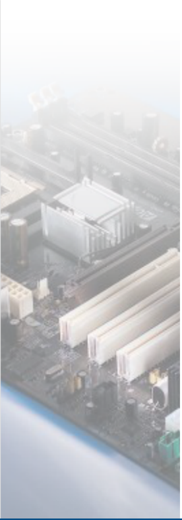


# Equivalence Checking using Gröbner Bases


Amr Sayed-Ahmed[1]  Daniel Große[1,2]
Mathias Soeken[3]  Rolf Drechsler[1,2]

[1]University of Bremen, Germany
[2]DFKI GmbH, Germany
[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)

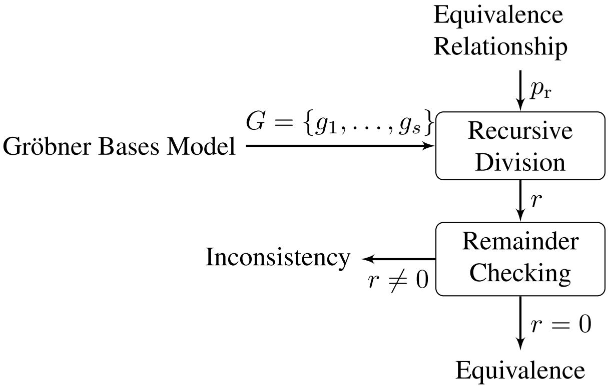
- Reverse Engineering
- Arithmetic Sweeping

Experimental Results

Conclusion

# Algebraic Decision Procedure

- Ideal Membership Testing:



# Modeling a Circuit as Gröbner Bases

- Modeling Logic Gates:

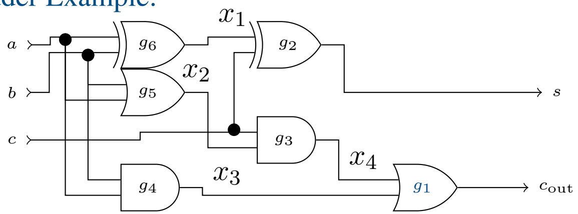
$$z = \neg a \Rightarrow g := -z + 1 - a \qquad z = a \oplus b \Rightarrow g := -z + a + b - 2ab$$
$$z = a \wedge b \Rightarrow g := -z + ab \qquad z = a \vee b \Rightarrow g := -z + a + b - ab$$

# Modeling a Circuit as Gröbner Bases

- Modeling Logic Gates:

$$z = \neg a \Rightarrow g := -z + 1 - a \qquad z = a \oplus b \Rightarrow g := -z + a + b - 2ab$$
$$z = a \wedge b \Rightarrow g := -z + ab \qquad z = a \vee b \Rightarrow g := -z + a + b - ab$$

- Full Adder Example:




leading monomial ↘ tail terms ↙

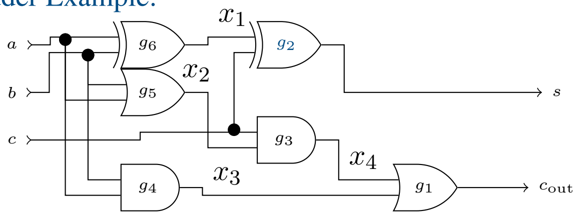
$$g_1 := -c_{out} \boxed{-x_4x_3 + x_4 + x_3}$$

# Modeling a Circuit as Gröbner Bases

- Modeling Logic Gates:

$$z = \neg a \Rightarrow g := -z + 1 - a \qquad z = a \oplus b \Rightarrow g := -z + a + b - 2ab$$
$$z = a \wedge b \Rightarrow g := -z + ab \qquad z = a \vee b \Rightarrow g := -z + a + b - ab$$

- Full Adder Example:




leading monomial ↘ tail terms ↙

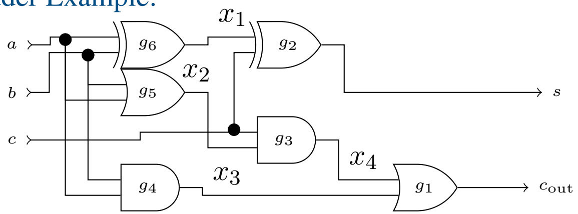
$$g_1 := -c_{\mathrm{out}} \boxed{-x_4x_3 + x_4 + x_3} \qquad g_2 := -s - 2x_1c + x_1 + c$$

# Modeling a Circuit as Gröbner Bases

- Modeling Logic Gates:

$$z = \neg a \Rightarrow g := -z + 1 - a \qquad z = a \oplus b \Rightarrow g := -z + a + b - 2ab$$
$$z = a \wedge b \Rightarrow g := -z + ab \qquad z = a \vee b \Rightarrow g := -z + a + b - ab$$

- Full Adder Example:




leading monomial　　tail terms

$$g_1 := \underbrace{-c_{\text{out}}}_{\text{leading monomial}} \boxed{-x_4x_3 + x_4 + x_3} \qquad g_2 := -s - 2x_1c + x_1 + c$$
$$g_3 := -x_4 + x_2c \qquad g_4 := -x_3 + ab$$
$$g_5 := -x_2 - ab + a + b \qquad g_6 := -x_1 - 2ab + a + b$$

# Modeling a Circuit as Gröbner Bases

- Modeling Logic Gates:

$$z = \neg a \Rightarrow g := -z + 1 - a \qquad z = a \oplus b \Rightarrow g := -z + a + b - 2ab$$
$$z = a \wedge b \Rightarrow g := -z + ab \qquad z = a \vee b \Rightarrow g := -z + a + b - ab$$

- Full Adder Example:

leading monomial ↘ ↙ tail terms

$$g_1 := -c_{\text{out}} \boxed{-x_4x_3 + x_4 + x_3} \qquad g_2 := -s - 2x_1c + x_1 + c$$
$$g_3 := -x_4 + x_2c \qquad g_4 := -x_3 + ab$$
$$g_5 := -x_2 - ab + a + b \qquad g_6 := -x_1 - 2ab + a + b$$

- Leading monomials are relatively prime $\Longrightarrow$ The model is Gröbner bases

# Ideal Membership Testing

- Following Full Adder Example: specification polynomial $p_{\mathrm{r}} := -2c_{\mathrm{cout}} - s + c + b + a$
- Its model

$$\begin{aligned} g_1 &:= -c_{\mathrm{out}} \boxed{-x_4x_3 + x_4 + x_3} & g_2 &:= -s - 2x_1c + x_1 + c \\ g_3 &:= -x_4 + x_2c & g_4 &:= -x_3 + ab \\ g_5 &:= -x_2 - ab + a + b & g_6 &:= -x_1 - 2ab + a + b \end{aligned}$$

- *Recursive Division*:

# Ideal Membership Testing

- Following Full Adder Example: specification polynomial
  $p_r := -2c_{cout} - s + c + b + a$
- Its model

$$g_1 := -c_{out} \boxed{-x_4x_3 + x_4 + x_3} \quad g_2 := -s - 2x_1c + x_1 + c$$
$$g_3 := -x_4 + x_2c \quad g_4 := -x_3 + ab$$
$$g_5 := -x_2 - ab + a + b \quad g_6 := -x_1 - 2ab + a + b$$

- *Recursive Division*:
  $p_r := -2c_{cout} - s + c + b + a \xrightarrow{g_1}$
  $-s \boxed{+2x_4x_3 - 2x_4 - 2x_3} + c + b + a \xrightarrow{g_2}$

# Ideal Membership Testing

- Following Full Adder Example: specification polynomial $p_\mathrm{r} := -2c_\mathrm{cout} - s + c + b + a$
- Its model

$$
\begin{aligned}
g_1 &:= -c_\mathrm{out} \boxed{-x_4x_3 + x_4 + x_3} & g_2 &:= -s - 2x_1c + x_1 + c \\
g_3 &:= -x_4 + x_2c & g_4 &:= -x_3 + ab \\
g_5 &:= -x_2 - ab + a + b & g_6 &:= -x_1 - 2ab + a + b
\end{aligned}
$$

- *Recursive Division*:
$\xrightarrow{g_2} 2x_4x_3 - 2x_4 - 2x_3 + 2x_1c - x_1 + b + a \xrightarrow{g_3}$

# Ideal Membership Testing

- Following Full Adder Example: specification polynomial
  $p_{\mathrm{r}} := -2c_{\mathrm{cout}} - s + c + b + a$
- Its model

$$
\begin{aligned}
g_1 &:= -c_{\mathrm{out}} \boxed{-x_4x_3 + x_4 + x_3} & g_2 &:= -s - 2x_1c + x_1 + c \\
g_3 &:= -x_4 + x_2c & g_4 &:= -x_3 + ab \\
g_5 &:= -x_2 - ab + a + b & g_6 &:= -x_1 - 2ab + a + b
\end{aligned}
$$

- *Recursive Division*:
  $\xrightarrow{g_3} 2x_3x_2c - 2x_3 - 2x_2c + 2x_1c - x_1 + b + a \xrightarrow{g_4}$

# Ideal Membership Testing

- Following Full Adder Example: specification polynomial
  $p_r := -2c_{\text{cout}} - s + c + b + a$
- Its model

$$
\begin{aligned}
g_1 &:= -c_{\text{out}} \boxed{-x_4x_3 + x_4 + x_3} & g_2 &:= -s - 2x_1c + x_1 + c \\
g_3 &:= -x_4 + x_2c & g_4 &:= -x_3 + ab \\
g_5 &:= -x_2 - ab + a + b & g_6 &:= -x_1 - 2ab + a + b
\end{aligned}
$$

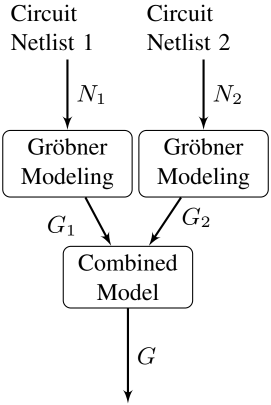
- *Recursive Division*:

$$
\xrightarrow{g_4} 2x_2cba - 2x_2c + 2x_1c - x_1 - 2ba + b + a
$$

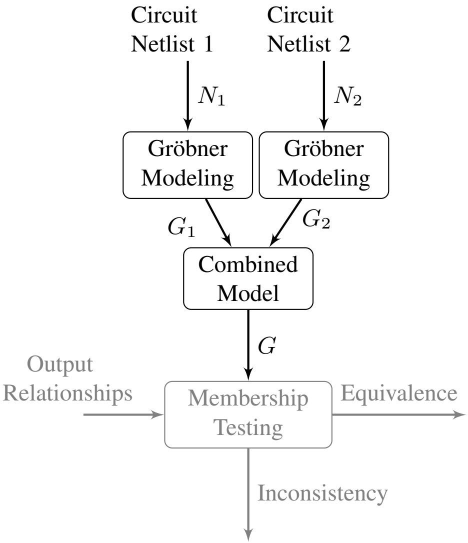
$$
\xrightarrow{g_5} 2x_1c - x_1 + 4cba - 2ca - 2cb - 2ab + b + a \xrightarrow{g_6} 0
$$

# Outline

# Flow of ACEC

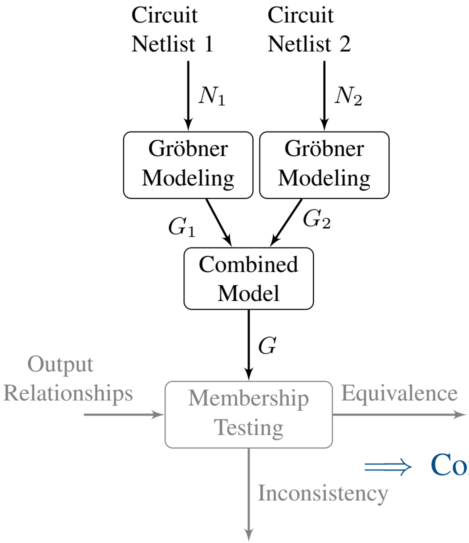

Circuit
Netlist 1
Circuit
Netlist 2
$N_1$
$N_2$
Gröbner
Modeling
Gröbner
Modeling
$G_1$
$G_2$
Combined
Model
$G$

# Flow of ACEC



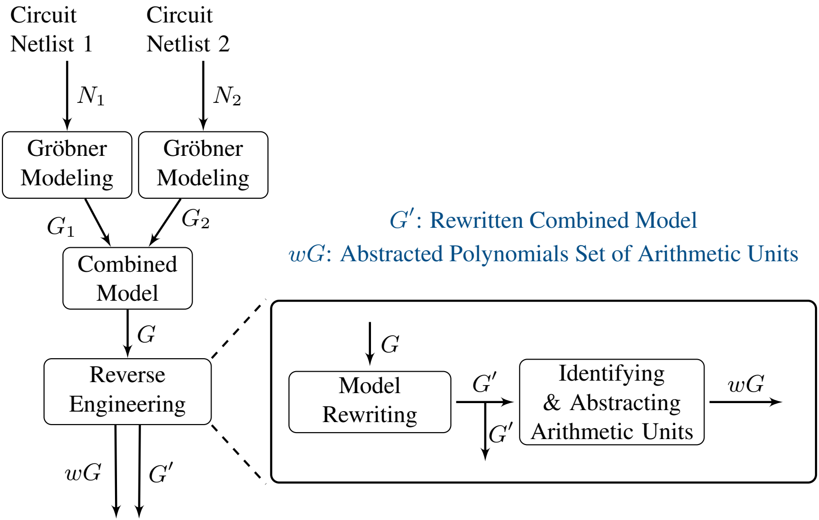
Circuit Netlist 1
Circuit Netlist 2
$N_1$
$N_2$
Gröbner Modeling
Gröbner Modeling
$G_1$
$G_2$
Combined Model
$G$
Output Relationships
Membership Testing
Equivalence
Inconsistency

# Flow of ACEC




$\Longrightarrow$ Computationally Infeasible

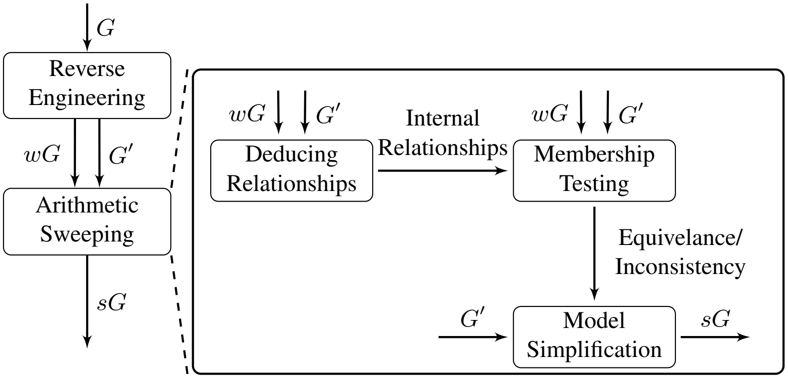
# Flow of ACEC




$G'$: Rewritten Combined Model

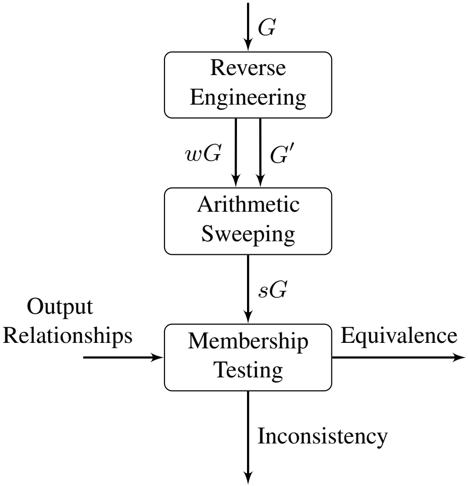
$wG$: Abstracted Polynomials Set of Arithmetic Units

# Flow of ACEC



$G$
Reverse Engineering
$wG$
$G'$
Arithmetic Sweeping
$sG$
$wG$
$G'$
Deducing Relationships
Internal Relationships
$wG$
$G'$
Membership Testing
Equivelance/ Inconsistency
$G'$
Model Simplification
$sG$

# Flow of ACEC


$G$
Reverse Engineering
$wG$
$G'$
Arithmetic Sweeping
$sG$
Output Relationships
Membership Testing
Equivalence
Inconsistency

# 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:
  $g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba$
  $g_2 : -c_{\text{out}} - 2cba + cb + ca + ba$
- 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:
  $g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba$
  $g_2 : -c_{\text{out}} - 2cba + cb + ca + ba$
- 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:
  $g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba$
  $g_2 : -c_{\text{out}} - 2cba + cb + ca + ba$
- 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:
  $g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba$
  $g_2 : -c_{\text{out}} - 2cba + cb + ca + ba$
- 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:
  $g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba$
  $g_2 : -c_{\text{out}} - 2cba + cb + ca + ba$
- 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:
  $g_1 : -s + c + b + a + 4cba - 2cb - 2ca - 2ba$
  $g_2 : -c_{\mathrm{out}} - 2cba + cb + ca + ba$
- 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:

$$2g_2 + g_1 \rightarrow g_{\mathrm{r}} : -2c_{\mathrm{out}} - s + c + b + a$$

# Reverse Engineering: 1) Model Rewriting

- *XOR rewriting* preserves inputs and outputs of chains of XOR gates
- Parallel Adder Model:
  $c_2 = D_2 \vee (X_2 \wedge D_1) \vee (X_2 \wedge X_1 \wedge D_0) \implies g_1 := -c_2 + X_2X_1a_2b_2a_1b_1a_0b_0 - X_2X_1a_1b_1a_0b_0 - X_2X_1a_2b_2a_0b_0 - X_2a_2b_2a_1b_1 + X_2X_1a_0b_0 + X_2a_1b_1 + a_2b_2$

$$
\begin{aligned}
s_2 &= X_2 \oplus c_1 && \implies g_2 := -s_2 - 2c_1X_2 + c1 + X_2 \\
c_1 &= D_1 \vee (X_1 \wedge D_0) && \implies g_3 := -c_1 - X_1a_1b_1a_0b_0 + X_1a_0b_0 + a_1b_1 \\
s_1 &= X_1 \oplus c_0 && \implies g_4 := -s_1 - 2c_0X_1 + c_0 + X_1 \\
c_0 &= D_0 && \implies g_5 := -c_0 + a_0b_0 \\
s_0 &= X_0 && \implies g_6 := -s_0 + X_0 \\
X_i &= a_i \oplus b_i && \implies g_{k-i-1} := -X_i - 2a_ib_i + b_i + a_i \\
D_i &= a_i \wedge b_i && \implies g_{k-i} := -D_i + a_ib_i
\end{aligned}
$$

# Reverse Engineering: 1)Model Rewriting

- *Common rewriting* preserves shared variables between polynomials
- Parallel adder model after XOR rewriting:
  $g_1 := -c_2 + X_2X_1a_0b_0 + X_2a_1b_1 + a_2b_2$

$$
\begin{aligned}
g_2 &:= -s_2 - 2c_1X_2 + c1 + X_2 \\
g_3 &:= -c_1 + X_1a_0b_0 + a_1b_1 \\
g_4 &:= -s_1 - 2c_0X_1 + c_0 + X_1 \\
g_5 &:= -c_0 + a_0b_0 \\
g_6 &:= -s_0 + X_0 \\
g_7 &:= -X_0 - 2a_0b_0 + b_0 + a_0 \\
g_8 &:= -X_1 - 2a_1b_1 + b_1 + a_1 \\
g_9 &:= -X_2 - 2a_2b_2 + b_2 + a_2
\end{aligned}
$$

- $X_0$, $c_0$ and $c_1$ will be eliminated

# Reverse Engineering: 1)Model Rewriting

- *Common rewriting* preserves shared variables between polynomials
- Parallel adder model after XOR rewriting:

$g_1 := -c_2 + X_2X_1a_0b_0 + X_2a_1b_1 + a_2b_2$

$$
\begin{aligned}
g_2 &:= -s_2 - 2c_1X_2 + c1 + X_2 \\
g_3 &:= -c_1 + X_1a_0b_0 + a_1b_1 \\
g_4 &:= -s_1 - 2c_0X_1 + c_0 + X_1 \\
g_5 &:= -c_0 + a_0b_0 \\
g_6 &:= -s_0 + X_0 \\
g_7 &:= -X_0 - 2a_0b_0 + b_0 + a_0 \\
g_8 &:= -X_1 - 2a_1b_1 + b_1 + a_1 \\
g_9 &:= -X_2 - 2a_2b_2 + b_2 + a_2
\end{aligned}
$$

- $X_0$, $c_0$ and $c_1$ will be eliminated

# Reverse Engineering: 1)Model Rewriting

- *Common rewriting* preserves shared variables between polynomials
- Parallel adder model after XOR rewriting:
  $g_1 := -c_2 + X_2X_1a_0b_0 + X_2a_1b_1 + a_2b_2$
  $$\begin{aligned} g_2 &:= -s_2 - 2c_1X_2 + c1 + X_2 \\ g_3 &:= -c_1 + X_1a_0b_0 + a_1b_1 \\ g_4 &:= -s_1 - 2c_0X_1 + c_0 + X_1 \\ g_5 &:= -c_0 + a_0b_0 \\ g_6 &:= -s_0 + X_0 \\ g_7 &:= -X_0 - 2a_0b_0 + b_0 + a_0 \\ g_8 &:= -X_1 - 2a_1b_1 + b_1 + a_1 \\ g_9 &:= -X_2 - 2a_2b_2 + b_2 + a_2 \end{aligned}$$
- $X_0$, $c_0$ 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*:

$$2g_1 + g_2 \to g_{\text{res}} := -2c_2 \boxed{+2X_2 X_1 a_0 b_0 + 2X_2 a_1 b_1} + 2a_2 b_2 - s_2 \boxed{-2X_2 X_1 a_0 b_0 - 2X_2 a_1 b_1} + X_2 + X_1 a_0 b_0 + a_1 b_1$$

# Reverse Engineering: 2) Extracting Arithmetic Units

- Parallel adder model after common rewriting:

$$g_1 := -c_2 + X_2X_1a_0b_0 + X_2a_1b_1 + a_2b_2$$

$$g_2 := -s_2 - 2X_2X_1a_0b_0 - 2X_2a_1b_1 + X_2 + X_1a_0b_0 + a_1b_1$$
$$g_4 := -s_1 - 2X_1a_0b_0 + a_0b_0 + X_1$$
$$g_6 := -s_0 + -2a_0b_0 + b_0 + a_0$$
$$g_8 := -X_1 - 2a_1b_1 + b_1 + a_1$$
$$g_9 := -X_2 - 2a_2b_2 + b_2 + a_2$$

- Abstraction by *Gaussian elimination*:

$$2g_1 + g_2 \to g_{\text{res}} := -2c_2 \boxed{\cancel{+2X_2X_1a_0b_0 + 2X_2a_1b_1}} + 2a_2b_2 - s_2 \boxed{\cancel{-2X_2X_1a_0b_0 - 2X_2a_1b_1}} + X_2 + X_1a_0b_0 + a_1b_1$$

# 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*:

$$g_{\text{res}} := -2c_2 - s_2 + X_2 + X_1a_0b_0 + 2a_2b_2 + a_1b_1$$
$$2g_{\text{res}} + g_4 \rightarrow g_{\text{res}} := -4c_2 - 2s_2 - s_1 + 2X_2 + X_1 + 4a_2b_2 + 2a_1b_1 + a_0b_0$$

# 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*:

  $g_{\text{res}} := -4c_2 - 2s_2 - s_1 + 2X_2 + X_1 + 4a_2b_2 + 2a_1b_1 + a_0b_0$

  $2g_{\text{res}} + g_6 \rightarrow g_{\text{res}} :=$

  $-8c_2 - 4s_2 - 2s_1 - s_0 + 4X_2 + 2X_1 + 8a_2b_2 + 4a_1b_1 + b_0 + a_0$

# 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:

  $g_1 := -c_2 + X_2X_1a_0b_0 + X_2a_1b_1 + a_2b_2$

  $g_2 := -s_2 - 2X_2X_1a_0b_0 - 2X_2a_1b_1 + X_2 + X_1a_0b_0 + a_1b_1$

  $g_4 := -s_1 - 2X_1a_0b_0 + a_0b_0 + X_1$

  $g_6 := -s_0 + -2a_0b_0 + b_0 + a_0$

  $g_8 := -X_1 - 2a_1b_1 + b_1 + a_1$

  $g_9 := -X_2 - 2a_2b_2 + b_2 + a_2$

- Abstraction by *Gaussian elimination*:

  $g_{\text{res}} := -8c_2 - 4s_2 - 2s_1 - s_0 + 4X_2 + 8a_2b_2 + 2b_1 + 2a_1 + b_0 + a_0$

  $g_{\text{res}} + 4g_9 \rightarrow$

  $\boxed{g_{\text{res}} := -8c_2 - 4s_2 - 2s_1 - s_0 + 4b_2 + 4a_2 + 2b_1 + 2a_1 + b_0 + a_0}$

# Outline

Symbolic Computation

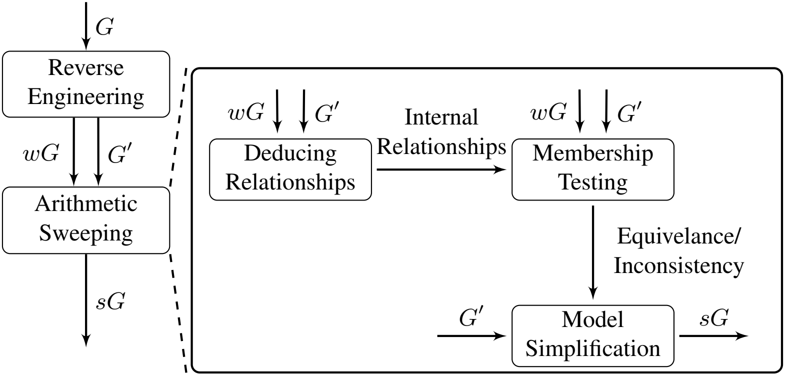
Algebraic Combinational Equivalence Checking (ACEC)

- Reverse Engineering
- Arithmetic Sweeping

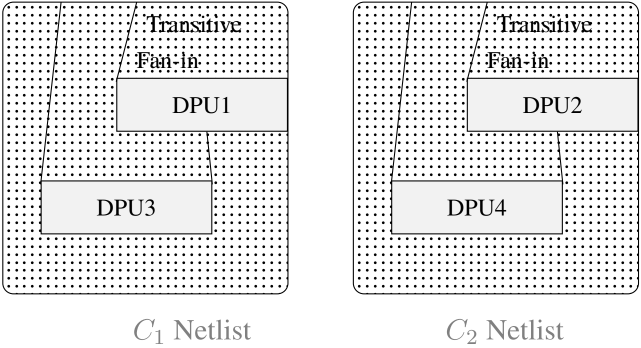
Experimental Results

Conclusion

# Arithmetic Sweeping

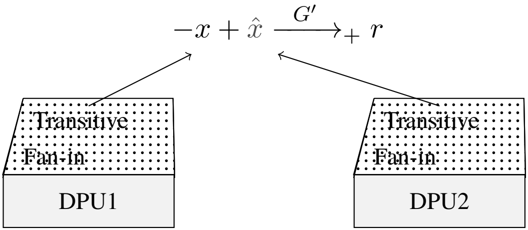


# Deducing Relationships




- Partitioning the combined model based on the extracted arithmetic information

# Deducing and Testing Relationships




- Deducing and testing bit relationships between variables of the transitive fan-in of arithmetic units

# Deducing and Testing Relationships

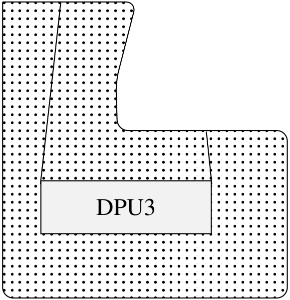
DPU1

DPU2

$$-2^{n-1}s_{n-1} - \cdots - s_0 + 2^{n-1}\hat{s}_{n-1} + \cdots + \hat{s}_0 \xrightarrow{wG}_{+} r$$

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

# Model Simplification

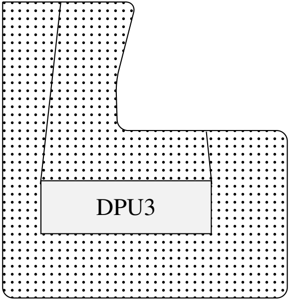



$C_1$ Netlist

$C_2$ Netlist

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

# Model Simplification




$C_1$ Netlist

$C_2$ Netlist

- 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

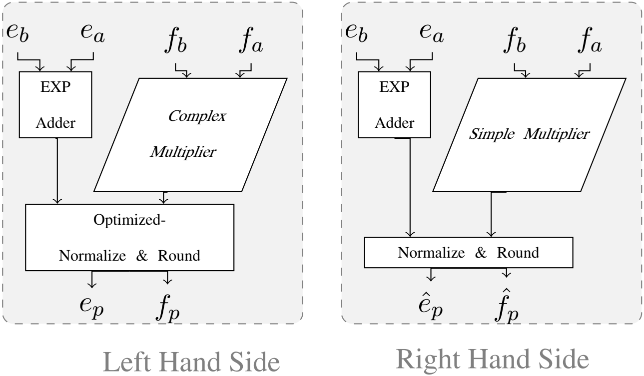



Figure: Compared FP Multiplier Circuits

# Experimental Results

| Multiplier Architecture | FP operand # bits | Commercial (h:m:s) | ABC (h:m:s) | ACEC (h:m:s) |
|---|---|---|---|---|
| SP-CT-BK | 16 | 00:08:50 | TO | 00:01:42 |
| SP-WT-CH | 16 | 00:09:08 | TO | 00:01:44 |
| SP-CT-BK | 24 | TO | TO | 00:17:49 |
| SP-WT-CH | 24 | TO | TO | 00:25:58 |
| SP-CT-BK | 32 | TO | TO | 02:24:01 |
| SP-WT-CH | 32 | TO | TO | 03:41:43 |

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

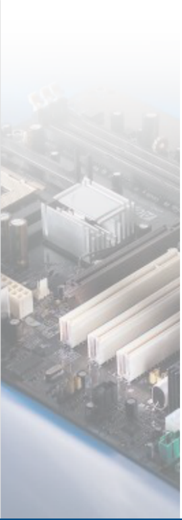
- New algebraic equivalence checking technique for circuits that combine data-path and control logic
  - New reverse engineering algorithm to extract and abstract arithmetic components
  - Arithmetic sweeping based on input and output boundaries of the abstracted components
  - Efficient polynomial representation (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[1] Daniel Große[1,2]
Mathias Soeken[3] Rolf Drechsler[1,2]

[1]University of Bremen, Germany
[2]DFKI GmbH, Germany
[3]EPFL, Switzerland

Email: asahmed@informatik.uni-bremen.de

FMCAD, October 2016