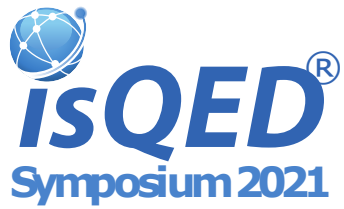
Word-Level Multi-Fix Rectifiability of Finite Field Arithmetic Circuits



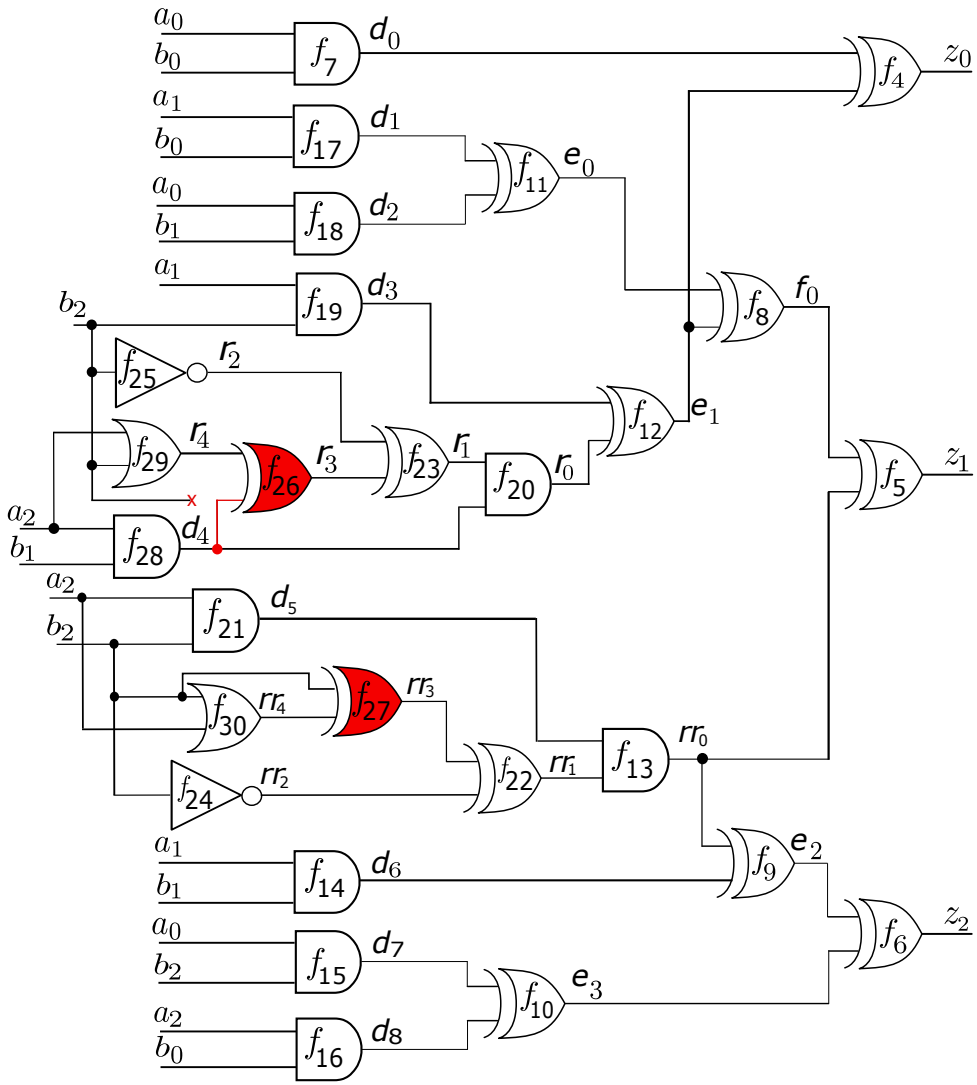
Vikas Rao[1], Irina Ilioaea[2], Haden Ondricek[1], Priyank Kalla[1], and Florian Enescu[3]

[1]Electrical & Computer Engineering, University of Utah
[2]Department of Mathematics, Louisiana State University Shreveport
[3]Mathematics & Statistics, Georgia State University

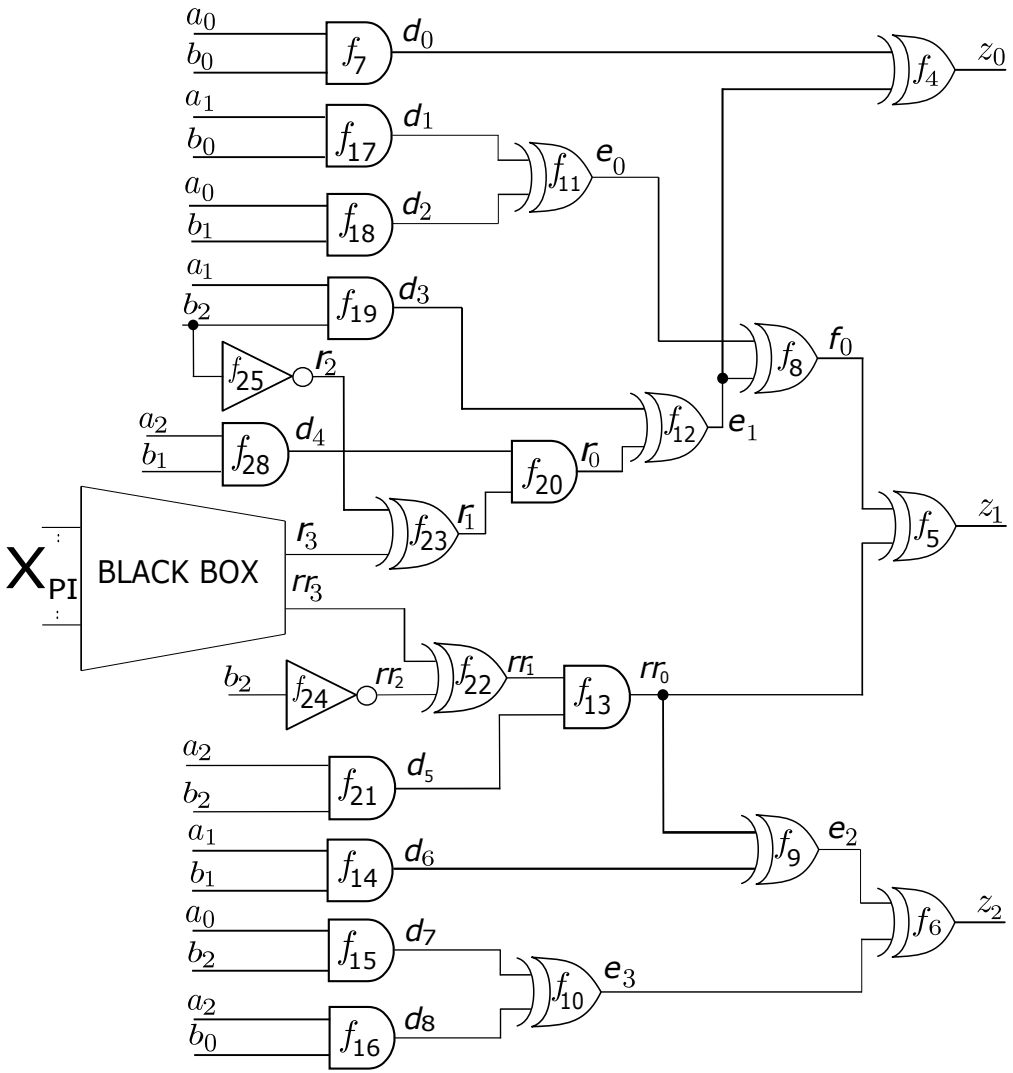
Outline

- Problem Description and Motivation
- Preliminaries
- Unified Framework
  - Mathematical Challenges
- Rectifiability Check
- Implementation
- Experimental Results
- Summary and Future work

Problem Description: Rectification



Problem Description: Rectification



Motivation: Finite Field Circuits

- Applications:
  - Cryptography: RSA, Ellyptic Curve Cryptography (ECC)
  - Error Correcting Codes, Digital Signal Processing, RFID, etc.

Motivation: Finite Field Circuits

- Applications:
  - Cryptography: RSA, Ellyptic Curve Cryptography (ECC)
  - Error Correcting Codes, Digital Signal Processing, RFID, etc.
    - Crypto-system bugs can leak secret keys [*Biham. et al*, Crypto'08]
    - RFID tag cloning could cause counterfeiting [*Batina. et al*, Security'09]

Motivation: Finite Field Circuits

- Applications:
  - Cryptography: RSA, Ellyptic Curve Cryptography (ECC)
  - Error Correcting Codes, Digital Signal Processing, RFID, etc.
    - Crypto-system bugs can leak secret keys [*Biham. et al*, Crypto'08]
    - RFID tag cloning could cause counterfeiting [*Batina. et al*, Security'09]
  - Large datapath sizes (n) in ECC crypto systems
    - $n = 163, 233, 283, 409, 571$ (NIST standard)

Motivation: Finite Field Circuits

- Applications:
  - Cryptography: RSA, Ellyptic Curve Cryptography (ECC)
  - Error Correcting Codes, Digital Signal Processing, RFID, etc.
    - Crypto-system bugs can leak secret keys [*Biham. et al*, Crypto'08]
    - RFID tag cloning could cause counterfeiting [*Batina. et al*, Security'09]
  - Large datapath sizes (n) in ECC crypto systems
    - $n = 163, 233, 283, 409, 571$ (NIST standard)
- Rectification Motivation:
  - Automated debugging
  - Synthesize sub-functions as opposed to complete redesign

Preliminaries: Finite fields

- Fields - set of elements over which operations $(+, \cdot, /)$ can be performed
  - Ex. $\mathbb{R}, \mathbb{Q}, \mathbb{C}$

Preliminaries: Finite fields

- Fields - set of elements over which operations $(+, \cdot, /)$ can be performed
  - Ex. $\mathbb{R}, \mathbb{Q}, \mathbb{C}$
- Finite fields (Galois fields) - Finite set of elements

Preliminaries: Finite fields

- Fields - set of elements over which operations $(+, \cdot, /)$ can be performed
  - Ex. $\mathbb{R}, \mathbb{Q}, \mathbb{C}$
- Finite fields (Galois fields) - Finite set of elements
  - Ex. $\mathbb{F}_q$, where $q = p^n,\ p = \textit{prime},\ n \in \mathbb{Z}_{\geq 1}$
    - With $n = 1,\ \textit{and}\ p = 2, \mathbb{F}_2 = \mathbb{B} = \{0, 1\}$

Preliminaries: Finite fields

- Fields - set of elements over which operations $(+, \cdot, /)$ can be performed
  - Ex. $\mathbb{R}, \mathbb{Q}, \mathbb{C}$
- Finite fields (Galois fields) - Finite set of elements
  - Ex. $\mathbb{F}_q$, where $q = p^n,\ p = \textit{prime},\ n \in \mathbb{Z}_{\geq 1}$
    - With $n = 1,\ \textit{and}\ p = 2, \mathbb{F}_2 = \mathbb{B} = \{0, 1\}$
  - On circuits, $p = 2$, $n =$ data-operand width

Preliminaries: Finite fields

- Fields - set of elements over which operations $(+, \cdot, /)$ can be performed
  - Ex. $\mathbb{R}, \mathbb{Q}, \mathbb{C}$
- Finite fields (Galois fields) - Finite set of elements
  - Ex. $\mathbb{F}_q$, where $q = p^n,\ p = \textit{prime},\ n \in \mathbb{Z}_{\geq 1}$
    - With $n = 1,\ \textit{and}\ p = 2, \mathbb{F}_2 = \mathbb{B} = \{0, 1\}$
  - On circuits, $p = 2$, $n =$ data-operand width
- Hardware cryptography extensively based on $\mathbb{F}_{2^n}$ (we use $\mathbb{F}_{2^n}$)

Preliminaries: Field Construction

- $\mathbb{F}_{2^n} = \mathbb{F}_2[x] \pmod{P_n(x)}$

Preliminaries: Field Construction

- $\mathbb{F}_{2^n} = \mathbb{F}_2[x] \pmod{P_n(x)}$
  - $P_n(x) \in \mathbb{F}_2[x]$ irreducible polynomial of degree n

Preliminaries: Field Construction

- $\mathbb{F}_{2^n} = \mathbb{F}_2[x] \pmod{P_n(x)}$
  - $P_n(x) \in \mathbb{F}_2[x]$ irreducible polynomial of degree n
  - Operations performed $\pmod{P_n(x)}$
  - Coefficients reduced $\pmod{2}$

Preliminaries: Field Construction

- $\mathbb{F}_{2^n} = \mathbb{F}_2[x] \pmod{P_n(x)}$
  - $P_n(x) \in \mathbb{F}_2[x]$ irreducible polynomial of degree n
  - Operations performed $\pmod{P_n(x)}$
  - Coefficients reduced $\pmod{2}$

- Construct $\mathbb{F}_{2^3}$:
  - Use $P_3(x) = x^3 + x + 1$ or $P_3(x) = x^3 + x^2 + 1$

Preliminaries: Field Construction

- $\mathbb{F}_{2^n} = \mathbb{F}_2[x] \pmod{P_n(x)}$
  - $P_n(x) \in \mathbb{F}_2[x]$ irreducible polynomial of degree n
  - Operations performed $\pmod{P_n(x)}$
  - Coefficients reduced $\pmod{2}$

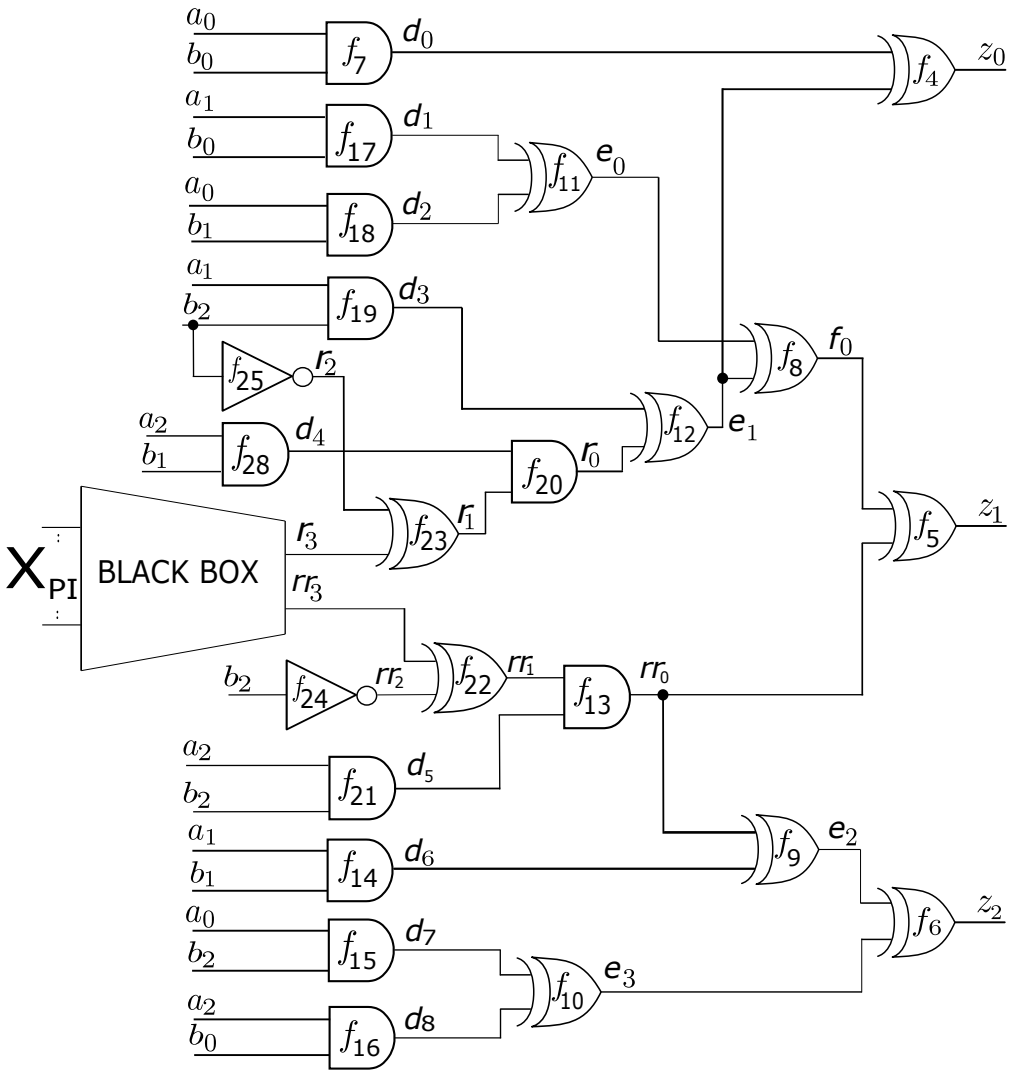
- Construct $\mathbb{F}_{2^3}$:
  - Use $P_3(x) = x^3 + x + 1$ or $P_3(x) = x^3 + x^2 + 1$
    - Fields are isomorphic

Preliminaries: Field Construction

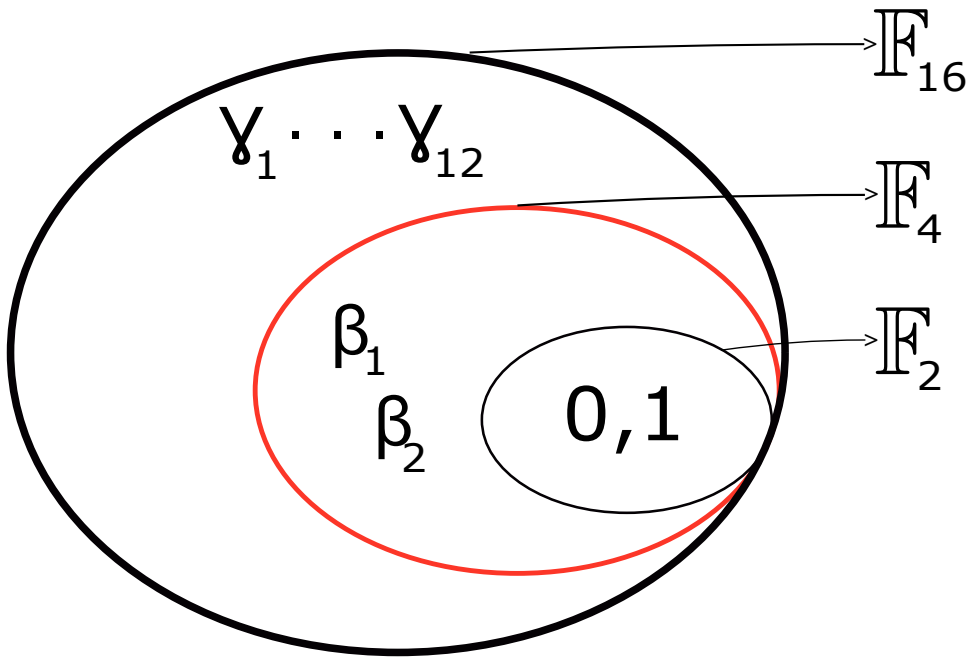
- $\mathbb{F}_{2^n} = \mathbb{F}_2[x] \pmod{P_n(x)}$
  - $P_n(x) \in \mathbb{F}_2[x]$ irreducible polynomial of degree n
  - Operations performed $\pmod{P_n(x)}$
  - Coefficients reduced $\pmod{2}$

- Construct $\mathbb{F}_{2^3}$:
  - Use $P_3(x) = x^3 + x + 1$ or $P_3(x) = x^3 + x^2 + 1$
    - Fields are isomorphic
    - Root of one is not the same as the other

Problem Description: Field Containment



Field Containment



$$\mathbb{F}_2 \subset \mathbb{F}_4 \subset \mathbb{F}_{16}$$

MFR Challenges: $\mathbb{F}_{2^k}$ and $P_k(x)$

- Smallest k is $LCM(n, m)$
  - $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^n}$ and $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^m}$
  - $\mathbb{F}_{2^k} = \mathbb{F}_2[x] \pmod{P_k(x)}$
    - $P_k(x)$ is a degree-k primitive polynomial; $P_k(\alpha) = 0$

MFR Challenges: $\mathbb{F}_{2^k}$ and $P_k(x)$

- Smallest k is $LCM(n, m)$
  - $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^n}$ and $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^m}$
  - $\mathbb{F}_{2^k} = \mathbb{F}_2[x] \pmod{P_k(x)}$
    - $P_k(x)$ is a degree-k primitive polynomial; $P_k(\alpha) = 0$
- What $P_K(x)$ should be used for constructing $\mathbb{F}_{2^k}$

MFR Challenges: $\mathbb{F}_{2^k}$ and $P_k(x)$

- Smallest k is $LCM(n, m)$
  - $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^n}$ and $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^m}$
  - $\mathbb{F}_{2^k} = \mathbb{F}_2[x] \pmod{P_k(x)}$
    - $P_k(x)$ is a degree-k primitive polynomial; $P_k(\alpha) = 0$
- What $P_K(x)$ should be used for constructing $\mathbb{F}_{2^k}$
- Mathematical challenge: Given $P_n(x)$ and $P_m(x)$, compute $P_k(x)$ such that $P_n(\gamma) = P_m(\beta) = P_k(\alpha) = 0$

MFR Challenges: $\mathbb{F}_{2^k}$ and $P_k(x)$

- Smallest k is $LCM(n, m)$
  - $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^n}$ and $\mathbb{F}_{2^k} \supset \mathbb{F}_{2^m}$
  - $\mathbb{F}_{2^k} = \mathbb{F}_2[x] \pmod{P_k(x)}$
    - $P_k(x)$ is a degree-k primitive polynomial; $P_k(\alpha) = 0$
- What $P_K(x)$ should be used for constructing $\mathbb{F}_{2^k}$
- Mathematical challenge: Given $P_n(x)$ and $P_m(x)$, compute $P_k(x)$ such that $P_n(\gamma) = P_m(\beta) = P_k(\alpha) = 0$
  - How are elements α, β, and γ related?

Contribution: Computing $P_k(x)$

- Property of finite fields, for any element $\phi \in \mathbb{F}_q$, $\phi^{q-1} = 1$

Contribution: Computing $P_k(x)$

- Property of finite fields, for any element $\phi \in \mathbb{F}_q$, $\phi^{q-1} = 1$
  - $\gamma = \alpha^{(2^k-1)/(2^n-1)} = \alpha^{\lambda}$
  - $\beta = \alpha^{(2^k-1)/(2^m-1)} = \alpha^{\mu}$

Contribution: Computing $P_k(x)$

- Property of finite fields, for any element $\phi \in \mathbb{F}_q$, $\phi^{q-1} = 1$
  - $\gamma = \alpha^{(2^k-1)/(2^n-1)} = \alpha^{\lambda}$
  - $\beta = \alpha^{(2^k-1)/(2^m-1)} = \alpha^{\mu}$
- Univariate Polynomial Factorization (UPF)

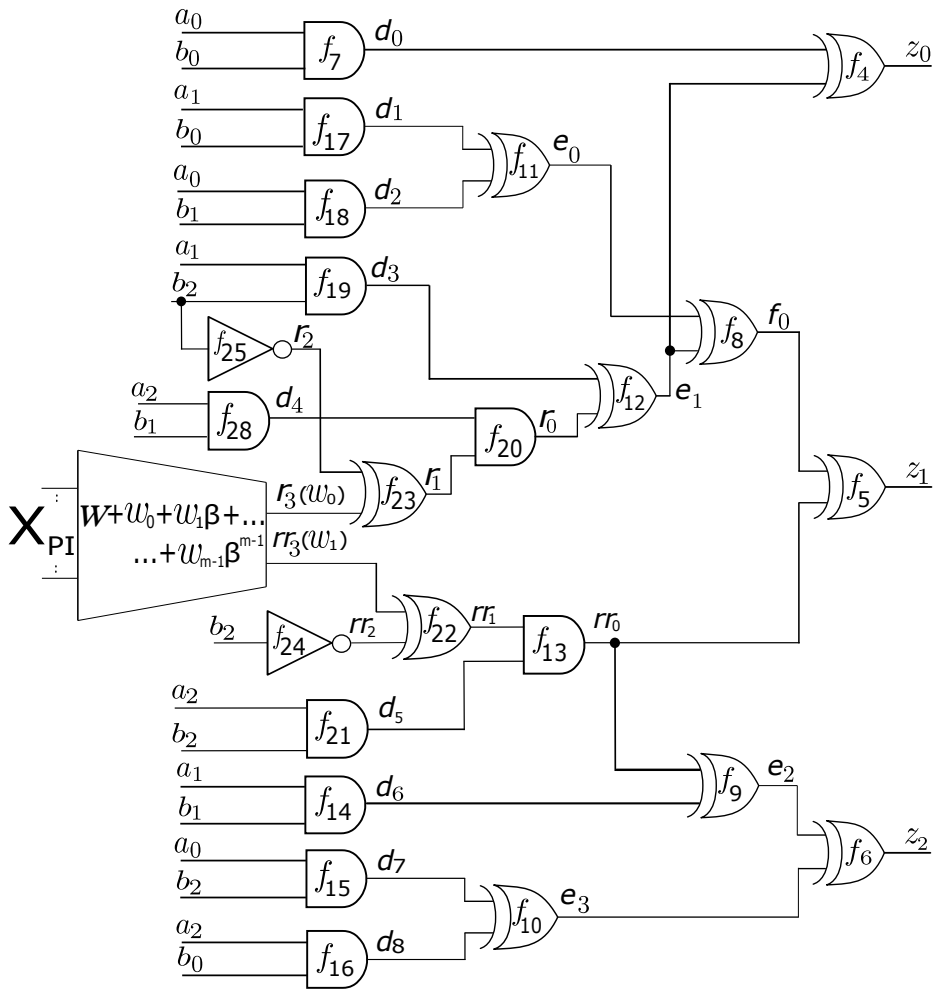
Contribution: Computing $P_k(x)$

- Property of finite fields, for any element $\phi \in \mathbb{F}_q$, $\phi^{q-1} = 1$
  - $\gamma = \alpha^{(2^k-1)/(2^n-1)} = \alpha^{\lambda}$
  - $\beta = \alpha^{(2^k-1)/(2^m-1)} = \alpha^{\mu}$
- Univariate Polynomial Factorization (UPF)
  - Obtain UPFs of $P_n(x^{\lambda})$ and $P_m(x^{\mu})$ in $\mathbb{F}_2[x]$

Contribution: Computing $P_k(x)$

- Property of finite fields, for any element $\phi \in \mathbb{F}_q$, $\phi^{q-1} = 1$
  - $\gamma = \alpha^{(2^k-1)/(2^n-1)} = \alpha^{\lambda}$
  - $\beta = \alpha^{(2^k-1)/(2^m-1)} = \alpha^{\mu}$
- Univariate Polynomial Factorization (UPF)
  - Obtain UPFs of $P_n(x^{\lambda})$ and $P_m(x^{\mu})$ in $\mathbb{F}_2[x]$
- Then, $\exists P_k(x) \in \mathbb{F}_2[x]$ as a common factor of $P_n(x^{\lambda})$ and $P_m(x^{\mu})$, such that:
  - $P_k(x)$ is a degree-k primitive polynomial in $\mathbb{F}_2[x]$ with $P_k(\alpha) = 0$

Application: Word-level representation



Patch function modeled as a 2-bit-vector word (m=2), $f_W : W + r_3 + \beta \cdot rr_3$

Application: Computing $P_k(x)$

- $P_3(x) = x^3 + x + 1, \quad P_2(x) = x^2 + x + 1, \quad \gamma = \alpha^9, \quad \beta = \alpha^{21}$

Application: Computing $P_k(x)$

- $P_3(x) = x^3 + x + 1, \;\; P_2(x) = x^2 + x + 1, \;\; \gamma = \alpha^9, \;\; \beta = \alpha^{21}$
- Composite field: $k = LCM(2, 3) = 6$

Application: Computing $P_k(x)$

- $P_3(x) = x^3 + x + 1$, $P_2(x) = x^2 + x + 1$, $\gamma = \alpha^9$, $\beta = \alpha^{21}$
- Composite field: $k = LCM(2, 3) = 6$
  - $UPF(P_3(x^9)) = (x^9)^3 + (x^9) + 1 =$
    $(x^6 + x^5 + x^2 + x + 1)(x^6 + x^5 + 1)(x^6 + x^4 + x^3 + x + 1)(x^6 +$
    $x^4 + x^2 + x + 1)(x^3 + x + 1)$;

Application: Computing $P_k(x)$

- $P_3(x) = x^3 + x + 1, \;\; P_2(x) = x^2 + x + 1, \;\; \gamma = \alpha^9, \;\; \beta = \alpha^{21}$
- Composite field: $k = LCM(2, 3) = 6$
  - $UPF(P_3(x^9)) = (x^9)^3 + (x^9) + 1 =$
    $(x^6 + x^5 + x^2 + x + 1)(x^6 + x^5 + 1)(x^6 + x^4 + x^3 + x + 1)(x^6 +$
    $x^4 + x^2 + x + 1)(x^3 + x + 1);$
  - $UPF(P_2(x^{21})) = (x^{21})^2 + (x^{21}) + 1 =$
    $(x^6 + x^5 + x^2 + x + 1)(x^6 + x^5 + 1)(x^6 + x^4 + x^3 + x + 1)(x^6 +$
    $x^5 + x^3 + x^2 + 1)(x^6 + x^5 + x^4 + x + 1)(x^6 + x + 1)(x^6 + x^3 + 1);$

Circuit Polynomials and Setup

- Ring $R = \mathbb{F}_{2^k}[Z, A, B, \ldots, W, r_3, rr_3, \ldots, a_0, a_1, \ldots, b_1, b_2]$

Circuit Polynomials and Setup

- Ring $R = \mathbb{F}_{2^k}[Z, A, B, \ldots, W, r_3, rr_3, \ldots, a_0, a_1, \ldots, b_1, b_2]$
- Circuit polynomials under a term order $>$:

$$f_1 : Z + z_0 + \gamma \cdot z_1 + \gamma^2 \cdot z_2; \qquad f_{22} : rr_1 + rr_3 + rr_2;$$
$$f_2 : A + a_0 + \gamma \cdot a_1 + \gamma^2 \cdot a_2; \qquad f_{23} : r_1 + r_2 + r_3;$$
$$f_3 : B + b_0 + \gamma \cdot b_1 + \gamma^2 \cdot b_2; \qquad f_{26} : r_3 + r_4 + d_4;$$
$$f_4 : z_0 + d_0 + e_1; \qquad f_{27} : rr_3 + rr_4 + b_2;$$
$$f_5 : z_1 + f_0 + rr_0; \qquad \ldots$$
$$\ldots \qquad f_{30} : rr_4 + a_2 + b_2 + a_2 b_2;$$
$$\ldots \qquad f_W : W + r_3 + \beta \cdot rr_3;$$

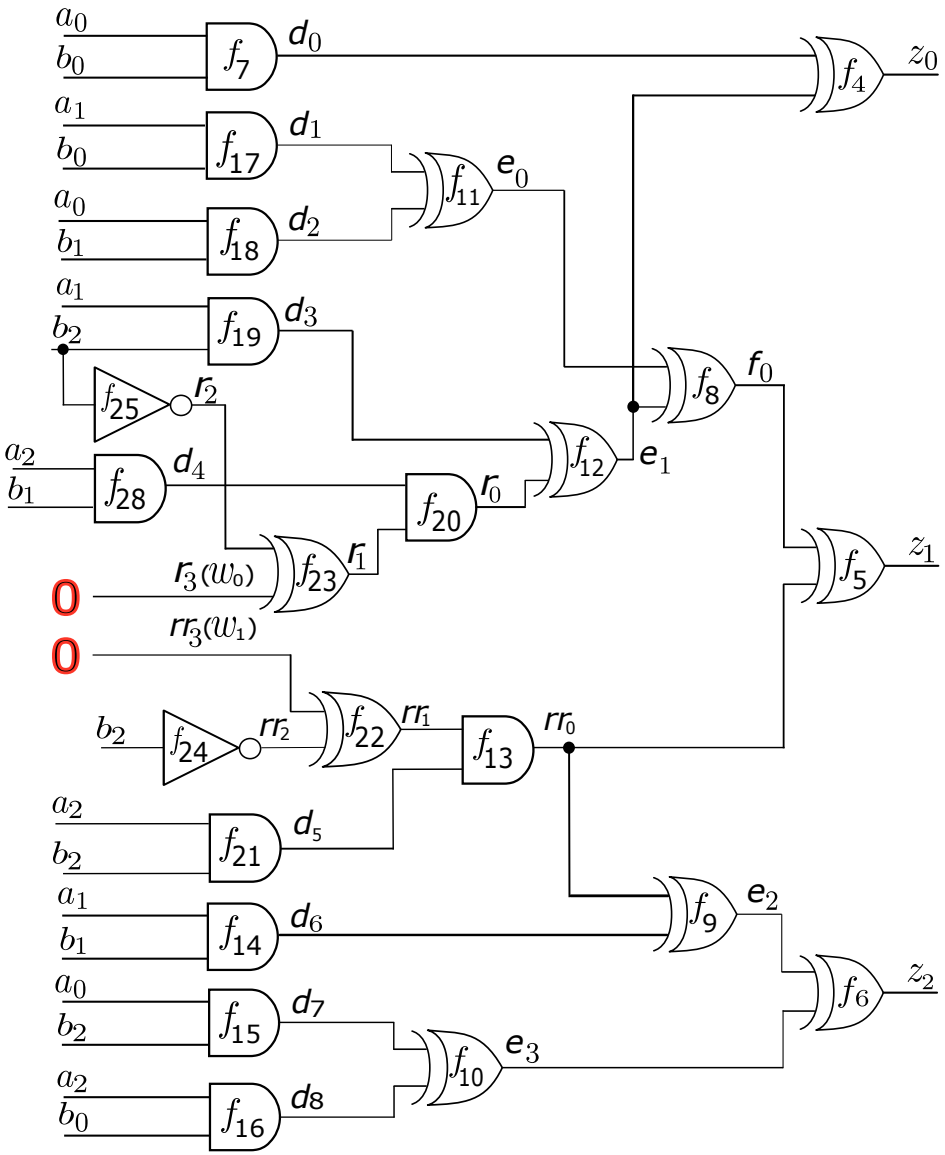
Circuit Polynomials and Setup

- Ring $R = \mathbb{F}_{2^k}[Z, A, B, \ldots, W, r_3, rr_3, \ldots, a_0, a_1, \ldots, b_1, b_2]$
- Circuit polynomials under a term order $>$:

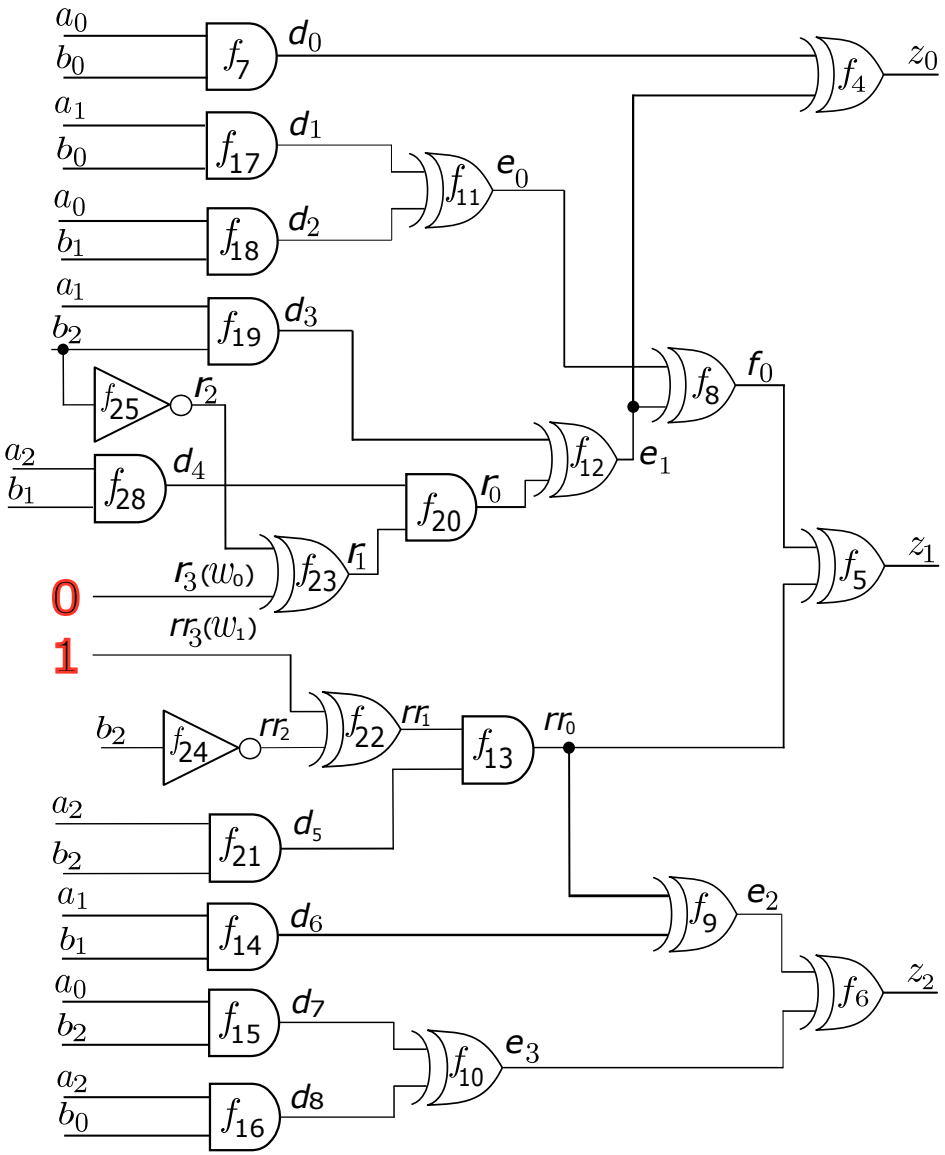
$$
\begin{aligned}
f_1 &: Z + z_0 + \gamma \cdot z_1 + \gamma^2 \cdot z_2; & f_{22} &: rr_1 + rr_3 + rr_2; \\
f_2 &: A + a_0 + \gamma \cdot a_1 + \gamma^2 \cdot a_2; & f_{23} &: r_1 + r_2 + r_3; \\
f_3 &: B + b_0 + \gamma \cdot b_1 + \gamma^2 \cdot b_2; & f_{26} &: r_3 + r_4 + d_4; \\
f_4 &: z_0 + d_0 + e_1; & f_{27} &: rr_3 + rr_4 + b_2; \\
f_5 &: z_1 + f_0 + rr_0; & &\ldots \\
&\ldots & f_{30} &: rr_4 + a_2 + b_2 + a_2 b_2; \\
&\ldots & f_W &: W + r_3 + \beta \cdot rr_3;
\end{aligned}
$$

- $F = \{f_1, \ldots, f_{30}, f_W\}$
- $F_0 = \{a_0^2 - a_0, \ldots, z_2^2 - z_2, A^8 - A, \ldots, Z^8 - Z, W^4 - W\}$.

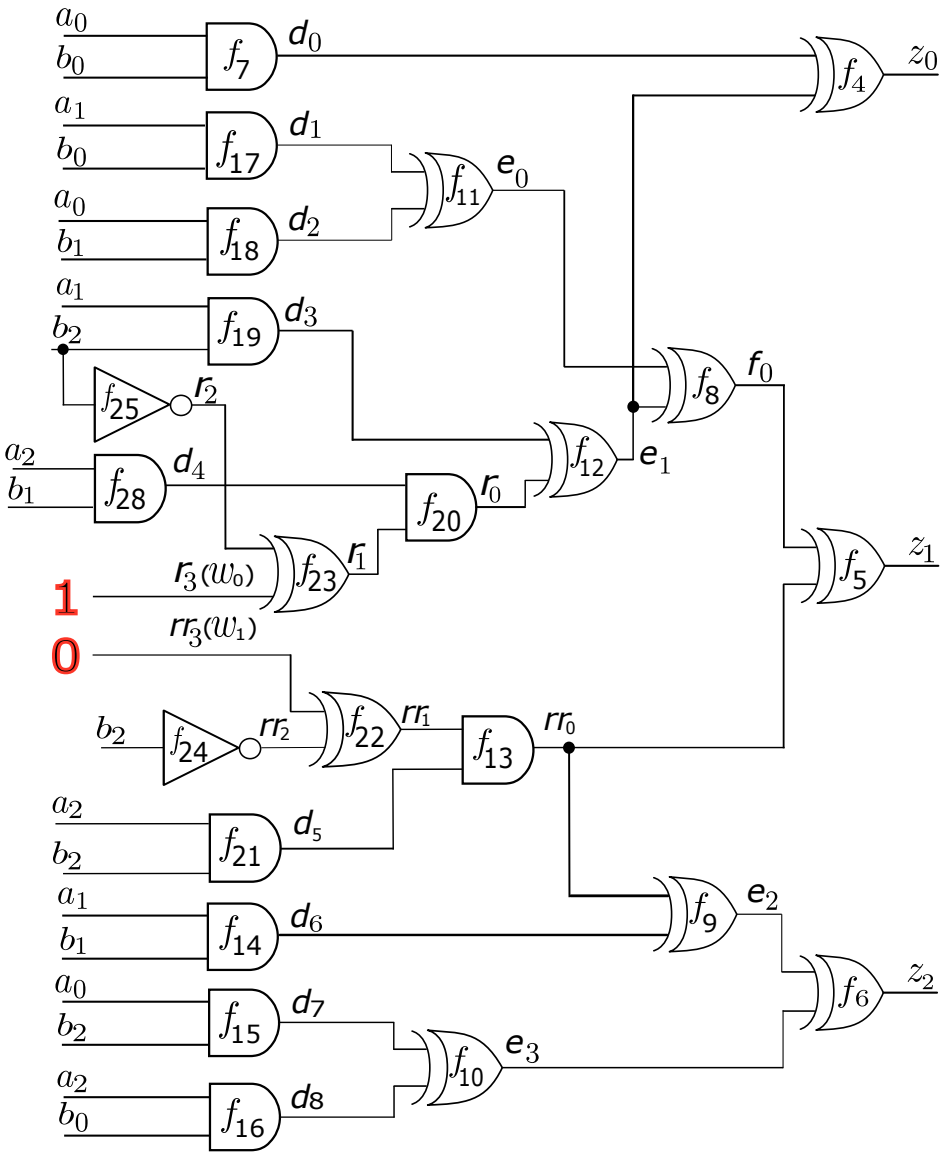
Rectification check: Remainder generation



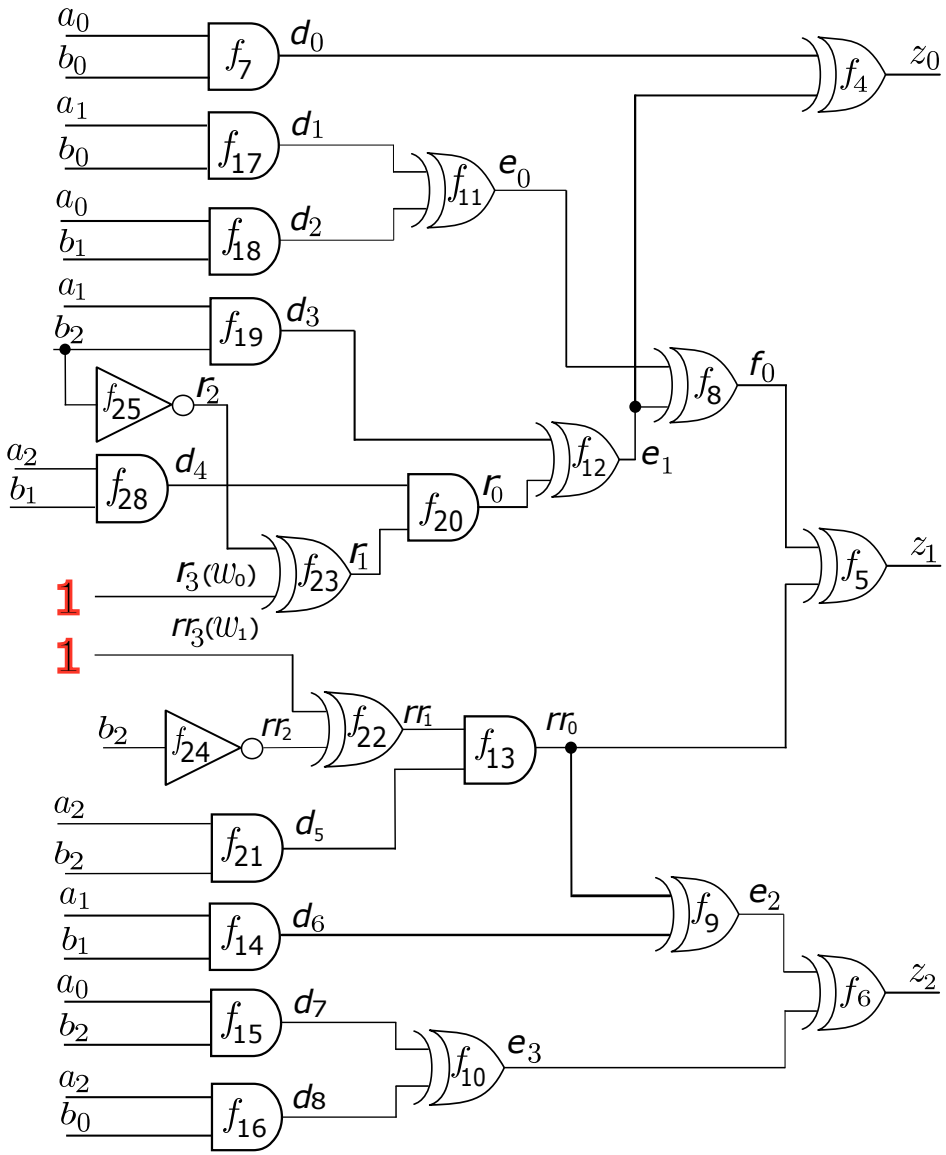
Rectification check: Remainder generation



Rectification check: Remainder generation



Rectification check: Remainder generation



Contribution: Multi-fix Rectification Check

- Constructing the F_l':
  - F_1', where $F_1'[f_W] = W + \delta(1) = W$,
  - F_2', where $F_2'[f_W] = W + \delta(2) = W + 1$,
  - F_3', where $F_3'[f_W] = W + \delta(3) = W + \beta$,
  - F_4', where $F_4'[f_W] = W + \delta(4) = W + \beta^2$

Contribution: Multi-fix Rectification Check

- Constructing the F'_l:
  - F'_1, where $F'_1[f_W] = W + \delta(1) = W$,
  - F'_2, where $F'_2[f_W] = W + \delta(2) = W + 1$,
  - F'_3, where $F'_3[f_W] = W + \delta(3) = W + \beta$,
  - F'_4, where $F'_4[f_W] = W + \delta(4) = W + \beta^2$

- Reducing the specification $f : Z + A \cdot B$:
  - $rem_1 = f \xrightarrow{F'_1 \cup F_0}_+ \alpha^{27}(a_2 b_1 b_2) + \alpha^{36}(a_2 b_2)$
  - $rem_2 = f \xrightarrow{F'_2 \cup F_0}_+ \alpha^{27}(a_2 b_1 b_2 + a_2 b_1) + \alpha^{36}(a_2 b_2)$
  - $rem_3 = f \xrightarrow{F'_3 \cup F_0}_+ \alpha^{27}(a_2 b_1 b_2)$
  - $rem_4 = f \xrightarrow{F'_4 \cup F_0}_+ \alpha^{27}(a_2 b_1 b_2 + a_2 b_1)$

Contribution: Multi-fix Rectification Check

- Constructing the F'_l:
  - F'_1, where $F'_1[f_W] = W + \delta(1) = W$,
  - F'_2, where $F'_2[f_W] = W + \delta(2) = W + 1$,
  - F'_3, where $F'_3[f_W] = W + \delta(3) = W + \beta$,
  - F'_4, where $F'_4[f_W] = W + \delta(4) = W + \beta^2$
- Reducing the specification $f : Z + A \cdot B$:
  - $rem_1 = f \xrightarrow{F'_1 \cup F_0}_+ \alpha^{27}(a_2 b_1 b_2) + \alpha^{36}(a_2 b_2)$
  - $rem_2 = f \xrightarrow{F'_2 \cup F_0}_+ \alpha^{27}(a_2 b_1 b_2 + a_2 b_1) + \alpha^{36}(a_2 b_2)$
  - $rem_3 = f \xrightarrow{F'_3 \cup F_0}_+ \alpha^{27}(a_2 b_1 b_2)$
  - $rem_4 = f \xrightarrow{F'_4 \cup F_0}_+ \alpha^{27}(a_2 b_1 b_2 + a_2 b_1)$
- $rem_1 \cdot rem_2 \cdot rem_3 \cdot rem_4 \xrightarrow{F_0}_+ 0$
- Target W with nets r_3 and rr_3 admits MFR

Implementation: Boolean Polynomials and ZDDs

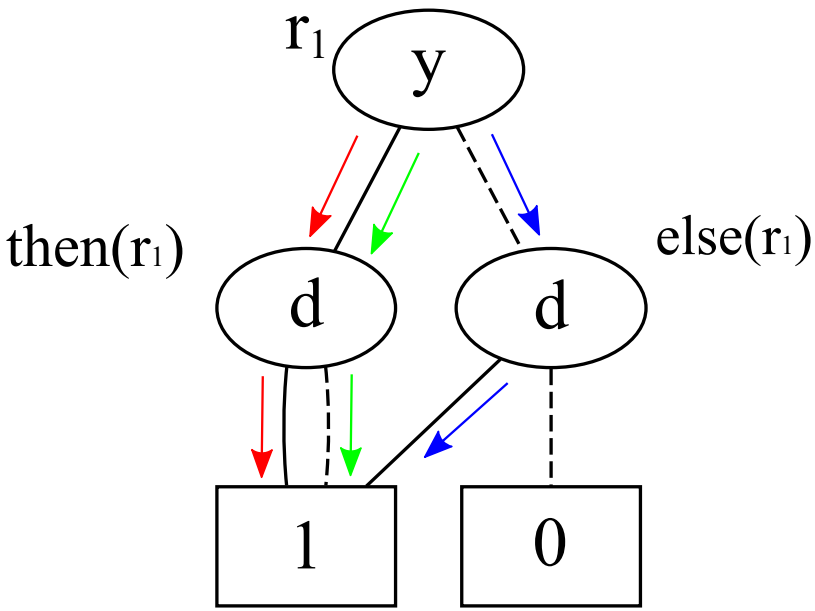
- Boolean polynomials as unate cube sets
  - Monomial: a product of positive literals or a cube
  - Polynomial: set of such cubes

Implementation: Boolean Polynomials and ZDDs

- Boolean polynomials as unate cube sets
  - Monomial: a product of positive literals or a cube
  - Polynomial: set of such cubes
- ZDDs efficient for manipulating unate cube sets [Minato, DAC'93]

Implementation: Boolean Polynomials and ZDDs

- Boolean polynomials as unate cube sets
  - Monomial: a product of positive literals or a cube
  - Polynomial: set of such cubes
- ZDDs efficient for manipulating unate cube sets [Minato, DAC'93]
- $r_1 = yd + y + d$ as $\{yd, y, d\}$

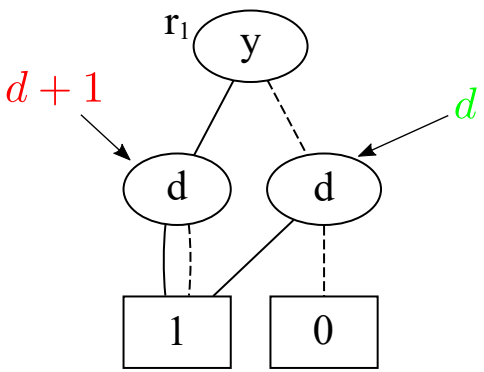


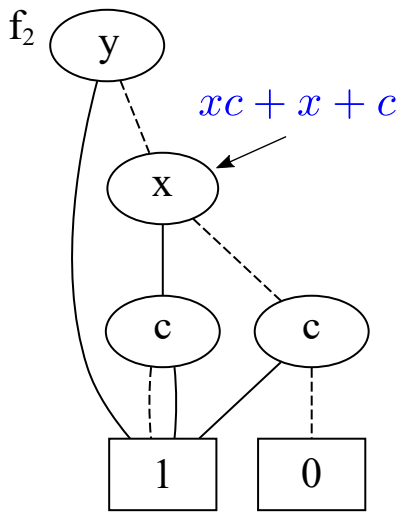
Paths terminating in 1: yd, y, d.

Improved Reduction Using ZDDs

- $r_1 = yd + y + d$, $f_2 = y + xc + x + c$, $r_1 \xrightarrow{f_2}_+$

$$
\begin{aligned}
&(yd + y + d) + (d + 1) \cdot (y + xc + x + c) \pmod 2 \\
&= 2 \cdot (yd + y) + d + (d + 1) \cdot (xc + x + c) \pmod 2 \\
&= d + (d + 1) \cdot (xc + x + c) \pmod 2
\end{aligned}
$$





- One step reduction: $else(r_1) + then(r_1) \cdot else(f_2)$ [Algorithm 6]

Implementation

- Custom software:

Implementation

- Custom software:
  - Reduction using ZDDs for remainder generation

Implementation

- Custom software:
  - Reduction using ZDDs for remainder generation
  - Singular to compute $P_k(x)$ and model composite field

Implementation

- Custom software:
  - Reduction using ZDDs for remainder generation
  - Singular to compute $P_k(x)$ and model composite field
  - Custom high level finite field engine

Implementation

- Custom software:
  - Reduction using ZDDs for remainder generation
  - Singular to compute $P_k(x)$ and model composite field
  - Custom high level finite field engine
    - Bit-vector and coefficient computations
    - Rectification check

Implementation

- Custom software:
  - Reduction using ZDDs for remainder generation
  - Singular to compute $P_k(x)$ and model composite field
  - Custom high level finite field engine
    - Bit-vector and coefficient computations
    - Rectification check
- Experiments performed on a 3.5GHz Intel(R) Core^{TM} i7-4770K Quad-Core CPU with 32 GB RAM

MFR Experiments: Mastrovito

n = Datapath Size, m = target word size, k = composite field size (degree of $P_k(X)$), AM = Maximum resident memory utilization in Mega Bytes, #G = Number of gates $\times 10^3$, #BO = Number of faulty outputs, PBS = Required time for PolyBori setup (ring declaration/poly collection/spec collection), VMS = Required time for verification, polynomial factorization and computing $P_k(X)$, and MFR setup, RC = Required time for MFR check, TE = Required time for total execution

n	***m***	***k***	**AM**	**#G**	**#BO**	**PBS**	**VMS**	**RC**	**TE**
16	5	80	100	0.8	6	0.04	0.06	0.12	0.22
32	5	160	120	2.8	8	0.13	0.12	0.4	0.65
163	5	815	550	69.8	6	6.04	3.36	11.9	21.3
233	2	466	750	119	3	13	1.2	0.01	14.2
283	2	566	1300	190	2	38	4.2	0.1	42.3
409	2	818	2400	384	2	190	5	0.1	195
571	2	1042	5000	827	5	2150	12	0.1	2162

MFR Experiments: Montgomery

n = Datapath Size, m = target word size, k = composite field size (degree of $P_k(X)$), AM = Maximum resident memory utilization in Mega Bytes, #G = Number of gates $\times 10^3$, #BO = Number of faulty outputs, PBS = Required time for PolyBori setup (ring declaration/poly collection/spec collection), VMS = Required time for verification, polynomial factorization and computing $P_k(X)$, and MFR setup, RC = Required time for MFR check, TE = Required time for total execution

n	*m*	*k*	AM	#G	#BO	PBS	VMS	RC	TE
16	5	80	100	0.9	16	0.04	0.56	35.6	36
32	5	160	120	2.8	32	0.13	0.57	27.6	28.3
163	5	815	550	57.5	128	5.2	6.8	262	274
233	2	466	750	112	233	11.5	3.5	360	375
283	2	566	1300	171	283	35	11	1503	1549
409	2	818	2400	340	409	134	10	4920	5064
571	2	1042	5000	663	12	1313	82	0.2	1395

MFR Experiments: Point Addition

n = Datapath Size, m = target word size, k = composite field size (degree of $P_k(X)$), AM = Maximum resident memory utilization in Mega Bytes, #G = Number of gates $\times 10^3$, #BO = Number of faulty outputs, PBS = Required time for PolyBori setup (ring declaration/poly collection/spec collection), VMS = Required time for verification, polynomial factorization and computing $P_k(X)$, and MFR setup, RC = Required time for MFR check, TE = Required time for total execution

n	m	k	AM	#G	#BO	PBS	VMS	RC	TE
16	5	80	100	0.9	7	0.06	0.11	1.73	1.9
32	5	160	120	2.9	13	0.18	0.8	134	135
163	5	815	550	71.6	22	15.7	4.7	15	35.4
233	2	466	750	122	233	19.2	2.15	0.15	21.5
283	2	566	1300	208	4	80.4	6.1	0.1	86.6
409	2	818	2400	368	409	220	10	2007	2237
571	2	1042	5000	813	5	2583	27	880	3490

Conclusion and Future work

- Algebraic approach for *m*-target MFR checking
  - Efficiency derived by interpreting targets as a bit-vector

Conclusion and Future work

- Algebraic approach for *m*-target MFR checking
  - Efficiency derived by interpreting targets as a bit-vector
- New mathematical insights for unified framework
  - Field incompatibility
  - Primitive polynomial computation

Conclusion and Future work

- Algebraic approach for *m*-target MFR checking
  - Efficiency derived by interpreting targets as a bit-vector
- New mathematical insights for unified framework
  - Field incompatibility
  - Primitive polynomial computation

- Computation of rectification function at the word-level
  - $W = a_2 b_1 b_2 + \beta \cdot a_2 b_2$
  - $r_3 = (a_2 \wedge b_1 \wedge b_2), \quad rr_3 = (a_2 \wedge b_2)$

Conclusion and Future work

- Algebraic approach for *m*-target MFR checking
  - Efficiency derived by interpreting targets as a bit-vector
- New mathematical insights for unified framework
  - Field incompatibility
  - Primitive polynomial computation

- Computation of rectification function at the word-level
  - $W = a_2 b_1 b_2 + \beta \cdot a_2 b_2$
  - $r_3 = (a_2 \wedge b_1 \wedge b_2), \quad rr_3 = (a_2 \wedge b_2)$
- Define and formulate existence of don't cares at the word-level

Conclusion and Future work

- Algebraic approach for *m*-target MFR checking
  - Efficiency derived by interpreting targets as a bit-vector
- New mathematical insights for unified framework
  - Field incompatibility
  - Primitive polynomial computation

- Computation of rectification function at the word-level
  - $W = a_2 b_1 b_2 + \beta \cdot a_2 b_2$
  - $r_3 = (a_2 \wedge b_1 \wedge b_2), \quad rr_3 = (a_2 \wedge b_2)$
- Define and formulate existence of don't cares at the word-level
- Extend the approach to integer arithmetic circuits

THANK YOU

Email: vikas.k.rao@utah.edu