Equation Solvers over GF(2)

Subhadeep Banik

University of Lugano

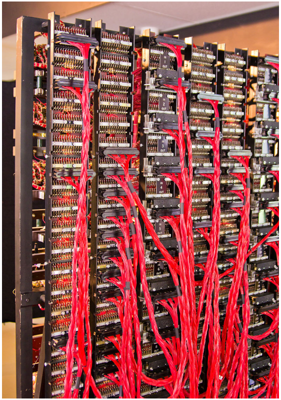
Security Privacy and Applied Cryptographic Engineering,

Indian Institute of Technology, Roorkee

14th December 2023

Sustainable Cryptanalysis

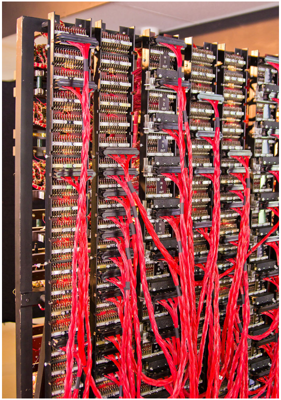
- The idea of making dedicated machines to attack ciphers is not new.



Source: Wikipedia

Sustainable Cryptanalysis

- Why should we care to make circuits ?



Source: Wikipedia

In Software

- Most general purpose CPU's have the following structure
  [A] A processing unit having logic gates and registers
  [B] A control unit having an instruction register and a program counter
  [C] Primary memory that stores data and instructions
  [D] Secondary memory, usually an external mass storage.

- Any computational step of the algorithm
  → First broken down into a sequence of instructions
  → Resides in the control unit.
  → Fetches data from the primary/secondary memory,
  → Processed in the processing unit.

In Software

- Dedicated Circuit
  [A] Collection of logic elements
  [B] Assembled specifically for the given task
  [C] Thus executes them with much greater efficiency.
  [D] Faster and more energy efficient.

- Gaussian Elimination of a 656×656 matrix
  $\rightarrow$ Dedicated circuit [SMITH, Bogdanov et al. 07]
  $\rightarrow$ Requires 86 ms.
  $\rightarrow$ On a Linux 800 MHz PIII PC would take around 40 minutes,
  $\rightarrow$ We can execute computationally heavy tasks on such circuits.

Introduction: Solving an Equation System

- Given m eqns $P_1, P_2, \ldots, P_m$ of n variables over GF(2) of max degree d.
  - $\rightarrow$ Usually $m = n$, sometimes $m > n$
  - $\rightarrow$ Each equation is a multivariate polynomial over GF(2)
  - $\rightarrow$ The algebraic degree d is usually small.
  - $\rightarrow$ Task: find a common root: $r \in \{0,1\}^n$ such that $P_i(r) = 0, \ \forall \ i$.

- Problem arises in many cryptographic contexts.
  - $\rightarrow$ Block ciphers with low multiplicative complexities like LowMC
  - $\rightarrow$ Given single pt/ct: solving low degree polynomials.
  - $\rightarrow$ Signature schemes like UOV.
  - $\rightarrow$ Cryptanalysis: solving quadratic polynomials over GF(2).

Linear Systems

If Equations are Linear ($d = 1$)

LSE (m equations, n variables)

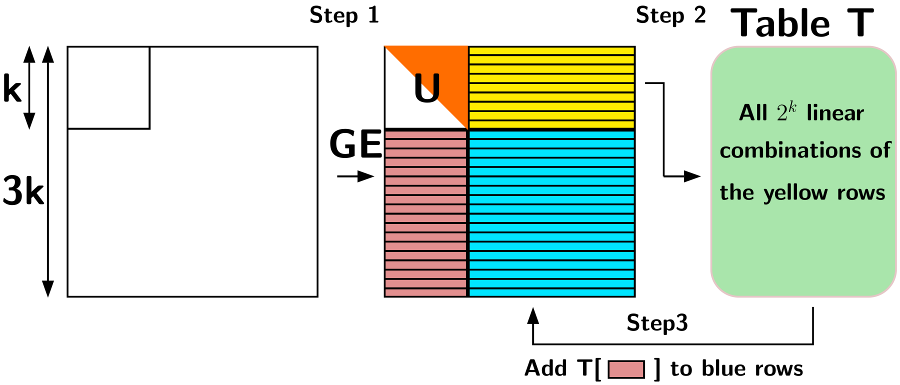
- Typical LSE

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

- Equivalently $A\vec{x} = \vec{b}$

- Linear equations can be Solved by Gaussian Elimination (GE) efficiently.
- GE takes n^3 operations in the worst case.
- Given a linear system of form $A\vec{x} = \vec{b}$
  - $\rightarrow$ Convert to equivalent system $U \cdot \vec{x} = \vec{b'}$, where U is upper-triangular.
  - $\rightarrow$ Done by applying elementary row operations.

MR4I



Popular in SW

- used in computer algebra packages like SAGE.
- It is interesting to see how far this cam be applied in HW

Gaussian Elimination

Gaussian_Elimination(A, n)

Input: Matrix $A \in \{0,1\}^{n \times n}$: Input matrix

```
for each column k = 1 → n do
    s := k;
    while a_sk = 0 do
        s := s + 1;
    end
    Swap row a⃗_k with row a⃗_s;
    for each row i = 1 → n do
        if i ≠ k and a_ik = 1 then
            a_ij := a_ij ⊕ a_kj
        end
    end
end
```

SMITH - Architecture

- GE requires 2 operations: row addition/row swap

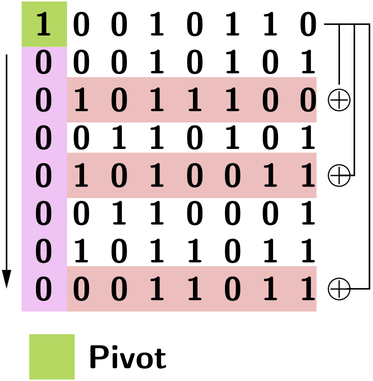
1	0	0	1	0	1	1	0
0	0	0	1	0	1	0	1
1	1	0	0	1	0	1	0
0	0	1	1	0	1	0	1
1	1	0	0	0	1	0	1
0	0	1	1	0	0	0	1
0	1	0	1	1	0	1	1
1	0	0	0	1	1	0	1

Pivot

- Pivot is the top-left element of unprocessed rows.

SMITH - Architecture

- GE requires 2 operations: row addition/row swap



- Column sweep once pivot is fixed.

SMITH - Architecture

- GE requires 2 operations: row addition/row swap

1	0	0	1	0	1	1	0
0	0	0	1	0	1	0	1
0	1	0	1	1	1	0	0
0	0	1	1	0	1	0	1
0	1	0	1	0	0	1	1
0	0	1	1	0	0	0	1
0	1	0	1	1	0	1	1
0	0	0	1	1	0	1	1

Pivot

- Pivot can not be zero. So swap with next available row which is **unprocessed**.

SMITH - Architecture

- GE requires 2 operations: row addition/row swap

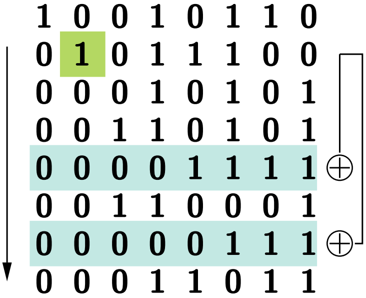
1	0	0	1	0	1	1	0
0	1	0	1	1	1	0	0
0	0	0	1	0	1	0	1
0	0	1	1	0	1	0	1
0	1	0	1	0	0	1	1
0	0	1	1	0	0	0	1
0	1	0	1	1	0	1	1
0	0	0	1	1	0	1	1

Pivot

- Swap is done. Ready for next row operation.

SMITH - Architecture

- GE requires 2 operations: row addition/row swap



Pivot

- Second column is cleared.

Circuit Issues

- Assume each element is stored in a single flip-flop.

1	0	0	1	0	1	1	0
0	0	0	1	0	1	0	1
0	1	0	1	1	1	0	0
0	0	1	1	0	1	0	1
0	1	0	1	0	0	1	1
0	0	1	1	0	0	0	1
0	1	0	1	1	0	1	1
0	0	0	1	1	0	1	1

Pivot

- Pivot is ever changing. How to keep track of it ?
- Can not swap with already processed row...
- How to select next row for swap?

SMITH - Architecture

- Initially both circuit and state are same
- Two additional registers Vec1 and Vec2
- To keep track of control flow.

1	0	0	1	0	1	1	0
0	0	0	1	0	1	0	1
1	1	0	0	1	0	1	0
0	0	1	1	0	1	0	1
1	1	0	0	0	1	0	1
0	0	1	1	0	0	0	1
0	1	0	1	1	0	1	1
1	0	0	0	1	1	0	1

Pivot

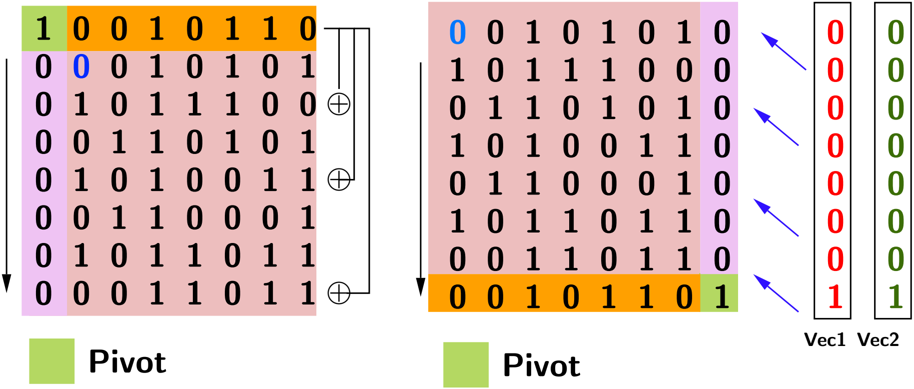
1	0	0	1	0	1	1	0
0	0	0	1	0	1	0	1
1	1	0	0	1	0	1	0
0	0	1	1	0	1	0	1
1	1	0	0	0	1	0	1
0	0	1	1	0	0	0	1
0	1	0	1	1	0	1	1
1	0	0	0	1	1	0	1

Pivot

Vec1	Vec2
0	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0

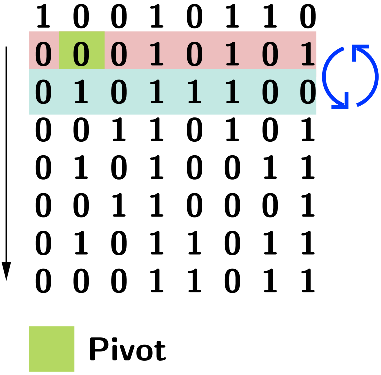
SMITH - Architecture

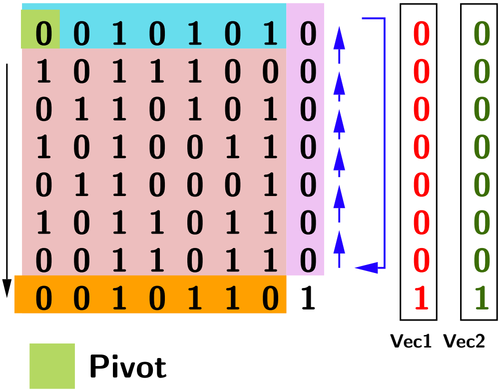
- Diagonal flow brings a_{11} to bottom-left (next pivot shifts to (1,1)!!!)
- The parts in red, blue and orange move accordingly.
- Orange part moves without xor operation/ purple is all zero



SMITH - Architecture

- Xor rule: New $a_{ij} = a_{i+1,\ j+1} \oplus a_{1,\ j+1} \cdot a_{i+1,1}$
- Vec1=Vec1 $\ll$ 1 $||$ 1 (last row is processed).
- Vec2=Vec2 $\lll$ 1 (last row is current row).





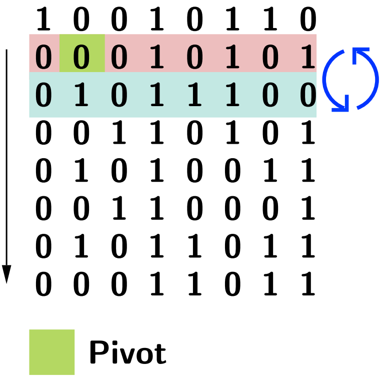
SMITH - Xor rule

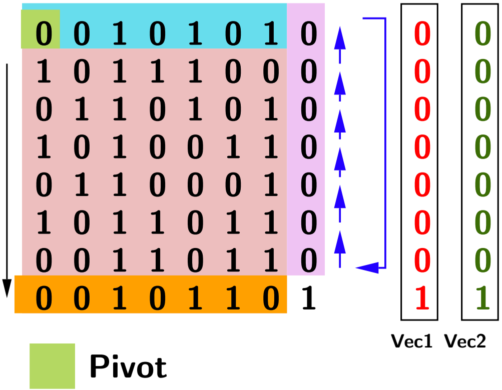
Row operations

$$\begin{bmatrix} 1 & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \rightarrow \begin{bmatrix} a_{22} \oplus (a_{12} \cdot a_{21}) & a_{23} \oplus (a_{13} \cdot a_{21}) & \cdots & 0 \\ a_{32} \oplus (a_{12} \cdot a_{31}) & a_{33} \oplus (a_{13} \cdot a_{31}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n2} \oplus (a_{12} \cdot a_{n1}) & a_{n3} \oplus (a_{13} \cdot a_{n1}) & \cdots & 0 \\ a_{12} & a_{13} & \cdots & 1 \end{bmatrix}$$

SMITH - Architecture

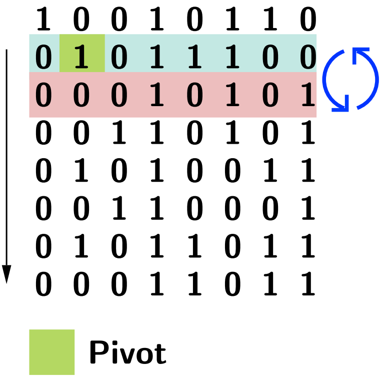
- Pivot is already at (1,1) and zero $\rightarrow$ Now row swaps are required
- Instead of row swap, we circularly rotate unprocessed rows till $a_{11} \neq 0$.
- Need to do same operation to the $\vec{b}$ in $A\vec{x} = \vec{b}$.

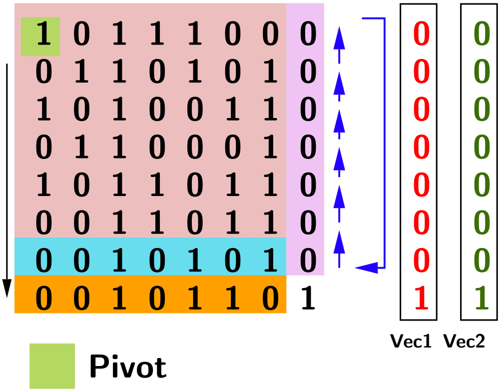




SMITH - Architecture

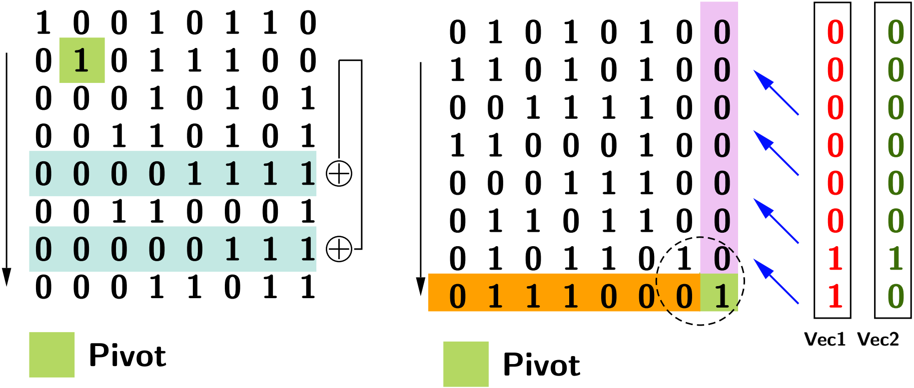
- After one swap pivot is already non-zero
- Ready for next row operation.
- Again we do diagonal movement on unprocessed rows.





SMITH - Architecture

- After row operation: last 2 cols are cleared.
- Bottom left corner becomes identity.
- Stops when Vec1 is all one.



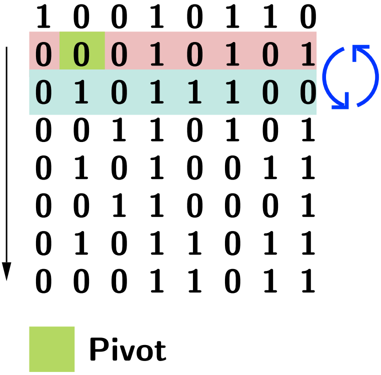
SMITH - Swap rule

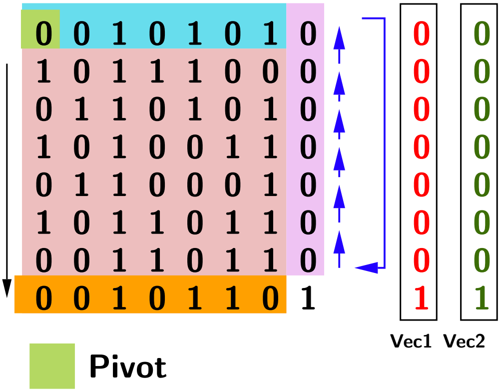
Row Swap

$$\begin{bmatrix} \vec{a_1} \\ \vec{a_2} \\ \vec{a_3} \\ \vdots \\ {\color{blue}\vec{a_i}} \\ \\ {\color{red}\vec{a_{i+1}}} \\ {\color{red}\vec{a_{i+2}}} \\ {\color{red}\vec{a_{i+3}}} \\ {\color{red}\vdots} \\ {\color{red}\vec{a_n}} \end{bmatrix} \rightarrow \begin{bmatrix} {\color{blue}\vec{a_i}} \\ \vec{a_1} \\ \vec{a_2} \\ \vdots \\ \vec{a_{i-1}} \\ \\ {\color{red}\vec{a_{i+1}}} \\ {\color{red}\vec{a_{i+2}}} \\ {\color{red}\vec{a_{i+3}}} \\ {\color{red}\vdots} \\ {\color{red}\vec{a_n}} \end{bmatrix}$$

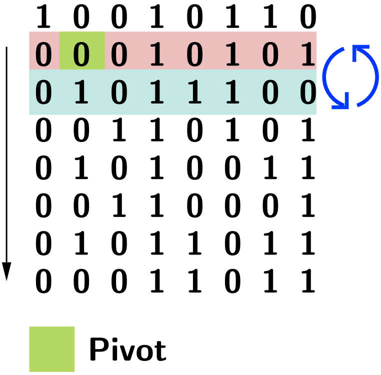
SMITH - RUNTIME

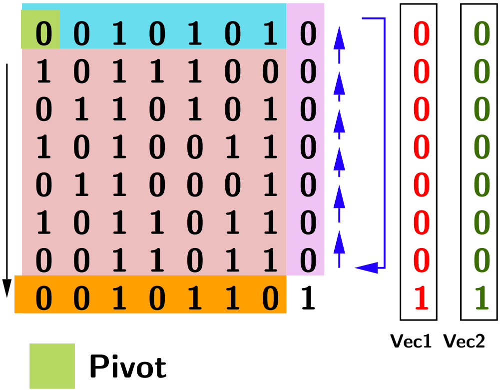
- What is the average Runtime for $n \times n$ matrix???
- At least n row xors are required.
- Additional time depends on number of row swaps.





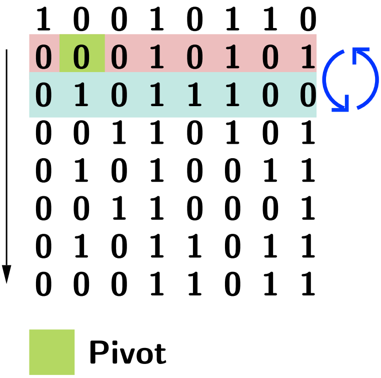
- If Pivot == 1 ($p = \frac{1}{2}$), no swaps are required (zero additional time)
- After one circular rotate if pivot==1, no further swaps are necessary.
- Thus one additional cycle in this case ($p = \frac{1}{4}$).

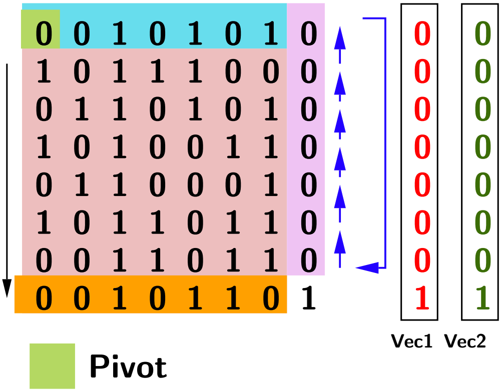




SMITH - RUNTIME

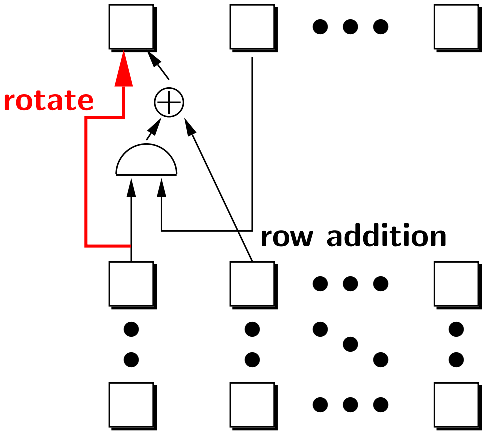
- If Pivot $==1$ after t rotations $(p=\frac{1}{2^{t+1}})$, t additional time.
- $E=0\cdot\frac{1}{2}+1\cdot\frac{1}{4}+2\cdot\frac{1}{8}+\cdots+t\cdot\frac{1}{2^{t+1}}+\cdots\approx 1$.
- Every row xor needs one extra cycle on average $\rightarrow$ $2n$ cycles.





SMITH - Each flip-flop

- Constant Depth circuit: enables low critical path even as $n \to \infty$
- Control signals to figure out which input overwrites the flip-flop
- Use Vec1 and Vec2 to design control signals



SMITH - Use cases

Matrix Square and Full rank

- Stops when Vec1 is all one...
- At the end of computation the FF array holds the identity matrix
- Same operations done on $\vec{b}$.
- If $\vec{b^*}$ is the final state of $\vec{b}$
  $\rightarrow \vec{x} = \vec{b^*}$ is the unique solution of $A\vec{x} = \vec{b}$.

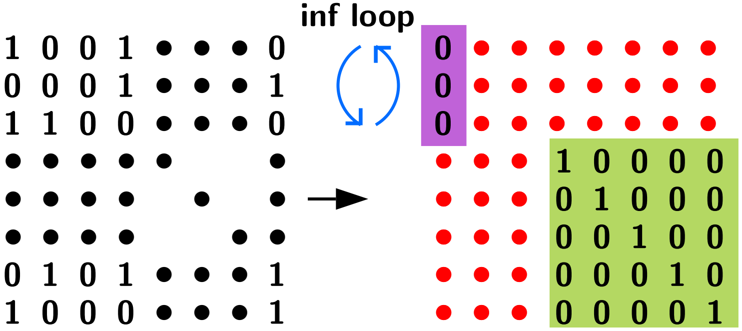
Proof

Original Equation $A\vec{x} = \vec{b}$
Unique solution to this is $\vec{x} = A^{-1}\vec{b}$
$A \rightarrow I$ is only possible
$\rightarrow$ Iff product of all linear operations on A equals A^{-1}
Same operations on b gives $\vec{b} \rightarrow \vec{b^*} = A^{-1}\vec{b}$
QED

SMITH - Use cases

Matrix Square and NOT Full rank

- Never Stops → Vec1 is never all one...
- At the end of computation bottom part holds the identity matrix
- Does not yield any meaningful solution.
- Additional counter logic required to stop infinite loop
  → Stop if counter > # Remaining rows.



SMITH - Use cases

Matrix Non-Square and Overdefined

- That is $m > n$, there are more equations than variables.
- The system is solvable iff

$$A \to \left(\frac{0}{I_n}\right), \quad \vec{b} \to \left(\frac{0}{\vec{b^*}}\right)$$

- Again Additional counter logic required to stop infinite loop
  $\to$ Stop if counter $> m - n$.

- No solution if for any $\vec{t} \neq 0$:

$$A \to \left(\frac{0}{I_n}\right), \quad \vec{b} \to \left(\frac{\vec{t}}{\vec{b^*}}\right)$$

SMITH - Other Uses

Matrix Multiplication

- Observe the following

$$D = \begin{pmatrix} I_n & A & 0 \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}, \quad D^{-1} = \begin{pmatrix} I_n & A & A \cdot B \\ 0 & I_n & B \\ 0 & 0 & I_n \end{pmatrix}$$

- Matrix multiplication $A \cdot B$ is possible given $3n \times 3n$ space
- Also observe if

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}, \quad B = \begin{pmatrix} \vec{b_1} \\ \vec{b_2} \\ \vdots \\ \vec{b_n} \end{pmatrix}$$

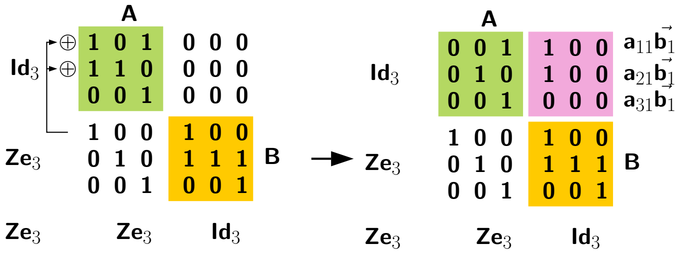
- Then we have

$$A \cdot B = \begin{pmatrix} a_{11}\vec{b_1} \\ a_{21}\vec{b_1} \\ \vdots \\ a_{n1}\vec{b_1} \end{pmatrix} \oplus \begin{pmatrix} a_{12}\vec{b_2} \\ a_{22}\vec{b_2} \\ \vdots \\ a_{n2}\vec{b_2} \end{pmatrix} \oplus \cdots \oplus \begin{pmatrix} a_{1n}\vec{b_n} \\ a_{2n}\vec{b_n} \\ \vdots \\ a_{nn}\vec{b_n} \end{pmatrix}$$

SMITH - Other Uses

Matrix Multiplication

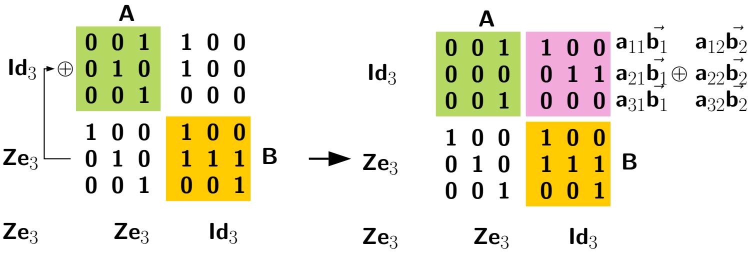
$$A \cdot B = \begin{pmatrix} a_{11}\vec{b_1} \\ a_{21}\vec{b_1} \\ \vdots \\ a_{n1}\vec{b_1} \end{pmatrix} \oplus \begin{pmatrix} a_{12}\vec{b_2} \\ a_{22}\vec{b_2} \\ \vdots \\ a_{n2}\vec{b_2} \end{pmatrix} \oplus \cdots \oplus \begin{pmatrix} a_{1n}\vec{b_n} \\ a_{2n}\vec{b_n} \\ \vdots \\ a_{nn}\vec{b_n} \end{pmatrix}$$



SMITH - Other Uses

Matrix Multiplication

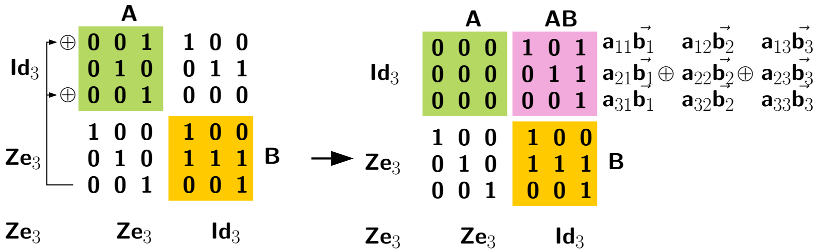
$$A \cdot B = \begin{pmatrix} a_{11}\vec{b_1} \\ a_{21}\vec{b_1} \\ \vdots \\ a_{n1}\vec{b_1} \end{pmatrix} \oplus \begin{pmatrix} a_{12}\vec{b_2} \\ a_{22}\vec{b_2} \\ \vdots \\ a_{n2}\vec{b_2} \end{pmatrix} \oplus \cdots \oplus \begin{pmatrix} a_{1n}\vec{b_n} \\ a_{2n}\vec{b_n} \\ \vdots \\ a_{nn}\vec{b_n} \end{pmatrix}$$



SMITH - Other Uses

Matrix Multiplication

$$A \cdot B = \begin{pmatrix} a_{11}\vec{b_1} \\ a_{21}\vec{b_1} \\ \vdots \\ a_{n1}\vec{b_1} \end{pmatrix} \oplus \begin{pmatrix} a_{12}\vec{b_2} \\ a_{22}\vec{b_2} \\ \vdots \\ a_{n2}\vec{b_2} \end{pmatrix} \oplus \cdots \oplus \begin{pmatrix} a_{1n}\vec{b_n} \\ a_{2n}\vec{b_n} \\ \vdots \\ a_{nn}\vec{b_n} \end{pmatrix}$$



SMITH- Hermite Canonical form

Over/Under-determined systems

- Given $Ax = b$. A and b is first padded with null rows/cols to get a square matrix A^* and extended column vector b^*.
- A^* converted to its Hermite Canonical Form H
- Do row operations$[A^* : I : b^*] \rightarrow [H : G : d]$
- Any general solution to $Ax = b$ is of the form $d + (I + H)z$ for any z,
  $\rightarrow$ The columns of $I + H$ form a basis for the null space of A

SMITH- Hermite Canonical form

1	*	0	0	*	0	0	0
0	0	0	0	0	0	0	0
0	0	1	0	*	0	0	0
0	0	0	1	*	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0
0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	1

Over/Under-determined systems

- Upper triangular.
- If $h_{ii} = 0$, the row must be null
- If $h_{ii} = 1$, the col must be unit vector

SMITH- Hermite Canonical form

$$
\begin{matrix}
1 & * & 0 & 0 & * & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & * & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & * & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{matrix}
$$

Over/under-determined systems

- The SMITH circuit alone is insufficient.
- Other operations are required
- Depth is no longer constant (OPEN PROBLEM)

Quadratic Systems

Quadratic Systems

When $d = 2$

- Solving degree $d > 2$ equations is NP-complete.
- Hardness of solving quadratics leveraged to construct PKC's.
  $\rightarrow$ HFE, QUARTZ, UOV, SFLASH etc.
- Quadratic over $GF(2)$ has a maximum of $S = 1 + n + \binom{n}{2}$ non-zero coefficients:

$$P: \; c + \; a_1x_1 + a_2x_2 + \cdots + a_nx_n + \; a_{12}x_1x_2 + a_{13}x_1x_3 + \cdots a_{n-1,n}x_{n-1}x_n$$

- Evaluating one poly over one point needs approx S bit-ops.

Standard Exhaustve search

When $d = 2$, m equations, n variables

- Quadratic over $GF(2)$ has a maximum of $S = 1 + n + \binom{n}{2}$ non-zero coefficients:

$$P: \ c + \ a_1 x_1 + a_2 x_2 + \cdots + a_n x_n + \ a_{12} x_1 x_2 + a_{13} x_1 x_3 + \cdots a_{n-1,n} x_{n-1} x_n$$

- Evaluating one poly over 2^n point needs approx $S \cdot 2^n$ bit-ops.
- Half of them are roots of P
  $\rightarrow$ Evaluate them 2^{n-1} over the next polynomial.
  $\rightarrow$ Total of $S \cdot 2^{n-1}$ operations.

- Third equation needs $S \cdot 2^{n-2}$ operations and so on ...

$$Comp = 2^n \cdot \left[S + \frac{S}{2} + \frac{S}{4} + \cdots \right] \approx S \cdot 2^{n+1} = O(n^2 2^n)$$

- Roots are points that are zeros that survive till the end.

Fast Exhaustve search [BCC+10/BCC+13]

Intuition

- Use Gray codes: $g_i = i \oplus (i \gg 1)$.
- Gray codes of successive integers differ by only 1 bit.

$$\begin{aligned} g_0 &= 000 \\ g_1 &= 001 \\ g_2 &= 011 \\ g_3 &= 010 \\ g_4 &= 110 \\ g_5 &= 111 \\ g_6 &= 101 \\ g_7 &= 100 \end{aligned}$$

- We traverse the input space of f in a Gray code manner.
- From knowledge of $f(g_i)$: we can evaluate $f(g_{i+1})$ efficiently without having to evaluate the entire function.

Taylor Expansion

The Idea

- $f(g_0) = f(\vec{0})$ is just the constant term of f.
- t is the bit-position where g_j and g_{j+1} differ
  $\rightarrow$ Let's say we already have the value of $f_i(g_j)$

$$f(g_{j+1}) = f(g_j) \oplus \frac{\delta f}{\delta x_t}(g_j). \quad (1)$$

- Here $\frac{\delta f}{\delta x_t}$ is the 1st order derivative of the function f at the point x_t.
- For example if $f = x_1x_2 \oplus x_3 \oplus x_1x_4x_5$,
  $\rightarrow \frac{\delta f}{\delta x_1} = x_2 \oplus x_4x_5$ and $\frac{\delta f}{\delta x_2} = x_1$, $\frac{\delta f}{\delta x_3} = 1$ etc.
- Derivative has degree one less than f and is easier to compute.

Taylor Expansion

Example

- If $f = x_1x_2 \oplus x_3 \oplus x_1x_4 \oplus x_5$,
  $\rightarrow \frac{\delta f}{\delta x_1} = x_2 \oplus x_4$ and $\frac{\delta f}{\delta x_2} = x_1$, $\frac{\delta f}{\delta x_3} = 1$, $\frac{\delta f}{\delta x_4} = x_1$ and $\frac{\delta f}{\delta x_5} = 1$.
- If original is quadratic derivatives are linear!!
  $\rightarrow \frac{\delta^2 f}{\delta x_1 x_2} = 1$, $\frac{\delta^2 f}{\delta x_1 x_3} = 0$ etc
  $\rightarrow$ Second derivatives are constant..
- Start with $f(00000) = 0$, next we find $f(g_1) = f(00001)$

$$f(00001) = f(00000) \oplus \frac{\delta f}{\delta x_1}(00000) = 0 \oplus x_2 + \oplus x_4|_{00000} = 0$$

- Then $f(g_2) = f(00011)$

$$f(00011) = f(00001) \oplus \frac{\delta f}{\delta x_2}(00001) = 0 \oplus x_1|_{00001} = 1$$

- Each next step takes evaluation of linear equation

Efficient Exhaustive Search for solutions over F_2

Main Theorem

All the zeroes of a single multivariate polynomial f in n variables of degree d can be found in essentially $d \cdot 2^n$ bit operations (plus a negligible overhead), using n^{d-1} bits of read-write memory, and accessing n^d bits of constants, after an initialization phase of negligible complexity $O(n^{2d})$.

- You need to pre-compute all derivatives.
- Precomputation needs time and energy and space.
- Precomputation required for each new equation system.
- Works best if $d = 2$ or lower.

Systems of arbitrary degree

Construct Truth tables

Truth Tables

$x_0x_1x_2$	P_0	P_1	P_2	$\cdots$	P_m	$\bigvee P_i$
000	0	1	1		0	1
001	1	0	0		1	1
010	0	1	1		1	1
011	1	1	0		0	1
100	0	0	0		0	0
				$\vdots$		
110	0	1	0		1	1
111	0	1	1		0	1

Root=100

Truth Tables

- Evaluation of a function at all points of its space. How can they help?

Möbius Transform

Möbius Transform

- Given the algebraic equation of any n-variable Boolean function, how to evaluate it over all the 2^n points of its input domain (i.e. find truth table) ?
- Given truth table of a Boolean function how to deduce its algebraic equation ?
- Answer to both the above is Möbius Transform.
- It is a linear, involutive transform that does both the above.
- Requires $n \cdot 2^{n-1}$ bit-operations.

Möbius Transform

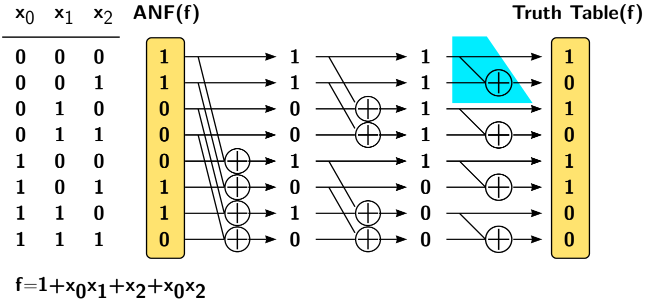


Figure: Möbius transform on $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. The blue shaded component represents one butterfly unit.

Salient Points

- Note we have lexicographical indexing.
- $t_6 = 1 \Rightarrow 6 = (110)_2 \Rightarrow$ the ANF contains the $x_0x_1 = x_0^1 \cdot x_1^1 \cdot x_2^0$ term.

Möbius Transform

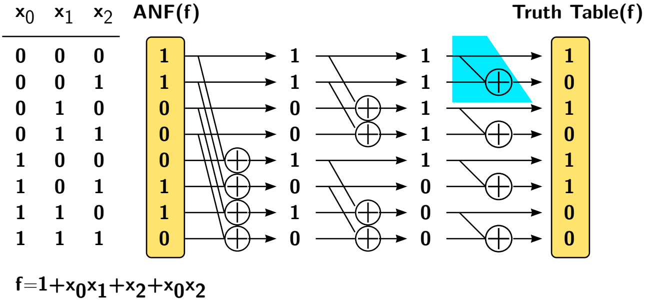


Figure: Möbius transform on $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. The blue shaded component represents one butterfly unit.

Salient Points

- n stages and 2^{n-1} xors per stage.
- Involutive: the same operations on ANF will give back TT.

The Mathematics

- If $\vec{v} = [v_0, v_1, \ldots, v_{2^n-1}]$ be the truth-table of f (note $v_i = f(i)$).
- If $\vec{u} = [u_0, u_1, \ldots, u_{2^n-1}]$ be the ANF of f.
- Then it is well known that

$$\vec{v} = M_n \cdot \vec{u}$$

- Note $M = m_{ij}$ is such that

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

- Eg $100 \preceq 101$, but $011 \not\preceq 100$ since 011 exceeds 100 in the last 2 bit-locations.

The Mathematics

- M_n is well studied in literature: Lower triangular + Involutive.
- Since $M_n = M_n^{-1}$, both $\vec{v} = M_n \cdot \vec{u}$ and $\vec{u} = M_n \cdot \vec{v}$ hold.
- 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.

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

Exponential circuits: The circuit Expmob1

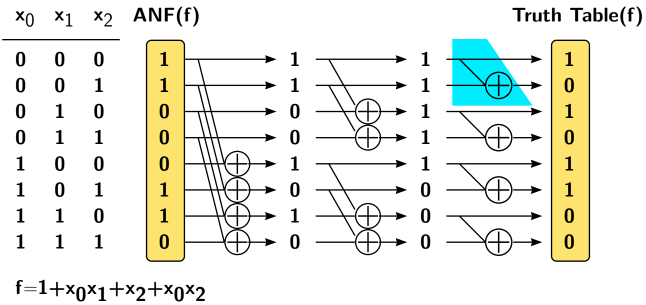


Figure: Möbius transform on $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. The blue shaded component represents one butterfly unit.

- Huge combinatorial circuit that stacks the stages one by one.
- Calculates in one single clock cycle: $n \cdot 2^{n-1}$ xor gates.

Exponential circuits: The circuit Expmob2

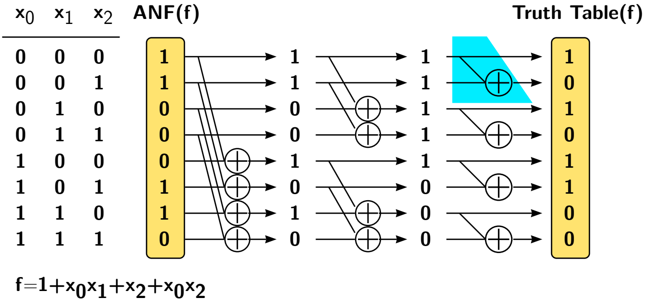


Figure: Möbius transform on $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. The blue shaded component represents one butterfly unit.

- Round based circuit: One stage in one clock cycle.
- Calculates in one n clock cycles: 2^{n-1} xor gates + Register of 2^n bits.

Exponential circuits: The circuit Expmob2

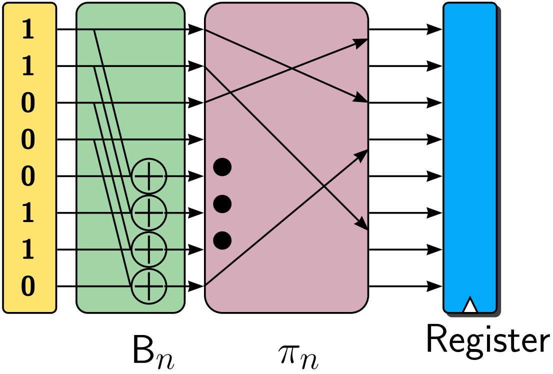
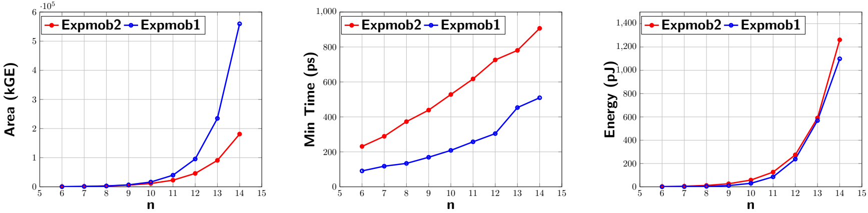


Figure: Round based Circuit.

- $\pi_n(2x) = x$, and $\pi_n(2x+1) = 2^{n-1} + x$ for all $0 \leq x < 2^{n-1}$
- If P_n is the permutation matrix for π_n, it can be shown $M_n = (P_n \cdot B_n)^n$.

Results (Nangate 15nm Open Cell Library)



(a) Area (b) Time (c) Energy

Figure: Synthesis results for **Expmob1** and **Expmob2** circuits

Degree Bound Functions

Polynomial number of Coeffciients

- ANF of Linear function: $n+1$ coefficients.
- ANF of Quadratic function: $\binom{n}{2}+n+1$ coefficients.
- ANF of Degree d function: $\binom{n}{\downarrow d}=\sum_{i=0}^{d}\binom{n}{i}$ coefficients $\in O(n^d)$.
- Challenge: With a register of size $\binom{n}{\downarrow d}$, can we compute the transform?

Take a look back

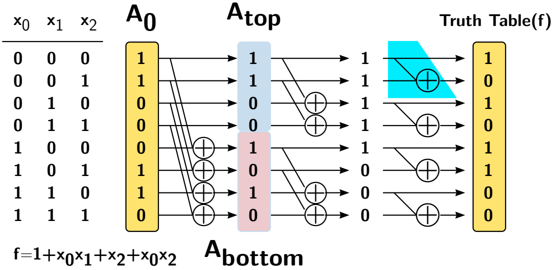


Figure: Round based Circuit.

- First stage $A_0 \rightarrow$ vectors A_{top} and A_{bottom}.
- A_{top} is actually ANF vector for $f(0, x_1, x_2)$ (in $n-1$ variables!!)
- A_{bottom} is actually ANF vector for $f(1, x_1, x_2)$ (in $n-1$ variables!!)
- Recursively apply Möbius Transform to these smaller vectors

Möbius Transform with Polynomial Space [Din21]

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 step, i.e.  leaf nodes of recursion tree */
if n=d then
    Use the formula B = M_n · 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, ↓d) bits.
    /* Compute the 2 operations of the butterfly layer */
1   Store 1st butterfly output i.e. A_top in T (requires no xors).
    Call Möbius (T, n-1, d)
2   Store 2nd butterfly output i.e. A_bottom in T (requires some xors).
    Call Möbius (T, n-1, d)
end
```

where $B = M_n \cdot A_0$ and the array size is $\binom{n-1}{\downarrow d}$.

Recursion tree

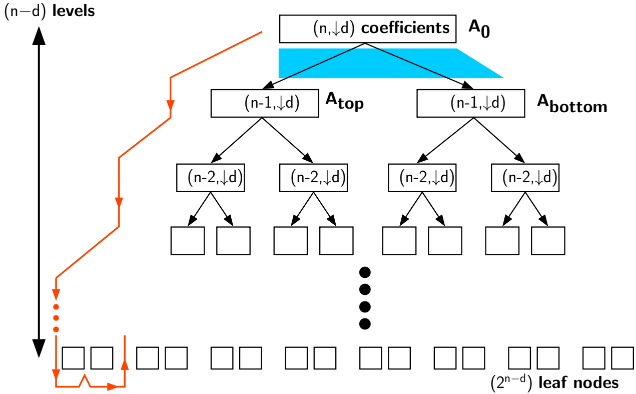


Figure: Recursion tree for the Möbius Transform algorithm. The blue shaded component roughly represents one arm of the butterfly unit.

- The Tree requires Depth first Traversal
- In Software this requires context switches, every time we traverse one level down.
- Mapping to hardware non trivial.

Circuit Sketch Polymob1

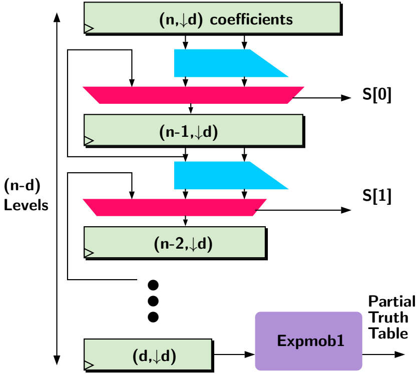


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- Primitive attempt to map algorithm to hw: can this work ?
- Each level needs own storage of size $\binom{n-i}{\downarrow d}$
- Task 1: Can we prove the space requirement is $O(n^{d+1})$??

Proof

- 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 [?] which states that $\sum_{m=d}^{n} \binom{m}{d} = \binom{n+1}{d+1}$. Note that expanding out $S(n,d)$ we get

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

Proof (contd)

- 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

$$\mathcal{P}(d+1) : \sum_{i=0}^{d+1} \binom{n}{i} < n^d + \binom{n}{d+1}$$

$$< n^d + \frac{n^{d+1}}{(d+1)!} = n^d \left(1 + \frac{n}{(d+1)!}\right) < n^{d+1}$$

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

Circuit Sketch Polymob1

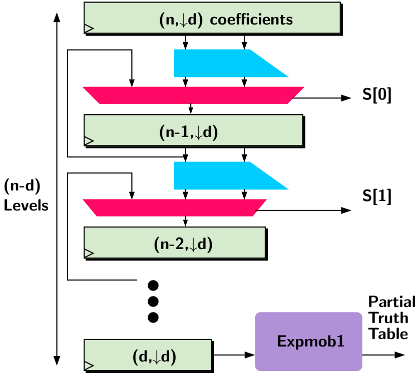


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- One reg of size $\binom{n}{\downarrow d}$ for A_0, but only one reg of size $\binom{n-1}{\downarrow d}$.
- If level 2 stores A_{top}, it must preserve this till its entire left sub-tree is executed.
- Only then overwrite to A_{bottom}.

Circuit Sketch Polymob1

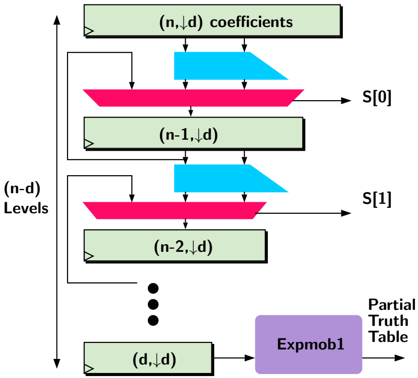


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- Multiplexer select signals control the flow.
- 3:1 multiplexer $\rightarrow$ Either preserve state or overwrite with $A_{\text{top/bottom}}$
- However only 2:1 mux is sufficient.

A bit of notation

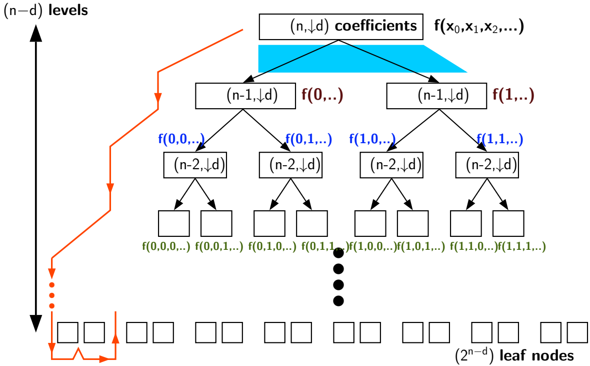


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- Every level sets one bit in the function argument.

A bit of notation

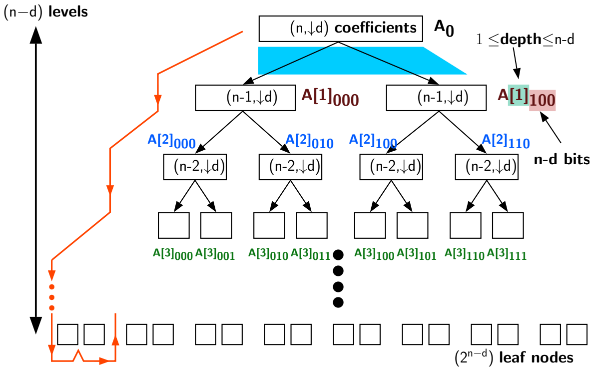


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- Let us label each ANF as $A[depth]_{bits}$

A bit of notation

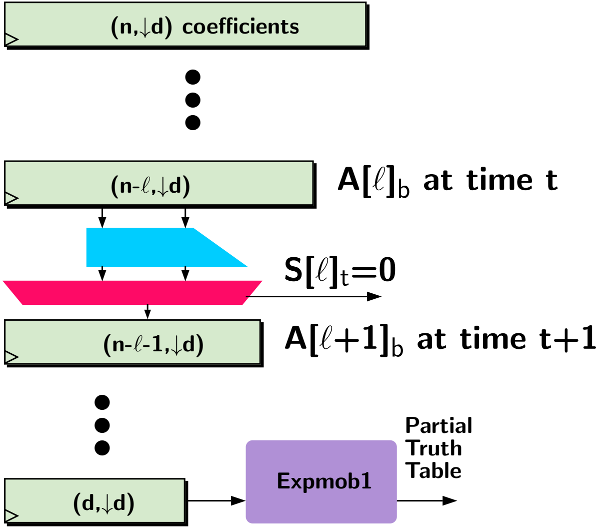


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

A bit of notation

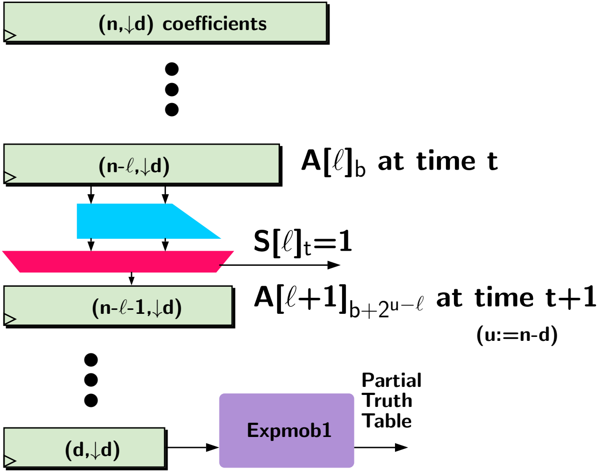
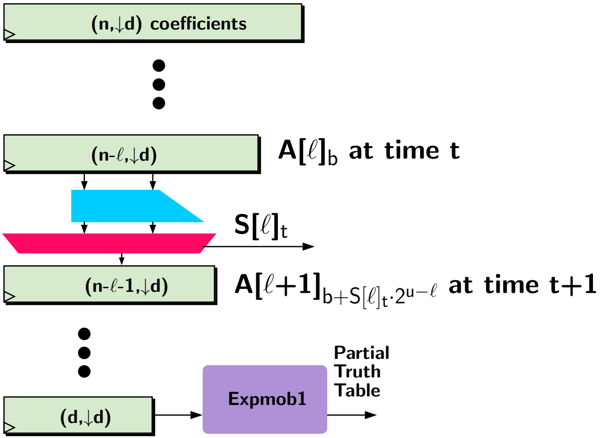
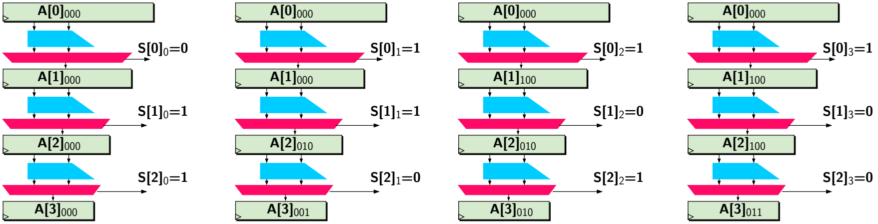


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

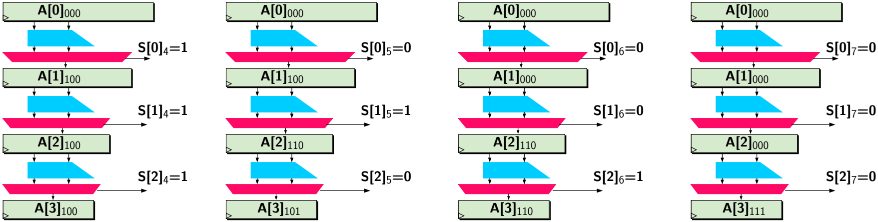
A bit of notation



Simulation $n = 5, d = 2$



(a) t=0 (b) t=1 (c) t=2 (d) t=3



(e) t=4 (f) t=5 (g) t=6 (h) t=7

Convert to Set of Equations

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	$4 \cdot S[0]_6$	$4 \cdot S[0]_5 + 2 \cdot S[1]_6$	$4 \cdot S[0]_4 + 2 \cdot S[1]_5 + S[2]_6$

- Left Column needs to be 0,1,2,3,...7
- Solve the integer equation system: look for solutions in $\{0, 1\}$

General Case (u:=n-d)

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

- Solve the integer equation system: look for solutions in $\{0, 1\}$
- Does Solution exist ? Is solution implementable ?

General Case (u:=n-d)

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

- Look at the i-th column shaded in green (note $j = u-1-i$)
- $S[j]_t$ is the $i+1$-th lsb of $(i+1),(i+2),\ldots$, i.e. the $(i+1)$-th lsb of $t+i+1$.

General Case (u:=n-d)

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

- A u-bit decimal up-counter for the variable t.
- A series of u incrementers to generate $t+1,\ t+2,\ldots,t+u$.

Circuit is implementable in logarithmic depth

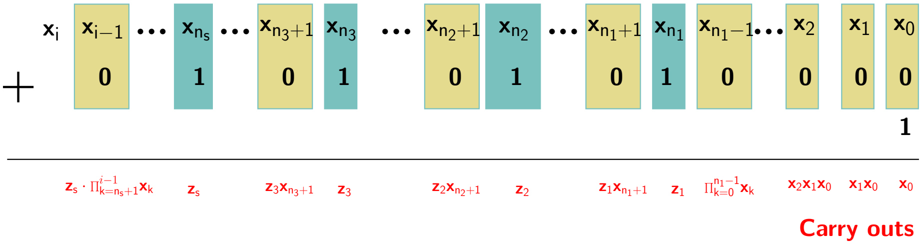


Figure: Visual representation of the addition $t+i+1$

- Having the whole incrementer circuit is unnecessary.
- We are only interested in $(i+1)$-th lsb of $t+i+1$.
- The expression is $x_i \oplus z_s \prod_{k=n_s+1}^{i-1} x_k$.
- Can be implemented using $2\log_2 u$ depth.

Polysolve1

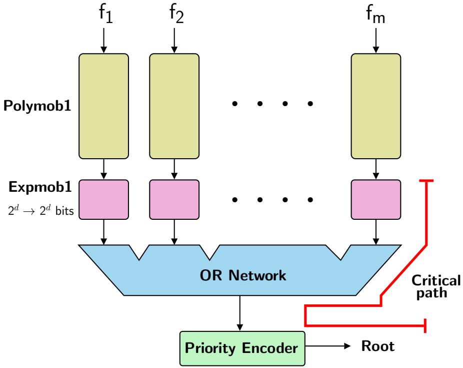


Figure: Hardware Solver **Polysolve1**

- After OR-ing, Priority Encoder gives the location of 1st 0 in the table.
- The solver will extract one root per partial truth table.
- Note large critical path !!

Polysolve2

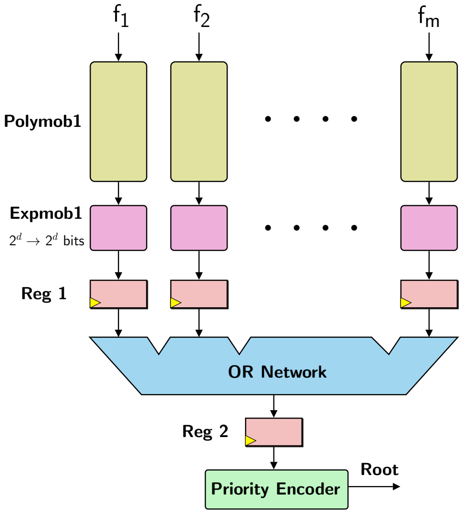
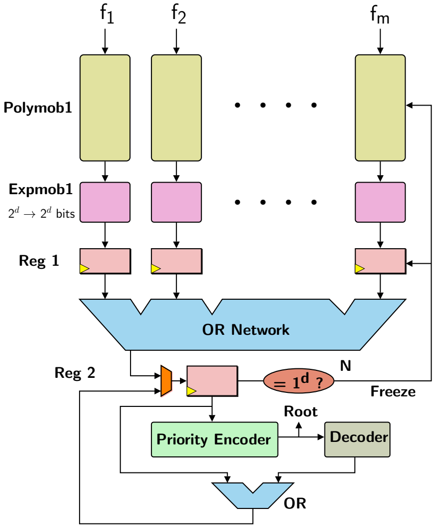


Figure: Hardware Solver **Polysolve2**

- Pipelining reduces the length of critical path.

Polysolve3



Example

If $d = 4$, and the **OR** of the truth tables is $T_0 = 1011\ 1111\ 1111\ 0111$

- At $\tau = 0$ Penc outputs 0001
- Decoder op $D_0 = 0100\ 0000\ 0000\ 0000$
- $T_1 = T_0 \vee D_0 = 1111\ 1111\ 1111\ 0111$
- $HW(T_1) = HW(T_0) + 1$, and is written back to **Reg2**.

- At $\tau = 1$ Penc outputs next root 1100
- We have $D_1 = 0000\ 0000\ 0000\ 1000$.
- $T_2 = T_1 \vee D_1 = 1111\ 1111\ 1111\ 1111$

which is now the all one string.

Polysolve3

Problem

The critical path of priority encoder+ decoder increases as d increases

- *Task 2: How to reduce it ?*

Polysolve3

Problem

The critical path of priority encoder+ decoder increases as d increases

- *Task 2: How to reduce it ?*

Solution

The Enc+Dec basically flips last 0 from a string

Other solutions exist n OR $n+1$??

$$
\begin{array}{rr}
\mathbf{n}: & 1100\ 0101\ 0111 \\
 & +1 \\
\hline
\mathbf{n+1}: & 1100\ 0101\ 1000 \\
\mathbf{n}: & 1100\ 0101\ 0111 \\
\hline
\mathbf{n} \vee \mathbf{n+1}: & 1100\ 0101\ 1111
\end{array}
$$

Polysolve3

Problem

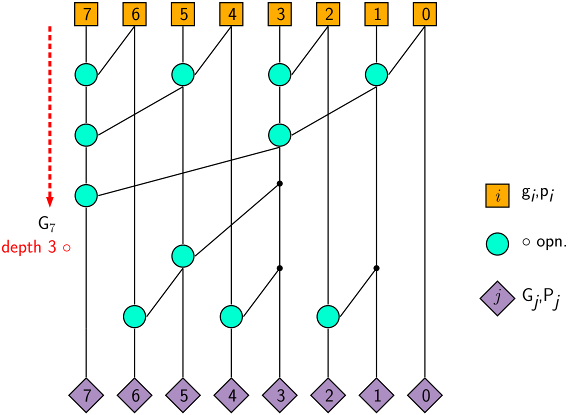
The critical path of priority encoder+ decoder increases as d increases

- *Task 2: How to reduce it ?*

Solution

- In stead of Encoder followed by Decoder, we can do Encoder and n OR $n+1$ block in parallel.
- Simultaneously fishes root+ flips zero.

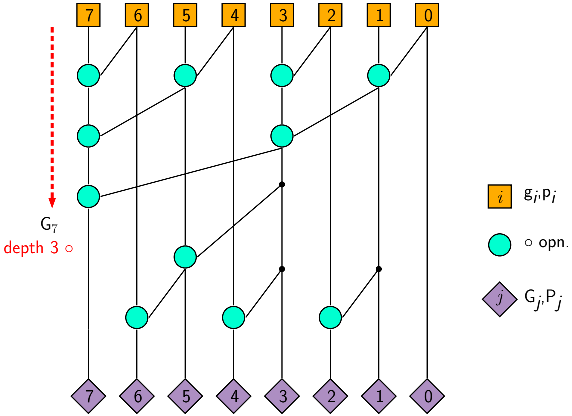
Polysolve3: Brent Kung adder



Generate/propagate

- $g_i = a_i \cdot b_i$ and $p_i = a_i \oplus b_i$
- $G_0, P_0 = g_0, p_0$ and $G_i, P_i = (g_i, p_i) \circ (G_{i-1}, P_{i-1})$
- $(x_1, y_1) \circ (x_2, y_2) = (x_1 \vee (y_1 \cdot x_2), y_1 \cdot y_2)$.

Polysolve3: Brent Kung adder



Generate/propagate

- Has logarithmic depth
- Can be used with carry-select approach
- TO get faster adders for arbitrary d.

Time taken

Problem

- *Plain Circuit takes 2^{n-d} cycles.*
- *Task 3: How much time does this take if there are R roots ?*

Time taken

Problem

- *Plain Circuit takes 2^{n-d} cycles.*
- *Task 3: How much time does this take if there are R roots ?*

Solution

- If partial truth table has $r_i = 0$ roots: no additional cycle.
- If partial truth table has $r_i = 1$ roots: one additional cycle.
- If partial truth table has $r_i = 2$ roots: two additional cycle.
- Therefore $1 + r_i$ cycles per partial TT

$$\sum_{i=1}^{2^{n-d}} 1 + r_i = 2^{n-d} + \sum r_i = 2^{n-d} + R$$

.

Energy Efficiency

Wasteful Computation

- Suppose we have 50 equations in 50 variables.
  - → The common solution of 1st 10 equations is 100.
  - → Evaluating Möbius Transform for the remaining equations⇒Evaluating 40 equations at 2^{50} points each.
  - → Evaluating 40 equations at 10 points is sufficient !!!!
- We found energy efficient solution for this.
- The idea is to filter any common root of first 10 eqns using Dot-product circuit.

Energy Efficiency

Wasteful Computation

- First run Möbius Transform on a small number of μ equations.
  $\rightarrow$ Find the common solution set Λ of 1st μ equations.
  $\rightarrow$ The remaining equations has to be checked only on above set

- So how do we this?
- Each $r \in \Lambda$ has to be evaluated on $m - \mu$ equations.

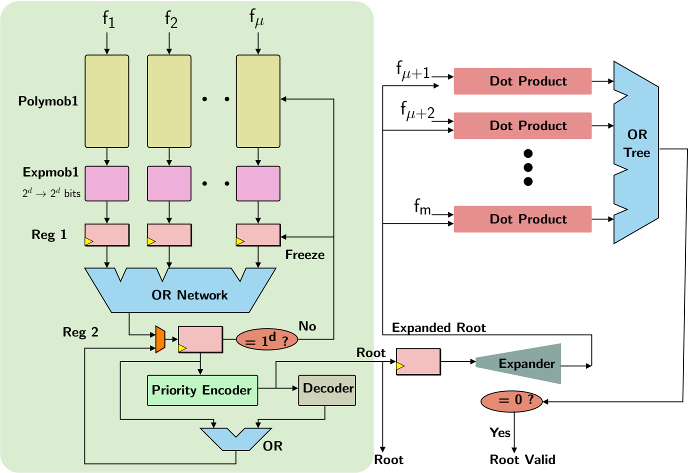
Tools

Circuit components

- Root expander: RE(n,d):$\{0,1\}^n \to \{0,1\}^{\binom{n}{\downarrow d}}$.
  $\to$ Eg.RE$(4,3)$ over the vector $(x_0, x_1, x_2, x_3) = (1,0,1,1)$
  $\to$ const value$=1$, $x_0x_1 = 0$, $x_0x_2 = 1$, $x_0x_3 = 1, x_1x_2 = 0$, $x_1x_3 = 0$, $x_2x_3 = 1$, $x_0x_1x_2 = 0, x_0x_1x_3 = 0$, $x_0x_2x_3 = 1$, $x_1x_2x_3 = 0$.
  $\to$ The expanded root **r**$=1111\ 0001\ 110$
  $\to$ Total hardware overhead is $\binom{n}{\downarrow d} - n$ **AND** gates.

- Dot-Product: Eg $f = 1 \oplus x_0 \oplus x_2 \oplus x_0x_1 \oplus x_2x_3$.
  $\to$ Vector Description **v**$=1011\ 0001\ 001$.
  $\to$ The dot-product **r** $\cdot$ **v** $= 0$, equals $f(r)$.
  $\to$ $\binom{n}{\downarrow d}$ **AND** gates and $\binom{n}{\downarrow d} - 1$ **XOR**

Circuit



PolySolve3 instance with μ equations

Time taken

Problem

- *Plain Circuit takes* $2^{n-d} + R$ *cycles.*
- *Task 3: How much time does this take if there are* μ *instances ?*

Time taken

Problem

- *Plain Circuit takes $2^{n-d} + R$ cycles.*
- *Task 3: How much time does this take if there are μ instances ?*

Lemma

Let $f_1, f_2, \ldots, f_\mu$ be iid 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 $2^{n-\mu}$.

- So $R + 2^{n-d} = 2^{n-\mu} + 2^{n-d}$ cycles on average.

Energy

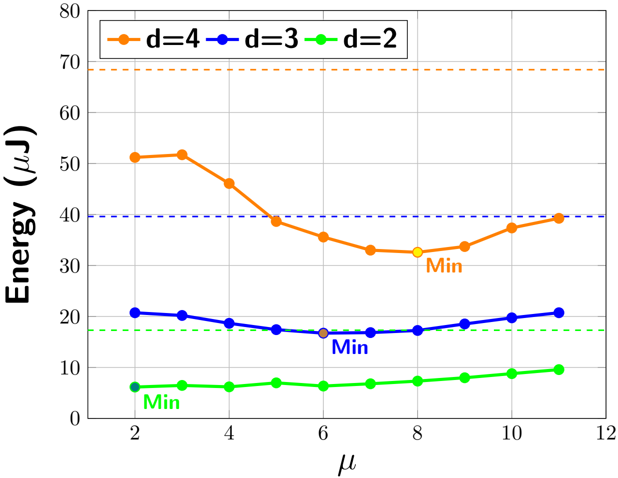


Figure: Energy consumption for varying μ for $n = m = 20$. The colored dashed lines show the energy consumed in the **Polysolve3** circuit for the corresponding equation systems.

Depth Bound trees

Algorithm 2: 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 step, i.e.  leaf nodes of recursion tree */
if n=d then
    Use the formula B = M_n · 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, ↓d) bits.
    /* Compute the 2 operations of the butterfly layer */
1   Store 1st butterfly output i.e. A_top in T (requires no xors).
    Call Möbius (T, n-1, d)
2   Store 2nd butterfly output i.e. A_bottom in T (requires some xors).
    Call Möbius (T, n-1, d)
end
```

Depth Bound trees

Algorithm 3: 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 step, i.e.  leaf nodes of recursion tree */
if n=h>d then
    Use the formula B = M_n · 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, ↓d) bits.
    /* Compute the 2 operations of the butterfly layer */
1   Store 1st butterfly output i.e. A_top in T (requires no xors).
    Call Möbius (T, n − 1, d)
2   Store 2nd butterfly output i.e. A_bottom in T (requires some xors).
    Call Möbius (T, n − 1, d)
end
```

Where: $B = M_n \cdot A_0$; size $\binom{n-1}{\downarrow d}$; A_{top}; A_{bottom}.

Depth boundedness

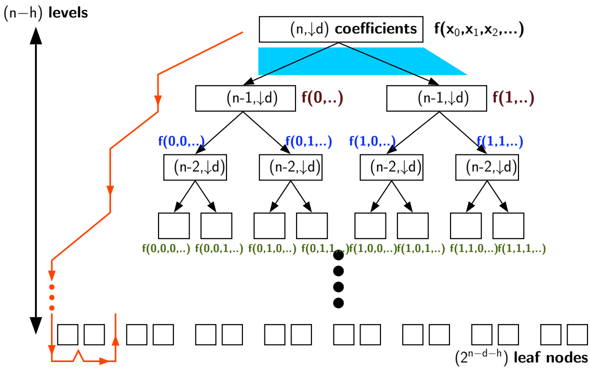


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- Number of levels shrink to $n - h$ from $n - d$.
- Faster computation: $2^{n-\mu} + 2^{n-d} \to 2^{n-\mu} + 2^{n-h}$.

Depth boundedness

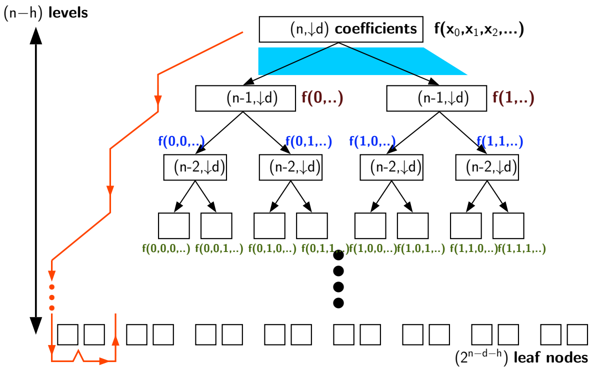


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- Downside **Expmob1**, encoder needed over $h > d$ bits.
- Increases the critical path.

Energy reduction

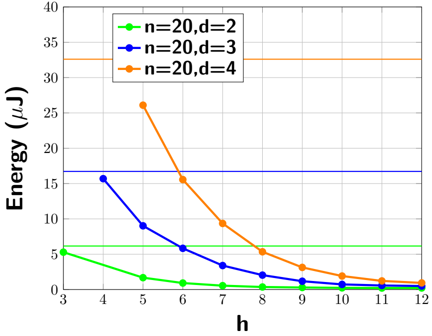


Figure: Energy decrease with increasing h for new solvers for $n = 20$, $h = \mu$. The colored horizontal lines indicate the best possible energy consumption for the full depth circuit for the same equation system.

SAKURA-X

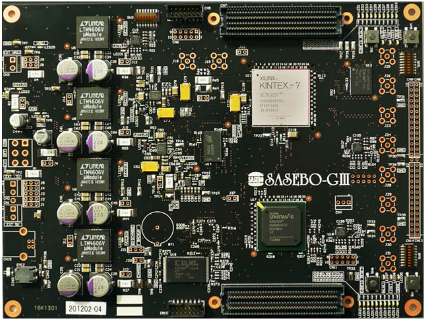


Figure: SAKURA-X

Proof of Concept

- SAKURA-X mainly built for side-channel experiments, limited computational power.
- We could solve quadratic equations of upto 50 variables in 8 hours.
- TODO $\rightarrow$ Implement on an FPGA cluster and solve upto 100 variables.

Conclusion

- Given m equations in n variables over $GF(2)$.
- Asymptotically, all the solutions can be found using a circuit of area $\propto m \cdot n^{d+2}$.
- This is not energy-efficient however: Möbius Transform does a lot of redundant computations.
- Circuit for energy efficiency also proposed.

THANK YOU