Method for Multiplier Verification Employing Boolean Equivalence Checking and Arithmetic Bit Level Description

U. Krautz[1], M. Wedler[1], W. Kunz[1] &
K. Weber[2], C. Jacobi[2], M. Pflanz[2]

[1]University of Kaiserslautern - Germany
[2]IBM Forschung und Entwicklungs GmbH - Germany

13th Asia and South Pacific Design Automation Conference, 2008

Outline

Outline

General Arithmetic Circuit Verification

- formal property checking of significant importance
- arithmetic circuits still "show stoppers"
- no universal framework for arithmetics, instead special "engineered" solutions
- useful for highly regular designs
- limited in case of full custom logic designs
- multipliers particular hard to verify
  - hardware multipliers common in processors
  - hard to generate compact canonical representation from bit level
  - for verification often equivalence check against reference (reference and design have to share large structural similarities)

General Arithmetic Circuit Verification

- formal property checking of significant importance
- arithmetic circuits still “show stoppers”
- no universal framework for arithmetics, instead special “engineered” solutions
- useful for highly regular designs
- limited in case of full custom logic designs
- multipliers particular hard to verify
  - hardware multipliers common in processors
  - hard to generate compact canonical representation from bit level
  - for verification often equivalence check against reference (reference and design have to share large structural similarities)

General Arithmetic Circuit Verification

- formal property checking of significant importance
- arithmetic circuits still “show stoppers”
- no universal framework for arithmetics, instead special “engineered” solutions
- useful for highly regular designs
- limited in case of full custom logic designs
- multipliers particular hard to verify
  - hardware multipliers common in processors
  - hard to generate compact canonical representation from bit level
  - for verification often equivalence check against reference (reference and design have to share large structural similarities)

General Arithmetic Circuit Verification

- formal property checking of significant importance
- arithmetic circuits still “show stoppers”
- no universal framework for arithmetics, instead special “engineered” solutions
- useful for highly regular designs
- limited in case of full custom logic designs
- multipliers particular hard to verify
  - hardware multipliers common in processors
  - hard to generate compact canonical representation from bit level
  - for verification often equivalence check against reference (reference and design have to share large structural similarities)

General Arithmetic Circuit Verification

- formal property checking of significant importance
- arithmetic circuits still “show stoppers”
- no universal framework for arithmetics, instead special “engineered” solutions
- useful for highly regular designs
- limited in case of full custom logic designs
- multipliers particular hard to verify
  - hardware multipliers common in processors
  - hard to generate compact canonical representation from bit level
  - for verification often equivalence check against reference (reference and design have to share large structural similarities)

General Arithmetic Circuit Verification

- formal property checking of significant importance
- arithmetic circuits still “show stoppers”
- no universal framework for arithmetics, instead special “engineered” solutions
- useful for highly regular designs
- limited in case of full custom logic designs
- multipliers particular hard to verify
  - hardware multipliers common in processors
  - hard to generate compact canonical representation from bit level
  - for verification often equivalence check against reference (reference and design have to share large structural similarities)

Outline

Previous Work

- Binary Moment Diagrams (*BMD) [Bryant, Cheng, Hamaguchi]
  - lack of robustness
- functional decomposition [Chang, Cheng, Fujita, Chen, Aagaard, Seger, Kaivola, Narasimhan]
  - prove internal properties
  - compose global proof of sub-goals
  - manual decomposition
  - non-trivial mapping of lowest proof level to design
- comparison of reference and design based on 1bit-adder network [Stoffel, Wedler]
  - extraction of adder network from design and reference (exponential number of possibilities)
  - equivalence proven by simple calculus

Previous Work

- Binary Moment Diagrams (*BMD) [Bryant, Cheng, Hamaguchi]
  - lack of robustness
- functional decomposition [Chang, Cheng, Fujita, Chen, Aagaard, Seger, Kaivola, Narasimhan]
  - prove internal properties
  - compose global proof of sub-goals
  - manual decomposition
  - non-trivial mapping of lowest proof level to design
- comparison of reference and design based on 1bit-adder network [Stoffel, Wedler]
  - extraction of adder network from design and reference (exponential number of possibilities)
  - equivalence proven by simple calculus

Previous Work

- Binary Moment Diagrams (*BMD) [Bryant, Cheng, Hamaguchi]
  - lack of robustness
- functional decomposition [Chang, Cheng, Fujita, Chen, Aagaard, Seger, Kaivola, Narasimhan]
  - prove internal properties
  - compose global proof of sub-goals
  - manual decomposition
  - non-trivial mapping of lowest proof level to design
- comparison of reference and design based on 1bit-adder network [Stoffel, Wedler]
  - extraction of adder network from design and reference (exponential number of possibilities)
  - equivalence proven by simple calculus

Outline

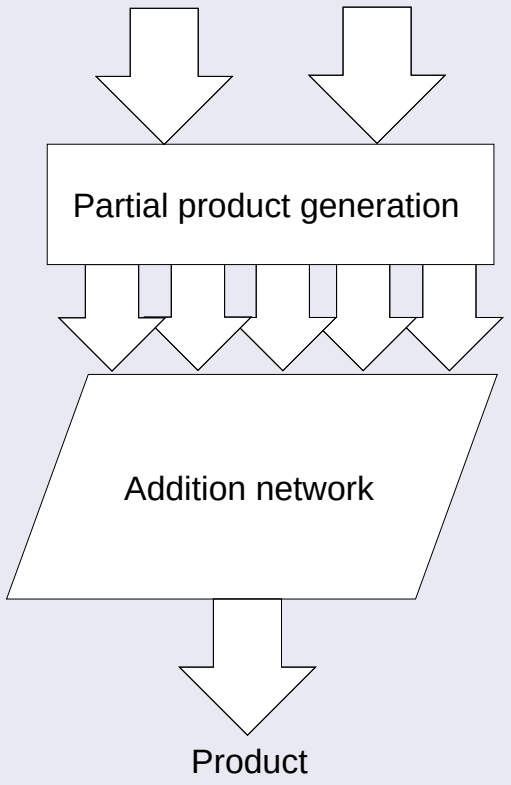
A Simple Multiplier

Algorithm

- integer multiplication $((a_0, \ldots, a_n) \cdot (b_0, \ldots, b_m) = p)$
- basic multiplication (grade school algorithm):

$$\left.\begin{array}{rll} (a_0, \ldots, a_n) & \cdot 2^0 & \cdot b_0 \\ +(a_0, \ldots, a_n) & \cdot 2^1 & \cdot b_1 \\ \vdots & & \vdots \\ +(a_0, \ldots, a_n) & \cdot 2^{m-1} & \cdot b_{m-1} \\ +(a_0, \ldots, a_n) & \cdot 2^m & \cdot b_m \end{array}\right\}$$

Structure



Full Custom Multipliers

- custom multipliers @IBM developed on bit-level
- various optimizations
  - no half-adder instances (just "AND", "XOR"-gates)
  - full-adders spread across cycles
  - no booth-encoder instances (just shifter, multiplexer)
  - constant bits in adder-tree
  - hot-one/ hot-two representation
- no word level information available (hard to extract)

Outline

Arithmetic Reference Description Language

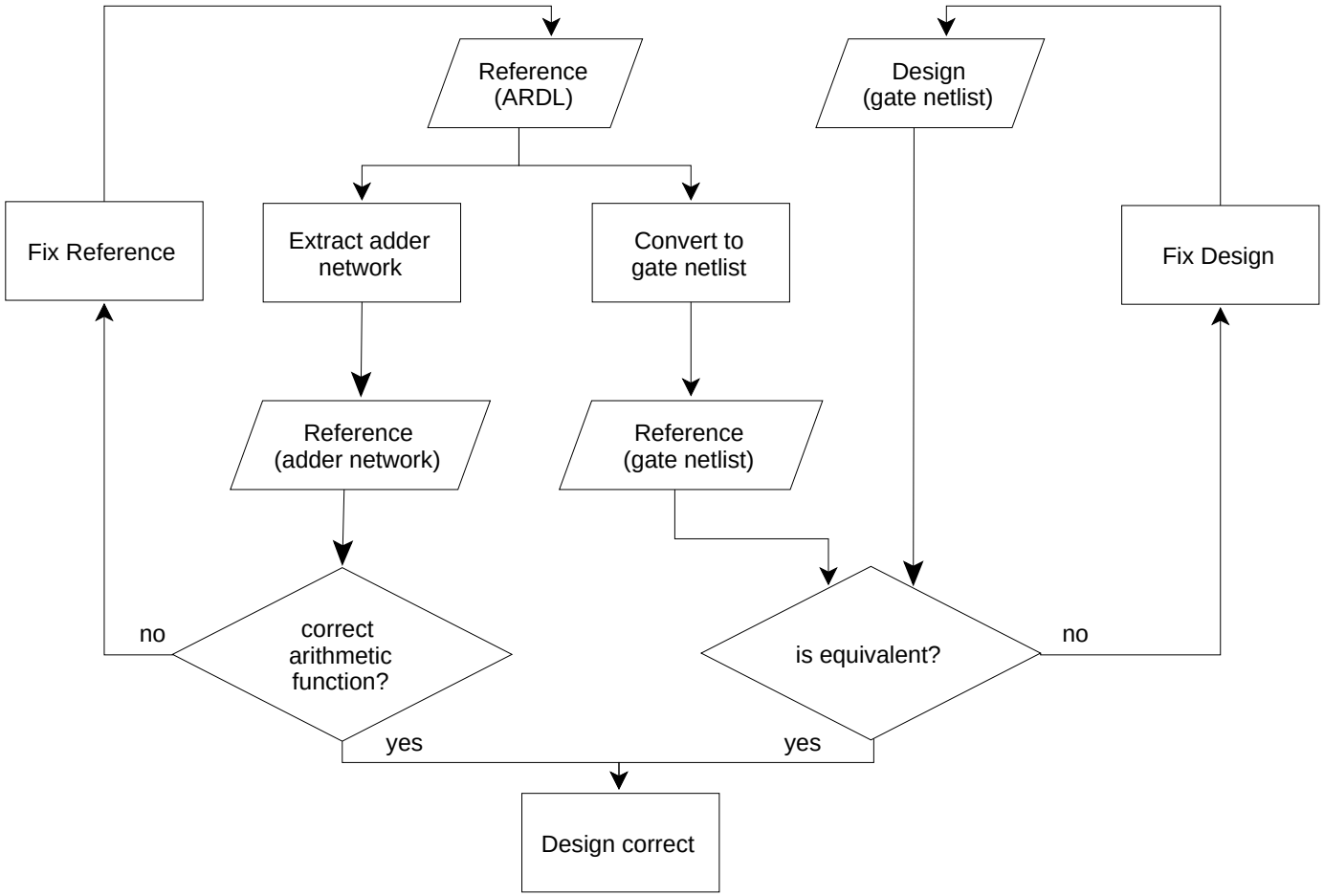
- utilized in verification
- specify multiplier on word level:
  - arithmetic by functions
  - structure by interconnection between functions
- syntax close to HD languages (Verilog / VHDL)
  - used by designer, not verification engineer
- developed at beginning of design process
  - formalization of typical considerations before implementing a design

ARDL Extract - 3Bit Radix2 Multiplier Example

```
variables {
a: in (0 to 2);
b: in (0 to 2);
...}
pp_def{
ppb(0) <= gen_pp(a(0 to 2), booth22(0 & 0 & b(0)));
ppb(1) <= gen_pp(a(0 to 2), booth22(b( 0 to 2)));
...}
tree_def{
s1a <= sum32 (ppb(0), ppb(1), ppb(2));
c1a <= carry32 (ppb(0), ppb(1), ppb(2));
sum <= s1a;
carry <= c1a
...}
```

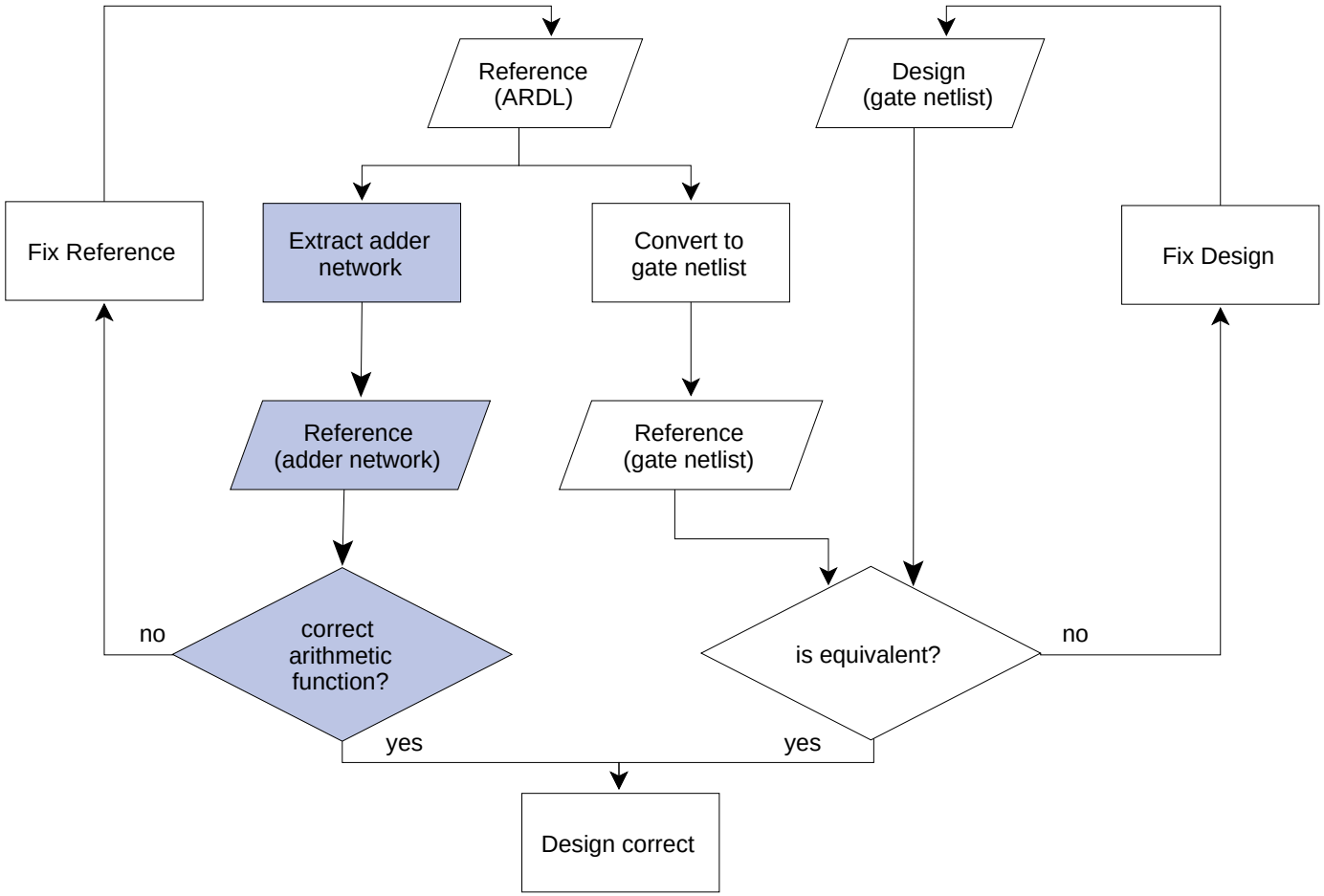
Outline

Approach Overview



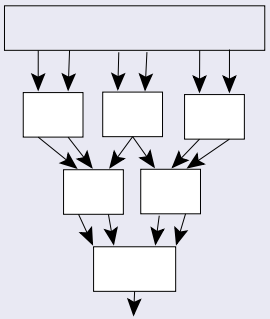
Outline

Overview Arithmetic Proof



ARDL to Structure

structural view



- input: ARDL of multiplier design (given by designer)

functional view

$a = (a_0, \ldots, a_n)$,
$b = (b_0, \ldots, b_m)$
$pp_j = a \cdot B_j$,
$B_j = -2ab_{j+1} + ab_j + ab_{j-1}$
$prod = \sum_{j=0}^{m} pp_j$

- arithm. functions derived from ARDL structure (automatically)
  - Booth encoding
  - adder network

Structure to Bit Arithmetic

functional view

$a = (a_0, \ldots, a_n)$,
$b = (b_0, \ldots, b_m)$
$pp_j = a \cdot B_j$,
$B_j = -2ab_{j+1} + ab_j + ab_{j-1}$
$prod = \sum_{j=0}^{m} pp_j$

- arithm. functions derived from ARDL structure (automatically)
  - Booth encoding
  - adder network

bit arithmetic

$$\begin{array}{cccc} 0 & 0 & 0 & -2a_0b_2 \\ 0 & a_0b_1 & a_1b_1 & a_2b_1 \\ a_1b_0 & a_0b_2 & a_1b_2 & a_2b_2 \end{array}$$

- transformation to basic multiplier definition (automatically)
  - successful/unsuccessful $\rightarrow$ correct/incorrect arithmetic

Example Booth encoded 3bit multiplier

initial situation

$$
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -2a_0b_2 & -2a_1b_2 & -2a_2b_2 \\
0 & 0 & a_0b_1 & a_1b_1 & a_2b_1 & 0 & 0 \\
0 & 0 & a_0b_2 & a_1b_2 & a_2b_2 & 0 & 0 \\
0 & 0 & -2a_0b_0 & -2a_1b_0 & -2a_2b_0 & 0 & 0 \\
a_0b_0 & a_1b_0 & a_2b_0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
p_{-1} & p_0 & p_1 & p_2 & p_3 & p_4 &
\end{array}
$$

Example Booth encoded 3bit multiplier

after transformation

$$
\begin{array}{cccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -a_0 b_2 & -a_1 b_2 & -a_2 b_2 & 0 \\
0 & 0 & a_0 b_1 & a_1 b_1 & a_2 b_1 & 0 & 0 \\
0 & 0 & 0 & 2a_0 b_2 & 2a_1 b_2 & 2a_2 b_2 & 0 \\
0 & -a_0 b_0 & -a_1 b_0 & -a_2 b_0 & 0 & 0 & 0 \\
0 & 2a_0 b_0 & 2a_1 b_0 & 2a_2 b_0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
p_{-1} & p_0 & p_1 & p_2 & p_3 & p_4 &
\end{array}
$$

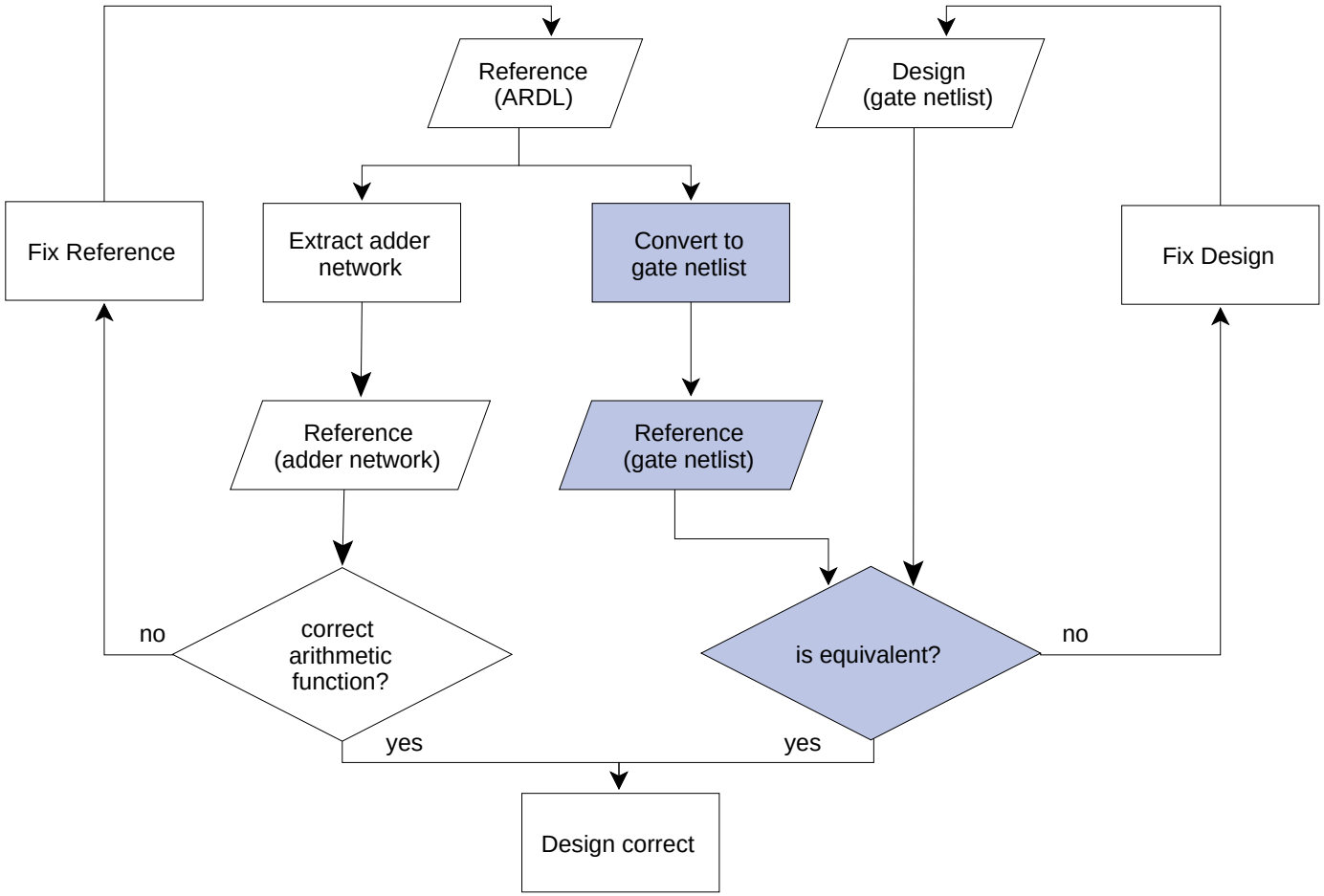
Example Booth encoded 3bit multiplier

after summation

$$
\begin{array}{ccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & a_0 b_1 & a_1 b_1 & a_2 b_1 & 0 & 0 \\
0 & 0 & 0 & a_0 b_2 & a_1 b_2 & a_2 b_2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & a_0 b_0 & a_1 b_0 & a_2 b_0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
p_{-1} & p_0 & p_1 & p_2 & p_3 & p_4 &
\end{array}
$$

Outline

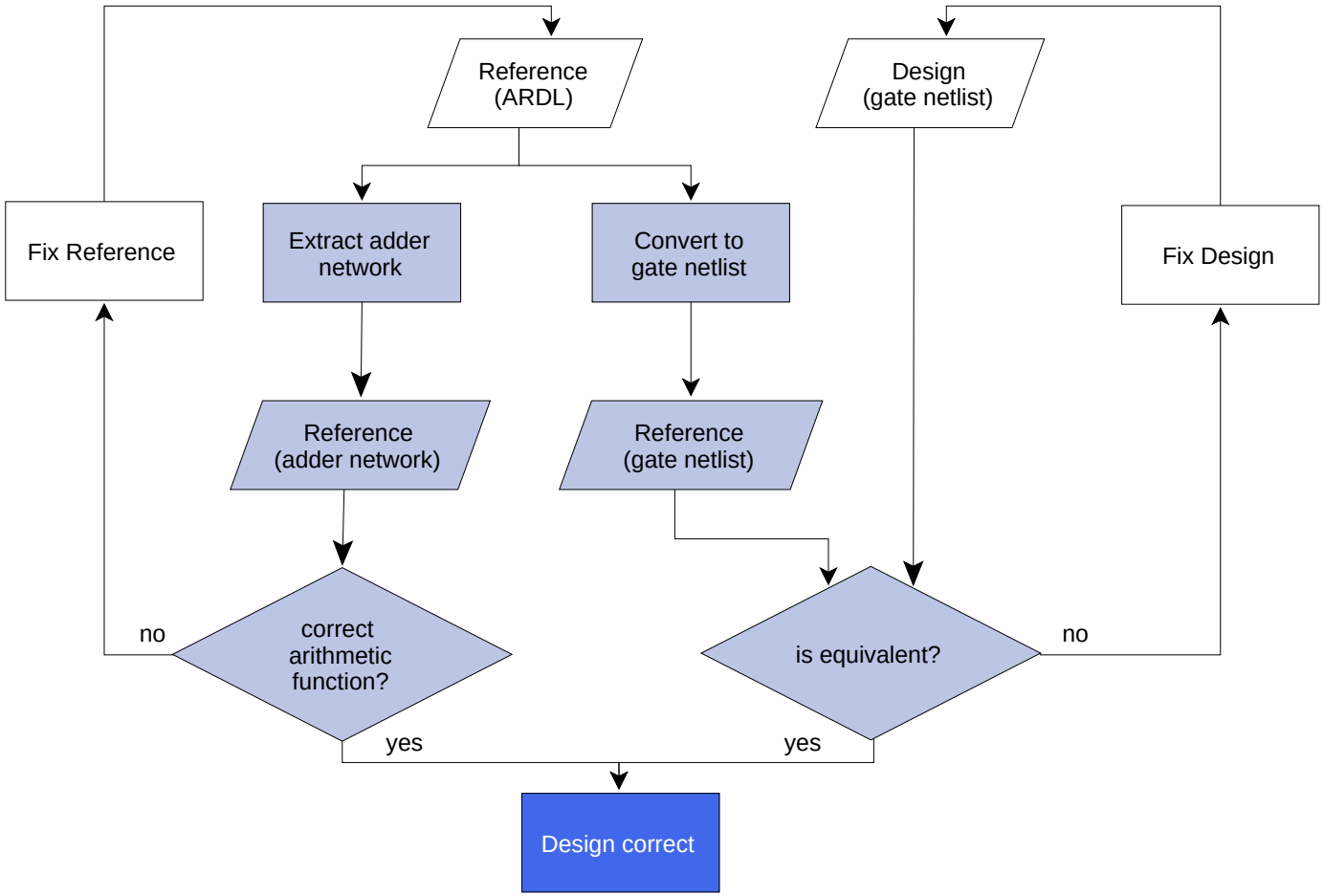
Overview - Equivalence Check



Equivalence Check

- ARDL structure translated into gate-list reference
- functions synthesized into pre-defined blocks (Booth-encoder, adder,...)
- equivalence checked of reference against design (standard SAT solver)
- successful check requires similarities between reference and design
  - similar inputs to adder-tree (same Booth-encoding)
  - adder-tree topology
  - assignments to adders (order of inputs)
- similarities through design concept in ARDL
  - similarities result through methodology
  - designer's responsibility

Overview - Final



Scope of Method

- arithmetic circuits modeled at word-level, with easy to derive bit-level
  - data-path verification
  - arithmetic otherwise hard to obtain from gate netlist
  - flexible
- multicycle operations through unrolling

Outline

Implementation/ Experiments

- prototype implementation (PERL)
- used on several industrial multipliers (IBM)
- complex instructions
  - parallel operations in wide multipliers
  - require different ARDL for each instruction

Operation (operand's bit width)	cpu time AP	EC
4x4	0.6s	2s
8x8	1s	2s
8x8+8x8+8x8+8x8	9s	2s
16x16+16x16	9s	10s
24x24	7s	10s
53x53	8min	15s
64x64	14min	21s

Conclusion

- ARDL suited for binary multiplier designs (FPU, FXU, ...)
- manual effort negligible (formalization of typical design considerations)
- applied to complex instructions (multiply-add)
- unsigned, signed multiplication possible

Summary

- Verification method for multipliers:
- special reference description in ARDL (Arithmetic Reference Description Language)
- ARDL reference for simple bit arithmetic check - transformation to basic multiplication
- ARDL reference for construction of gate-list representation - equivalence check against design

Thank you

Questions?