Inference of

Numerical Relations from

Digital Circuits

Enric Rodríguez-Carbonell   Jordi Cortadella

Universitat Politècnica de Catalunya
Barcelona

Overview of the Talk

1. Introduction
2. Overview of the Method
3. Simple Example: Binary Addition
4. Abstract Domain
5. Inductive Method
6. Working with Small Coefficients
7. Future Work

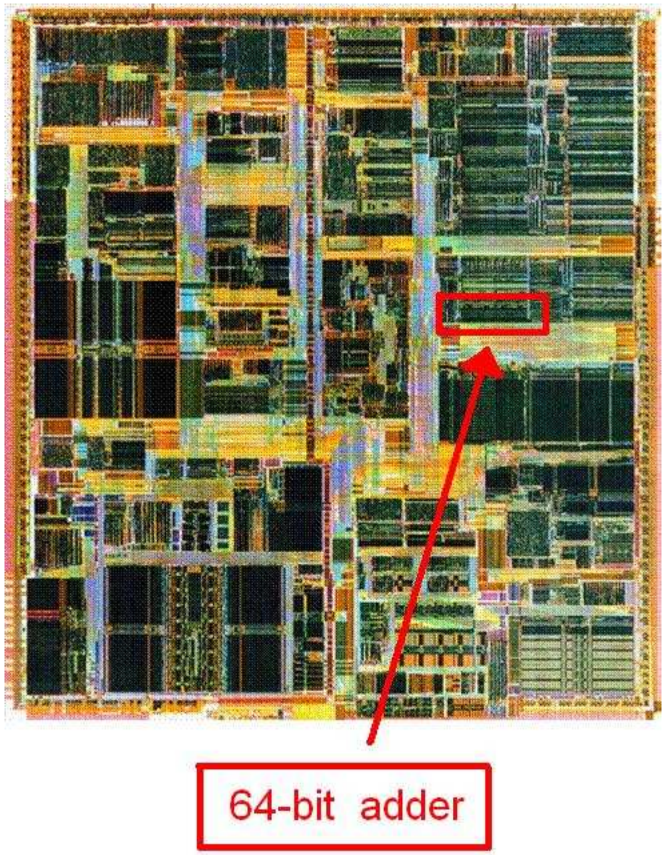
Introduction
Need for Hardware Verification

Errors in hardware are:

- *very* costly:
  - Pentium division bug cost Intel **0.5 billion $**
  - Wide Field Infrared Explorer (WIRE) spacecraft from NASA failed soon after launch
- irreversible: no patches possible once product is on market

Need for Hardware Verification to Increase Reliability !

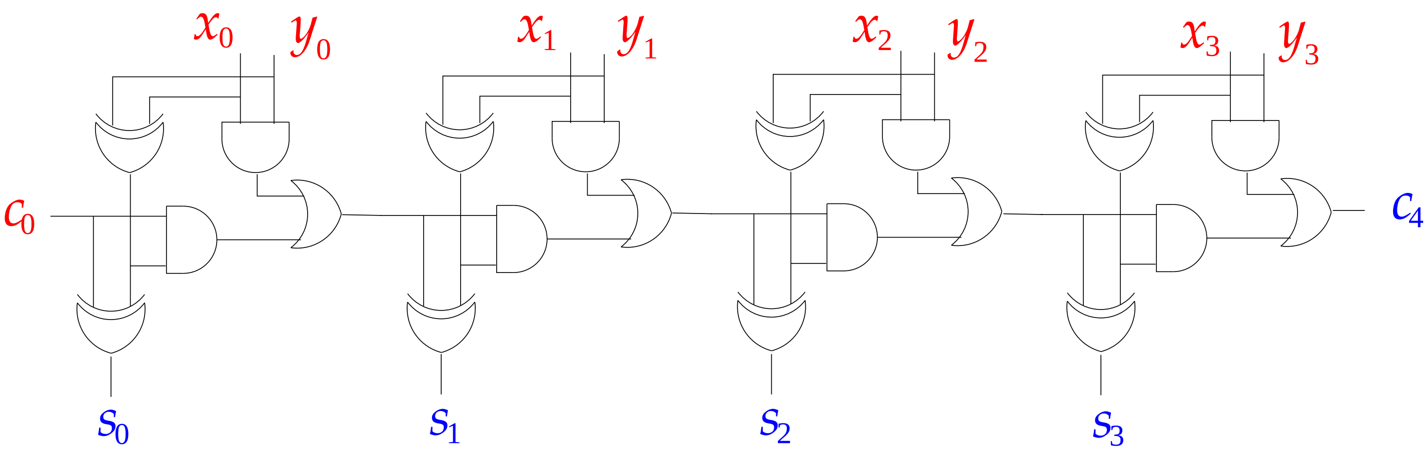
Introduction
Verifying Hardware



- When verifying hardware we have:
  - Gate list
  - High-level specification
- **PROBLEM:** Huge gap !
- **SOLUTION:** Abstraction
  Reverse engineering discovers properties hidden in circuits

Introduction

Abstracting Circuits



$\bar{x}$, $\bar{y}$, $\bar{s}$: 4-bit integers

$$\bar{s} + 16c_4 = c_0 + \bar{x} + \bar{y}$$

Introduction
Arithmetic Circuits are Difficult

- Arithmetic circuits are difficult to verify
- BDD's representing multipliers have huge size
- Current techniques cannot handle real-sized multipliers
- Arithmetics has not been sufficiently exploited

$\Longrightarrow$ **Combination logics/arithmetics**

Overview of the Talk

1. Introduction
2. **Overview of the Method**
3. Simple Example: Binary Addition
4. Abstract Domain
5. Inductive Method
6. Working with Small Coefficients
7. Future Work

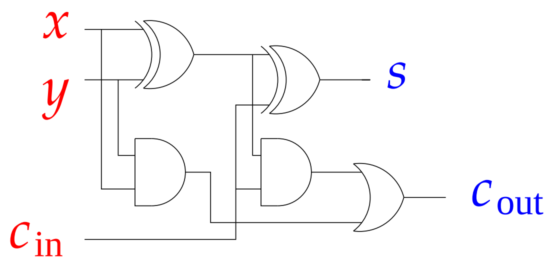
Overview of the Method

- **GOAL:** extract numerical relations from arithmetic circuits
- **APPLICATION:** preprocessing step to alleviate formal verification with other methods
- Boolean values abstracted to integers
- Boolean functions abstracted to polynomials
- Gaussian elimination used to infer numerical relations

Overview of the Talk

1. **Introduction**
2. **Overview of the Method**
3. **Simple Example: Binary Addition**
4. **Abstract Domain**
5. **Inductive Method**
6. **Working with Small Coefficients**
7. **Future Work**

Simple Example: Binary Addition
Full Adder



- Full adder: sum of two bits with carry in and carry out
- Input signals: x, y, c_{in}
- Output signals: s, c_{out}
- **GOAL**: generate the equation

$$s + 2c_{\text{out}} = x + y + c_{\text{in}}$$

Simple Example: Binary Addition

From Boolean Functions to Polynomials

$$
\begin{aligned}
x \text{ AND } y &= xy \\
x \text{ XOR } y &= x + y - 2xy \\
x \text{ OR } y &= x + y - xy \\
\text{NOT } x &= 1 - x
\end{aligned}
$$

$$x \in \{0, 1\} \implies x^2 = x$$

$$
\begin{aligned}
s &= x \text{ XOR } y \text{ XOR } c_{\text{in}} \\
c_{\text{out}} &= (x \text{ AND } y) \text{ OR } (x \text{ AND } c_{\text{in}}) \text{ OR } (y \text{ AND } c_{\text{in}})
\end{aligned}
$$

$$
\begin{aligned}
s &= x + y - 2xy + c_{\text{in}} - 2c_{\text{in}}x - 2c_{\text{in}}y + 4c_{\text{in}}xy \\
c_{\text{out}} &= xy + c_{\text{in}}x + c_{\text{in}}y - c_{\text{in}}^2 xy - x^2 y c_{\text{in}} - xy^2 c_{\text{in}} + x^2 y^2 c_{\text{in}}^2
\end{aligned}
$$

$$
\begin{aligned}
s &= x + y - 2xy + c_{\text{in}} - 2c_{\text{in}}x - 2c_{\text{in}}y + 4c_{\text{in}}xy \\
c_{\text{out}} &= xy + c_{\text{in}}x + c_{\text{in}}y - 2c_{\text{in}}xy
\end{aligned}
$$

Simple Example: Binary Addition
Applying Gaussian Elimination

- Non-linear terms are considered as new variables
- Variables eliminated using Gaussian elimination

$$
\begin{aligned}
s &= x + y + c_{\mathsf{in}} - 2xy - 2c_{\mathsf{in}}x - 2c_{\mathsf{in}}y + 4c_{\mathsf{in}}xy \\
c_{\mathsf{out}} &= xy + c_{\mathsf{in}}x + c_{\mathsf{in}}y - 2c_{\mathsf{in}}xy \\
&\Downarrow \\
s + 2c_{\mathsf{out}} &= x + y + c_{\mathsf{in}}
\end{aligned}
$$

- Sometimes the aimed equation has non-linear terms:
  for carry look-ahead,

$$2^n \cdot (G + Pc_{\mathsf{in}}) + \sum_{i=0}^{n-1} 2^i s_i = c_{\mathsf{in}} + \sum_{i=0}^{n-1} 2^i (x_i + y_i)$$

$\longrightarrow$ Heuristics to select the terms to be eliminated

Overview of the Talk

1. **Introduction**
2. **Overview of the Method**
3. **Simple Example: Binary Addition**
4. **Abstract Domain**
5. **Inductive Method**
6. **Working with Small Coefficients**
7. **Future Work**

Abstract Domain

- ABSTRACT VALUES:
  vector spaces of polynomials with coefficients in $\mathbb{Q}$

- ABSTRACTION FUNCTION α

$$\begin{array}{rcl} \alpha : \mathcal{P}(\{0,1\}^n) & \longrightarrow & \{\text{vector spaces in } \mathbb{Q}[x_1,\dots,x_n]\} \\ B & \longmapsto & \{\textbf{vector space of} \\ & & \textbf{polynomials evaluating to 0 on } B\} \end{array}$$

- CONCRETIZATION FUNCTION γ

$$\begin{array}{rcl} \gamma : \{\text{vector spaces in } \mathbb{Q}[x_1,\dots,x_n]\} & \longrightarrow & \mathcal{P}(\{0,1\}^n) \\ V & \longmapsto & \{\textbf{zeros of } V \textbf{ in } \{0,1\}^n\} \end{array}$$

Abstract Domain

Equations of output variables as	$\longrightarrow$	Polynomial
boolean functions of input variables	$\longrightarrow$	equations

- Not all consequences of equations are linear combinations
  - **Linear algebra not complete !!**
  - **Ideals of polynomials** (Gröbner bases) **bad complexity**

- **INTERMEDIATE SOLUTION:**
  - approximate ideal generated by equations
  - add new equations by multiplying by monomials, using $x_i^2 = x_i$

$\longrightarrow$ Heuristics to select new equations to add

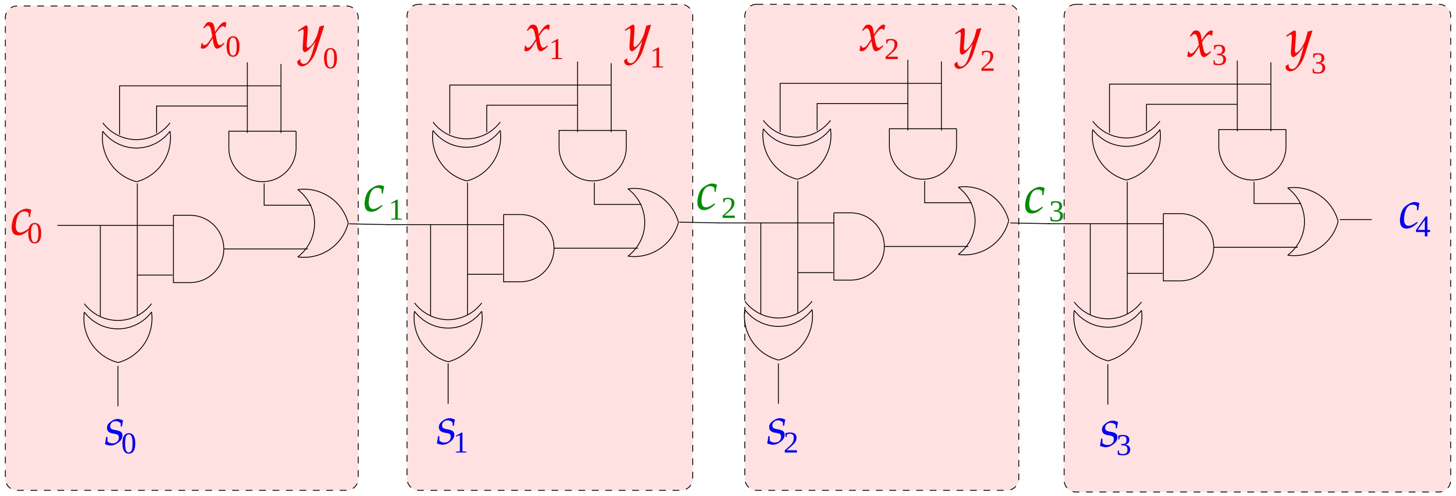
Overview of the Talk

1. Introduction
2. Overview of the Method
3. Simple Example: Binary Addition
4. Abstract Domain
5. **Inductive Method**
6. Working with Small Coefficients
7. Future Work

Inductive Method

- **PROBLEM:** Not feasible for big number of variables
- **SOLUTION:**
  - Decompose circuit into black-boxes inductively
  - Behaviour of black boxes described by polynomials
  - Bigger black boxes built from smaller black boxes
  - *Local* signals (neither *input* nor *output*) eliminated by Gaussian elimination

Inductive Method
Example: 4-bit Carry-Ripple Adder



$$
\begin{aligned}
s_0 + 2c_1 &= c_0 + x_0 + y_0 \\
s_1 + 2c_2 &= c_1 + x_1 + y_1 \\
s_2 + 2c_3 &= c_2 + x_2 + y_2 \\
s_3 + 2c_4 &= c_3 + x_3 + y_3
\end{aligned}
$$

$$s_0 + 2s_1 + 4s_2 + 8s_3 + 16c_4 = c_0 + x_0 + 2x_1 + 4x_2 + 8x_3 + y_0 + 2y_1 + 4y_2 + 8y_3$$

$$\bar{s} + 16c_4 = c_0 + \bar{x} + \bar{y}$$

Overview of the Talk

1. **Introduction**
2. **Overview of the Method**
3. **Simple Example: Binary Addition**
4. **Abstract Domain**
5. **Inductive Method**
6. **Working with Small Coefficients**
7. **Future Work**

Working with Small Coefficients

- Coefficients in numerical relations we are interested are $\pm 2^i$
- Coefficients may be **very** large in computations
  - Exact arithmetic is slow
  - Risk of overflow
- **Use finite fields for the coefficients !**
- **Advantages:**
  - Coefficients can be represented with few bits
  - Arithmetics can be tabulated at compile-time
- **Disadvantages:**
  - Not sound
  - ... but results can be later checked

Working with Small Coefficients

- Let p be an odd prime number such that 2 generates $\mathbb{Z}_p^*$
- There are **many** such prime numbers
- Let $q = (p-3)/2$. Then:

$$\begin{aligned} \mathbb{Z}_p^* \;=\; & \{-2^q, -2^{q-1}, ..., -2^2, -2, -1, 1, \\ & 1, 2, 2^2, ..., 2^q\} \end{aligned}$$

- Heuristic approach:
  1. Work with polynomials with coefficients in the finite field
  2. Once result computed, translate back into coefficients as powers of 2

Working with Small Coefficients

$\pm 2^i$	Z_{19}^*
-256	10
-128	5
-64	12
-32	6
-16	3
-8	11
-4	15
- 2	17
-1	18
1	1
2	2
4	4
8	8
16	16
32	13
64	7
128	14
256	9

8-BIT ADDER

$$s_0 + 2s_1 + 4s_2 + 8s_3 + 16s_4 + 13s_5 + 7s_6 + 14s_7 + 10c_4 = c_0 + x_0 + 2x_1 + 4x_2 + 8x_3 + 16x_4 + 13x_5 + 7x_6 + 14x_7 + y_0 + 2y_1 + 4y_2 + 8y_3 + 16y_4 + 13y_5 + 7y_6 + 14y_7$$

$$\Downarrow$$

$$s_0 + 2s_1 + 4s_2 + 8s_3 + 16s_4 + 32s_5 + 64s_6 + 128s_7 + 256c_4 = c_0 + x_0 + 2x_1 + 4x_2 + 8x_3 + 16x_4 + 32x_5 + 64x_6 + 128x_7 + y_0 + 2y_1 + 4y_2 + 8y_3 + 16y_4 + 32y_5 + 64y_6 + 128y_7$$

Overview of the Talk

1. Introduction
2. Overview of the Method
3. Simple Example: Binary Addition
4. Abstract Domain
5. Inductive Method
6. Working with Small Coefficients
7. Future Work

Future Work

- Heuristics for eliminating terms in Gaussian elimination
- Heuristics for adding new equations
- Implementation in progress
- Regularity-based techniques for partitioning circuits
- Application to adders and multipliers
- Integration to a verification system