2025
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
TAIWAN

Automatically Fix RTL Lint Violations with GenAI

Chih-Jen Wei, Phison

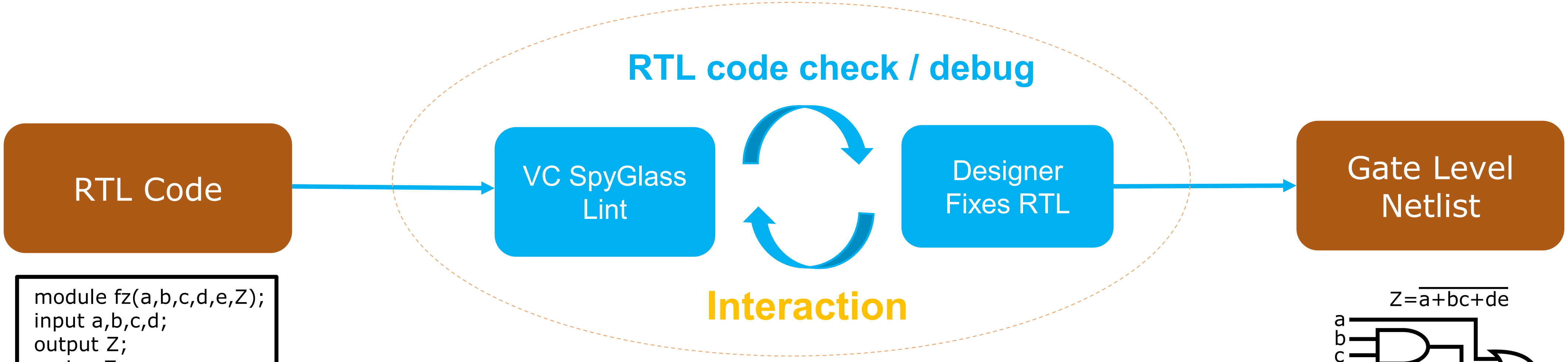
PHISON SYNOPSYS® accellera SYSTEMS INITIATIVE

Agenda

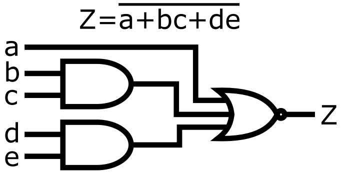
- Problem Statement
- How to Automatically Fix RTL Code With AI?
  - Using Phison in-house solution
  - The results of using Phison in-house solution
- Fine-Tuning with Phison aiDAPTIV+ to Improve Correction Rate
  - Training LLM for better results
- Conclusions
- Future Works

Problem Statement

- During initial stage of using VC SpyGlass Lint for RTL code development, debugging often results in numerous rule violations, which can consume a significant amount of time for designers.
- Therefore, we propose leveraging automatic generation capabilities of AI to automatically fix RTL, reducing time developers spend on manual corrections and thus improving efficiency.

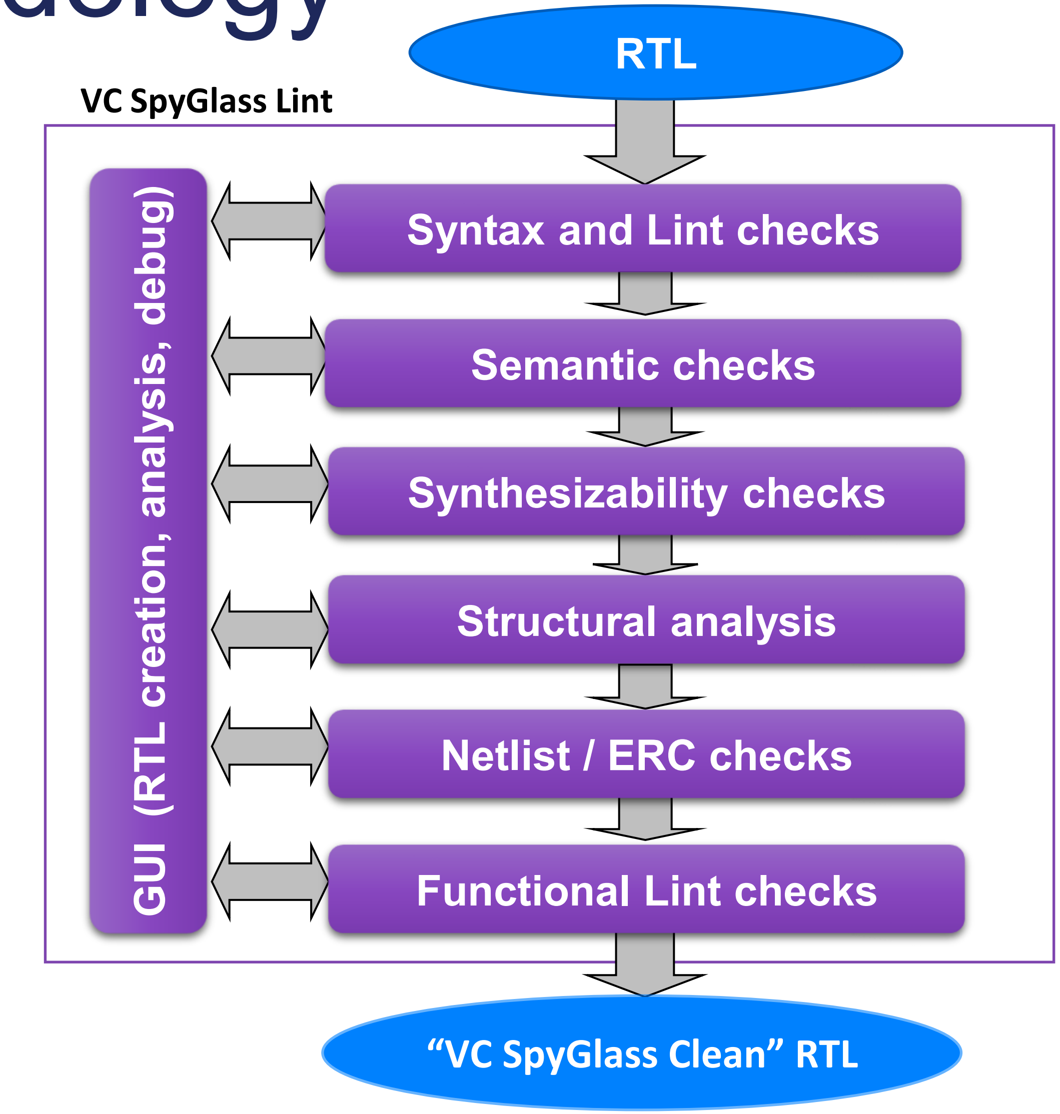


```
module fz(a,b,c,d,e,Z);
input a,b,c,d;
output Z;
assign Z =
 ~(a|(b&c)|(d&e));
endmodule
```



VC SpyGlass Lint Methodology

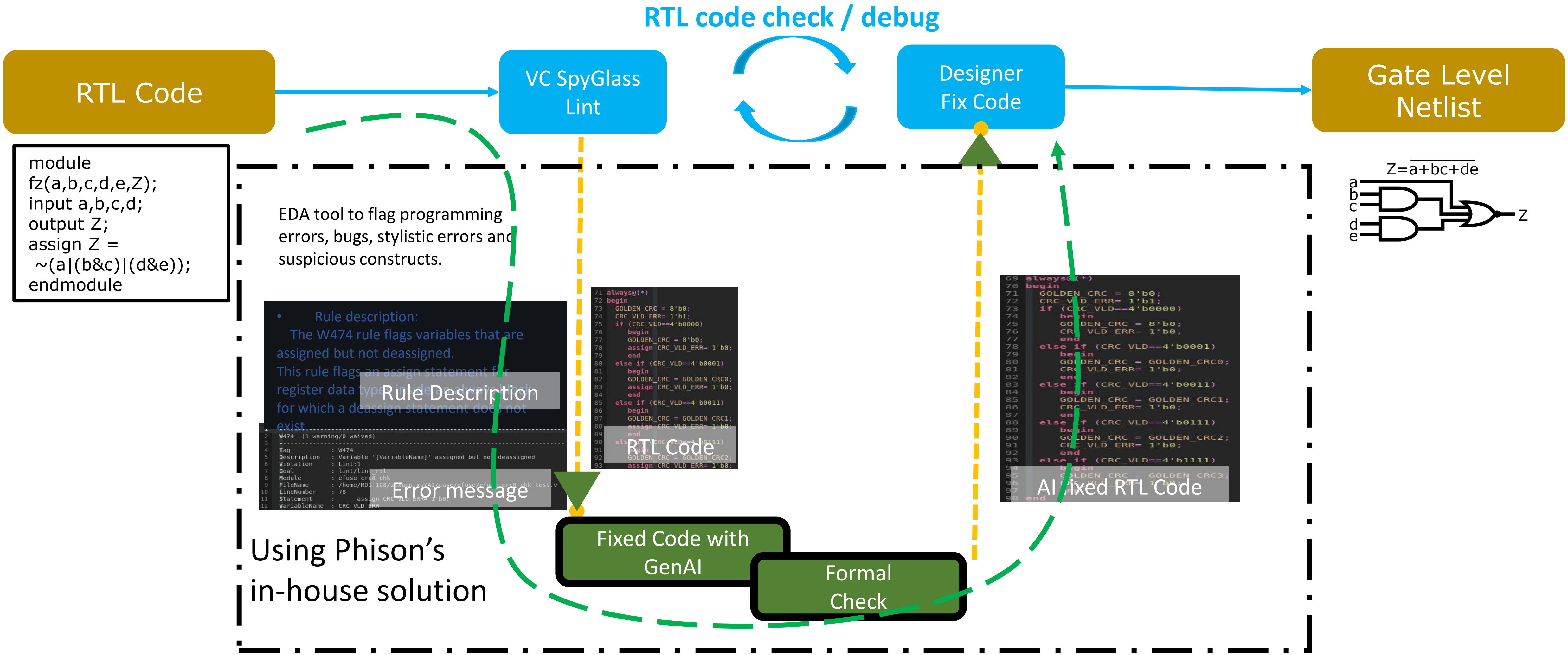
- Find & fix bugs at source (correct-by-construction RTL)
  - Compliance to coding guidelines, STARC, OpenMore, Morelint, etc.
  - Synthesizability & simulation issues
  - Structural, logical and connectivity issues
  - Electrical rule checks
- Verilog, VHDL, SystemVerilog and mixed RTL support
- Structured methodology and templates help tackle design issues systematically
- Comprehensive waiver support
- Easy debug with cross-probe to RTL in Verdi GUI



How to Automatically Fix RTL Code With AI?

Using Phison in-house solution

Automatically Fix RTL Lint Violations Using AI



```
module
fz(a,b,c,d,e,Z);
input a,b,c,d;
output Z;
assign Z =
 ~(a|(b&c)|(d&e));
endmodule
```

EDA tool to flag programming errors, bugs, stylistic errors and suspicious constructs.

- Rule description:

  The W474 rule flags variables that are assigned but not deassigned.
  This rule flags an assign statement for register data types inside an always block, for which a deassign statement does not exist.

```
W474  (1 warning/0 waived)
-----------------------------------------------------------------------------
Tag            : W474
Description    : Variable '[VariableName]' assigned but not deassigned
Violation      : Lint:1
Goal           : lint/lint_rtl
Module         : efuse_crc8_chk
FileName       : /home/RD1_IC8/zixuan_xu/AI/case/efuse/efuse_crc8_chk_test.v
LineNumber     : 78
Statement      :     assign CRC_VLD_ERR= 1'b0;
VariableName   : CRC_VLD_ERR
```

```
always@(*)
begin
  GOLDEN_CRC = 8'b0;
  CRC_VLD_ERR= 1'b1;
  if (CRC_VLD==4'b0000)
     begin
     GOLDEN_CRC = 8'b0;
     assign CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0001)
     begin
     GOLDEN_CRC = GOLDEN_CRC0;
     assign CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0011)
     begin
     GOLDEN_CRC = GOLDEN_CRC1;
     assign CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0111)
     begin
     GOLDEN_CRC = GOLDEN_CRC2;
     assign CRC_VLD_ERR= 1'b0;
```

```
always@(*)
begin
  GOLDEN_CRC = 8'b0;
  CRC_VLD_ERR= 1'b1;
  if (CRC_VLD==4'b0000)
     begin
     GOLDEN_CRC = 8'b0;
     CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0001)
     begin
     GOLDEN_CRC = GOLDEN_CRC0;
     CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0011)
     begin
     GOLDEN_CRC = GOLDEN_CRC1;
     CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0111)
     begin
     GOLDEN_CRC = GOLDEN_CRC2;
     CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b1111)
     begin
     GOLDEN_CRC = GOLDEN_CRC3;
     CRC_VLD_ERR= 1'b0;
     end
end
```

Using Phison’s
in-house solution

Prerequisites for Automatic Gen. of Lint Viol. Fixes

Selecting Correct LLM, Server Platform, Prompt, Token, and Rule

LLM Selection

Select a Suitable AI Model for Generating RTL Code

Server Platform

Select a platform with reasonable pricing and high quality as the basis for AI-generated content

Prompt & Token

The size of the token and the accuracy of the prompt will significantly affect the accuracy and completeness of the AI-generated content

Classification & Statistics

Statistical Analysis of Commonly Waived Lint Rules and Categorization of Lint Rules Suitable for AI Correction

Violation Rule Classification & Statistics

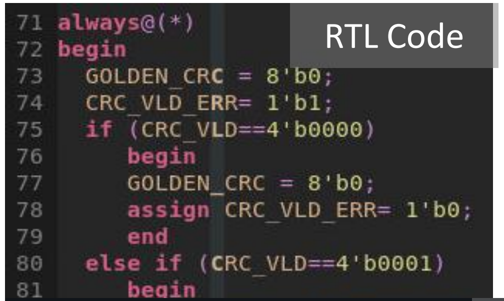
- Choosing Lint rules to be targeted for automated generation of RTL Fixes
- Designer first categorizes violations to determine which types of violations have potential to be successfully corrected by AI and will also compile statistics on the most common violations in each project.

LINT RULE	Classification	Project 1 Violation Number
STARC05-3.3.1.4b	C	9448
W164b	B	4397
AsgnToOneBit-ML	A	2553
ImproperRangeIndex-ML	C	2205
W416	B	2050
W164a	B	1215
W154	A	924
FlopClockConstant	C	753
W362	B	700
NoExprInPort-ML	A	502
STARC05-2.3.1.4	C	465
ResetFlop-ML	C	341
W241	C	160

LINT RULE	Classification	Project 2 Violation Number
STARC05-3.3.1.4b	C	1171
W164a	B	906
W287a	C	718
W362	B	440
STARC05-2.2.3.1	A	385
RptNegEdgeFF-ML	C	354
W164b	C	329
NoExprInPort-ML	B	294
ResetFlop-ML	C	280
W443	C	236
W241	C	229

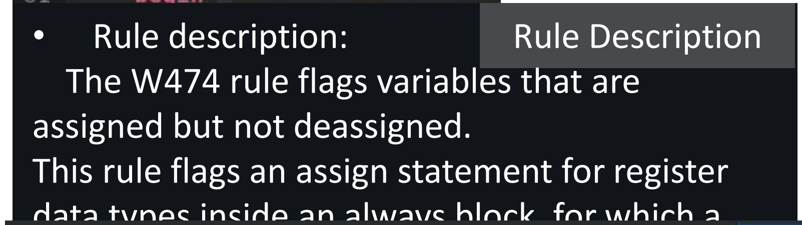
A: Can be Corrected, Accuracy 90%~100%
B: Potentially Correctable, Accuracy 60%~80%
C: Not Suitable, Requires Human Intervention

Providing Prompt for GenAI to Fix RTL Code



RTL Code

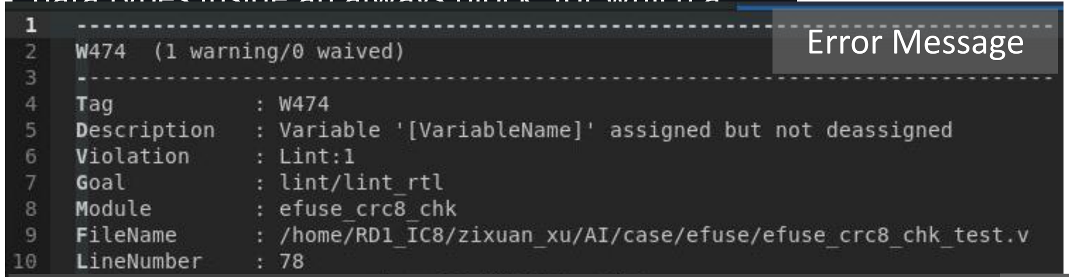
```
always@(*)
begin
  GOLDEN_CRC = 8'b0;
  CRC_VLD_ERR= 1'b1;
  if (CRC_VLD==4'b0000)
     begin
     GOLDEN_CRC = 8'b0;
     assign CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0001)
     begin
```



Rule Description

- Rule description:

The W474 rule flags variables that are assigned but not deassigned.
This rule flags an assign statement for register data types inside an always block, for which a



Error Message

```
--------------------------------------------------------------------------------
W474  (1 warning/0 waived)
--------------------------------------------------------------------------------
Tag              : W474
Description      : Variable '[VariableName]' assigned but not deassigned
Violation        : Lint:1
Goal             : lint/lint_rtl
Module           : efuse_crc8_chk
FileName         : /home/RD1_IC8/zixuan_xu/AI/case/efuse/efuse_crc8_chk_test.v
LineNumber       : 78
```

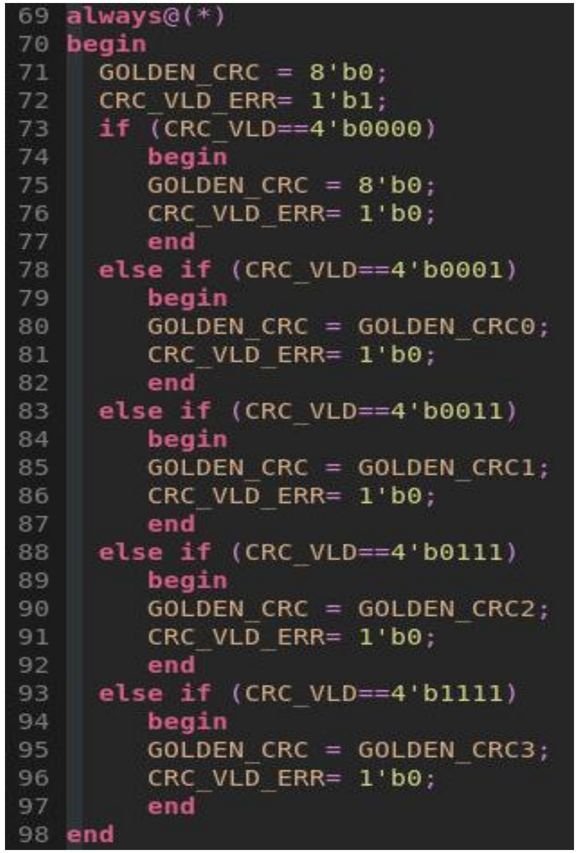
Prompt

System Prompt:
Clearly tell the AI what to do.
User Prompt:
Provide detailed items such as: analyze first, then fix the code, and provide examples for reference.

GenAI

Precise Inputs to GenAI is key.

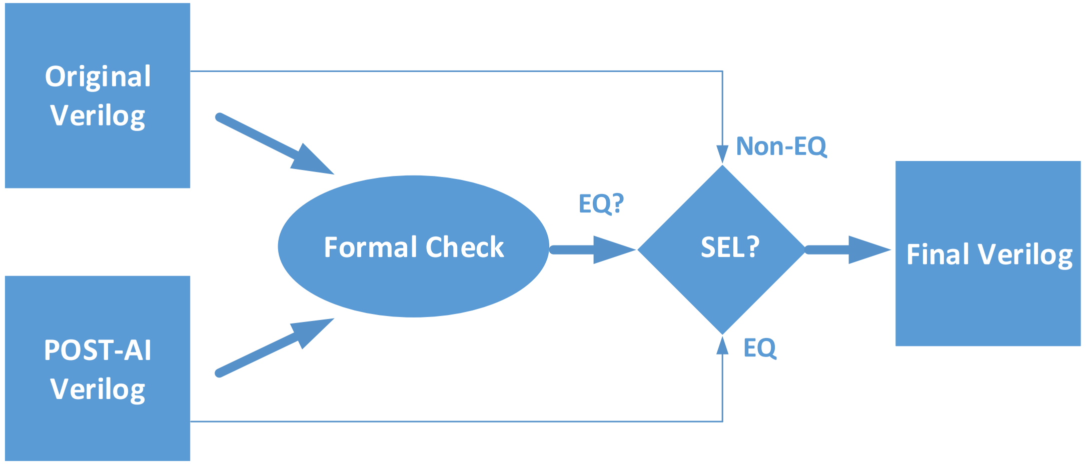
The RTL code has been corrected by AI

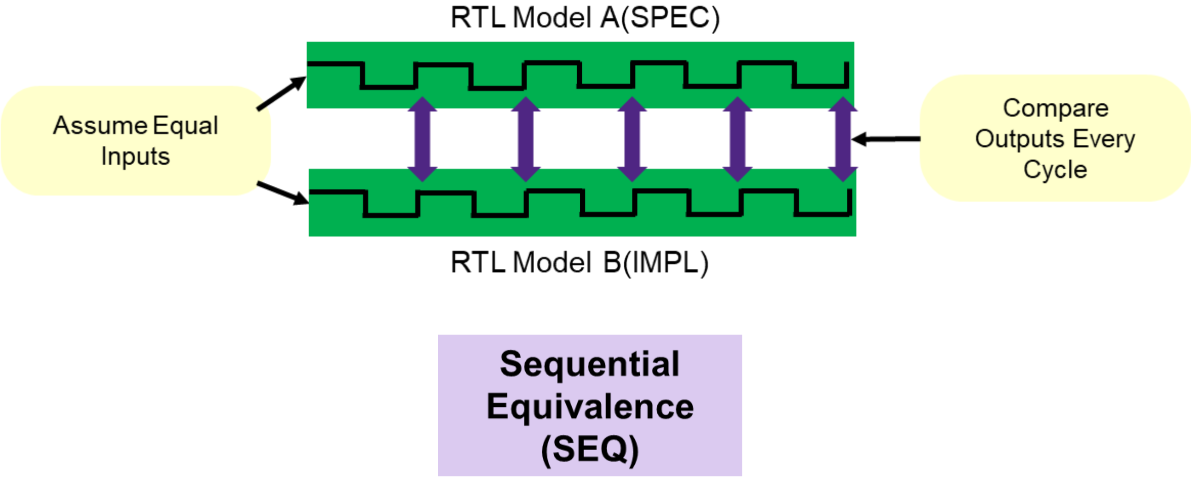


```
always@(*)
begin
  GOLDEN_CRC = 8'b0;
  CRC_VLD_ERR= 1'b1;
  if (CRC_VLD==4'b0000)
     begin
     GOLDEN_CRC = 8'b0;
     CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0001)
     begin
     GOLDEN_CRC = GOLDEN_CRC0;
     CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0011)
     begin
     GOLDEN_CRC = GOLDEN_CRC1;
     CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b0111)
     begin
     GOLDEN_CRC = GOLDEN_CRC2;
     CRC_VLD_ERR= 1'b0;
     end
  else if (CRC_VLD==4'b1111)
     begin
     GOLDEN_CRC = GOLDEN_CRC3;
     CRC_VLD_ERR= 1'b0;
     end
end
```

Importance of Formal Verification in AI Generation Processes

- **Necessity of Formal Verification**
  - AI-generated results are probabilistic
  - There is a possibility of errors
- **Challenges in Automating RTL Code Correction**
  - Designers still need to confirm the correctness of AI corrections themselves
  - This affects the efficiency and purpose of automated RTL code correction
- **Solution**
  - Synopsys VC Formal Sequential Equivalence Checking (SEQ) can be used as an alternative tool
  - Adopt third-party Formal tools for post-generation verification





How to Automatically Fix RTL Code With AI?

The results of using Phison in-house solution

Automatically Fix Code on Real Project

The following table presents real project cases where the flow was used to:

Lint Rule	Initial Violations (A)	POST-AI Remaining[1] Violations at once (B)	Correction Rate (C)
AsgnToOneBit-ML	2553	1167	**54.29%**
W154	920	152	**83.48%**
STARC05-2.2.3.1	9	2	**77.78%**
NoExprInPort-ML	492	340	**30.89%**
W164b	4395	4294	**2.3%**
W164a	1211	1132	**6.52%**
W416	2050	1970	**3.9%**
W362	700	611	**11.9%**

Post-AI Correction Violations Handling: At once vs In segments

Correcting multiple rules simultaneously in a single block has been shown to affect the correction rate (RTL Fixes). Therefore, executing corrections in segments can effectively improve accuracy.

Lint Rule	Post-AI Correction Rate at once	Post-AI Correction[1] Rate in segments	Once vs SEG. DIFF. Rate	Segments
AsgnToOneBit-ML	54.29%	**63.22%**	+8.93%	Segment 1
W154	83.48%	**88.26%**	+4.78%	
NoExprInPort-ML	30.89%	**42.48%**	+11.59%	
STARC05-2.2.3.1	77.78%	**77.78%**	+0%	
W164b	2.3%	**34.74%**	+32.44%	Segment 2
W164a	6.52%	**15.85%**	+9.33%	
W416	3.9%	**75.9%**	+72%	
W362	11.9%	**21.39%**	+9.49%	

RTL Correction Rate via Different AI Model Using Same Prompt

The same prompt might affect different model generation results as shown in following table:

Lint Rule	Initial Violations	Mistral Large[1] POST-AI Remaining Violations	LLAMA[2] POST-AI Remaining Violations	QWEN[3] POST-AI Remaining Violations
AsgnToOneBit-ML	2553	1167	2005	2026
W154	920	152	174	173
STARC05-2.2.3.1	9	2	2	2
NoExprInPort-ML	492	340	384	381
W164b	4395	4294	4239	4246
W164a	1211	1132	1275	1214
W416	2050	1970	1975	1982
W362	700	611	608	607
Correction Rate (Post-AI Cor. / Initial Vio.)	-	**21.58%**	**13.52%**	**13.77%**

Analysis of the Relationship between AI Correction Time and Token Size

The size of tokens indeed affects the time it takes for AI to generate content. Filtering out large token files can help shorten the generation time.

Lint Rule	Initial Violations	POST-AI Remaining[1] Violations	POST-AI Remaining[1] Violations
AsgnToOneBit-ML	2553	1167	1454
W154	920	152	268
STARC05-2.2.3.1	9	2	2
NoExprInPort-ML	492	340	316
W164b	4395	4294	4376
W164a	1211	1132	1138
W416	2050	1970	1997
W362	700	611	640
Token	-	<4000	<2500
Correction Rate (Post-AI Cor. / Initial Vio.)	-	**21.58%**	**17.34%**
Run time	-	**7H 29min**	**1H 55min**

Fine-Tuning with Phison aiDAPTIV+ to Improve Correction Rate

Training LLM for better results

Fine-Tuning a Model Using LoRA to Improve Correction Rate

We first fine-tuned W164b to confirm whether it could improve the correction rate.
The experimental results showed that the correction rate indeed increased.

Item	Initial Violation	POST-AI Remaining Violations	POST-AI Remaining Violations
W164b	4395	2868	**2746**
Correction Rate	-	34.74%	**37.51%**
Fine-Tuning	-	X	**V**

Extend GPU RAM: aiDAPTIV+
AI Model : Mistral Large 123b
Dataset : 1000 patterns
EPOCH : 3
Run time : 21H
GPU:A6000*8 ,RAM: 384GB

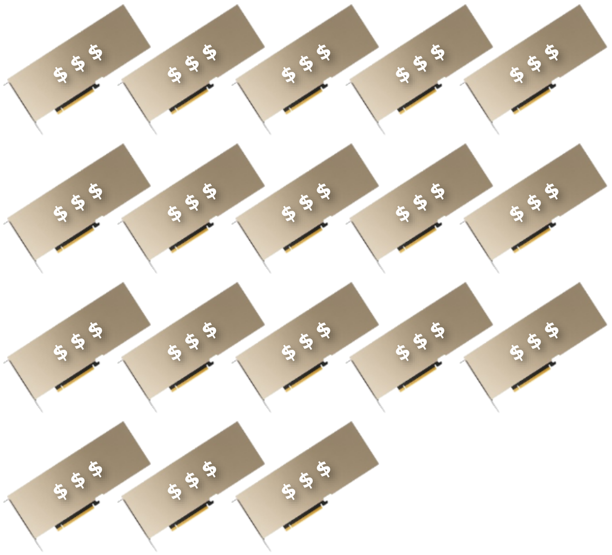
Phison's aiDAPTIVCache Removes HBM Limitation

Current AI Computing Architecture

Limited by GPU HBM

18 H100 GPU to run LLaM2 (70B)

(Requires 3 DGX Chassis)

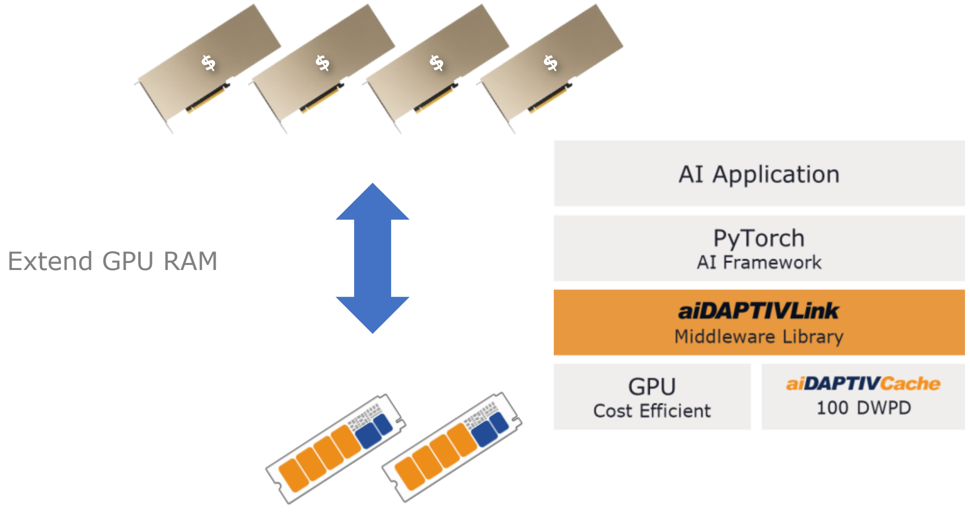


Phison's aiDAPTIV+ AI Computing Architecture

Unlimited Model Size[1]

Scale # GPU to match budget

(Requires 4 GPU and 2 SSD)



ai100E aiDAPTIVCache
w/ 100 DWPD

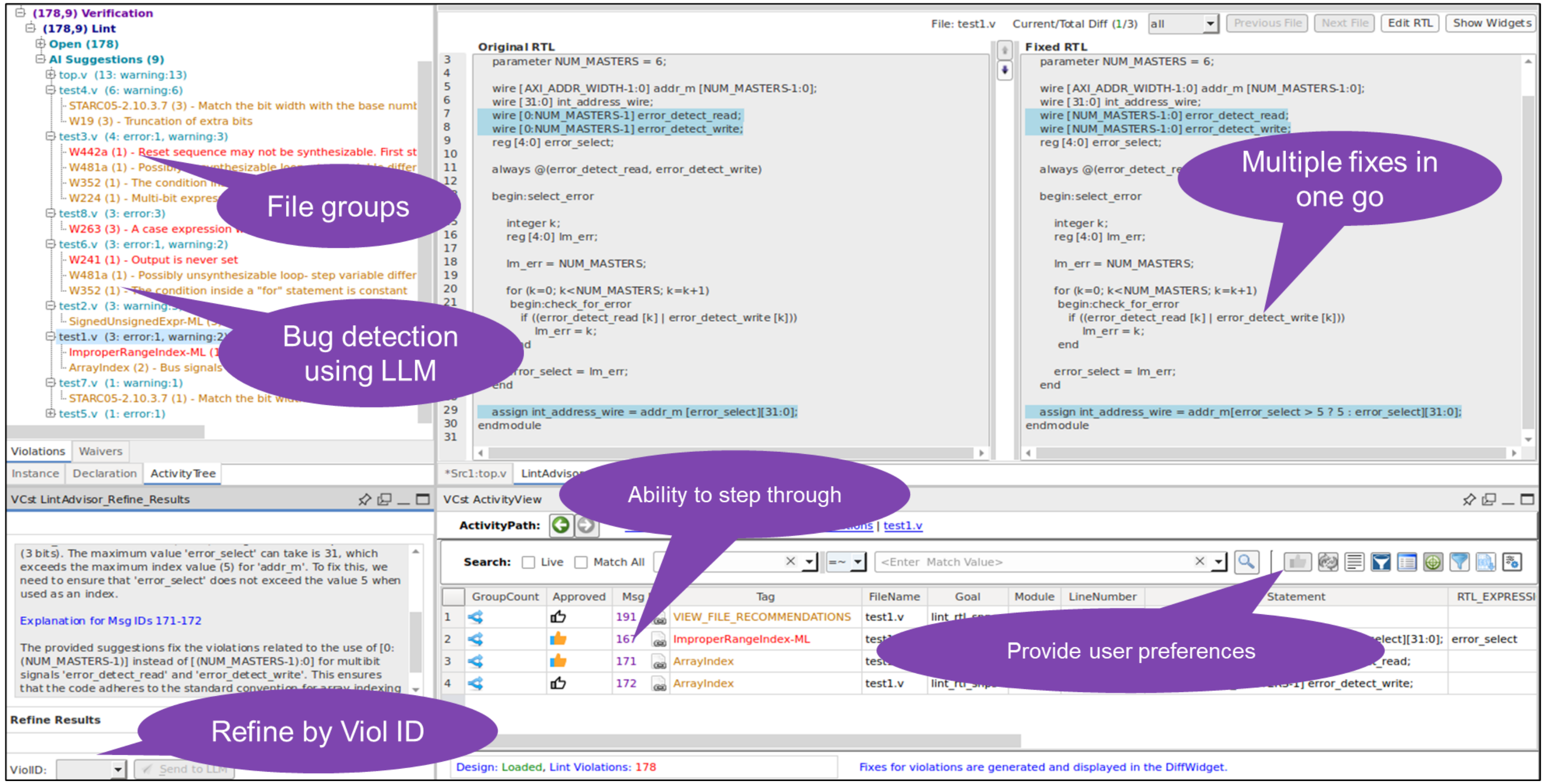
1: Based on aiDAPTIVCatch capacity

Conclusions

- **Automated Correction Using AI Generation Technology**:
  - By leveraging AI generation technology in conjunction with Lint violation reports, automated corrections were achieved, significantly improving work efficiency.
  - Formal tools were used to automatically verify consistency before and after corrections, ensuring the accuracy and reliability of each correction.
- **Significant Correction of Violations**:
  - Experimental results showed that this process can significantly correct at least 10% of violations, and in some cases, the correction rate is even higher.
- **Fine-tuning with Phison aiDAPTIV+**:
  - Using Phison aiDAPTIV^{+} with limited hardware support for fine-tuning further enhanced the accuracy of corrections, ensuring optimal correction outcomes.

Future Work

- We also plan to explore GenAI powered **VC SpyGlass Lint Advisor with Phison's aiDAPTIV+** AI Computing Architecture for
  - Automated RTL Lint violation fixes, verifying design specific custom bug finding and automated waiver generation



File groups

Bug detection using LLM

Multiple fixes in one go

Ability to step through

Provide user preferences

Refine by Viol ID

```
(178,9) Verification
  (178,9) Lint
    Open (178)
    AI Suggestions (9)
      top.v  (13: warning:13)
      test4.v  (6: warning:6)
        STARC05-2.10.3.7 (3) - Match the bit width with the base numb
        W19 (3) - Truncation of extra bits
      test3.v  (4: error:1, warning:3)
        W442a (1) - Reset sequence may not be synthesizable. First st
        W481a (1) - Possibly unsynthesizable loop- step variable differ
        W352 (1) - The condition in
        W224 (1) - Multi-bit expres
      test8.v  (3: error:3)
        W263 (3) - A case expression
      test6.v  (3: error:1, warning:2)
        W241 (1) - Output is never set
        W481a (1) - Possibly unsynthesizable loop- step variable differ
        W352 (1) - The condition inside a "for" statement is constant
      test2.v  (3: warning:3)
        SignedUnsignedExpr-ML (3)
      test1.v  (3: error:1, warning:2)
        ImproperRangeIndex-ML (1
        ArrayIndex (2) - Bus signals
      test7.v  (1: warning:1)
        STARC05-2.10.3.7 (1) - Match the bit wid
      test5.v  (1: error:1)
```

Original RTL

```
  parameter NUM_MASTERS = 6;

  wire [AXI_ADDR_WIDTH-1:0] addr_m [NUM_MASTERS-1:0];
  wire [31:0] int_address_wire;
  wire [0:NUM_MASTERS-1] error_detect_read;
  wire [0:NUM_MASTERS-1] error_detect_write;
  reg [4:0] error_select;

  always @(error_detect_read, error_detect_write)

  begin:select_error

     integer k;
     reg [4:0] lm_err;

     lm_err = NUM_MASTERS;

     for (k=0; k<NUM_MASTERS; k=k+1)
      begin:check_for_error
        if ((error_detect_read [k] | error_detect_write [k]))
           lm_err = k;
      end

     error_select = lm_err;
  end

  assign int_address_wire = addr_m [error_select][31:0];
endmodule
```

Fixed RTL

```
  parameter NUM_MASTERS = 6;

  wire [AXI_ADDR_WIDTH-1:0] addr_m [NUM_MASTERS-1:0];
  wire [31:0] int_address_wire;
  wire [NUM_MASTERS-1:0] error_detect_read;
  wire [NUM_MASTERS-1:0] error_detect_write;
  reg [4:0] error_select;

  always @(error_detect_re

  begin:select_error

     integer k;
     reg [4:0] lm_err;

     lm_err = NUM_MASTERS;

     for (k=0; k<NUM_MASTERS; k=k+1)
      begin:check_for_error
        if ((error_detect_read [k] | error_detect_write [k]))
           lm_err = k;
      end

     error_select = lm_err;
  end

  assign int_address_wire = addr_m[error_select > 5 ? 5 : error_select][31:0];
endmodule
```

(3 bits). The maximum value 'error_select' can take is 31, which exceeds the maximum index value (5) for 'addr_m'. To fix this, we need to ensure that 'error_select' does not exceed the value 5 when used as an index.

Explanation for Msg IDs 171-172

The provided suggestions fix the violations related to the use of [0:(NUM_MASTERS-1)] instead of [(NUM_MASTERS-1):0] for multibit signals 'error_detect_read' and 'error_detect_write'. This ensures that the code adheres to the standard convention for array indexing

Refine Results

ViolID: Send to LLM

ActivityPath: ons | test1.v

Search: □ Live □ Match All =~ <Enter Match Value>

	GroupCount	Approved	Msg	Tag	FileName	Goal	Module	LineNumber	Statement	RTL_EXPRESSI
1		👍	191	VIEW_FILE_RECOMMENDATIONS	test1.v	lint_rtl_sn				
2		👍	167	ImproperRangeIndex-ML	test				elect][31:0];	error_select
3		👍	171	ArrayIndex	tes				_read;	
4		👍	172	ArrayIndex	test1.v	lint_rtl_snp			ERS-1] error_detect_write;	

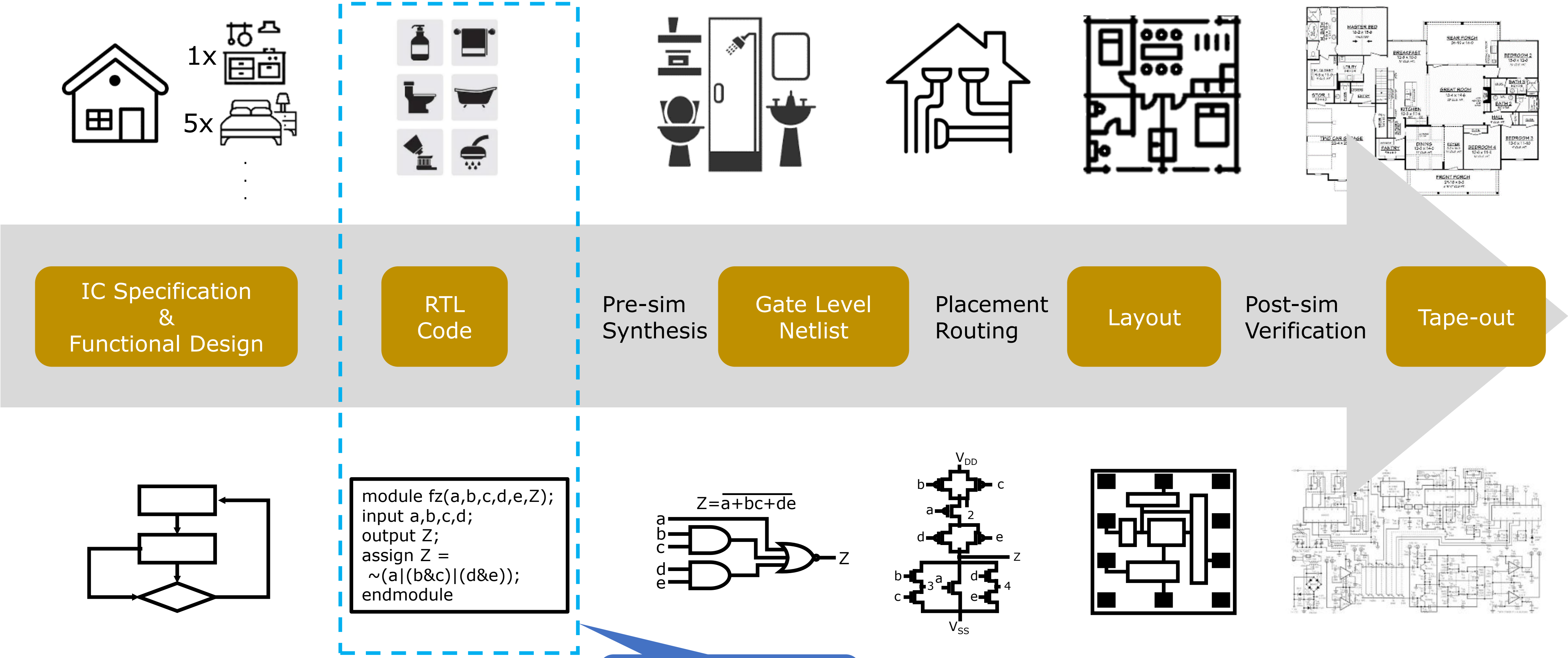
Design: Loaded, Lint Violations: 178

Fixes for violations are generated and displayed in the DiffWidget.

Final

IC Design Flow Illustration: House-Building

Focus of this session:
RTL code check / debug



IC Specification & Functional Design

RTL Code

Pre-sim Synthesis

Gate Level Netlist

Placement Routing

Layout

Post-sim Verification

Tape-out

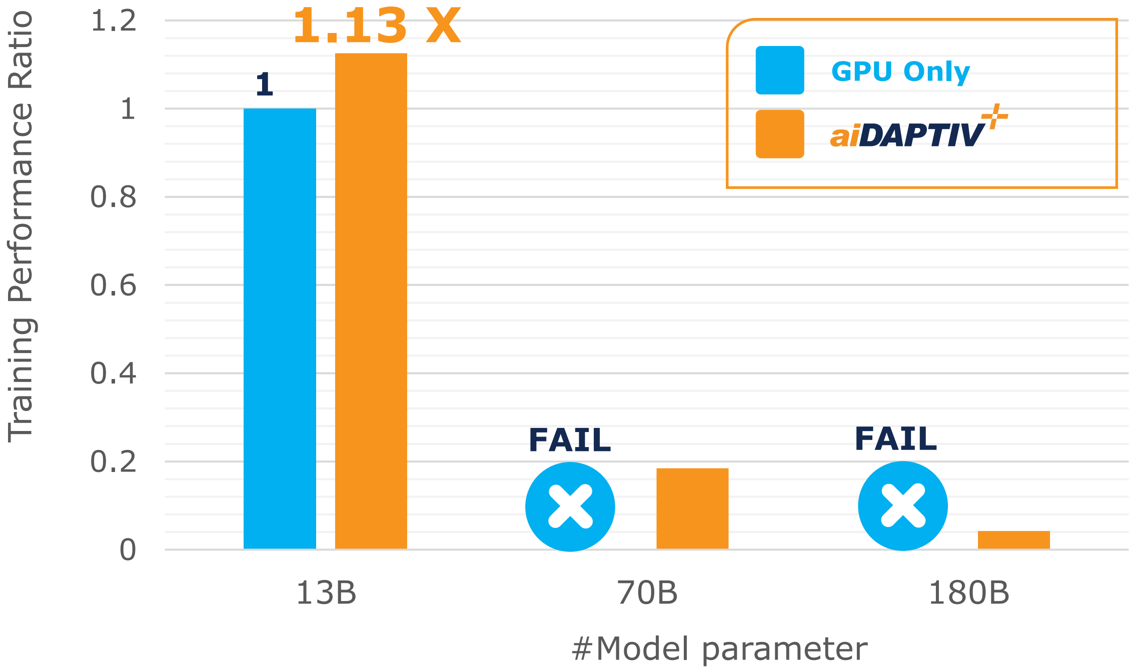
```
module fz(a,b,c,d,e,Z);
input a,b,c,d;
output Z;
assign Z =
 ~(a|(b&c)|(d&e));
endmodule
```

Use VC SpyGlass Lint to check RTL code

Performance Improvement with aiDAPTIV+

w/ aiDAPTIV+ AI Training Server

- The computing power and performance comparison for fine-tuning the LLAMA-2 13B/70B/ FALCON 180B model:



aiDAPTIV+

Make it possible for larger model training

&

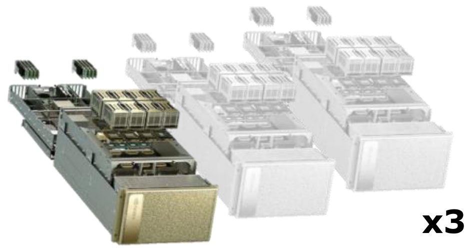
Enhances training performance in limit #GPU condition

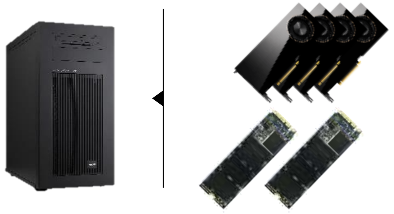
System Configuration

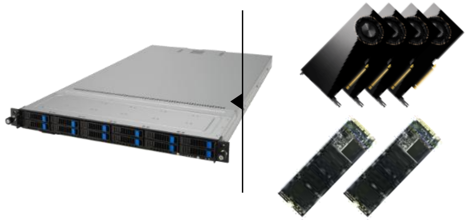
- CPU : Intel Xeon Gold6430
- GPU: **A6000*8**
- RAM: 64GB*16

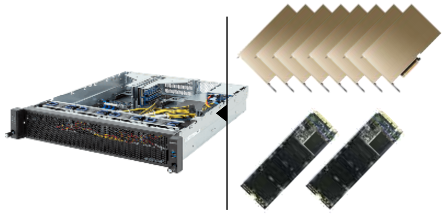
aiDAPTIV+ Benefit

Fine-tuning the **Llama 3.1 70B** model with **10M** tokens:









	DGX H100	Tower Workstation aiDAPTIV+ / 4000 Ada	Rack Workstation aiDAPTIV+ / A6000	Server aiDAPTIV+ / A6000
	• H100 GPU x24 • GPU Memory 1920GB • Without SSD	• 4000 Ada GPU x4 • GPU Memory 80GB • 2TB AI100E x2	• A6000 GPU x4 • GPU Memory 192GB • 2TB AI100E x2	• A6000 GPU x8 • GPU Memory 384GB • 2TB AI100E x2
Price (USD)	$1,920K	$40K ↓ 97.9%	$80K ↓ 95.8%	$100K ↓ 94.7%
Training Time (Hour)	0.3 H	7.6 H	4.3 H	2.2 H