Arithmetic Overflow Verification using Formal LINT

Kaiwen Chin, Esra Sahin Basaran, Kranthi Pamarthi

Renesas Electronics

accellera SYSTEMS INITIATIVE

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

Arithmetic Overflow Verification Challenge

- Arithmetic overflow verification:
  - Unsigned arithmetic
  - Signed arithmetic
- Traditional methods can be inefficient:
  - Dynamic simulation: Hard to be exhaustive
  - Structural LINT: Lots of false negatives
- Formal LINT = Structural LINT + Formal capability
  - Auto-generated SystemVerilog Assertions for Formal Verification
- Formal LINT looks promising
  - But…
  - The paper provides its prerequisite

Arithmetic Logic Category

- Unsigned Logic
  - The design implements unsigned arithmetic
  - No "signed" keyword
  - "Part select syntax" could be used for readability
  - Manual "zero-padding" at MSBs could be used for readability
- Example

```
wire [3:0] FUL_U1A, FUL_U1B; // Variables are intended to be unsigned.
wire [4:0] Y_U1A = FUL_U1A + FUL_U1B;
wire [4:0] Y_U1B = FUL_U1A[3:0] + FUL_U1B[3:0];
wire [4:0] Y_U1C = {1'b0,FUL_U1A[3:0]} + {1'b0,FUL_U1B[3:0]};
wire [4:0] Y_U1D = {1'b0,FUL_U1A} + {1'b0,FUL_U1B};
```

Arithmetic Logic Category (cont'd)

- Implicit Signed Logic
  - The design implements signed arithmetic
  - No "signed" keyword. Variable signedness is implied by its consuming logic.
  - Manual sign-extension for implicit signed variable must be used for correctness
  - Part-select syntax could be used for readability
  - Manual zero-padding at MSBs could be used for readability (for unsigned variables)
- Example

```
wire [3:0] FUL_U1A; // Variable is intended to be unsigned.
wire [3:0] FUL_S1B; // Variable is intended to be signed but not declared explicitly.
wire [4:0] Y_S1A = FUL_U1A + {FUL_S1B[3],FUL_S1B};
wire [4:0] Y_S1B = {1'b0,FUL_U1A} + {FUL_S1B[3],FUL_S1B};
wire [4:0] Y_S1C = {1'b0,FUL_U1A[3:0]} + {FUL_S1B[3],FUL_S1B[3:0]};
```

- Explicit Signed Logic
  - The design implements signed arithmetic
  - Signed variables are declared using "signed" keyword
  - No part-select syntax for explicit signed variables
  - No manual sign-extension for explicit signed variables
- Example

```
wire        [3:0] FUL_U1A; // Variable is intended to be unsigned.
wire        [3:0] FUL_U1B; // Variable is intended to be unsigned.
wire signed [3:0] FUL_S1C; // Variable is intended to be signed and declared explicitly.
wire signed [3:0] FUL_S1D; // Variable is intended to be signed and declared explicitly.
wire signed [4:0] Y_S1A = FUL_U1A + FUL_U1B;
wire signed [4:0] Y_S1B = FUL_S1C + FUL_S1D;
```

Formal LINT for Unsigned Logic

- Formal LINT proves Y_U3A has no overflow issue.

```
output [3:0] Y_U3A;
input  [3:0] FUL_U1A, FUL_U1B;
wire   [3:0] HLF_U1A = (FUL_U1A > 7) ? 7 : FUL_U1A;
wire   [3:0] HLF_U1B = (FUL_U1B > 7) ? 7 : FUL_U1B;
assign Y_U3A = HLF_U1A + HLF_U1B;
```

Formal LINT for Explicit Signed Logic

- Formal LINT proves Y_S3F has no overflow issue.

```
output signed [3:0] Y_S3F;
input  signed [3:0] FUL_S1A, FUL_S1B;
wire   signed [3:0] HLF_S1A, HLF_S1B;
assign HLF_S1A = (FUL_S1A > 3) ? 3 : (FUL_S1A < -4) ? -4 : FUL_S1A;
assign HLF_S1B = (FUL_S1B > 3) ? 3 : (FUL_S1B < -4) ? -4 : FUL_S1B;
assign Y_S3F = HLF_S1A + HLF_S1B;
```

Formal LINT for Implicit Signed Logic

- Formal LINT treat both operands as unsigned and flag error.

```
output [3:0] Y_SCf;
input  [3:0] FUL_S1e, FUL_S1f;
wire   [3:0] HLF_S1e = (FUL_S1e[3:2]==2'b01) ? 4'b0011 :
                       (FUL_S1e[3:2]==2'b10) ? 4'b1100 : FUL_S1e[3:0] ;
wire   [3:0] HLF_S1f = (FUL_S1f[3:2]==2'b01) ? 4'b0011 :
                       (FUL_S1f[3:2]==2'b10) ? 4'b1100 : FUL_S1f[3:0] ;
assign Y_SCf[3:0] = {HLF_S1e[3], HLF_S1e[3:0]} + {HLF_S1f[3], HLF_S1f[3:0]};
// Formal LINT doesn't know the operands are signed in the design intention.
```

Formal LINT is not for all of them!

- The key issue is variable's signedness information

Category	Pitfalls	Formal LINT limitation
Implicit Signed Logic	None	• Limitation: Lack of variable signedness information. • Formal LINT currently may not accurately analyze it. • Work-in-progress for EDA vendors
Explicit Signed Logic	Many (show you later)	• No showstopper for Formal LINT • Complementary checks required • Work-in-progress for EDA vendors
Unsigned Logic	None	• No limitation • Formal LINT is fully capable of its verification

Formal LINT is most promising for these two types.

Pitfall in Explicit Signed Logic

- Signed-to-unsigned conversion: Mixture of signed and unsigned in equation

```
wire [3:0] FUL_U1A;
wire signed [3:0] FUL_S1A, FUL_S1B;
wire [4:0] Y_U1B = FUL_U1A + FUL_S1A
```

Unsigned variable

Signed variable converted into unsigned during evaluation of the equation

Pitfall in Explicit Signed Logic (cont'd)

- Signed-to-unsigned conversion: Result of concatenation is treated as unsigned

```
wire signed [3:0] S4b_A, S4b_B;
wire signed [4:0] S5b_Y = {S4b_A[3],S4b_A} + {S4b_B[3],S4b_B};
```

- Signed-to-unsigned conversion: Part-select changes the variable into unsigned

```
wire signed [3:0] S4b_A, S4b_B;
wire signed [4:0] S5b_X = S4b_A[3:0] + S4b_B[3:0];
```

They will be automatically zero-padded instead of sign-extended

- Bad Sign Casting:

```
wire [3:0] A_U4b;
wire signed [3:0] B_S4b;
wire signed [5:0] X_S6b = A_U4b + B_S4b; // Signed-to-unsigned conversion

wire signed [5:0] Y_S6b = $signed(A_U4b) + B_S4b; // Bad sign casting

wire signed [5:0] Z_S6b = $signed({1'b0,A_U4b}) + B_S4b; // Good sign casting
```

Example: If A_U4b is "+15", $signed(A_U4b) will be interpreted as "-1".

Solution: Zero-padding before sign-casting

(Reference: Dr. Greg Tumbush, "Signed Arithmetic in Verilog 2001 – Opportunities and Hazards," in DVCON 2005)

- Bad signed constant:

```
wire signed [3:0] A_S4b;
wire signed [5:0] Y_S6b = A_S4b + 4'sd8;
wire signed [5:0] Z_S6b = A_S4b + 5'sd8;
```

Use a signed constant to keep A_S4b as signed. But "4'sd8" will be interpreted as "-8".

This is the right way.

Value range of 4-bit signed operand : +7 ~ -8
Value range of 5-bit signed operand : +15 ~ -16

Pitfall in Explicit Signed Logic (cont'd)

- Interim result overflow:

```
wire [3:0] B = 4'b0011;
wire [3:0] C = 4'b1110;
wire [3:0] Y2 = (B+C) >>> 1; // Y2 = 4'b0000
wire [3:0] Y4 = (B+C) / 2  ; // Y4 = 4'b1000
```

"B+C" evaluated as 4-bit expression → Overflow already before shift

"(B+C)/2" evaluated as 4-bit expression. Result is correct.

- Formal LINT should be accompanied by **pitfall checks**.
- We evaluated EDA tools from two vendors:

Check Items		Required LINT type	Commercial Solutions
Arithmetic overflow (LHS variable is not wide enough to hold result from RHS equation)		Formal	Covered
Pitfall checks		**Required LINT type**	**Commercial Solutions**
Signed-to-unsigned conversion	Due to mixed signed and unsigned operands in equation	Structural	Covered
	Due to part-select	Structural	Work in progress
	Due to concatenation	Structural	Work in progress
Bad sign-casting		Formal	Work in progress
Bad signed constant		Structural	Work in progress
Operator precedence of arithmetic shift (Tools won't know designer's intention)		n/a	n/a
Interim result overflow		Formal	Work in progress

REFERENCES

[1] "IEEE Standard for Verilog Hardware Description Language," in IEEE Std 1364-2005 (Revision of IEEE Std 1364-2001) , vol., no., pp.1-590, 7 April 2006, doi: 10.1109/IEEESTD.2006.99495.
[2] "IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language," in IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012) , vol., no., pp.1-1315, 22 Feb. 2018, doi: 10.1109/IEEESTD.2018.8299595
[3] Dr. Greg Tumbush, "Signed Arithmetic in Verilog 2001 – Opportunities and Hazards," in DVCON 2005

RENESAS

Special thanks to: Cadence, Synopsys