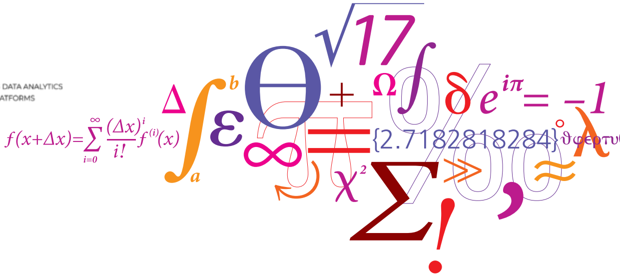
Compact equation solvers over GF(2)

Subhadeep Banik

University of Lugano

Dagstuhl Symmetric Key Workshop

22nd January 2024



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).

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.

Systems of arbitrary degree

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	Root=100
				$\vdots$			
110	0	1	0		1	1	
111	0	1	1		0	1	

Truth Tables

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

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

- Roots are indices at which all P_i's evaluate to zero, i.e. $\vee P_i = 0$

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

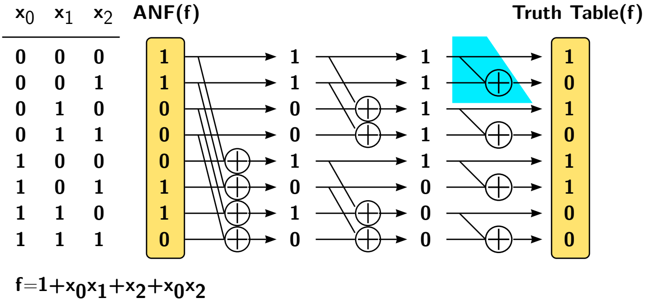


Figure: Möbius transform on $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. The blue shaded component represents one butterfly unit.

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

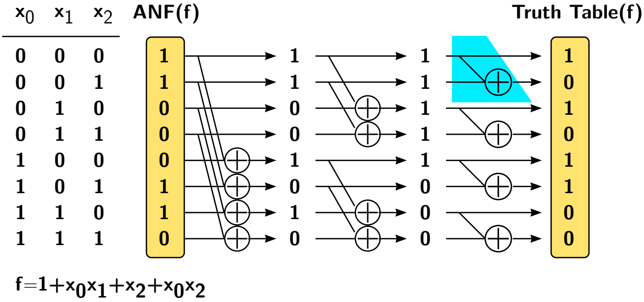


Figure: Möbius transform on $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. The blue shaded component represents one butterfly unit.

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

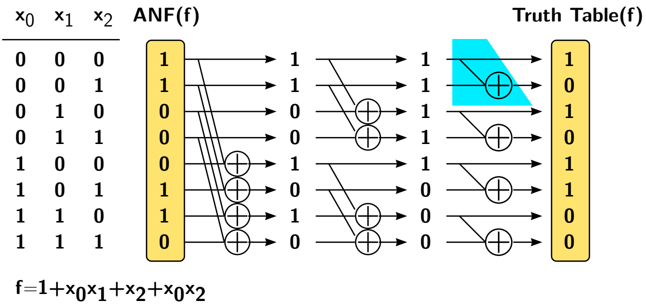


Figure: Möbius transform on $f = 1 \oplus x_0x_1 \oplus x_2 \oplus x_0x_2$. The blue shaded component represents one butterfly unit.

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

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

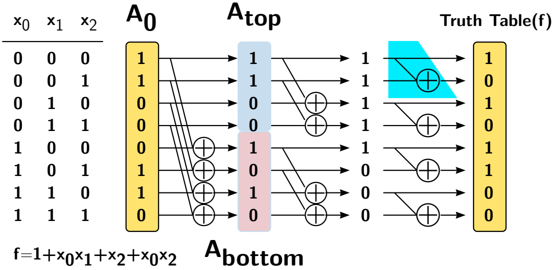


Figure: Round based Circuit.

- First stage A_0 → 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 $B = M_n \cdot A_0$ 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.
  `/* 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

Recursion tree

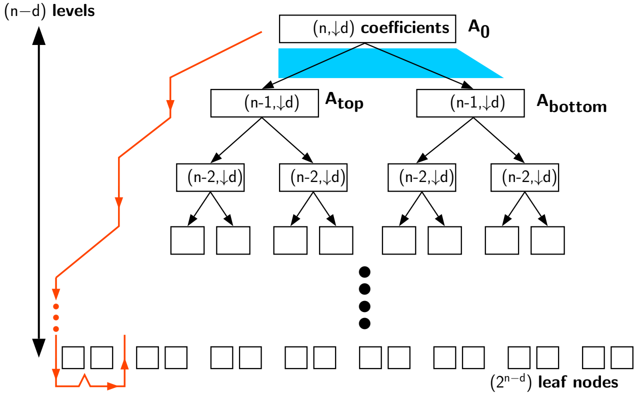


Figure: Recursion tree for the Möbius Transform algorithm. The blue shaded component roughly represents one arm of the butterfly unit.

- 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

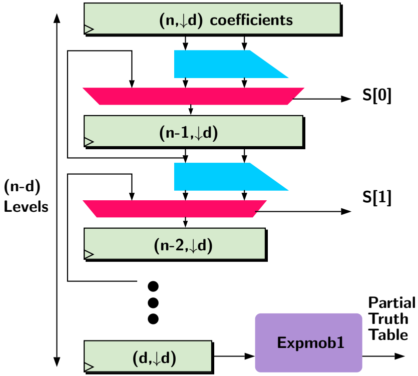


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- Primitive attempt to map algorithm to hw: can this work ?
- Each level needs own storage of size $\binom{n-i}{\downarrow d}$
- Let us see.

Circuit Sketch Polymob1

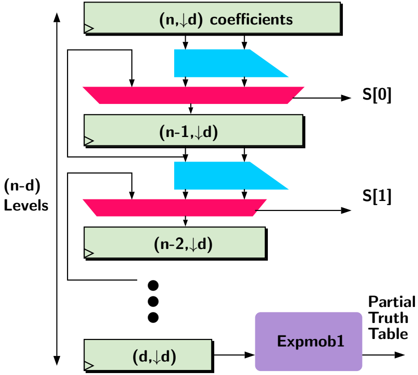


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

- 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

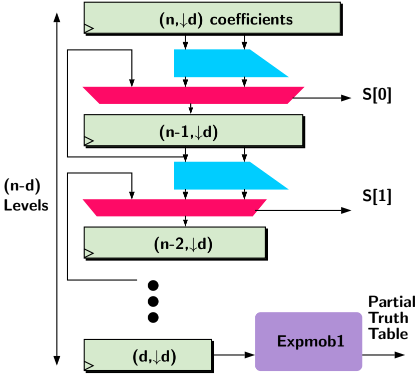
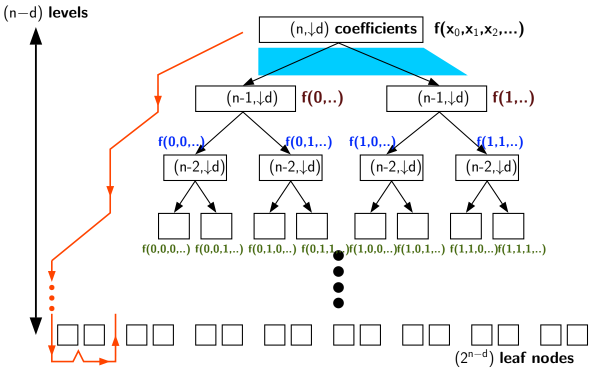


Figure: Hardware architecture **Polymob1** for the Möbius Transform algorithm. The blue shaded part roughly represents one arm of the butterfly unit.

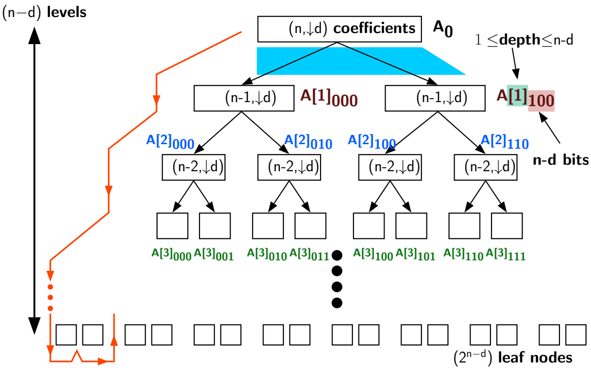
- Multiplexer select signals control the flow.
- 3:1 multiplexer $\rightarrow$ Either preserve state or overwrite with $A_{\text{top/bottom}}$
- However only 2:1 mux is sufficient.

A bit of notation



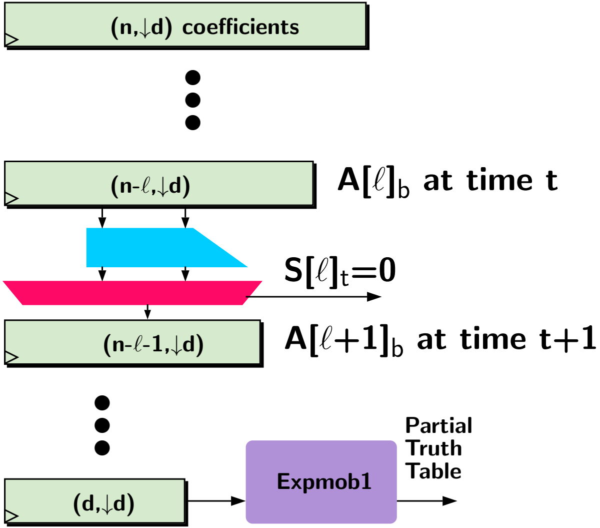
- Every level sets one bit in the function argument.

A bit of notation

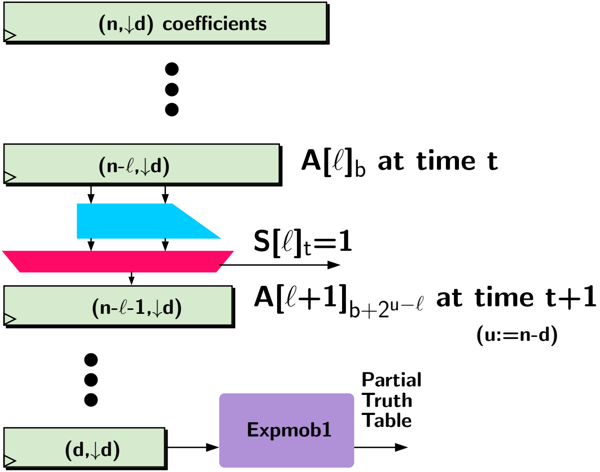


- Let us label each ANF as $A[depth]_{bits}$

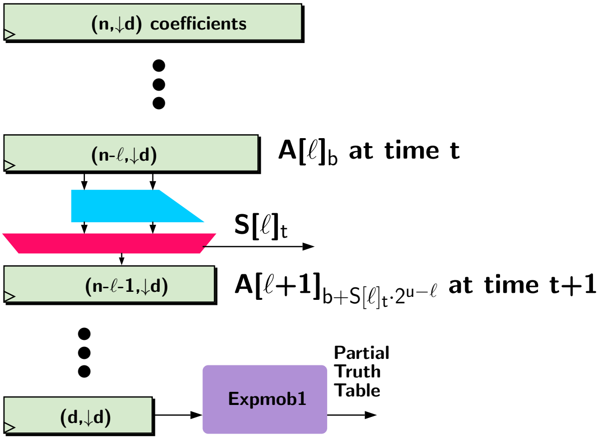
A bit of notation



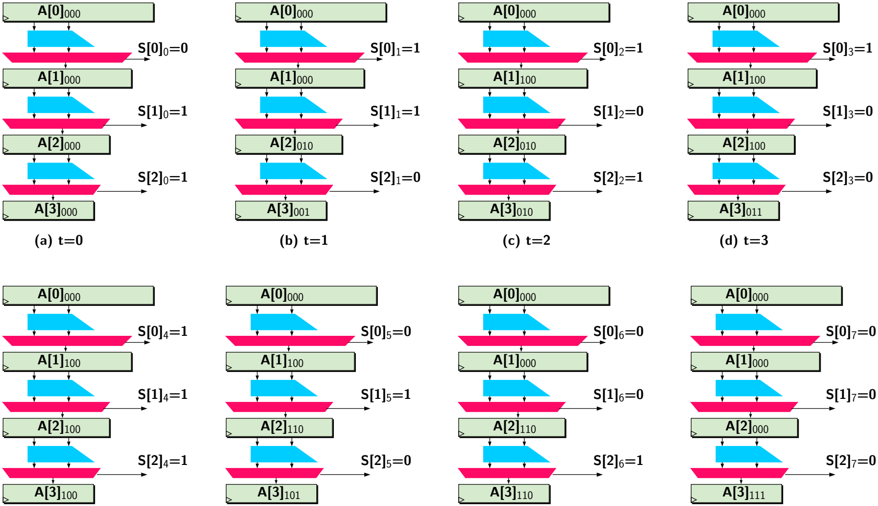
A bit of notation



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)

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

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

General Case (u:=n-d)

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

- 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)

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

- 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

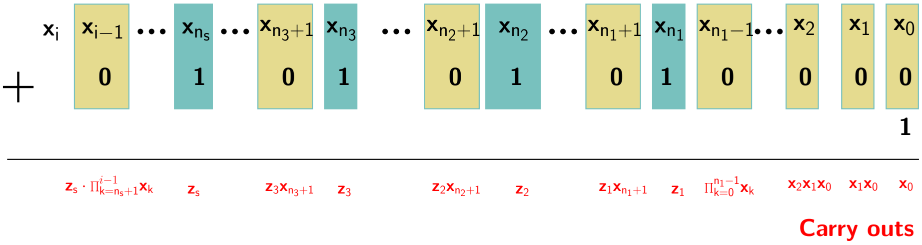


Figure: Visual representation of the addition $t + i + 1$

- 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

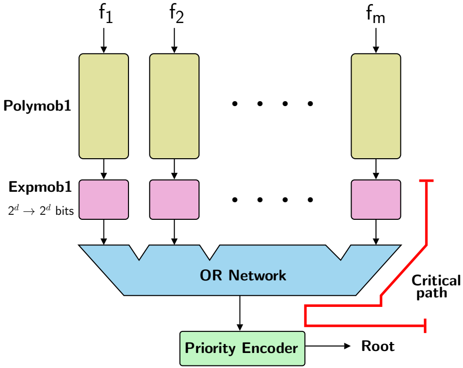


Figure: Hardware Solver **Polysolve1**

- 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

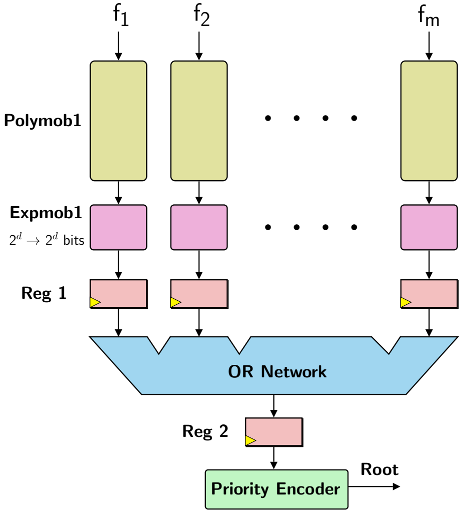
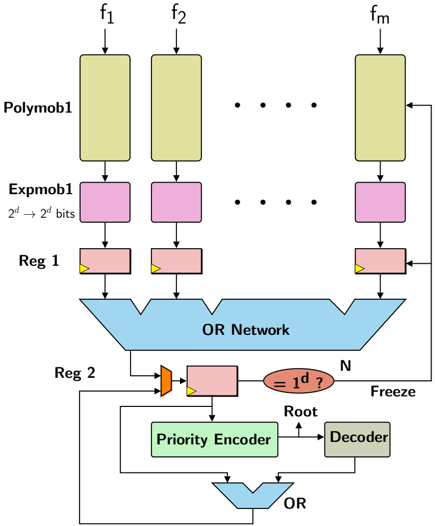


Figure: Hardware Solver **Polysolve2**

- 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.

SAKURA-X

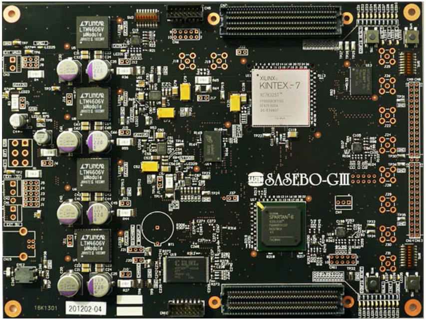


Figure: SAKURA-X

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 → 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 $\propto m \cdot n^{1+d}$.
- This is not energy-efficient however: Möbius Transform does a lot of redundant computations.
- Energy efficient solutions must additionally look at linear algebra.
- Please see **https://ia.cr/2023/948** for details

THANK YOU