Cevasco, Vojtech Cima, Radim Cmar, Dionysios Diamantopoulos, Fabrizio Ferrandi, Jan Martinovic, Gianluca Palermo, Michele Paolino, Antonio Parodi, Lorenzo Pittaluga, Daniel Raho, Francesco Regazzoni, Katerina Slaninová, and Christoph Hagleitner. EVEREST: A design environment for extreme-scale big data analytics on heterogeneous platforms. In *Design, Automation & Test in Europe Conference & Exhibition, DATE 2021, Grenoble, France, February 1-5, 2021*, pages 1320–1325. IEEE, 2021.

[Tho79] Clark D. Thompson. Area-time complexity for VLSI. In Michael J. Fischer, Richard A. DeMillo, Nancy A. Lynch, Walter A. Burkhard, and Alfred V. Aho, editors, *Proceedings of the 11h Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1979, Atlanta, Georgia, USA*, pages 81–88. ACM, 1979.