Equivalence Checking using Gröbner Bases

Amr Sayed-Ahmed\textsuperscript{1} \hspace{1cm} Daniel Große\textsuperscript{1,2}
Mathias Soeken\textsuperscript{3} \hspace{1cm} Rolf Drechsler\textsuperscript{1,2}

\textsuperscript{1}University of Bremen, Germany
\textsuperscript{2}DFKI GmbH, Germany
\textsuperscript{3}EPFL, Switzerland

Email: asahmed@informatik.uni-bremen.de

FMCAD, October 2016
Introduction

- Formal verification circumvents costly bugs

- Automated verification of floating-point circuits at gate level is still a major challenge

- The proposed algebraic technique is a fully automated verification for floating-point circuits
Introduction

- Formal verification circumvents costly bugs

- Automated verification of floating-point circuits at gate level is still a major challenge

- The proposed algebraic technique is a fully automated verification for floating-point circuits
Introduction

- Formal verification circumvents costly bugs

- Automated verification of floating-point circuits at gate level is still a major challenge

- The proposed algebraic technique is a fully automated verification for floating-point circuits
Outline

Symbolic Computation

Algebraic Combinational Equivalence Checking (ACEC)
  Reverse Engineering
  Arithmetic Sweeping

Experimental Results

Conclusion
Outline

Symbolic Computation

Algebraic Combinational Equivalence Checking (ACEC)
Reverse Engineering
Arithmetic Sweeping

Experimental Results

Conclusion
Algebraic Decision Procedure

- Ideal Membership Testing:

Gröbner Bases Model $G = \{g_1, \ldots, g_s\}$

Equivalence Relationship $p_r$

Recursive Division $r$

Remainder Checking $r \neq 0$

Inconsistency $r = 0$

Equivalence
Modeling a Circuit as Gröbner Bases

- Modeling Logic Gates:

  \[ z = \neg a \Rightarrow g := -z + 1 - a \]
  \[ z = a \land b \Rightarrow g := -z + ab \]
  \[ z = a \lor b \Rightarrow g := -z + a + b - ab \]
  \[ z = a \oplus b \Rightarrow g := -z + a + b - 2ab \]
Modeling a Circuit as Gröbner Bases

- **Modeling Logic Gates:**
  \[ z = \neg a \Rightarrow g := -z + 1 - a \]
  \[ z = a \land b \Rightarrow g := -z + ab \]
  \[ z = a \lor b \Rightarrow g := -z + a + b - ab \]

- **Full Adder Example:**

```
g_1 := -c_{out} (-x_4x_3 + x_4 + x_3)
g_2 := -x_1 - 2ab + a + b
g_3 := -x_2 - ab + a + b
g_4 := -x_3 - 2ab + a + b
g_5 := -x_4 - 1 - a
g_6 := -1 - a
```

Diagram of the full adder with labeled inputs and outputs.
Modeling a Circuit as Gröbner Bases

- **Modeling Logic Gates:**
  \[
  z = \neg a \Rightarrow g := -z + 1 - a \\
  z = a \land b \Rightarrow g := -z + ab \\
  z = a \lor b \Rightarrow g := -z + a + b - ab
  \]
  \[
  z = a \oplus b \Rightarrow g := -z + a + b - 2ab
  \]

- **Full Adder Example:**

  ![Full Adder Diagram]

  Leading monomial: \( g_1 := -c_{out} \left( -x_4 x_3 + x_4 + x_3 \right) \)
  Tail terms: \( g_2 := -s - 2x_1 c + x_1 + c \)
Modeling a Circuit as Gröbner Bases

- **Modeling Logic Gates:**
  \[
  z = \neg a \Rightarrow g := -z + 1 - a \quad z = a \oplus b \Rightarrow g := -z + a + b - 2ab \\
  z = a \land b \Rightarrow g := -z + ab \quad z = a \lor b \Rightarrow g := -z + a + b - ab
  \]

- **Full Adder Example:**

  ![Full Adder Diagram]

  leading monomial \quad \text{tail terms}

  \[
  g_1 := -c_{\text{out}}(x_4 + x_3 + x_4 + x_3) \\
  g_3 := -x_4 + x_2 c \\
  g_5 := -x_2 - ab + a + b \\
  g_2 := -s - 2x_1 c + x_1 + c \\
  g_4 := -x_3 + ab \\
  g_6 := -x_1 - 2ab + a + b
  \]
Modeling a Circuit as Gröbner Bases

- **Modeling Logic Gates:**

  \[
  z = \neg a \Rightarrow g := -z + 1 - a \\
  z = a \oplus b \Rightarrow g := -z + a + b - 2ab \\
  z = a \land b \Rightarrow g := -z + ab \\
  z = a \lor b \Rightarrow g := -z + a + b - ab
  \]

- **Full Adder Example:**

<table>
<thead>
<tr>
<th>leading monomial</th>
<th>tail terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g_1 := -c_{out} ( -x_4 x_3 + x_4 + x_3 ) )</td>
<td>( g_2 := -s - 2x_1 c + x_1 + c )</td>
</tr>
<tr>
<td>( g_3 := -x_4 + x_2 c )</td>
<td>( g_4 := -x_3 + ab )</td>
</tr>
<tr>
<td>( g_5 := -x_2 - ab + a + b )</td>
<td>( g_6 := -x_1 - 2ab + a + b )</td>
</tr>
</tbody>
</table>

- **Leading monomials are relatively prime** \( \implies \) **The model is Gröbner bases**
Ideal Membership Testing

- **Following Full Adder Example**: specification polynomial
  
  \[ p_r := -2c_{cout} - s + c + b + a \]

- Its model
  
  \[ g_1 := -c_{out} \left( -x_4x_3 + x_4 + x_3 \right) \]
  \[ g_2 := -s - 2x_1c + x_1 + c \]
  \[ g_3 := -x_4 + x_2c \]
  \[ g_4 := -x_3 + ab \]
  \[ g_5 := -x_2 - ab + a + b \]
  \[ g_6 := -x_1 - 2ab + a + b \]

- **Recursive Division**: 
Ideal Membership Testing

- **Following Full Adder Example**: specification polynomial
  \[ p_r := -2c_{\text{cout}} - s + c + b + a \]

- **Its model**
  \[
  \begin{align*}
  g_1 & := -c_{\text{out}}(-x_4x_3 + x_4 + x_3) \\
  g_3 & := -x_4 + x_2c \\
  g_5 & := -x_2 - ab + a + b \\
  g_2 & := -s - 2x_1c + x_1 + c \\
  g_4 & := -x_3 + ab \\
  g_6 & := -x_1 - 2ab + a + b
  \end{align*}
  \]

- **Recursive Division**:
  \[
  \begin{align*}
  p_r & := -2c_{\text{cout}} - s + c + b + a \xrightarrow{g_1} \\
  -s\left[2x_4x_3 - 2x_4 - 2x_3\right] + c + b + a & \xrightarrow{g_2} 
  \end{align*}
  \]
Ideal Membership Testing

► **Following Full Adder Example:** specification polynomial

\[ p_r := -2c_{cout} - s + c + b + a \]

► Its model

\[
\begin{align*}
g_1 &:= -c_{out} (-x_4 x_3 + x_4 + x_3) \\
g_2 &:= -s - 2x_1 c + x_1 + c \\
g_3 &:= -x_4 + x_2 c \\
g_4 &:= -x_3 + ab \\
g_5 &:= -x_2 - ab + a + b \\
g_6 &:= -x_1 - 2ab + a + b
\end{align*}
\]

► **Recursive Division:**

\[
\begin{array}{r}
\frac{g_2}{\rightarrow 2x_4 x_3 - 2x_4 - 2x_3 + 2x_1 c - x_1 + b + a}
\end{array}
\]
Ideal Membership Testing

- **Following Full Adder Example**: specification polynomial
  \[ p_r := -2c_{\text{cout}} - s + c + b + a \]

- Its model
  \[ g_1 := -c_{\text{out}} \left( -x_4 x_3 + x_4 + x_3 \right) \]
  \[ g_2 := -s - 2x_1 c + x_1 + c \]
  \[ g_3 := -x_4 + x_2 c \]
  \[ g_4 := -x_3 + ab \]
  \[ g_5 := -x_2 - ab + a + b \]
  \[ g_6 := -x_1 - 2ab + a + b \]

- **Recursive Division**:
  \[ g_3 \rightarrow 2x_3 x_2 c - 2x_3 - 2x_2 c + 2x_1 c - x_1 + b + a \]
  \[ g_4 \rightarrow \]
Ideal Membership Testing

- **Following Full Adder Example:** specification polynomial
  \[ p_r := -2c_{cout} - s + c + b + a \]

- Its model
  \[
  \begin{align*}
  g_1 & := -c_{out} (x_4 x_3 + x_4 + x_3) \\
  g_2 & := -s - 2x_1 c + x_1 + c \\
  g_3 & := -x_4 + x_2 c \\
  g_4 & := -x_3 + a b \\
  g_5 & := -x_2 - a b + a + b \\
  g_6 & := -x_1 - 2a b + a + b 
  \end{align*}
  \]

- **Recursive Division:**
  \[
  \begin{align*}
  g_4 & \rightarrow 2x_2 cba - 2x_2 c + 2x_1 c - x_1 - 2ba + b + a \\
  g_5 & \rightarrow 2x_1 c - x_1 + 4cba - 2ca - 2cb - 2ab + b + a \\
  g_6 & \rightarrow 0
  \end{align*}
  \]
Outline

Symbolic Computation

Algebraic Combinational Equivalence Checking (ACEC)
  Reverse Engineering
  Arithmetic Sweeping

Experimental Results

Conclusion
Flow of ACEC

Circuit Netlist 1

\[ N_1 \]

Gröbner Modeling

\[ G_1 \]

Combined Model

\[ G \]

Circuit Netlist 2

\[ N_2 \]

Gröbner Modeling

\[ G_2 \]
Flow of ACEC

Circuit Netlist 1 → \( N_1 \) → Gröbner Modeling → \( G_1 \) → Combined Model

Circuit Netlist 2 → \( N_2 \) → Gröbner Modeling → \( G_2 \) → Combined Model

\( G \) → Membership Testing

Output Relationships → Equivalence → Inconsistency
Flow of ACEC

Circuit Netlist 1  →  Gröbner Modeling  →  Combined Model  →  Gröbner Modeling  →  Circuit Netlist 2

\[ N_1 \rightarrow G_1 \rightarrow G \rightarrow G_2 \rightarrow N_2 \]

Membership Testing

Output Relationships  →  Equivalence  →  Computationally Infeasible

Inconsistency
Flow of ACEC

Circuit Netlist 1

Gröbner Modeling

$N_1$

$G_1$

Combined Model

Reverse Engineering

$wG$

$G'$

Circuit Netlist 2

Gröbner Modeling

$N_2$

$G_2$

$G'$: Rewritten Combined Model

$wG$: Abstracted Polynomials Set of Arithmetic Units

Model Rewriting

Identifying & Abstracting Arithmetic Units

$wG$
Flow of ACEC

Reverse Engineering

Arithmetic Sweeping

$wG \downarrow \downarrow G'$

Deducing Relationships

Internal Relationships

Membership Testing

Equivelance/Inconsistency

Model Simplification

$sG$
Flow of ACEC

- Reverse Engineering
- Arithmetic Sweeping
- Membership Testing
  - Output Relationships
  - Inconsistency
- Equivalence
Outline

Symbolic Computation

Algebraic Combinational Equivalence Checking (ACEC)
  Reverse Engineering
  Arithmetic Sweeping

Experimental Results

Conclusion
Reverse Engineering

- Based on detecting carry bits propagation within arithmetic units (integer adders and multipliers)
- Full adder model revealing carry terms:
  \[ g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba \]
  \[ g_2 : -c_{\text{out}} - 2cba + cb + ca + ba \]
- Identifying subsets of polynomials that share carry terms, therefore, model arithmetic components
- Model rewriting is required for:
  - Revealing carry terms
  - Removing vanishing monomials (redundant monomials that always evaluate to zero)
- Abstraction by Gaussian elimination, for the full adder:
  \[ 2g_2 + g_1 \rightarrow g_r : -2c_{\text{out}} - s + c + b + a \]
Reverse Engineering

- Based on detecting carry bits propagation within arithmetic units (integer adders and multipliers)
- **Full adder model revealing carry terms:**
  
  $g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba$
  
  $g_2 : -c_{out} - 2cba + cb + ca + ba$

- Identifying subsets of polynomials that share carry terms, therefore, model arithmetic components
- Model rewriting is required for:
  
  - Revealing carry terms
  - Removing vanishing monomials (redundant monomials that always evaluate to zero)

- Abstraction by Gaussian elimination, for the full adder:

  $$2g_2 + g_1 \rightarrow g_r : -2c_{out} - s + c + b + a$$
Reverse Engineering

- Based on detecting carry bits propagation within arithmetic units (integer adders and multipliers)
- Full adder model revealing carry terms:
  \[ g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba \]
  \[ g_2 : -c_{out} - 2cba + cb + ca + ba \]
- Identifying subsets of polynomials that share carry terms, therefore, model arithmetic components
- Model rewriting is required for:
  - Revealing carry terms
  - Removing vanishing monomials (redundant monomials that always evaluate to zero)
- Abstraction by Gaussian elimination, for the full adder:
  \[ 2g_2 + g_1 \rightarrow g_r : -2c_{out} - s + c + b + a \]
Reverse Engineering

- Based on detecting carry bits propagation within arithmetic units (integer adders and multipliers)
- Full adder model revealing carry terms:
  \[ g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba \]
  \[ g_2 : -c_{out} - 2cba + cb + ca + ba \]
- Identifying subsets of polynomials that share carry terms, therefore, model arithmetic components
- Model rewriting is required for:
  - Revealing carry terms
  - Removing vanishing monomials (redundant monomials that always evaluate to zero)
- Abstraction by Gaussian elimination, for the full adder:
  \[ 2g_2 + g_1 \rightarrow g_r : -2c_{out} - s + c + b + a \]
Reverse Engineering

- Based on detecting carry bits propagation within arithmetic units (integer adders and multipliers)
- Full adder model revealing carry terms:
  \[ g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba \]
  \[ g_2 : -c_{\text{out}} - 2cba + cb + ca + ba \]
- Identifying subsets of polynomials that share carry terms, therefore, model arithmetic components
- Model rewriting is required for:
  - Revealing carry terms
  - Removing vanishing monomials (redundant monomials that always evaluate to zero)
- Abstraction by Gaussian elimination, for the full adder:
  \[ 2g_2 + g_1 \rightarrow g_r : -2c_{\text{out}} - s + c + b + a \]
Reverse Engineering

- Based on detecting carry bits propagation within arithmetic units (integer adders and multipliers)
- Full adder model revealing carry terms:
  \[ g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba \]
  \[ g_2 : -c_{out} - 2cba + cb + ca + ba \]
- Identifying subsets of polynomials that share carry terms, therefore, model arithmetic components
- Model rewriting is required for:
  - Revealing carry terms
  - Removing vanishing monomials (redundant monomials that always evaluate to zero)
- Abstraction by Gaussian elimination, for the full adder:
  \[ 2g_2 + g_1 \rightarrow g_r : -2c_{out} - s + c + b + a \]
Reverse Engineering

- Based on detecting carry bits propagation within arithmetic units (integer adders and multipliers)
- Full adder model revealing carry terms:
  \[ g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba \]
  \[ g_2 : -c_{\text{out}} - 2cba + cb + ca + ba \]
- Identifying subsets of polynomials that share carry terms, therefore, model arithmetic components
- Model rewriting is required for:
  - Revealing carry terms
  - Removing vanishing monomials (redundant monomials that always evaluate to zero)
- Abstraction by Gaussian elimination, for the full adder:
  \[ 2g_2 + g_1 \rightarrow g_r : -2c_{\text{out}} - s + c + b + a \]
Reverse Engineering: 1) Model Rewriting

- **XOR rewriting** preserves inputs and outputs of chains of XOR gates

- **Parallel Adder Model:**
  \[
  c_2 = D_2 \lor (X_2 \land D_1) \lor (X_2 \land X_1 \land D_0) \implies g_1 :=
  -c_2 + X_2 X_1 a_2 b_2 a_1 b_1 a_0 b_0 - X_2 X_1 a_1 b_1 a_0 b_0 - X_2 X_1 a_2 b_2 a_0 b_0 -
  X_2 a_2 b_2 a_1 b_1 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2
  \]

  \[
  s_2 = X_2 \oplus c_1 \implies g_2 := -s_2 - 2c_1 X_2 + c_1 + X_2
  
  c_1 = D_1 \lor (X_1 \land D_0) \implies g_3 := -c_1 - X_1 a_1 b_1 a_0 b_0 + X_1 a_0 b_0 + a_1 b_1
  
  s_1 = X_1 \oplus c_0 \implies g_4 := -s_1 - 2c_0 X_1 + c_0 + X_1
  
  c_0 = D_0 \implies g_5 := -c_0 + a_0 b_0
  
  s_0 = X_0 \implies g_6 := -s_0 + X_0
  
  X_i = a_i \oplus b_i \implies g_{k-i-1} := -X_i - 2a_i b_i + b_i + a_i
  
  D_i = a_i \land b_i \implies g_{k-i} := -D_i + a_i b_i
  \]
Reverse Engineering: 1) Model Rewriting

- **Common rewriting** preserves shared variables between polynomials

- Parallel adder model after XOR rewriting:
  
  \[ g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \]
  
  \[ g_2 := -s_2 - 2c_1 X_2 + c_1 + X_2 \]
  
  \[ g_3 := -c_1 + X_1 a_0 b_0 + a_1 b_1 \]
  
  \[ g_4 := -s_1 - 2c_0 X_1 + c_0 + X_1 \]
  
  \[ g_5 := -c_0 + a_0 b_0 \]
  
  \[ g_6 := -s_0 + X_0 \]
  
  \[ g_7 := -X_0 - 2a_0 b_0 + b_0 + a_0 \]
  
  \[ g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \]
  
  \[ g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 \]

- \[ X_0, c_0 \text{ and } c_1 \] will be eliminated
Reverse Engineering: 1) Model Rewriting

- **Common rewriting** preserves shared variables between polynomials
- **Parallel adder model after XOR rewriting:**
  \[
  g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \\
  g_2 := -s_2 - 2c_1 X_2 + c_1 + X_2 \\
  g_3 := -c_1 + X_1 a_0 b_0 + a_1 b_1 \\
  g_4 := -s_1 - 2c_0 X_1 + c_0 + X_1 \\
  g_5 := -c_0 + a_0 b_0 \\
  g_6 := -s_0 + X_0 \\
  g_7 := -X_0 - 2a_0 b_0 + b_0 + a_0 \\
  g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \\
  g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 \\
  
- **X_0, c_0 and c_1** will be eliminated
Reverse Engineering: 1) Model Rewriting

- Common rewriting preserves shared variables between polynomials
- Parallel adder model after XOR rewriting:
  \[ g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \]
  \[ g_2 := -s_2 - 2c_1 X_2 + c_1 + X_2 \]
  \[ g_3 := -c_1 + X_1 a_0 b_0 + a_1 b_1 \]
  \[ g_4 := -s_1 - 2c_0 X_1 + c_0 + X_1 \]
  \[ g_5 := -c_0 + a_0 b_0 \]
  \[ g_6 := -s_0 + X_0 \]
  \[ g_7 := -X_0 - 2a_0 b_0 + b_0 + a_0 \]
  \[ g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \]
  \[ g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 \]
- \( X_0, c_0 \) and \( c_1 \) will be eliminated
Reverse Engineering: 2) Extracting Arithmetic Units

▶ Parallel adder model after common rewriting:

\[ g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \]

\[ g_2 := -s_2 - 2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1 + X_2 + X_1 a_0 b_0 + a_1 b_1 \]

\[ g_4 := -s_1 - 2X_1 a_0 b_0 + a_0 b_0 + X_1 \]

\[ g_6 := -s_0 + -2a_0 b_0 + b_0 + a_0 \]

\[ g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \]

\[ g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 \]

▶ Abstraction by Gaussian elimination:
Reverse Engineering: 2) Extracting Arithmetic Units

- Parallel adder model after common rewriting:
  \[ g_1 := -c_2 + X_2X_1a_0b_0 + X_2a_1b_1 + a_2b_2 \]
  \[ g_2 := -s_2 - 2X_2X_1a_0b_0 - 2X_2a_1b_1 + X_2 + X_1a_0b_0 + a_1b_1 \]
  \[ g_4 := -s_1 - 2X_1a_0b_0 + a_0b_0 + X_1 \]
  \[ g_6 := -s_0 + -2a_0b_0 + b_0 + a_0 \]
  \[ g_8 := -X_1 - 2a_1b_1 + b_1 + a_1 \]
  \[ g_9 := -X_2 - 2a_2b_2 + b_2 + a_2 \]

- Abstraction by *Gaussian elimination*:
Reverse Engineering: 2) Extracting Arithmetic Units

- Parallel adder model after common rewriting:
  \[ g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \]
  \[ g_2 := -s_2 - 2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1 + X_2 + X_1 a_0 b_0 + a_1 b_1 \]
  \[ g_4 := -s_1 - 2X_1 a_0 b_0 + a_0 b_0 + X_1 \]
  \[ g_6 := -s_0 + -2a_0 b_0 + b_0 + a_0 \]
  \[ g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \]
  \[ g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 \]

- Abstraction by *Gaussian elimination*:
  \[ 2g_1 + g_2 \rightarrow g_{\text{res}} := -2c_2 + \overline{2X_2 X_1 a_0 b_0 + 2X_2 a_1 b_1} + 2a_2 b_2 - s_2 \overline{2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1} + X_2 + X_1 a_0 b_0 + a_1 b_1 \]
Reverse Engineering: 2) Extracting Arithmetic Units

- Parallel adder model after common rewriting:
  \[ g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \]
  \[ g_2 := -s_2 - 2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1 + X_2 + X_1 a_0 b_0 + a_1 b_1 \]
  \[ g_4 := -s_1 - 2X_1 a_0 b_0 + a_0 b_0 + X_1 \]
  \[ g_6 := -s_0 + -2a_0 b_0 + b_0 + a_0 \]
  \[ g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \]
  \[ g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 \]

- Abstraction by **Gaussian elimination**:
  \[ 2g_1 + g_2 \rightarrow g_{\text{res}} := -2c_2 \left[ +2X_2 X_1 a_0 b_0 + 2X_2 a_1 b_1 \right] + 2a_2 b_2 - \]
  \[ s_2 \left[ -2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1 \right] + X_2 + X_1 a_0 b_0 + a_1 b_1 \]
Reverse Engineering: 2) Extracting Arithmetic Units

- Parallel adder model after common rewriting:
  \[ g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \]
  \[ g_2 := -s_2 - 2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1 + X_2 + X_1 a_0 b_0 + a_1 b_1 \]
  \[ g_4 := -s_1 - 2X_1 a_0 b_0 + a_0 b_0 + X_1 \]
  \[ g_6 := -s_0 + -2a_0 b_0 + b_0 + a_0 \]
  \[ g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \]
  \[ g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 \]

- Abstraction by Gaussian elimination:
  \[ g_{\text{res}} := -2c_2 - s_2 + X_2 + X_1 a_0 b_0 + 2a_2 b_2 + a_1 b_1 \]
  \[ 2g_{\text{res}} + g_4 \rightarrow g_{\text{res}} := -4c_2 - 2s_2 - s_1 + 2X_2 + X_1 + 4a_2 b_2 + 2a_1 b_1 + a_0 b_0 \]
Reverse Engineering: 2) Extracting Arithmetic Units

- Parallel adder model after common rewriting:
  
  \[
  g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \\
  g_2 := -s_2 - 2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1 + X_2 + X_1 a_0 b_0 + a_1 b_1 \\
  g_4 := -s_1 - 2X_1 a_0 b_0 + a_0 b_0 + X_1 \\
  g_6 := -s_0 + -2a_0 b_0 + b_0 + a_0 \\
  g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \\
  g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 
  \]

- Abstraction by Gaussian elimination:
  
  \[
  g_{\text{res}} := -4c_2 - 2s_2 - s_1 + 2X_2 + X_1 + 4a_2 b_2 + 2a_1 b_1 + a_0 b_0 \\
  2g_{\text{res}} + g_6 \rightarrow g_{\text{res}} := -8c_2 - 4s_2 - 2s_1 - s_0 + 4X_2 + 2X_1 + 8a_2 b_2 + 4a_1 b_1 + b_0 + a_0
  \]
Reverse Engineering: 2) Extracting Arithmetic Units

- Parallel adder model after common rewriting:
  \[ 
  g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \\
  g_2 := -s_2 - 2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1 + X_2 + X_1 a_0 b_0 + a_1 b_1 \\
  g_4 := -s_1 - 2X_1 a_0 b_0 + a_0 b_0 + X_1 \\
  g_6 := -s_0 + -2a_0 b_0 + b_0 + a_0 \\
  g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \\
  g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 
  \]

- Abstraction by Gaussian elimination:
  \[ 
  g_{\text{res}} := -8c_2 - 4s_2 - 2s_1 - s_0 + 4X_2 + 2X_1 + 8a_2 b_2 + 4a_1 b_1 + b_0 + a_0 \\
  g_{\text{res}} + 2g_8 \rightarrow g_{\text{res}} := \\
  -8c_2 - 4s_2 - 2s_1 - s_0 + 4X_2 + 8a_2 b_2 + 2b_1 + 2a_1 + b_0 + a_0 
  \]
Reverse Engineering: 2) Extracting Arithmetic Units

- Parallel adder model after common rewriting:
  \[ g_1 := -c_2 + X_2 X_1 a_0 b_0 + X_2 a_1 b_1 + a_2 b_2 \]
  \[ g_2 := -s_2 - 2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1 + X_2 + X_1 a_0 b_0 + a_1 b_1 \]
  \[ g_4 := -s_1 - 2X_1 a_0 b_0 + a_0 b_0 + X_1 \]
  \[ g_6 := -s_0 + -2a_0 b_0 + b_0 + a_0 \]
  \[ g_8 := -X_1 - 2a_1 b_1 + b_1 + a_1 \]
  \[ g_9 := -X_2 - 2a_2 b_2 + b_2 + a_2 \]

- Abstraction by Gaussian elimination:
  \[ g_{\text{res}} := -8c_2 - 4s_2 - 2s_1 - s_0 + 4X_2 + 8a_2 b_2 + 2b_1 + 2a_1 + b_0 + a_0 \]
  \[ g_{\text{res}} + 4g_9 \rightarrow \]
  \[ g_{\text{res}} := -8c_2 - 4s_2 - 2s_1 - s_0 + 4b_2 + 4a_2 + 2b_1 + 2a_1 + b_0 + a_0 \]
Outline

Symbolic Computation

Algebraic Combinational Equivalence Checking (ACEC)
  Reverse Engineering
  Arithmetic Sweeping

Experimental Results

Conclusion
Arithmetic Sweeping

Reverse Engineering

Arithmetic Sweeping

\[ wG \]

\[ G' \]

\[ sG \]

Deducing Relationships

Internal Relationships

Membership Testing

Equivelance/Inconsistency

Model Simplification

\[ G' \]

\[ sG \]
Deducing Relationships

Partitioning the combined model based on the extracted arithmetic information
Deducing and testing bit relationships between variables of the transitive fan-in of arithmetic units
Deducing and Testing Relationships

- Testing the word relationship between output variables of compared arithmetic units, using the abstracted polynomials

\[-2^{n-1}s_{n-1} - \cdots - s_0 + 2^{n-1}\hat{s}_{n-1} + \cdots + \hat{s}_0 \xrightarrow{wG} + r\]
Merging proved equivalent variables simplifies the combined model dramatically.

Therefore, testing output relationships wrt. the simplified model is computationally feasible.
Model Simplification

- Merging proved equivalent variables simplifies the combined model dramatically.
- Therefore, testing output relationships wrt. the simplified model is computationally feasible.
Outline

Symbolic Computation

Algebraic Combinational Equivalence Checking (ACEC)

Reverse Engineering

Arithmetic Sweeping

Experimental Results

Conclusion
Experimental Results

Figure: Compared FP Multiplier Circuits
## Experimental Results

<table>
<thead>
<tr>
<th>Multiplier Architecture</th>
<th>FP operand # bits</th>
<th>Commercial h:m:s</th>
<th>ABC h:m:s</th>
<th>ACEC h:m:s</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP-CT-BK</td>
<td>16</td>
<td>00:08:50</td>
<td>TO</td>
<td>00:01:42</td>
</tr>
<tr>
<td>SP-WT-CH</td>
<td>16</td>
<td>00:09:08</td>
<td>TO</td>
<td>00:01:44</td>
</tr>
<tr>
<td>SP-CT-BK</td>
<td>24</td>
<td>TO</td>
<td>TO</td>
<td>00:17:49</td>
</tr>
<tr>
<td>SP-WT-CH</td>
<td>24</td>
<td>TO</td>
<td>TO</td>
<td>00:25:58</td>
</tr>
<tr>
<td>SP-CT-BK</td>
<td>32</td>
<td>TO</td>
<td>TO</td>
<td>02:24:01</td>
</tr>
<tr>
<td>SP-WT-CH</td>
<td>32</td>
<td>TO</td>
<td>TO</td>
<td>03:41:43</td>
</tr>
</tbody>
</table>

SP → Simple Partial Product  
WT → Wallace Tree  
CT → Compressor Tree  
CH → Carry Look Ahead Adder  
BK → Brent-Kung Adder  

TO=100 Hour
Outline

Symbolic Computation

Algebraic Combinational Equivalence Checking (ACEC)
Reverse Engineering
Arithmetic Sweeping

Experimental Results

Conclusion
Conclusion

- New algebraic equivalence checking technique for circuits that combine data-path and control logic
  - New reverse engineering algorithm to extract and abstract arithmetic components
  - Arithmetic sweeping based on input and output boundaries of the abstracted components
  - Efficient polynomial representation (negative-Davio decomposition)

- Checking equivalence of large floating-point multipliers which cannot be verified by state-of-art equivalence checkers
- Verifying heavy optimized circuits and dealing with non-equivalent circuits are still major challenges
Conclusion

- New algebraic equivalence checking technique for circuits that combine data-path and control logic
  - New reverse engineering algorithm to extract and abstract arithmetic components
  - Arithmetic sweeping based on input and output boundaries of the abstracted components
  - Efficient polynomial representation (negative-Davio decomposition)
- Checking equivalence of large floating-point multipliers which cannot be verified by state-of-art equivalence checkers
- Verifying heavy optimized circuits and dealing with non-equivalent circuits are still major challenges
Conclusion

- New algebraic equivalence checking technique for circuits that combine data-path and control logic
  - New reverse engineering algorithm to extract and abstract arithmetic components
  - Arithmetic sweeping based on input and output boundaries of the abstracted components
  - Efficient polynomial representation (negative-Davio decomposition)
- Checking equivalence of large floating-point multipliers which cannot be verified by state-of-art equivalence checkers
- Verifying heavy optimized circuits and dealing with non-equivalent circuits are still major challenges
Conclusion

- New algebraic equivalence checking technique for circuits that combine data-path and control logic
  - New reverse engineering algorithm to extract and abstract arithmetic components
  - Arithmetic sweeping based on input and output boundaries of the abstracted components
  - Efficient polynomial representation (negative-Davio decomposition)
- Checking equivalence of large floating-point multipliers which cannot be verified by state-of-art equivalence checkers
- Verifying heavy optimized circuits and dealing with non-equivalent circuits are still major challenges
Conclusion

- New algebraic equivalence checking technique for circuits that combine data-path and control logic
  - New reverse engineering algorithm to extract and abstract arithmetic components
  - Arithmetic sweeping based on input and output boundaries of the abstracted components
  - Efficient polynomial representation (negative-Davio decomposition)

- Checking equivalence of large floating-point multipliers which cannot be verified by state-of-art equivalence checkers

- Verifying heavy optimized circuits and dealing with non-equivalent circuits are still major challenges
Conclusion

- New algebraic equivalence checking technique for circuits that combine data-path and control logic
  - New reverse engineering algorithm to extract and abstract arithmetic components
  - Arithmetic sweeping based on input and output boundaries of the abstracted components
  - Efficient polynomial representation (negative-Davio decomposition)
- Checking equivalence of large floating-point multipliers which cannot be verified by state-of-art equivalence checkers
- Verifying heavy optimized circuits and dealing with non-equivalent circuits are still major challenges
Equivalence Checking using Gröbner Bases

Amr Sayed-Ahmed\textsuperscript{1} \quad Daniel Große\textsuperscript{1,2}
Mathias Soeken\textsuperscript{3} \quad Rolf Drechsler\textsuperscript{1,2}

\textsuperscript{1}University of Bremen, Germany
\textsuperscript{2}DFKI GmbH, Germany
\textsuperscript{3}EPFL, Switzerland

Email: asahmed@informatik.uni-bremen.de

FMCAD, October 2016