

Architecturally Speaking, Are We Cool?

Effectiveness of Qualifying C/C++
Hardware Model and Simulator
Environment Using Certitude C++

A Gutmann
Analog Devices Inc.

September 11, 2014
SNUG Boston

SNUG Paper

Based on “Architecturally Speaking, Are We Cool?
Effectiveness of Qualifying C/C++ Hardware Model and
Simulator Environment Using Certitude C++”, paper by
A Gutmann, John Hayden, David Brownell, and Marty Rowe



Why are we here?

Architect and Design Engineers' Challenges

Verification Environment Is Difficult

Importance of Verification Effectiveness and Measuring
Verification methodologies in C/C++

Exploring Functional Qualification Certitude C++ and
Results



Advantage of Using Functional Qualification tool

Architect and Design Engineers' Challenge

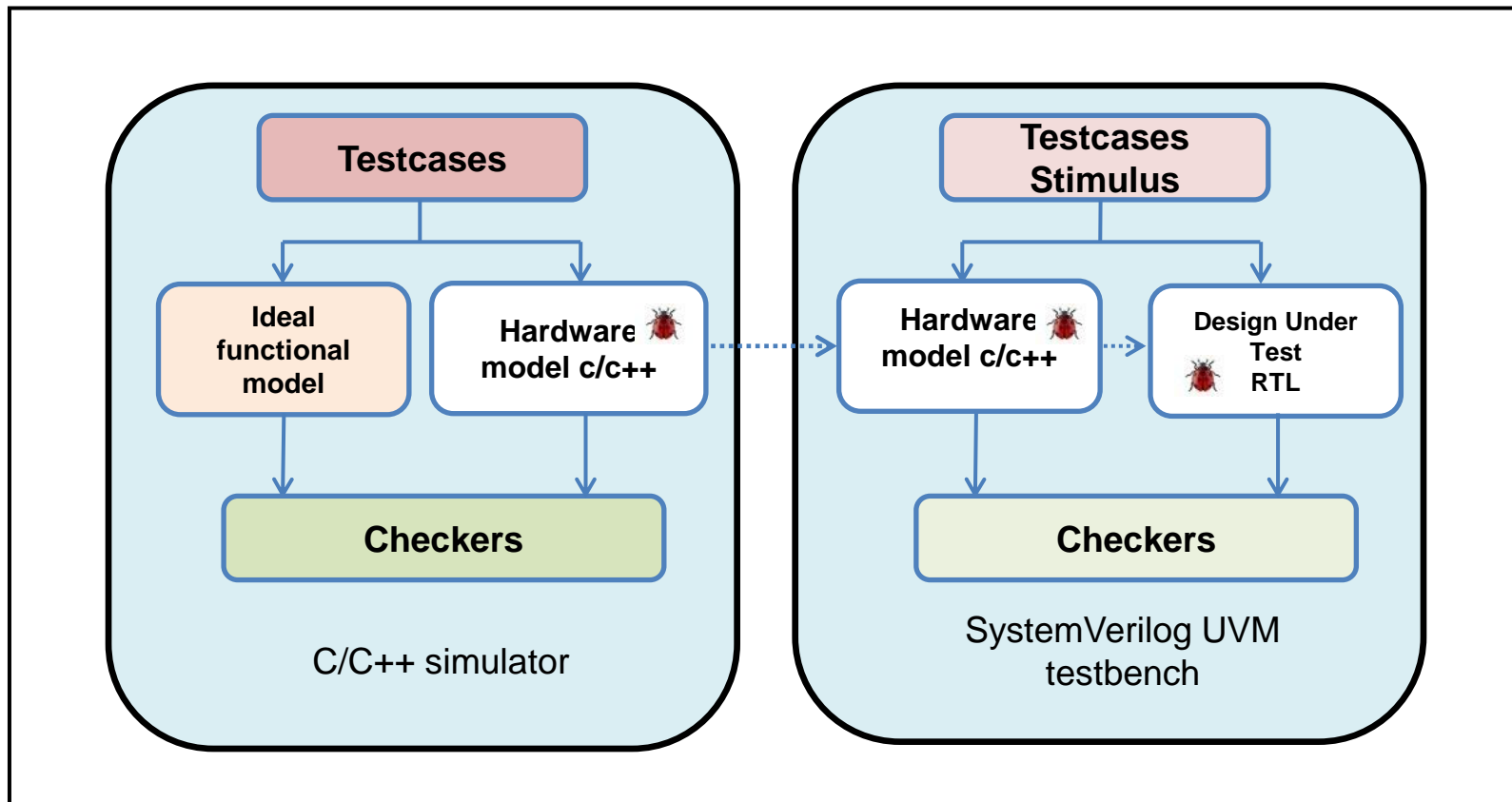


Importance of Verification Effectiveness and Measuring Verification methodologies in C/C++

- *Faster Tapeout*
- *Reduce number of respins.
(i.e. Eliminate bugs in our design)*
- *Better software and memory utilization.
(i.e. Better and more efficient coding style.)*
- *Good Quality in our software and system
verification before the Chip arrives*



Typical C/C++ Simulator vs. SystemVerilog UVM Testbench



Certitude C/C++

Certitude for RTL

Verification Environment Is Difficult

- C/C++ Simulator must cover all scenarios
- Every single functionality should be monitored, checked, and confirmed
- C/C++ testbench infrastructure reports all errors without any false positive
- Everything that is IMPORTANT should be checked!!!

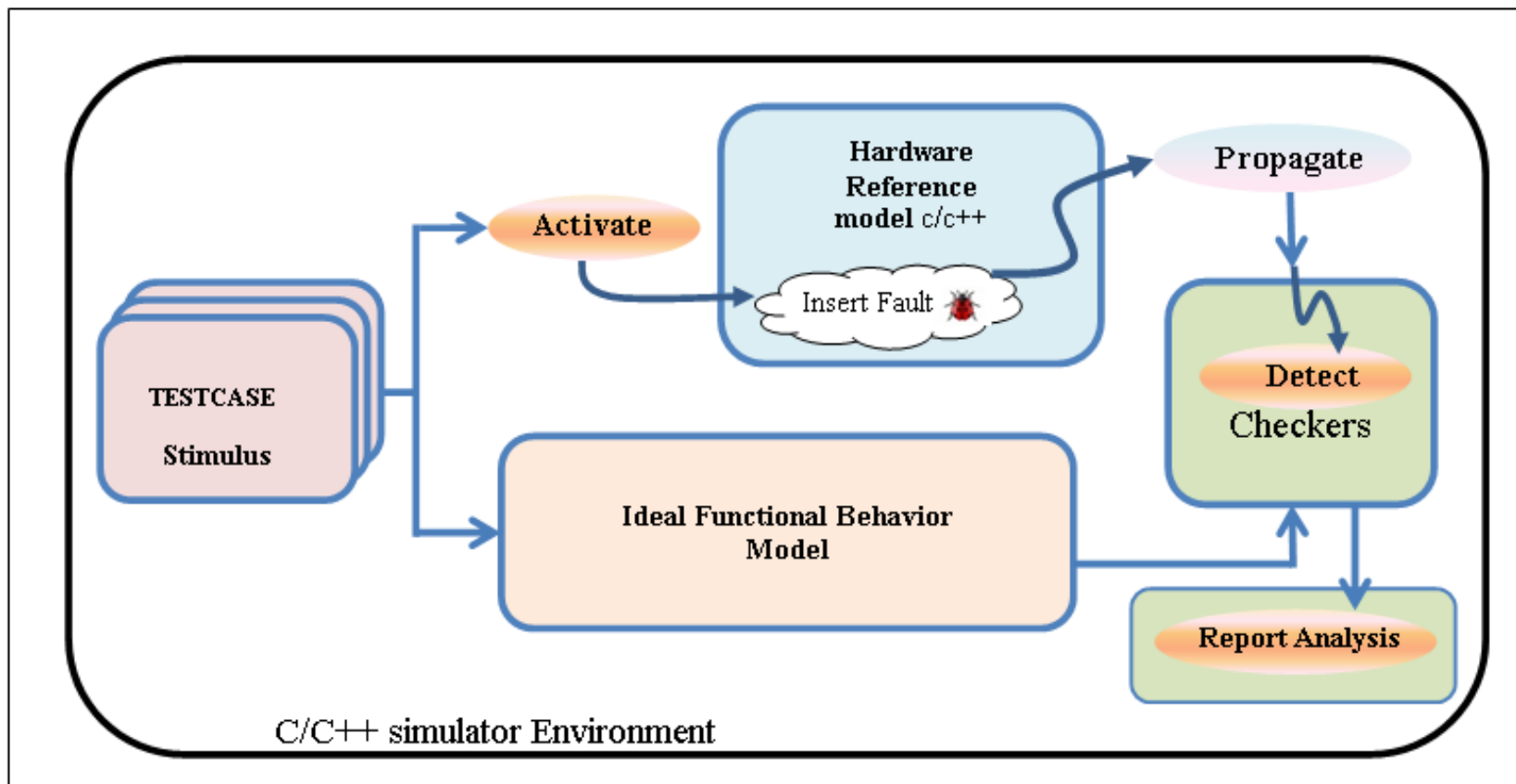


Effective Verification in C/C++

Detecting a bug, the C/C++ testbench must be able to:

- **Activate** the bug
- An effect of the bug must **Propagate** to an “output” of the program/model
- The C/C++ testbench infrastructure must **Detect** the behavior difference if bugs exist
- The C/C++ testbench must process and report failures properly

Qualification Steps with C/C++



*Effective Verification must be able to **activate**, **propagate**, and **detect** faults !!!*

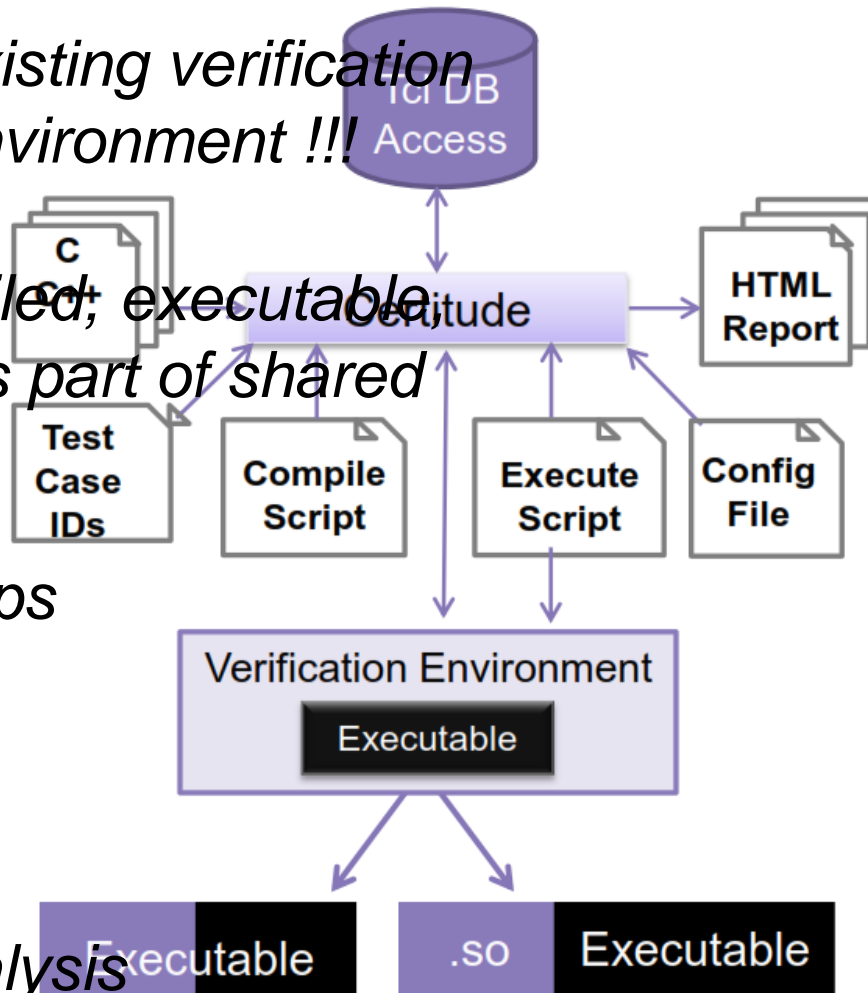
Certitude C/C++ Process and Flow Diagram

Leverages existing verification regression environment !!!

DUT is compiled, executable and linked as part of shared object

Certitude Steps

- Model
- Activate
- Detect
- Report analysis



How Fault Injection Works in Certitude C/C++?

- *Locally modified c/c++ code to insert fault*

```
a = b | c → a = b & c // fault changes operator
```

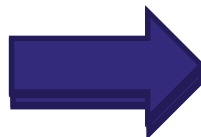
- *Simulates broken c/c++ code in verification environment*
 - Test failed == **GREAT!!!**
 - **If not, something is wrong in our c/c++ code or in our verification environment**

Types of Inserted Faults in C/C++

Fault Types	Fault Insertion
Condition Faults	Condition True, Condition False, Negated
Dead Faults	Dead Assign, Dead Call, Dead operator, Dead Assignment
Variable/Array Faults	ValueAtZero, ValueAtMax
Operator Faults	Swap Operators including prefix to postfix

Operation Fault:

`a = b + c;`



`a = b - c;`

Fault Classification

- **Non-Activated fault** : *No testcase capable of activating fault*
- **Non-Propagated (Weak)** : *Fault is activated, but no testcase has propagated it to detection phase.
E.g. an if-statement which is always false.*
- **Detected**: *At least one testcase has activated, propagated, and detected the fault*
- **Non-Detected**: *The fault has been activated but no testcase has detected it. This fault is propagated to the boundary of the output of hardware reference.*
- **Not Yet Qualified**: *The fault has not been fully processed yet.*

Certitude C/C++

Example of Fault Injection for a single line of code

1st : *If (0) { C += 8; D >>1; }*

2nd: *If (1) { C += 8; D >>1; }*

3rd: *If (!~~(A == 0 && B > -2)~~) { C += 8; D >>1; }*

4th: *If (A != 0 && B > -2) { C += 8; D >>1; }*

5th: *If (A == 0 // B > -2) { C += 8; D >>1; }*

6th: *If (A == 0 && B < -2) { C += 8; D >>1; }*

7th: *If (A == 0 && B >= -2) { C += 8; D >>1; }*

Certitude C/C++

Example of Fault Injection

`If (A == 0 && B > -2) { C += 8; D >> 1; }`

8th: If (A == 0 && B > +2) { C += 8; D >> 1; }

9th: If (A == 0 && B > -2) { C; D >> 1; }

10th: If (A == 0 && B > -2) { C -= 8; D >> 1; }

11th: If (A == 0 && B > -2) { C += 8; D << 1; }

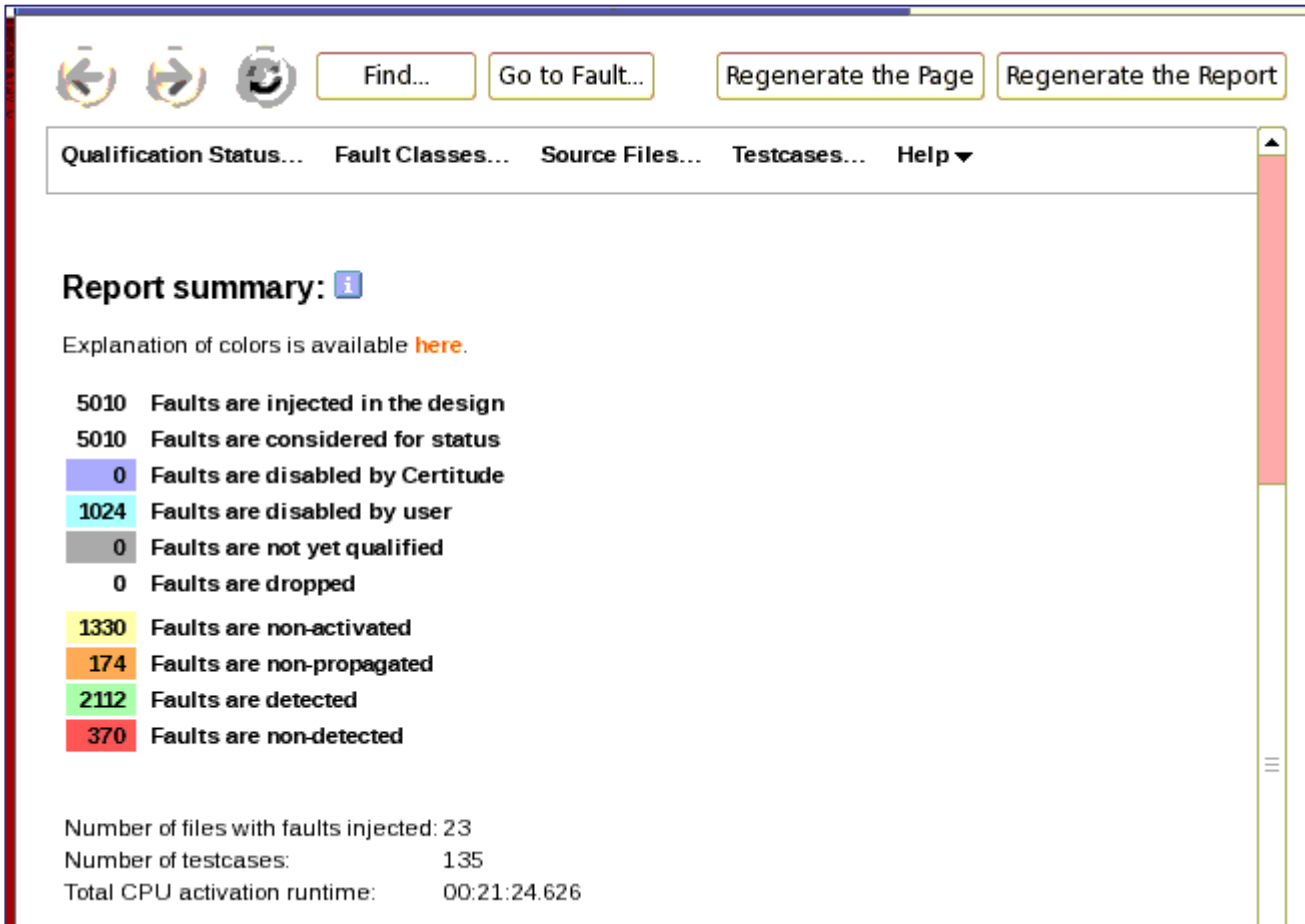
Certitude C/C++

Example of Fault Injection

```
function_helloWorld();
```

Fault injection: /* code removed */ // function_helloWorld()

Certitude C/C++ Internal HTML report example : Summary



Qualification Status... Fault Classes... Source Files... Testcases... Help ▼





Report summary: ⓘ

Explanation of colors is available [here](#).

5010	Faults are injected in the design
5010	Faults are considered for status
0	Faults are disabled by Certitude
1024	Faults are disabled by user
0	Faults are not yet qualified
0	Faults are dropped
1330	Faults are non-activated
174	Faults are non-propagated
2112	Faults are detected
370	Faults are non-detected

Number of files with faults injected: 23
 Number of testcases: 135
 Total CPU activation runtime: 00:21:24.626

Fault Type Color coded report

-  **Yellow** represents non-activated faults.
-  **Orange** represents non-propagated (weak) faults.
-  **Green** represents detected faults.
-  **Red** represents non-detected faults.

Certitude C/C++ Testcase Report

Testcase statistics for 'DSP Accelerator'

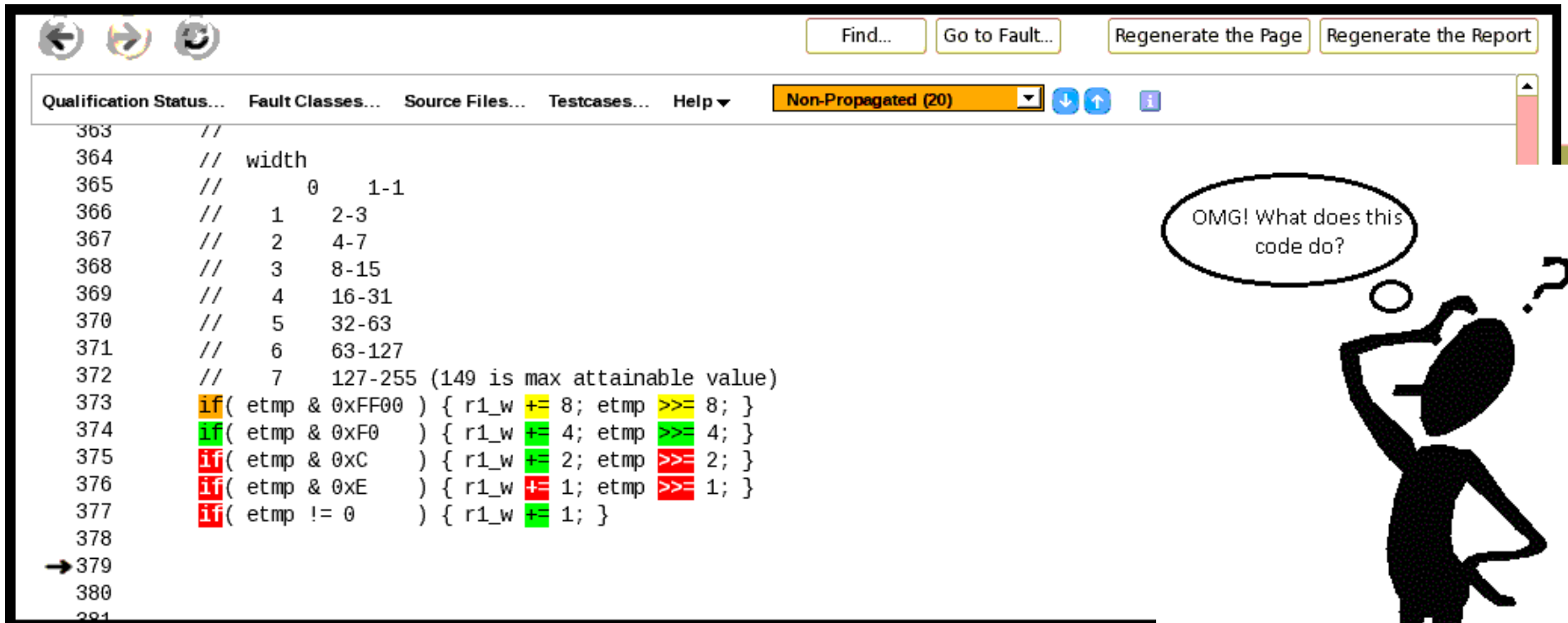
All durations are expressed using hh:mm:ss.milliseconds

This report was generated on: 2014-08-15 at 17:06:47

Group ▾	Testcase ▾	Id ▾	Runtime ▾	Activated Faults ▾	# Run for Detection ▾	Propagated Faults ▾	Detected Faults ▾
All Testcases (135)			00:21:24.626	2656	18945	0	2112
	dsp_acc001	1	00:00:00.508	453	106	0	0
	dsp_acc002	2	00:00:00.553	465	105	0	0
	dsp_acc003	3	00:00:00.764	456	101	0	0
	dsp_acc004	4	00:00:00.653	443	104	0	0
	dsp_acc005	5	00:00:00.848	537	101	0	0
	dsp_acc006	6	00:00:01.104	534	101	0	0

Certitude C/C++ Source Files Report

Source file statistics for 'DSP Accelerator'



Qualification Status... Fault Classes... Source Files... Testcases... Help ▼ Non-Propagated (20) ↓ ↑ i

```
363 //  
364 // width  
365 // 0 1-1  
366 // 1 2-3  
367 // 2 4-7  
368 // 3 8-15  
369 // 4 16-31  
370 // 5 32-63  
371 // 6 63-127  
372 // 7 127-255 (149 is max attainable value)  
373 if( etmp & 0xFF00 ) { r1_w += 8; etmp >>= 8; }  
374 if( etmp & 0xF0 ) { r1_w += 4; etmp >>= 4; }  
375 if( etmp & 0xC ) { r1_w += 2; etmp >>= 2; }  
376 if( etmp & 0xE ) { r1_w += 1; etmp >>= 1; }  
377 if( etmp != 0 ) { r1_w += 1; }  
378  
→ 379  
380  
381
```

OMG! What does this code do?

Certitude C/C++ Source Files Report

Qualification Status... Fault Classes... Source Files... Testcases... Help ▾ Non-Detected (61) ↓ ↑ ⓘ cmode

```
110
111 void tb_abort( void ) {
112     fprintf(stderr, "Testbench Abort");
113     if(abortCase == NULL) {
114         fprintf(stderr, ", abortCase ptr = NULL\n");
115     } else {
116         if( isTwoArgOp( abortCase->argstyle ) ) {
117             fprintf(stderr, ", case x,y = %16.8g,%16.8g = %.7a,%.7a",
118                 abortCase->x, abortCase->y, abortCase->x, abortCase->y );
119         } else {
120             fprintf(stderr, ", case x = %16.8g = %16.7a", abortCase->x, abortCase->x );
121         }
122         fprintf(stderr, "\n");
123     }
124 }
125 printf("\n*** RESULT: TEST ABORT ***\n");
126 tb_fflush();
127 raise(SIGABRT); // call the debugger
128 exit(1);        /* never get here */
129 }
130
```

Experience with Functional Qualification Certitude C/C++ and Results

Certitude C/C++ Easy Integration

Setting up simulator environment in Certitude C/C++ was approximately 2 hours by updating the following files, compile, and run.

- ***certitude_config.cer*** (setup config)
- ***certitude_compile*** (environment compilation step)
- ***certitude_batch*** (lsf job command)
- ***certitude_execute*** (script capturing “pass” or “fail”)
- ***certitude_testcases.cer*** (testcase names)
- ***certitude_hdl_files.cer*** (reference model and waivers)
- ***cert_testlist*** (testcase list).

******Certitude C/C++ License was required***



Simulator Environment Qualified Statistic

- *16 different algorithms*
- *Bit-accurate Hardware Reference model*

Source Code	Line Count
C/C++ Hardware Reference Model	9658
C/C++ Ideal Reference Model and Testbench	9,642
RTL in SystemVerilog	12,808
SystemVerilog UVM Testbench	17,313

Runtime and Data Volume

$$\text{Activate Runtime} = \frac{\#tests * test_time}{\text{Number of Servers}}$$

$$\text{Detect Runtime} = \frac{\# faults * \#tests * test_time}{\text{Number of servers}}$$

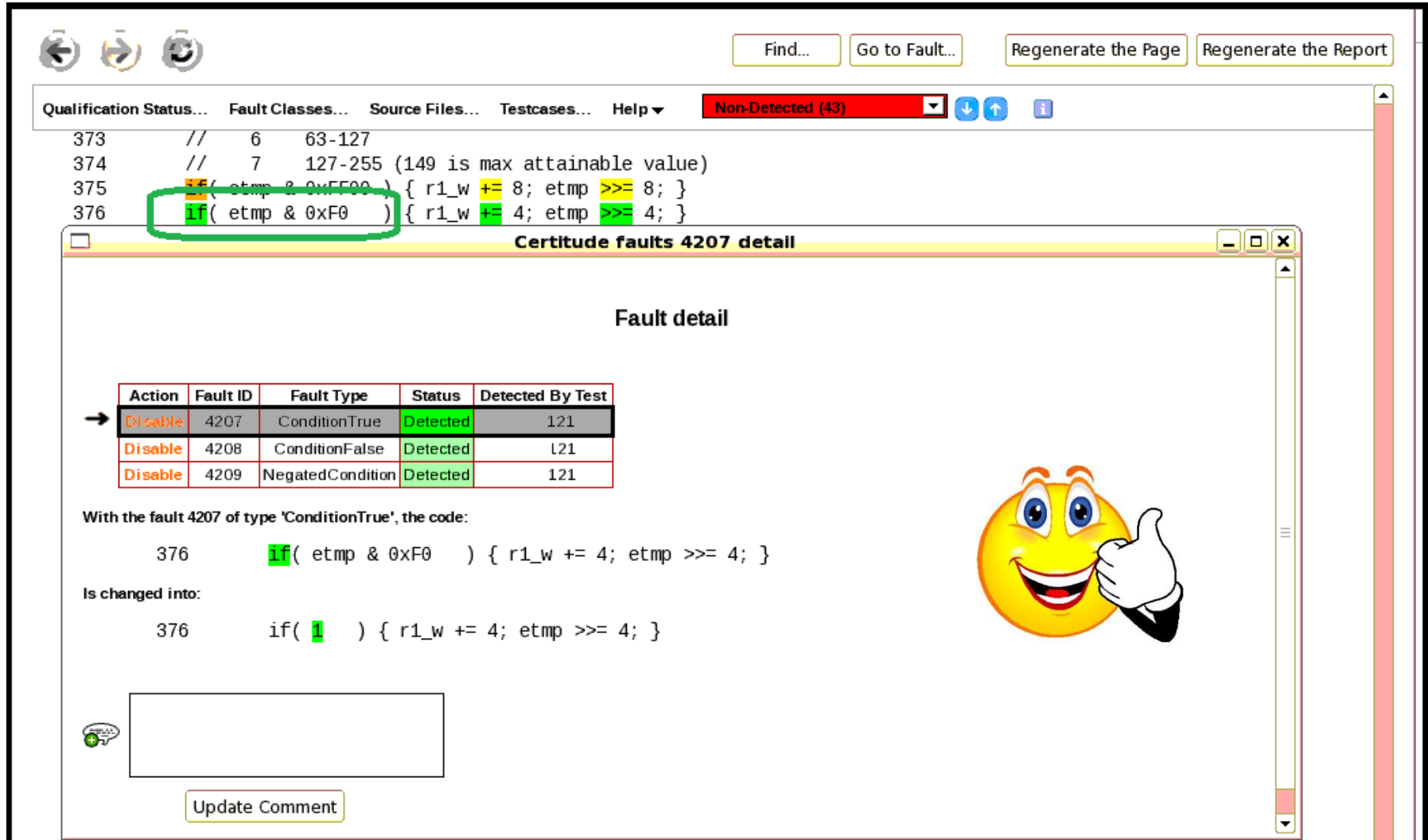
Phase	Short Regression Runtime
Model	1 minute
Activate	45 minutes
Detect	21 hours

Objective Goal and Certitude C/C++ Result

- *Plan ahead:*
 - When to run Certitude C/C++?
 - Who should run it?
- *Resource: Runtime, disk volume, and Result Analysis*
 - Analyzing your result... Prioritize to use your time wisely
 - Runtime and disk volume can be large
- *Level of severities matter*
 - Non-Detect faults are the highest priority
 - Non-Activated faults are the next high priority



Certitude C/C++ Report example ... Detected Fault



The screenshot shows the Certitude C/C++ Report interface. At the top, there are navigation buttons: Find..., Go to Fault..., Regenerate the Page, and Regenerate the Report. Below these is a toolbar with icons for back, forward, and search. The main panel displays a list of source files and testcases. A red bar indicates 'Non-Detected (43)'. The code snippet shows a fault detected at line 376: `if(etmp & 0xF0) { r1_w += 4; etmp >>= 4; }`. A green box highlights the `if` statement. Below the code, a window titled 'Certitude faults 4207 detail' is open, showing the 'Fault detail' section. It contains a table with columns: Action, Fault ID, Fault Type, Status, and Detected By Test. The table lists three faults: 4207 (ConditionTrue, Detected, 121), 4208 (ConditionFalse, Detected, 121), and 4209 (NegatedCondition, Detected, 121). Below the table, it states: 'With the fault 4207 of type 'ConditionTrue', the code:'. The code snippet is: `376 if(etmp & 0xF0) { r1_w += 4; etmp >>= 4; }`. Below this, it says 'Is changed into:' and shows the modified code: `376 if(1) { r1_w += 4; etmp >>= 4; }`. A yellow smiley face with a thumbs up is next to the modified code. At the bottom, there is a text box for comments and an 'Update Comment' button.

Action	Fault ID	Fault Type	Status	Detected By Test
Disable	4207	ConditionTrue	Detected	121
Disable	4208	ConditionFalse	Detected	121
Disable	4209	NegatedCondition	Detected	121

With the fault 4207 of type 'ConditionTrue', the code:

```
376 if( etmp & 0xF0 ) { r1_w += 4; etmp >>= 4; }
```

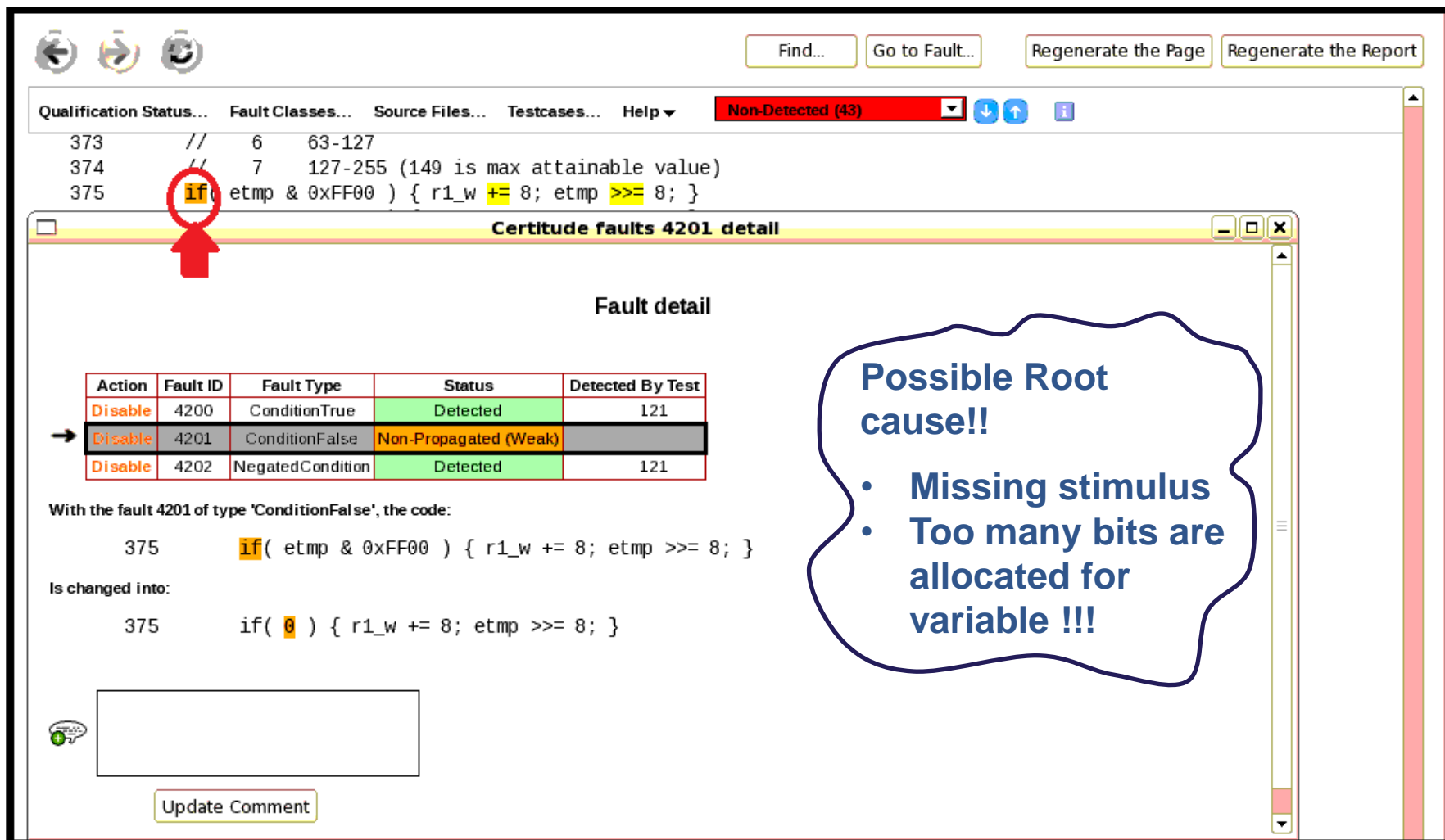
Is changed into:

```
376 if( 1 ) { r1_w += 4; etmp >>= 4; }
```

Update Comment

Certitude C/C++ Report example...

Non-Propagated Fault (Weak)



The screenshot shows the Certitude C/C++ Report interface. At the top, there are navigation buttons: Find..., Go to Fault..., Regenerate the Page, and Regenerate the Report. Below these is a menu bar with options: Qualification Status..., Fault Classes..., Source Files..., Testcases..., and Help. A dropdown menu shows 'Non-Detected (43)'.

The main content area displays a list of faults. The first fault is highlighted with a red circle and a red arrow pointing to it. The fault is identified as 'if(etmp & 0xFF00) { r1_w += 8; etmp >>= 8; }'.

Below the list, a window titled 'Certitude faults 4201 detail' is open, showing the 'Fault detail' for fault ID 4201. The detail is as follows:

Action	Fault ID	Fault Type	Status	Detected By Test
Disable	4200	ConditionTrue	Detected	121
→ Disable	4201	ConditionFalse	Non-Propagated (Weak)	
Disable	4202	NegatedCondition	Detected	121

With the fault 4201 of type 'ConditionFalse', the code:

```
375 if( etmp & 0xFF00 ) { r1_w += 8; etmp >>= 8; }
```

Is changed into:

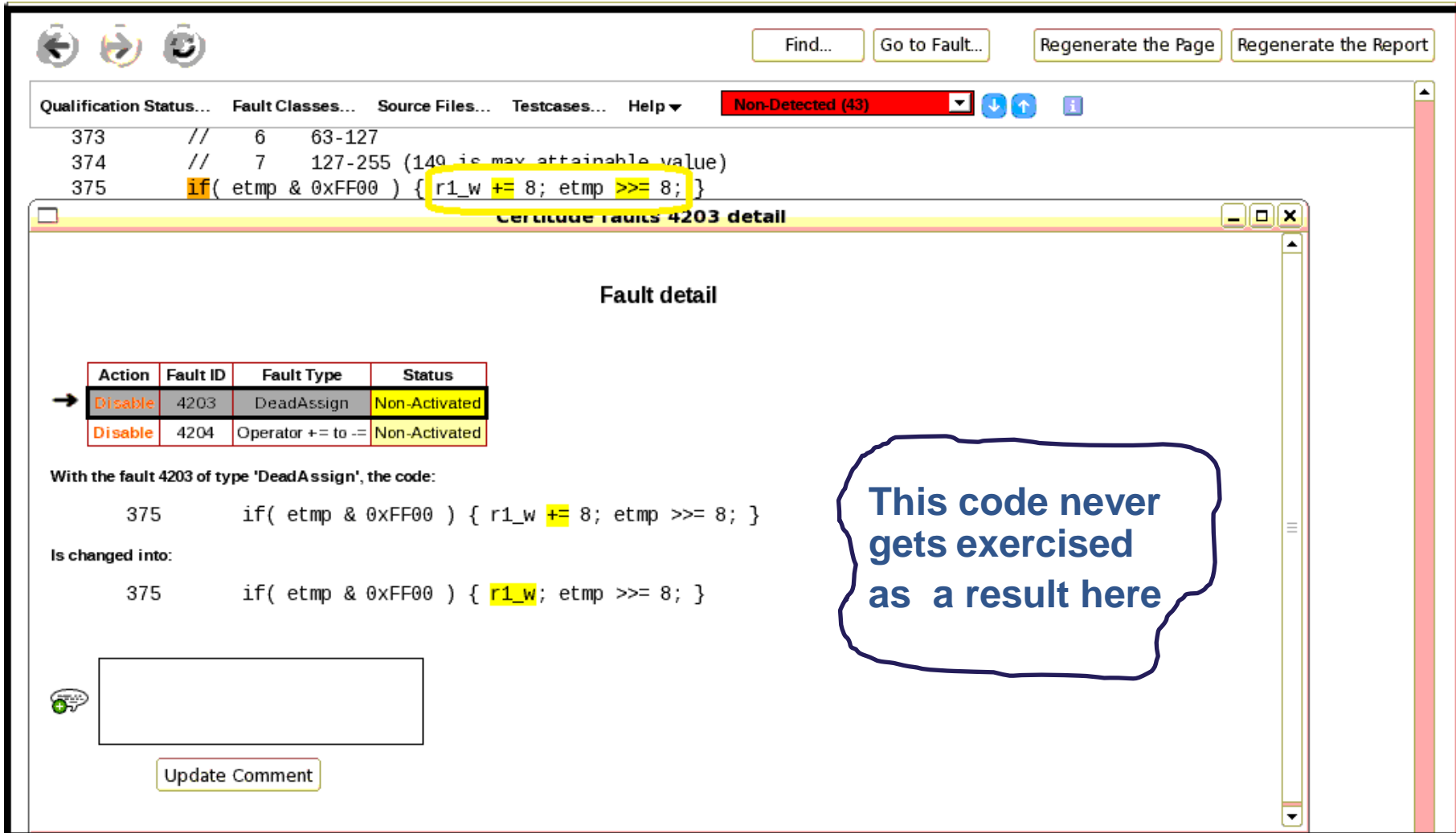
```
375 if( 0 ) { r1_w += 8; etmp >>= 8; }
```

At the bottom, there is a text box for comments and an 'Update Comment' button.

Possible Root cause!!

- Missing stimulus
- Too many bits are allocated for variable !!!

Certitude C/C++ Report example... Non-Activated Fault



The screenshot shows the Certitude C/C++ Report interface. At the top, there are navigation buttons: Find..., Go to Fault..., Regenerate the Page, and Regenerate the Report. Below these is a menu bar with options: Qualification Status..., Fault Classes..., Source Files..., Testcases..., and Help. A dropdown menu shows 'Non-Detected (43)'.

The main content area displays a list of faults. The following code snippet is highlighted:

```
373 // 6 63-127
374 // 7 127-255 (149 is max attainable value)
375 if( etmp & 0xFF00 ) { r1_w += 8; etmp >>= 8; }
```

The fault detail window is open, showing the following table:

Action	Fault ID	Fault Type	Status
Disable	4203	DeadAssign	Non-Activated
Disable	4204	Operator += to -=	Non-Activated

With the fault 4203 of type 'DeadAssign', the code:

```
375 if( etmp & 0xFF00 ) { r1_w += 8; etmp >>= 8; }
```

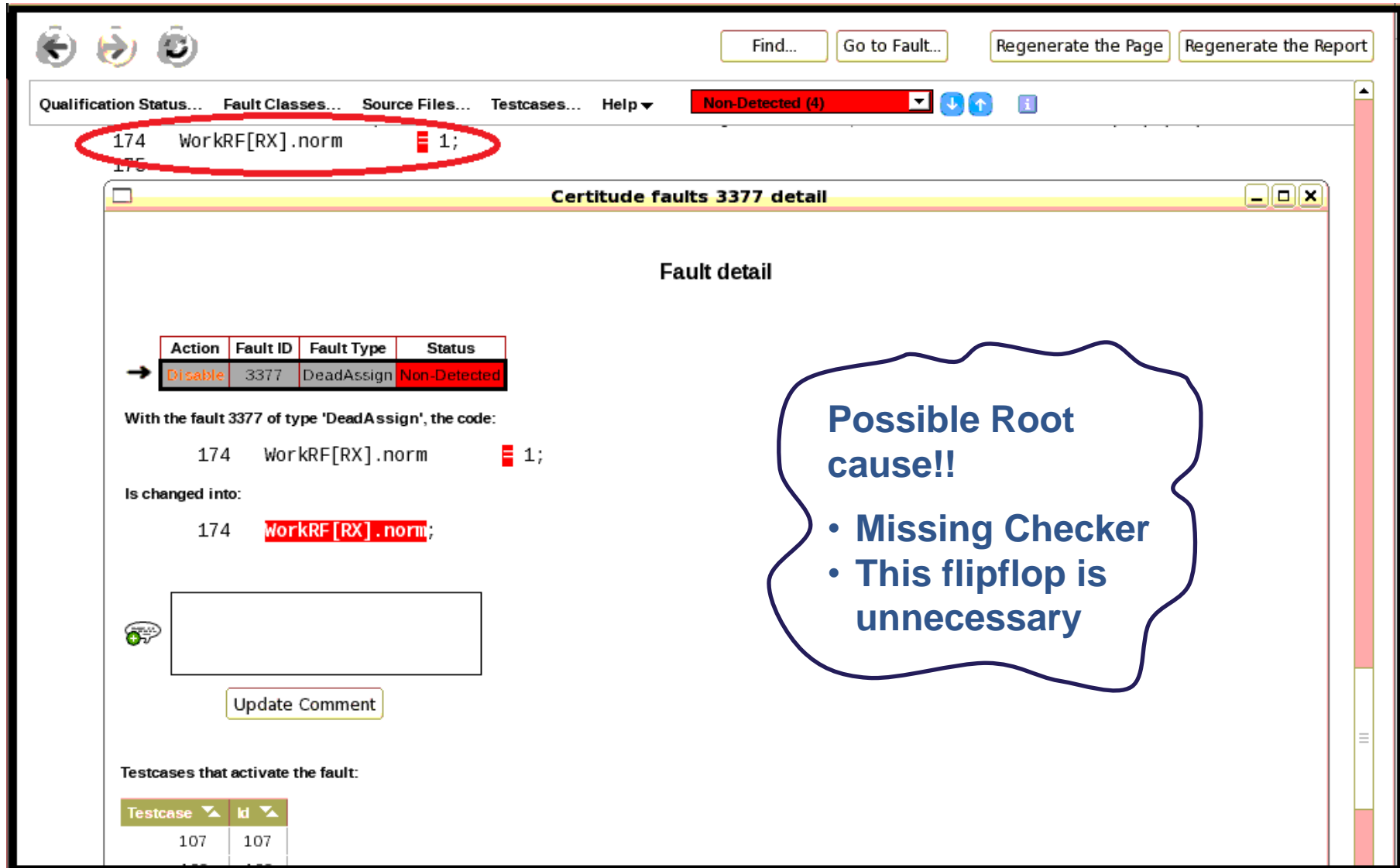
Is changed into:

```
375 if( etmp & 0xFF00 ) { r1_w; etmp >>= 8; }
```

A text box with a green plus icon and the text 'Update Comment' is visible at the bottom left.

A handwritten note in a blue bubble states: "This code never gets exercised as a result here".

Certitude C/C++ Report example... Non-Detected Fault



The screenshot shows the Certitude C/C++ Report interface. At the top, there are navigation buttons: Find..., Go to Fault..., Regenerate the Page, and Regenerate the Report. Below these is a menu bar with options: Qualification Status..., Fault Classes..., Source Files..., Testcases..., and Help. A red bar indicates 'Non-Detected (4)' faults. The main content area shows a list of faults, with one fault highlighted: 174 WorkRF[RX].norm 1;.

Certitude faults 3377 detail

Fault detail

Action	Fault ID	Fault Type	Status
→ Disable	3377	DeadAssign	Non-Detected

With the fault 3377 of type 'DeadAssign', the code:

```
174 WorkRF[RX].norm 1;
```

Is changed into:

```
174 WorkRF[RX].norm;
```

Update Comment

Testcases that activate the fault:

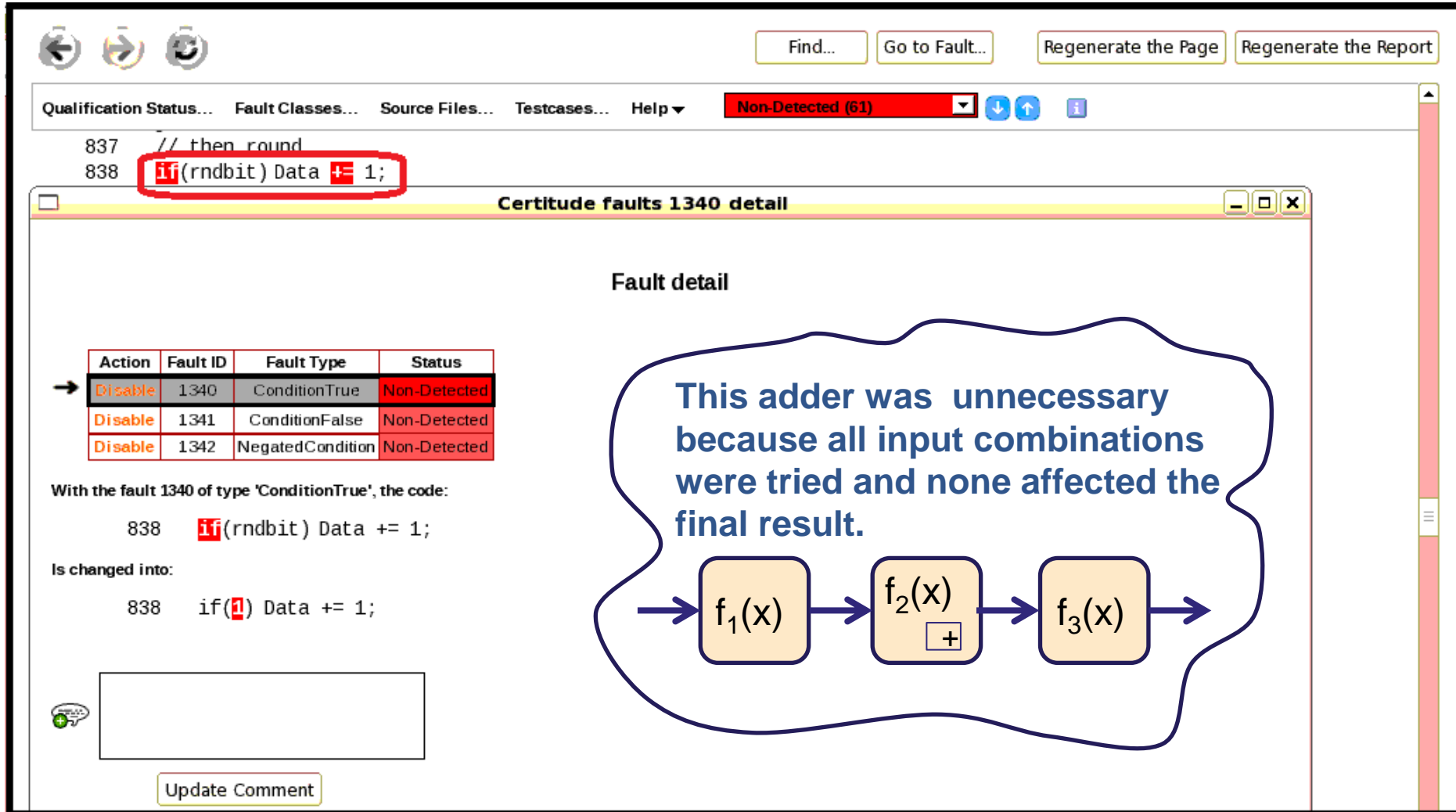
Testcase	Id
107	107

Possible Root cause!!

- Missing Checker
- This flipflop is unnecessary

Certitude C/C++ Report example...

Non-Detected Fault



The screenshot shows the Certitude C/C++ Report interface. At the top, there are navigation buttons: Find..., Go to Fault..., Regenerate the Page, and Regenerate the Report. Below these is a toolbar with icons for back, forward, and search. The main menu includes Qualification Status..., Fault Classes..., Source Files..., Testcases..., and Help. A dropdown menu shows 'Non-Detected (61)'. The code editor displays two lines of code: 837 // then round and 838 if(rndbit) Data += 1;. The 'if' statement is highlighted with a red box. A window titled 'Certitude faults 1340 detail' is open, showing the fault detail. The fault detail table lists three faults: 1340 (ConditionTrue, Non-Detected), 1341 (ConditionFalse, Non-Detected), and 1342 (NegatedCondition, Non-Detected). The first row is highlighted. Below the table, it states: 'With the fault 1340 of type 'ConditionTrue', the code: 838 if(rndbit) Data += 1; Is changed into: 838 if(1) Data += 1;'. A diagram shows a flow from $f_1(x)$ to $f_2(x)$ to $f_3(x)$. The $f_2(x)$ block contains a '+' sign, indicating an adder. A blue cloud-shaped callout points to the diagram with the text: 'This adder was unnecessary because all input combinations were tried and none affected the final result.'

Qualification Status... Fault Classes... Source Files... Testcases... Help ▾ Non-Detected (61)

837 // then round
838 if(rndbit) Data += 1;

Certitude faults 1340 detail

Fault detail

Action	Fault ID	Fault Type	Status
Disable	1340	ConditionTrue	Non-Detected
Disable	1341	ConditionFalse	Non-Detected
Disable	1342	NegatedCondition	Non-Detected

With the fault 1340 of type 'ConditionTrue', the code:

```
838 if(rndbit) Data += 1;
```

Is changed into:

```
838 if(1) Data += 1;
```

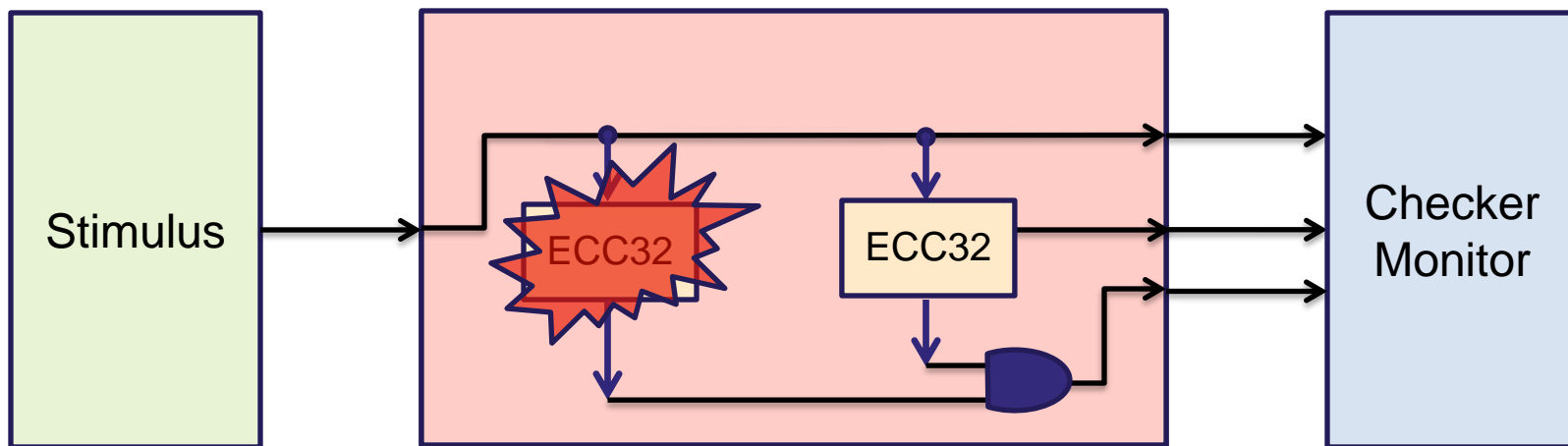
This adder was unnecessary because all input combinations were tried and none affected the final result.

$f_1(x) \rightarrow f_2(x) \rightarrow f_3(x)$

Update Comment

Another example of Certitude C/C++ Finding Design issues

A hardware reference model or software code:
Alarm2 = ECC32_calc(input, corrected_data2);
Alarm1 = ECC32_calc(input, corrected_data1);
IRQ = Alarm1 && Alarm2;
Output = corrected_data1;



If you run Certitude C/C++, You will find this. **X**

Advantage of Using Functional Qualification tool

The Good, The Bad, and The Ugly



*Sometimes the unexpected happens,
and we are human after all !!!*

The Bad and The Ugly :

BE SMART....

(Don't start the engine and throw the key away!!)

The Good!!!!

That is why we are here...

- You will find (many) things that aren't perfect in your design and verification environment
 - Low Power design
 - Software memory critical code/program
- Detect bad coding style or deadcode that weren't intended to be in the design
- You can get a good list of tests for Short Regression. Critical for design change, feature adding, and a quick respin.



Why should you run Certitude C/C++ too?

- *You can walk away feeling more confident as your design code is thoroughly interrogated and examined.*
 - “There is a checker for every single line of code or logic”
- *Improved C/C++ code results in higher quality RTL or software*
- *Objectively measure critical software code or Hardware model as well as your verification environment*
- *Find bugs earlier rather than waiting for device to come back*

Recommendation for Future Certitude C/C++ features



Acknowledgement

“Who We Should Thank!!!”

*John Hayden, David Brownell, Marty Rowe, ADI Norwood
and IPDC team, Synopsys, and Technical committees*



Always Have Fun !!

Thank You

Get It Done !!