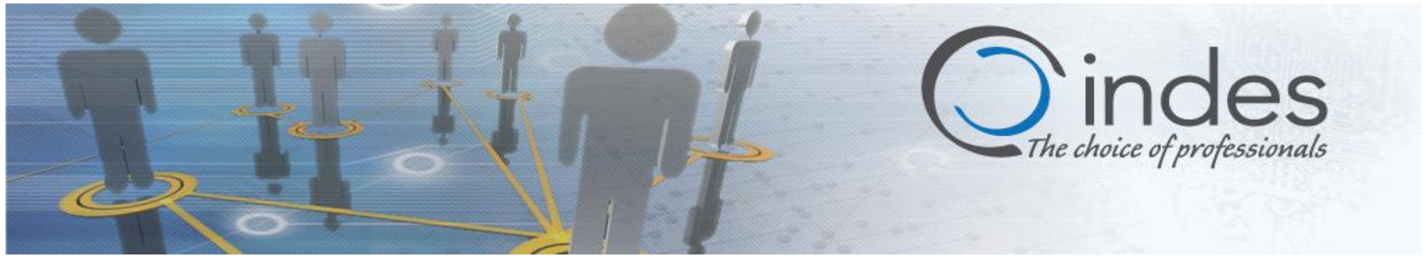
Testen van Embedded Systemen

Gerard Fianen



INDES-IDS BV - Embedded Software Development

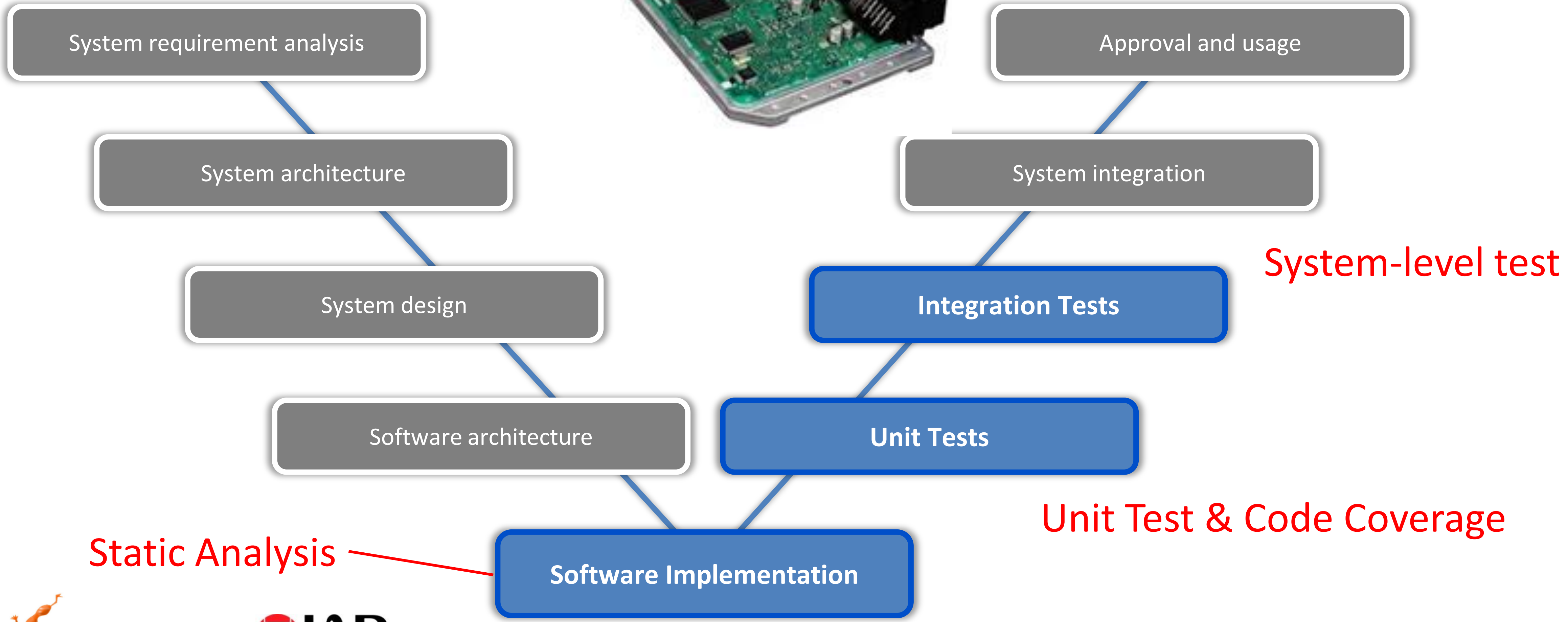
"The choice of professionals"

info@indes.com

www.indes.com/embedded

Tel: 0345 - 545.535

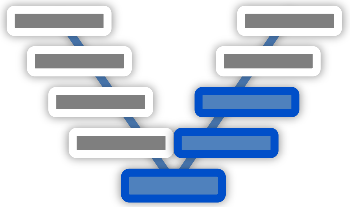
Test : The V-Model

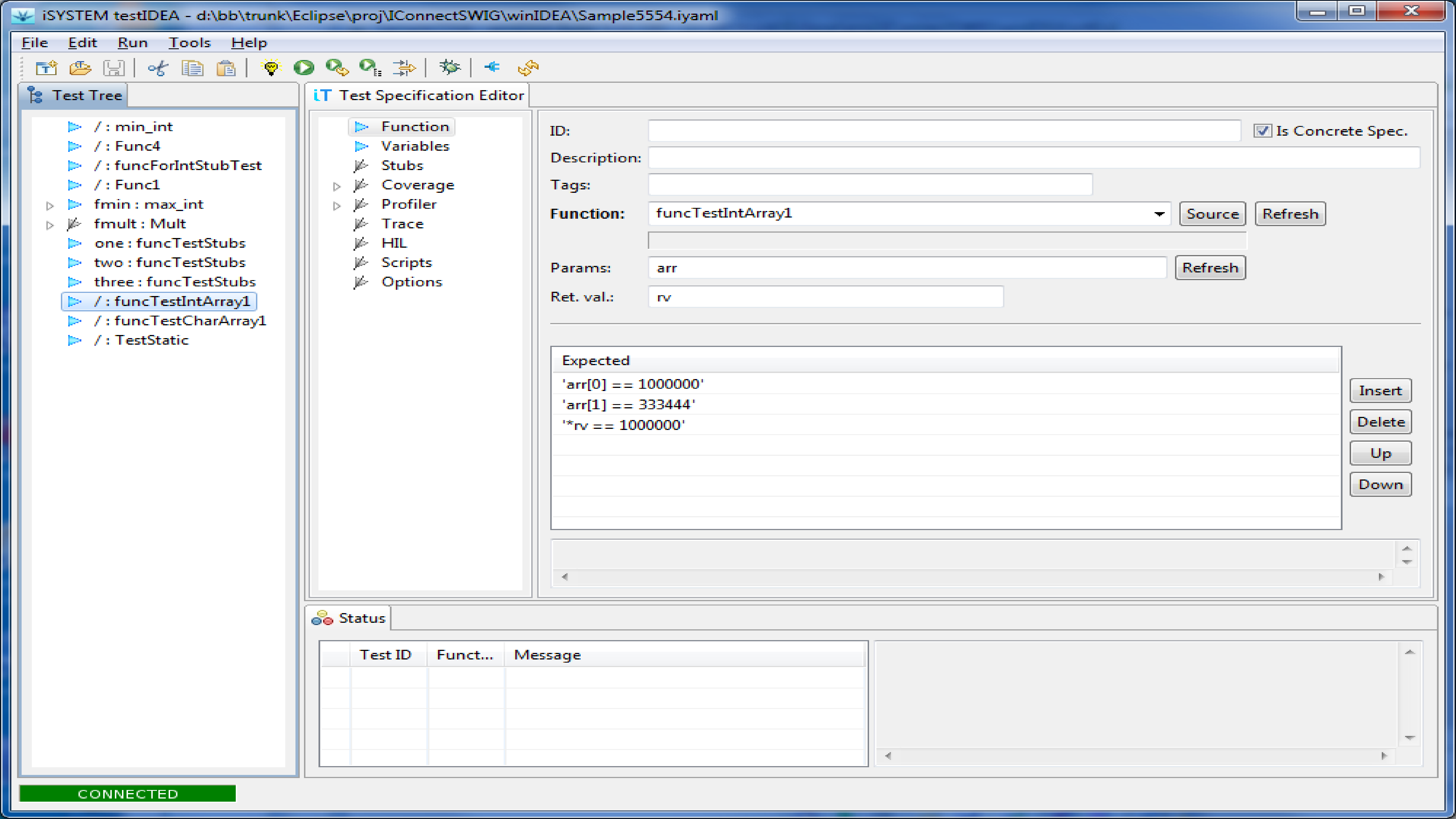


goanna IAR SYSTEMS

Unit Test & Code Coverage

- What is Unit Test ?





iSYSTEM testIDEA - d:\bb\trunk\Eclipse\proj\IConnectSWIG\winIDEA\Sample5554.iyaml

Test Tree:

- / : min_int
- / : Func4
- / : funcForIntStubTest
- / : Func1
- fmin : max_int
- fmult : Mult
- one : funcTestStubs
- two : funcTestStubs
- three : funcTestStubs
- / : funcTestIntArray1
- / : funcTestCharArray1
- / : TestStatic

Test Specification Editor sections:

- Function
- Variables
- Stubs
- Coverage
- Profiler
- Trace
- HIL
- Scripts
- Options

ID:

☑ Is Concrete Spec.

Description:

Tags:

Function: funcTestIntArray1 [Source] [Refresh]

Params: arr [Refresh]

Ret. val.: rv

Expected
'arr[0] == 1000000'
'arr[1] == 333444'
'*rv == 1000000'

[Insert] [Delete] [Up] [Down]

	Test ID	Funct...	Message

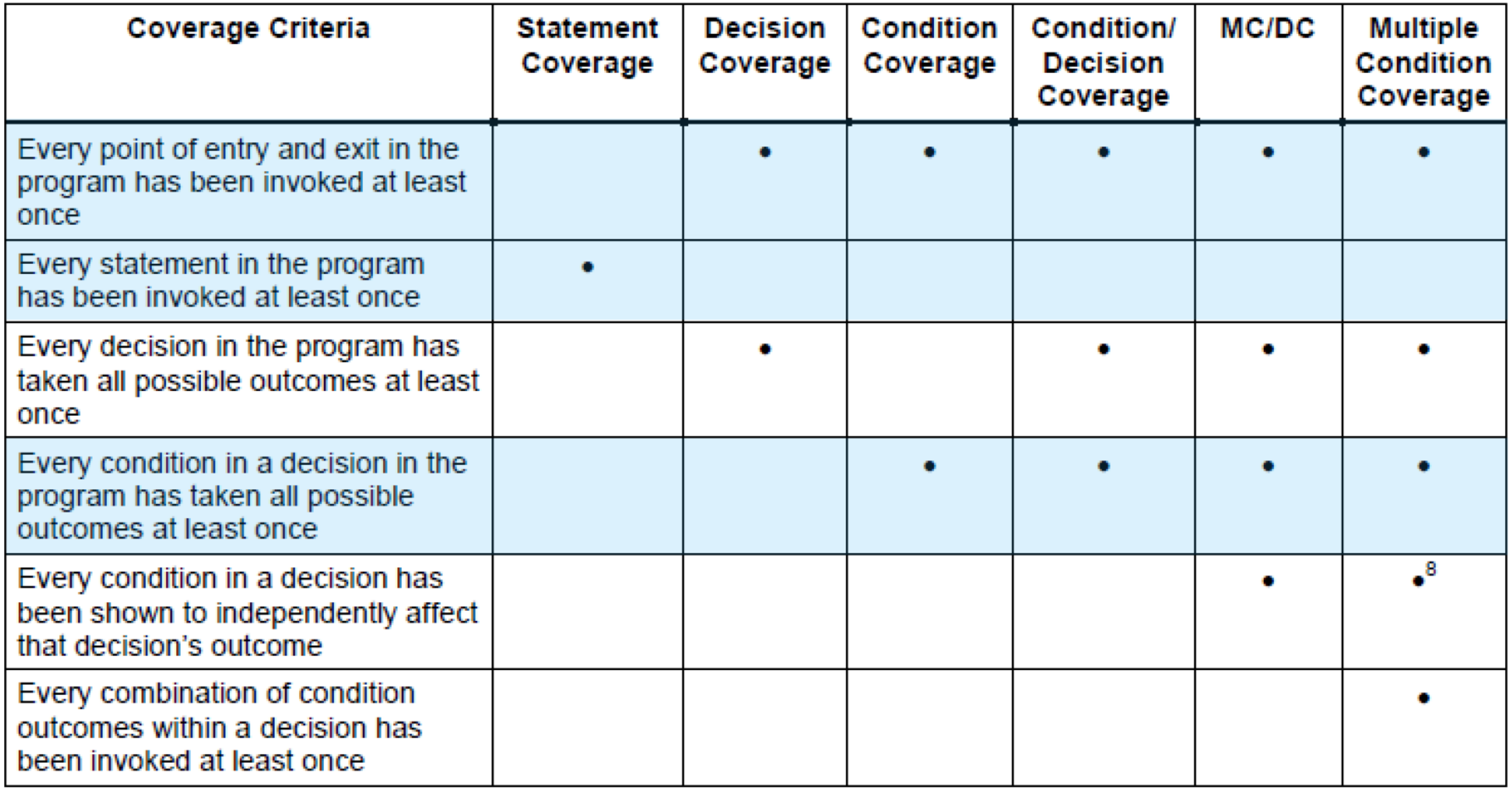
CONNECTED

Unit Test & Code Coverage

- What is Code Coverage ?
  - SC (Statement Coverage)
  - BC / DC (Branch / Decision Coverage)
  - CC (Condition Coverage)
  - MCDC (Modified Control Decision Coverage)

- Build an automatic regression Testsuite

Unit Test & Code Coverage



Coverage Criteria	Statement Coverage	Decision Coverage	Condition Coverage	Condition/ Decision Coverage	MC/DC	Multiple Condition Coverage
Every point of entry and exit in the program has been invoked at least once		•	•	•	•	•
Every statement in the program has been invoked at least once	•					
Every decision in the program has taken all possible outcomes at least once		•		•	•	•
Every condition in a decision in the program has taken all possible outcomes at least once			•	•	•	•
Every condition in a decision has been shown to independently affect that decision's outcome					•	•[8]
Every combination of condition outcomes within a decision has been invoked at least once						•

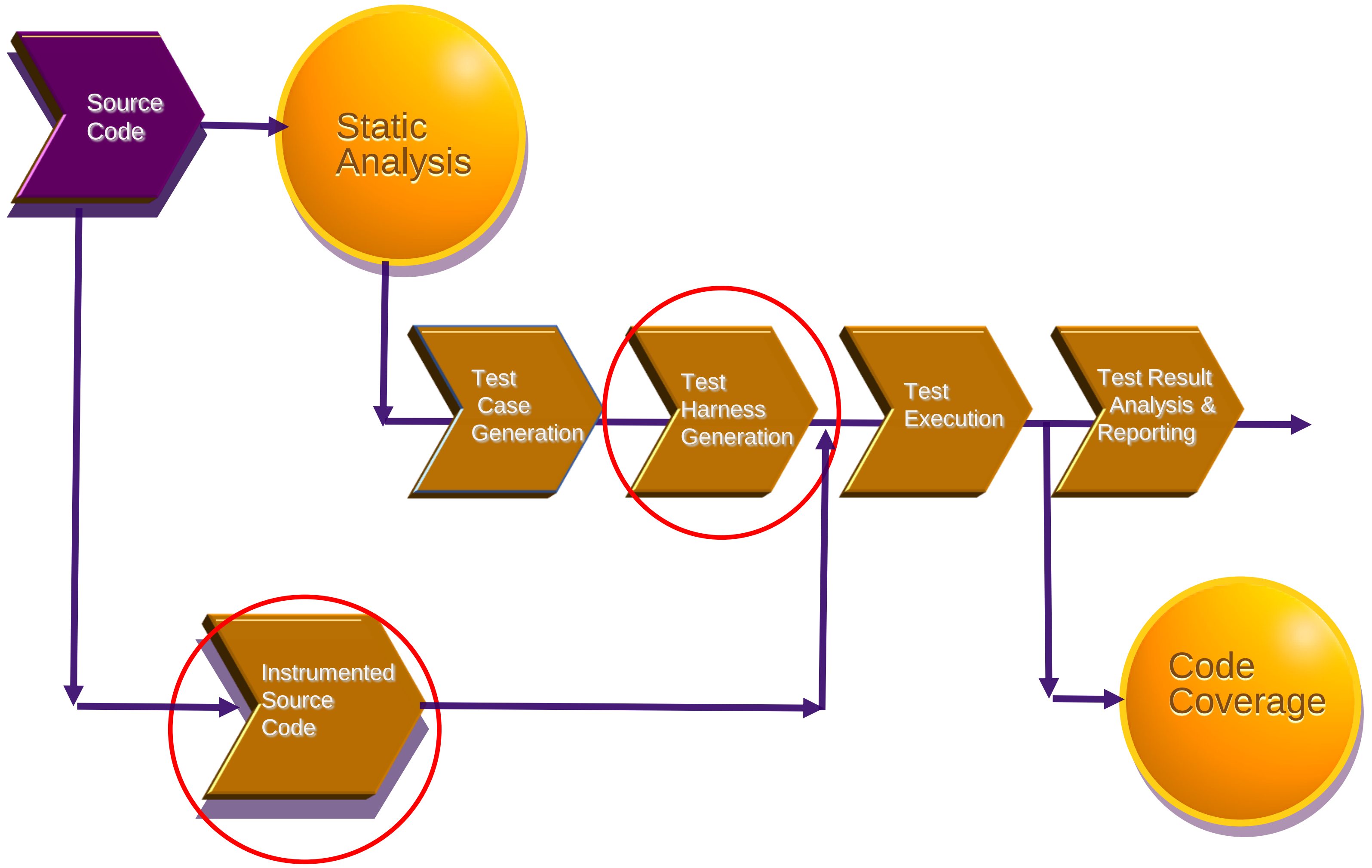
Unit Test & Code Coverage

for embedded systems

- On-Host testing versus On-Target Testing
- Real-Time versus non_Real-Time

Unit Test & Code Coverage

Traditional workflow



```
static void countProducts(void)
{
  struct CountedProduct * currentCountedProduct;
  uint32_t iProduct = 0U;
  const struct Product * currentProduct;

  ProductDatabase_resetCountedProducts();
  /* iterate over each product that has been scanned */
  while (iProduct < scannedProducts) {
    currentProduct = ShoppingBasket[iProduct];
    if ( currentProduct != NULL_POINTER )
    {
       currentCountedProduct = ProductDatabase_getCountedProduct(currentProduct);
       if (currentCountedProduct != NULL_POINTER)
       {
          (currentCountedProduct->count)++;
       }
    }
    iProduct++;
  }
}
```

met instrumentatie:

```
static void countProducts ( void)
  {
   int iCashregister_7zzqqzz = Cashregister_7zzqqzz (  6 ) ; /
* 37 */
    struct CountedProduct * currentCountedProduct ;
    uint32_t iProduct = 0U ;
    const struct Product * currentProduct ;
    ProductDatabase_resetCountedProducts () ;
    /* iterate over each product that has been scanned */
    while ( iProduct < scannedProducts )
      {
 { /* 32 */
        currentProduct = ShoppingBasket [ iProduct ] ;
        if ( currentProduct != ( ( void * ) 0 ) )
 {
            {
            currentCountedProduct =
ProductDatabase_getCountedProduct ( currentProduct ) ;
            if ( currentCountedProduct != ( ( void * ) 0 ) )
              {
                ( currentCountedProduct -> count ) ++ ;
              }

else
  Cashregister_7zzqqzz (  7 ) ; /* 4 */
            }
 }
else
  Cashregister_7zzqqzz (  8 ) ; /* 4 */
        iProduct ++ ;
    Cashregister_7zzqqzz (  9 ) ;} /* 6 */
      }
    Cashregister_7zzqqzz ( 10 ) ; /* 5 */
    Cashregister_7zzqqzz ( 11 ) ; /* 30 */
  }
```

Unit Test & Code Coverage

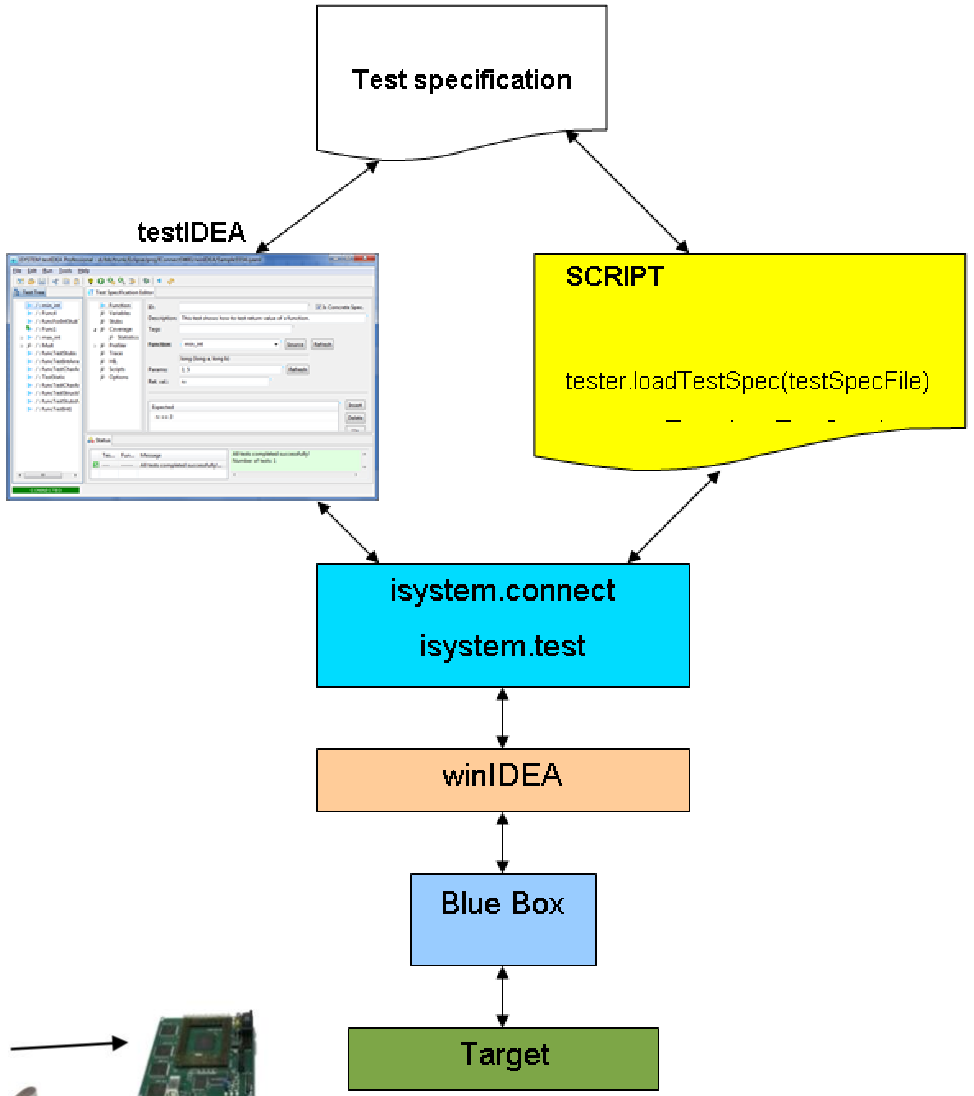
Problems with instrumentation

- Often tricky to integrate into the build process
- Code size increases
- Execution times are different
- Not all functions can be tested (drivers)
- Certification of the instrumenter (more complex TQP)

Unit Test & Code Coverage

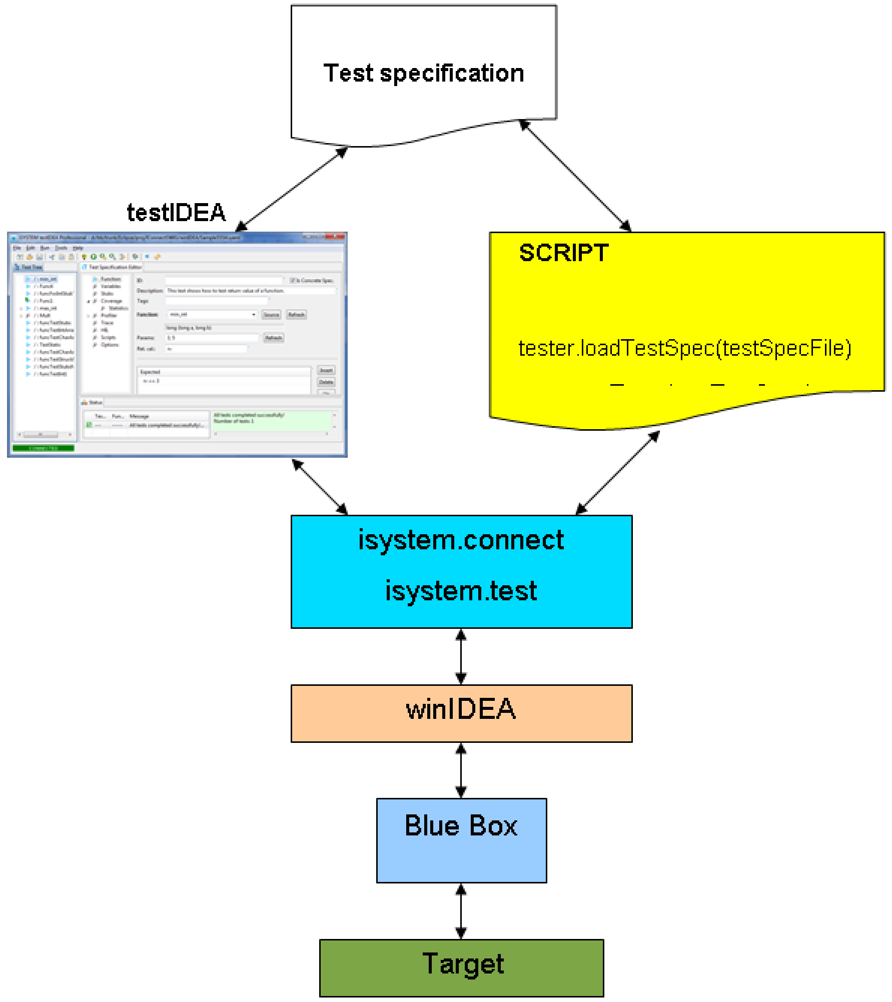
Alternative approach

- Integrate with the debugger
  - Full open API to all debugger functions
  - Fully integrated Unit Test creation
- Automate the process
- For coverage : Use Real-Time Trace
  - ETM, Nexus, Aurora, Full ICE



Benefits

- No instrumenter in the build process
- Test execution on the target system
- Test execution on the production code
  - on Machine code level
  - Cross compiler v.s. host-compiler
- No test driver / test harness needed
- Continuous code coverage Code coverage
- Fully integrated with Development process
  - Development engineer can develop the tests



Examples of advanced features

- Compare ‘golden’ with current Real-Time Execution trace
- Combine trace, performance analysis, code coverage and I/O stimuli with test runs
- Technology may be used for unit, integration and system test

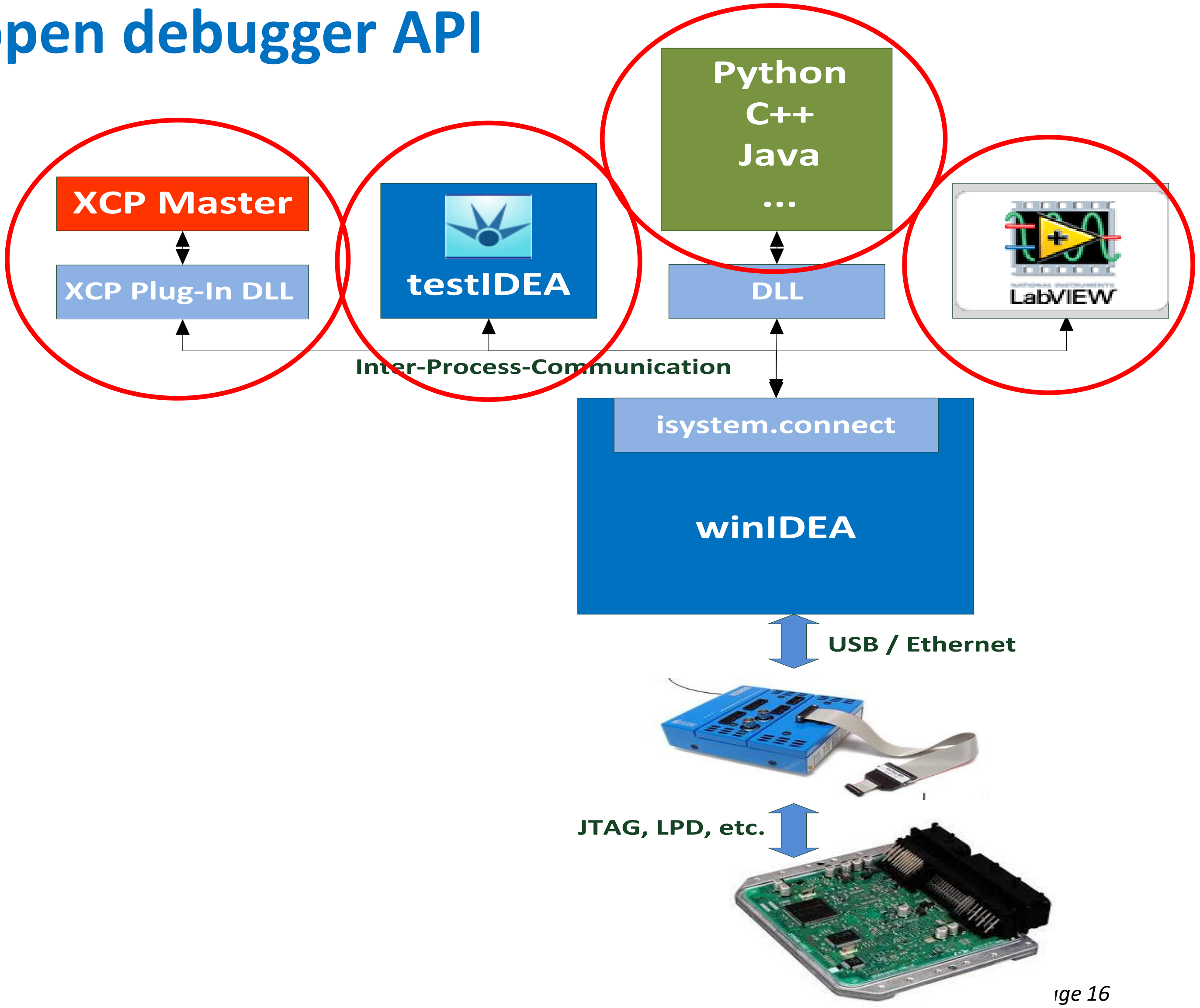
System test

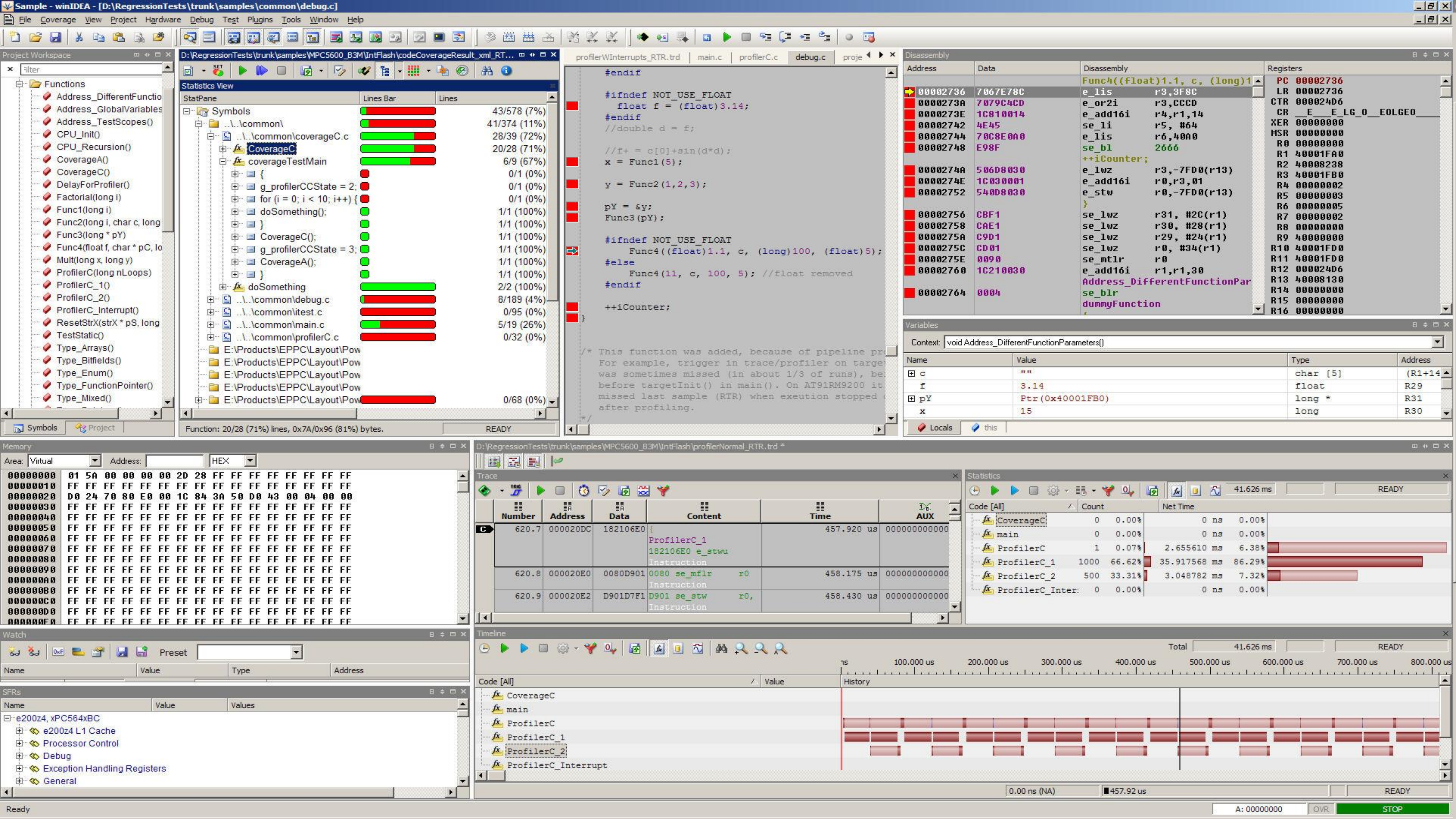
take advantage of the open debugger API

Available Features:

- Debugging:
  Download, Run/Stop, Break
  Symbol Information
  Read/Write Data Access
- Analysis:
  Trace
  Coverage
  Profiling
- IDE and Build System

- All testcases are parametric





Sample - winIDEA - [D:\RegressionTests\trunk\samples\common\debug.c]

File Coverage View Project Hardware Debug Test Plugins Tools Window Help

Project Workspace

- Functions
  - Address_DifferentFunctio
  - Address_GlobalVariables
  - Address_TestScopes()
  - CPU_Init()
  - CPU_Recursion()
  - CoverageA()
  - CoverageC()
  - DelayForProfiler()
  - Factorial(long i)
  - Func1(long i)
  - Func2(long i, char c, long
  - Func3(long * pY)
  - Func4(float f, char * pC, lo
  - Mult(long x, long y)
  - ProfilerC(long nLoops)
  - ProfilerC_1()
  - ProfilerC_2()
  - ProfilerC_Interrupt()
  - ResetStrX(strX * pS, long
  - TestStatic()
  - Type_Arrays()
  - Type_Bitfields()
  - Type_Enum()
  - Type_FunctionPointer()
  - Type_Mixed()

Symbols | Project

D:\RegressionTests\trunk\samples\MPC5600_B3M\IntFlash\codeCoverageResult_xml_RT...

Statistics View

StatPane	Lines Bar	Lines
Symbols		43/578 (7%)
..\..\common\		41/374 (11%)
..\..\common\coverageC.c		28/39 (72%)
CoverageC		20/28 (71%)
coverageTestMain		6/9 (67%)
{		0/1 (0%)
g_profilerCCState = 2;		0/1 (0%)
for (i = 0; i < 10; i++) {		0/1 (0%)
doSomething();		1/1 (100%)
}		1/1 (100%)
CoverageC();		1/1 (100%)
g_profilerCCState = 3;		1/1 (100%)
CoverageA();		1/1 (100%)
}		1/1 (100%)
doSomething		2/2 (100%)
..\..\common\debug.c		8/189 (4%)
..\..\common\itest.c		0/95 (0%)
..\..\common\main.c		5/19 (26%)
..\..\common\profilerC.c		0/32 (0%)
E:\Products\EPPC\Layout\Pow		
E:\Products\EPPC\Layout\Pow		
E:\Products\EPPC\Layout\Pow		
E:\Products\EPPC\Layout\Pow		
E:\Products\EPPC\Layout\Pow		0/68 (0%)

Function: 20/28 (71%) lines, 0x7A/0x96 (81%) bytes. READY

profilerWInterrupts_RTR.trd | main.c | profilerC.c | debug.c | proje

```
#endif

#ifndef NOT_USE_FLOAT
  float f = (float)3.14;
#endif
//double d = f;

//f+ = c[0]+sin(d*d);
x = Func1(5);

y = Func2(1,2,3);

pY = &y;
Func3(pY);

#ifndef NOT_USE_FLOAT
    Func4((float)1.1, c, (long)100, (float)5);
#else
    Func4(11, c, 100, 5); //float removed
#endif

++iCounter;
}


/* This function was added, because of pipeline pr
   For example, trigger in trace/profiler on targe
   was sometimes missed (in about 1/3 of runs), be
   before targetInit() in main(). On AT91RM9200 it
   missed last sample (RTR) when exeution stopped
   after profiling.
*/
```

Disassembly

```
Address    Data       Disassembly
                      Func4((float)1.1, c, (long)1
00002736   7067E78C   e_lis      r3,3F8C
0000273A   7079C4CD   e_or2i     r3,CCCD
0000273E   1C810014   e_add16i   r4,r1,14
00002742   4E45       se_li      r5, #64
00002744   70C8E0A0   e_lis      r6,40A0
00002748   E98F       se_bl      2666
                      ++iCounter;
0000274A   506D8030   e_lwz      r3,-7FD0(r13)
0000274E   1C030001   e_add16i   r0,r3,01
00002752   540D8030   e_stw      r0,-7FD0(r13)
                      }
00002756   CBF1       se_lwz     r31, #2C(r1)
00002758   CAE1       se_lwz     r30, #28(r1)
0000275A   C9D1       se_lwz     r29, #24(r1)
0000275C   CD01       se_lwz     r0, #34(r1)
0000275E   0090       se_mtlr    r0
00002760   1C210030   e_add16i   r1,r1,30
                      Address_DifferentFunctionPar
00002764   0004       se_blr
                      dummyFunction
```

Registers

```
 PC 00002736
 LR 00002736
CTR 000024D6
 CR __E___E_LG_0__EOLGEO____
XER 00000000
MSR 00000000
 R0 00000000
 R1 40001FA0
 R2 40008238
 R3 40001FB0
 R4 00000002
 R5 00000003
 R6 00000005
 R7 00000002
 R8 00000000
 R9 40000000
R10 40001FD0
R11 40001FD0
R12 000024D6
R13 40008130
R14 00000000
R15 00000000
R16 00000000
```

Variables

Context: void Address_DifferentFunctionParameters()

Name	Value	Type	Address
c	""	char [5]	(R1+14
f	3.14	float	R29
pY	Ptr(0x40001FB0)	long *	R31
x	15	long	R30

Locals | this

Memory

Area: Virtual Address: HEX

```
00000000  01 5A 00 00 00 00 2D 28 FF FF FF FF FF FF FF FF
00000010  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000020  D0 24 70 80 E0 00 1C 84 3A 50 D0 43 00 04 00 00
00000030  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000040  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000050  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000060  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000070  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000080  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
00000090  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000A0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000B0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000C0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000D0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
000000E0  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Watch

Preset

Name	Value	Type	Address

SFRs

Name	Value	Values
e200z4, xPC564xBC		
e200z4 L1 Cache		
Processor Control		
Debug		
Exception Handling Registers		
General		

D:\RegressionTests\trunk\samples\MPC5600_B3M\IntFlash\profilerNormal_RTR.trd *

Trace

```
Number   Address   Data       Content                  Time          AUX
 620.7   000020DC  182106E0   {                        457.920 us    000000000000
                              ProfilerC_1
                              182106E0 e_stwu
                              Instruction
 620.8   000020E0  0080D901   0080 se_mflr     r0      458.175 us    000000000000
                              Instruction
 620.9   000020E2  D901D7F1   D901 se_stw      r0,     458.430 us    000000000000
                              Instruction
```

Statistics

41.626 ms READY

Code [All]	Count		Net Time	
CoverageC	0	0.00%	0 ns	0.00%
main	0	0.00%	0 ns	0.00%
ProfilerC	1	0.07%	2.655610 ms	6.38%
ProfilerC_1	1000	66.62%	35.917568 ms	86.29%
ProfilerC_2	500	33.31%	3.048782 ms	7.32%
ProfilerC_Inter:	0	0.00%	0 ns	0.00%

Timeline

Total 41.626 ms READY

ns 100.000 us 200.000 us 300.000 us 400.000 us 500.000 us 600.000 us 700.000 us 800.000 us

Code [All]	Value	History
CoverageC		
main		
ProfilerC		
ProfilerC_1		
ProfilerC_2		
ProfilerC_Interrupt		

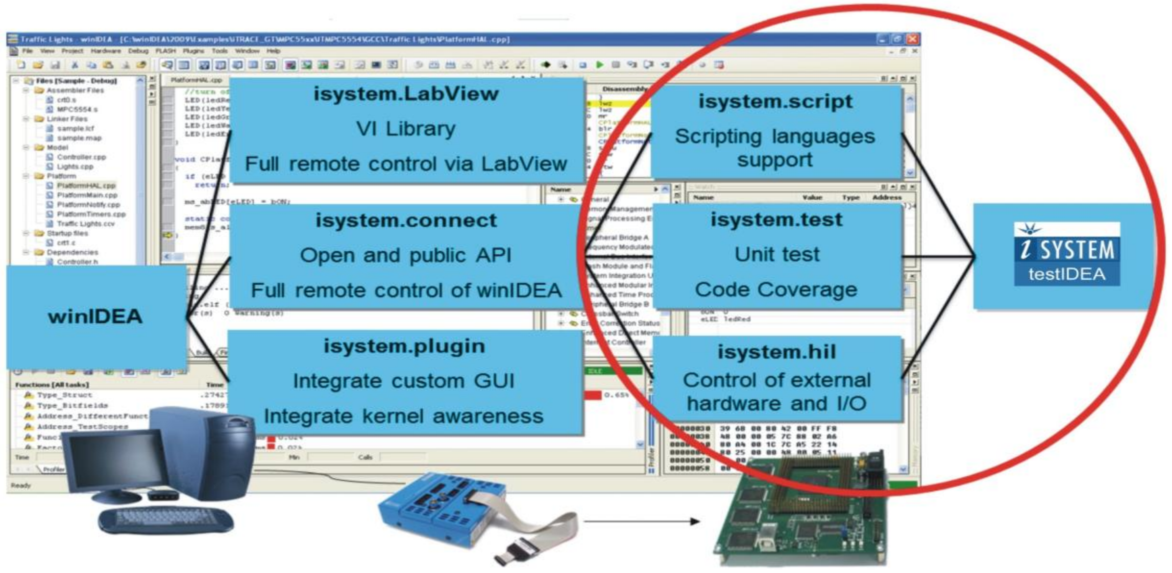
0.00 ns (NA) 457.92 us READY

Ready A: 00000000 OVR STOP

System test

Benefit of using debugger API

- Very easy to do complex tests
  - Debugger knows ‘everything’
  - Real-time access to memory, registers, variables, peripherals
  - All during Real-Time execution
- Simplifies System test fixtures
- All test cases are parametric
- Easy reuse of test cases



iSystem Test Results

Test Configuration	
Attribute	Value
report file	D:\winIDEA\2011\Examples\OnChip\PowerPC\MPC55xx\ITMPC5554\GCC\IntRAM\ErolsersterReport.xml
tester	Erol
date	20.06.2011
time	16:58:15
software	SW
hardware	HW
description	Desc
testSpecificationFile	D:\winIDEA\2011\Examples\OnChip\PowerPC\MPC55xx\ITMPC5554\GCC\IntRAM\PPC5554 ITMPC5554 GCC Int RAM.iyaml
wiWorkspacePath	D:\winIDEA\2011\Examples\OnChip\PowerPC\MPC55xx\ITMPC5554\GCC\IntRAM

Test ID	Function	Result
StubTest	Address_DifferentFunctionParameters	Pass
Tags	Description	
	Stub Test	

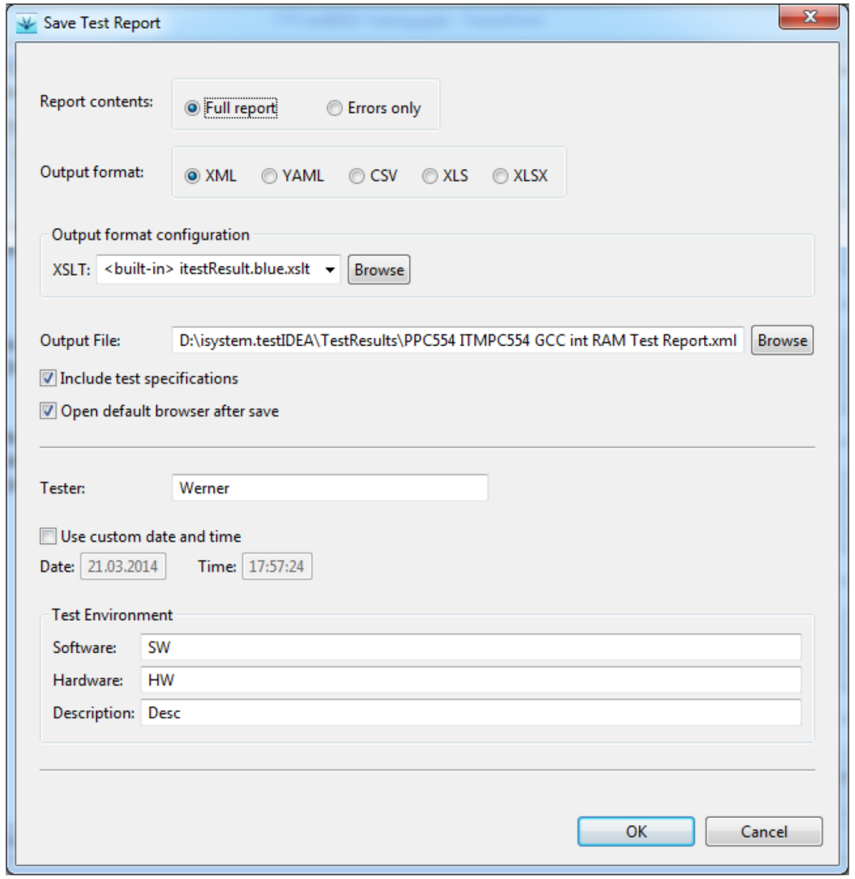
Test Specification

```
id: 'StubTest'
desc: 'Stub Test'
init:
  iCounter: 0
func: [Address_DifferentFunctionParameters]
stubs:
- func:
  - Func2
  - retV
  assign:
    retV: 2
expect:
- iCounter == 1
```

Test ID	Function	Result
GlobalVariableTest	Address_DifferentFunctionParameters	Pass
Tags	Description	
	Global Variable Test	

Test Specification

```
id: 'GlobalVariableTest'
desc: 'Global Variable Test'
init:
  iCounter: 0
func: [Address_DifferentFunctionParameters]
expect:
- iCounter == 1
```


Save Test Report

Report contents: [✓ Full report] Errors only

Output format: [✓ XML] YAML CSV XLS XLSX

Output format configuration

XSLT: <built-in> itestResult.blue.xslt

Output File: D:\isystem.testIDEA\TestResults\PPC554 ITMPC554 GCC int RAM Test Report.xml

☑ Include test specifications

☑ Open default browser after save

Tester: Werner

☐ Use custom date and time

Date: 21.03.2014 Time: 17:57:24

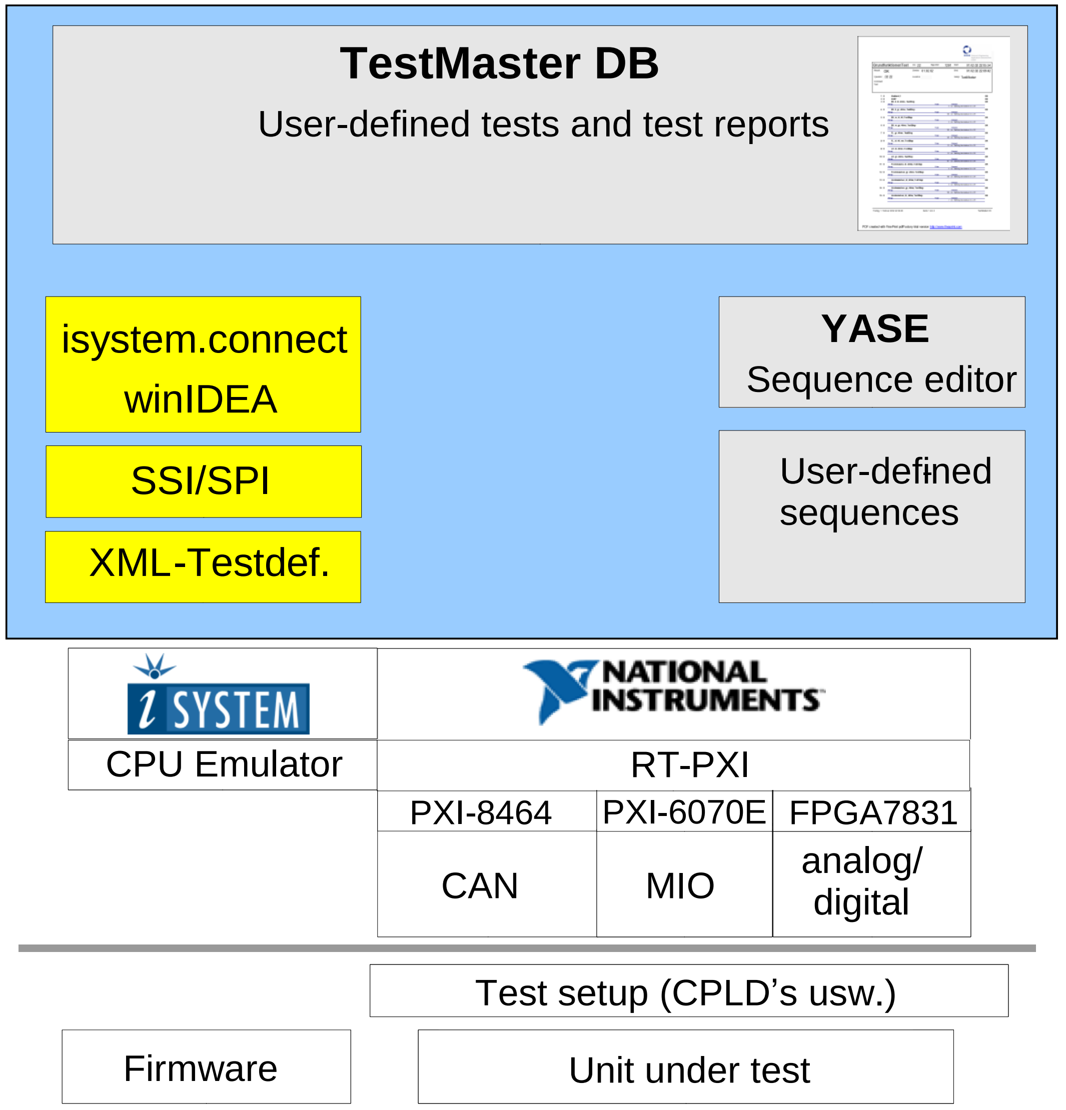
Test Environment

Software: SW

Hardware: HW

Description: Desc

Example: Hella (part of CASA@HELLA)



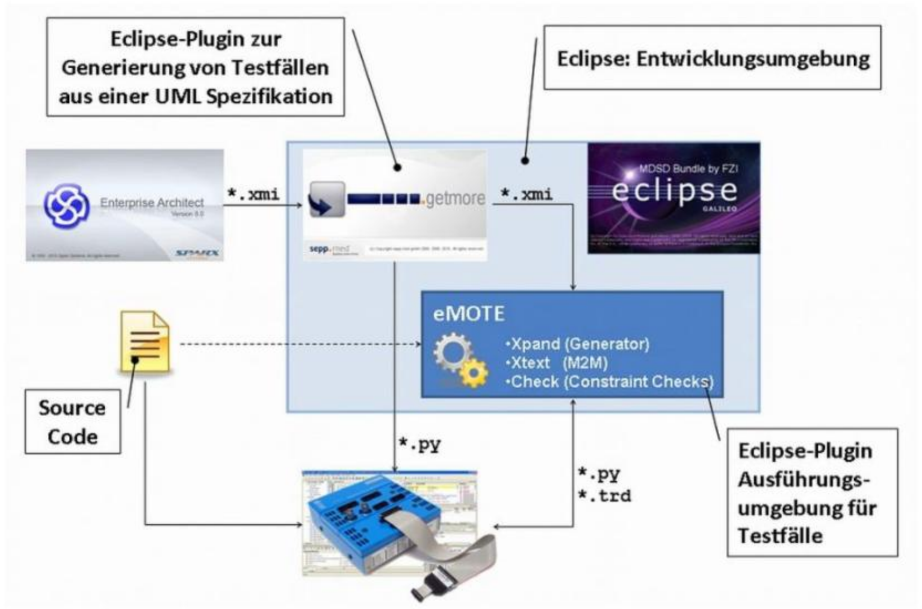
Configuration
Test definition

Measurement system

FIT Hardware

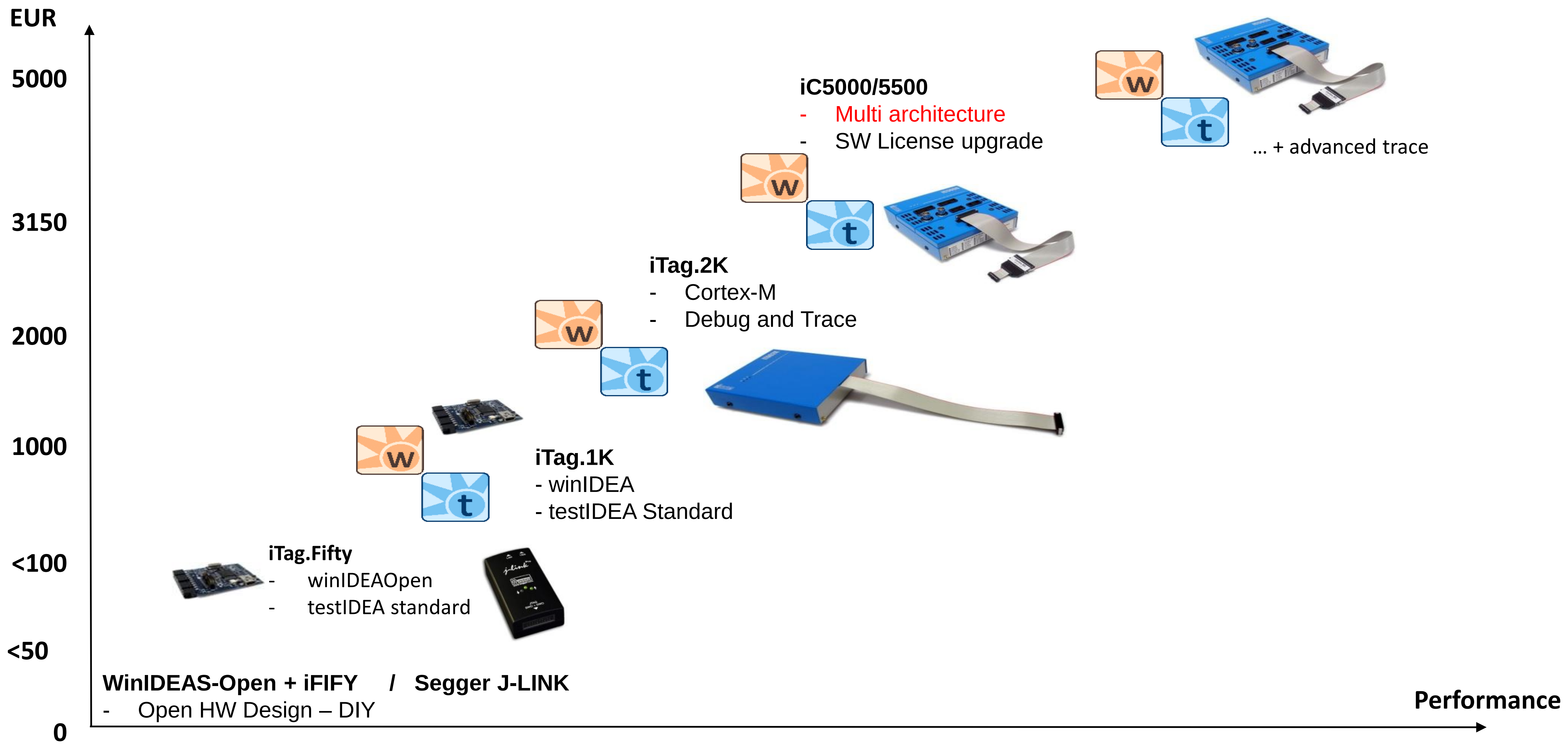
Advanced example:

eMOTE:*E*mbedded *Mo*del-based *Te*sting


Eclipse-Plugin zur Generierung von Testfällen aus einer UML Spezifikation
Eclipse: Entwicklungsumgebung
Enterprise Architect
*.xmi
.getmore
*.xmi
MDSD Bundle by FZI
eclipse
eMOTE
•Xpand (Generator)
•Xtext (M2M)
•Check (Constraint Checks)
Source Code
*.py
*.py
*.trd
Eclipse-Plugin Ausführungs-umgebung für Testfälle

- Funded by BMWi Germany
- Project partners: FZI Karlsruhe, sepp.med GmbH
- Project Number: KF2076903SS9

Wat kost dat ?



Download WinIDEA-OPEN at : http://isystem.com/index.php/download/winideaopen

WinIDEA Open

experience the environment yourself

Free Cortex-M software development and test platform supporting iSYSTEM's Cortex Tool iTAG, different 3rd party debug hardware (e.g., SEGGER J-Link , ST-LINK, CMSIS-DAP) and a large number of evaluation boards. winIDEA Open supports all major compilers and imposes no code size restrictions when used with the GNU toolchain (32K others) and is provided with no support. Therefore it is recommended to use winIDEA Open for evaluation and non-critical projects only.

Overall software features (no time or code size limitations):
Full GNU GCC toolchain 4.7 included
Unlimited code size with GCC compiler
Full featured winIDEA platform:

- Editor & build manager
- Flash programming
- HW and SW breakpoints
- Low and high-level debugging
- Device register view (SFRs)
- Python scripting
- Test tool testIDEA standard included (Unit Test / Code Coverage)
- RTOS aware debugging
- Interoperable with a wide range of tools through isystem.connect API

Optional upgrade to commercial winIDEA build (all compilers supported, full technical support, regular winIDEA updates, support for new microcontrollers, new winIDEA functionalities, ...). To upgrade, please INDES-IDS BV.

WinIDEA Open

Download WinIDEA-OPEN at : http://isystem.com/index.php/download/winideaopen

winIDEA Open currently supports the iSYSTEM iTAG.ZERO and iTAG.FIFTY as well as various third party debug hardware such as the SEGGER J-Link , ST-LINK and CMSIS-DAP in conjunction with many evaluation boards, see http://isystem.com/index.php/products/software/winidea-open

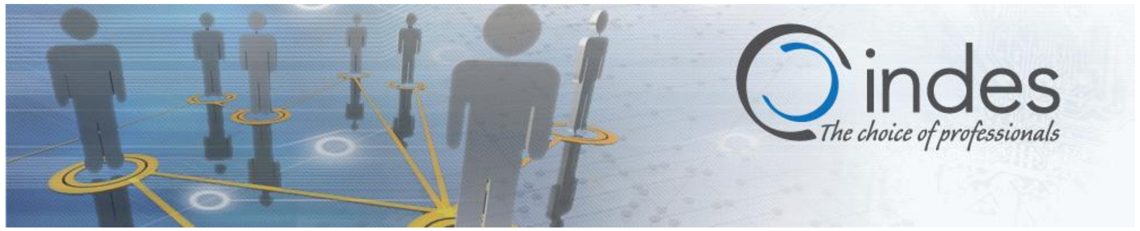
Voor meer informatie :

Gerard Fianen

gerard@indes.com

Tel : 0345 – 545.535

www.indes.com/embedded



Supporting slides

- More detailed information on Unit Test
  - Debugger knows ‘everything’
  - Real-time access to memory, registers, variables, peripherals
  - All during Real-Time execution
- More detailed information on Code Coverage using trace
- More on Certification

indes
The choice of professionals

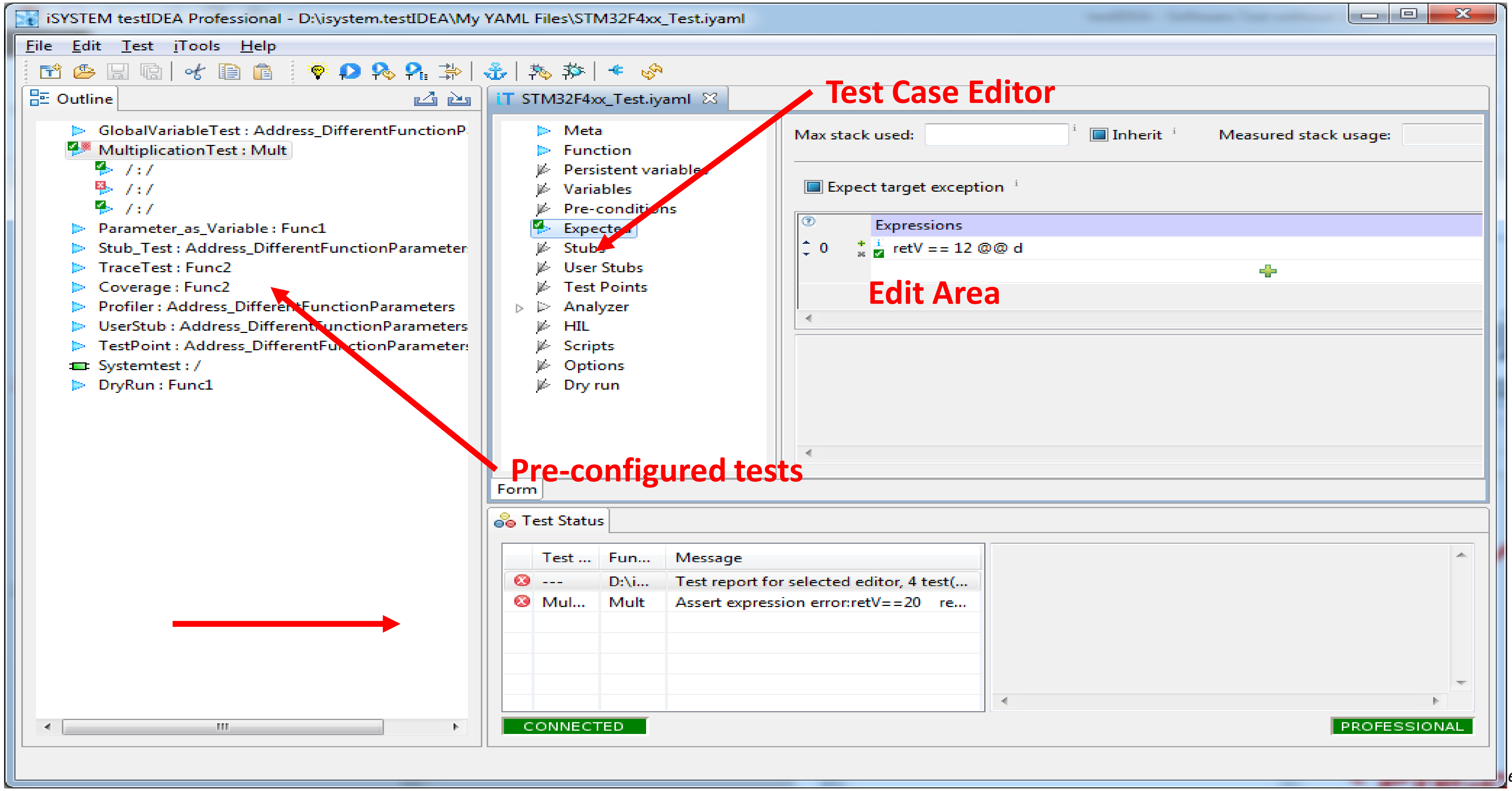
iSYSTEM Test Possibilities

iSYSTEM

EMBEDDED TOOLS FOR SINGLE AND MULTI-CORE

Software Development, Analysis, Test Automation and Certification

testIDEA in conjunction with winIDEA



GlobalVariableTest : Address_DifferentFunctionP
MultiplicationTest : Mult
/ : /
/ : /
/ : /
Parameter_as_Variable : Func1
Stub_Test : Address_DifferentFunctionParameter
TraceTest : Func2
Coverage : Func2
Profiler : Address_DifferentFunctionParameters
UserStub : Address_DifferentFunctionParameters
TestPoint : Address_DifferentFunctionParameter
Systemtest : /
DryRun : Func1

Meta
Function
Persistent variables
Variables
Pre-conditions
Expected
Stubs
User Stubs
Test Points
Analyzer
HIL
Scripts
Options
Dry run

Max stack used: [ ] Inherit Measured stack usage:

Expect target exception

	Expressions
0	retV == 12 @@ d

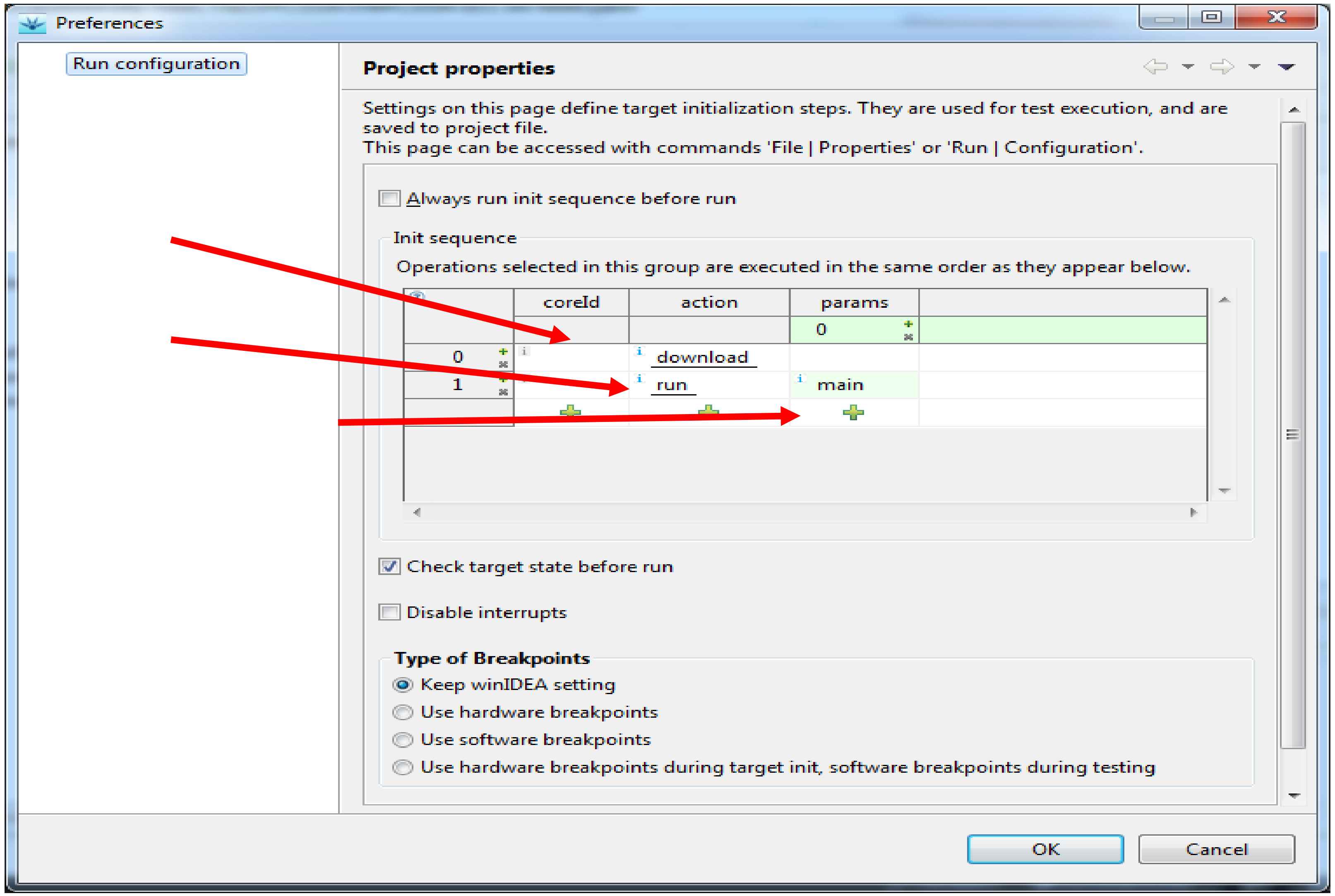
	Test ...	Fun...	Message
⊗	---	D:\i...	Test report for selected editor, 4 test(...
⊗	Mul...	Mult	Assert expression error:retV==20 re...

testIDEA Basic Configuration

Which core?

What to do?

Which parameter?



Project properties

Settings on this page define target initialization steps. They are used for test execution, and are saved to project file.
This page can be accessed with commands 'File | Properties' or 'Run | Configuration'.

[ ] Always run init sequence before run

Init sequence

Operations selected in this group are executed in the same order as they appear below.

	coreId	action	params
			0
0		download	
1		run	main

[✓] Check target state before run

[ ] Disable interrupts

Type of Breakpoints

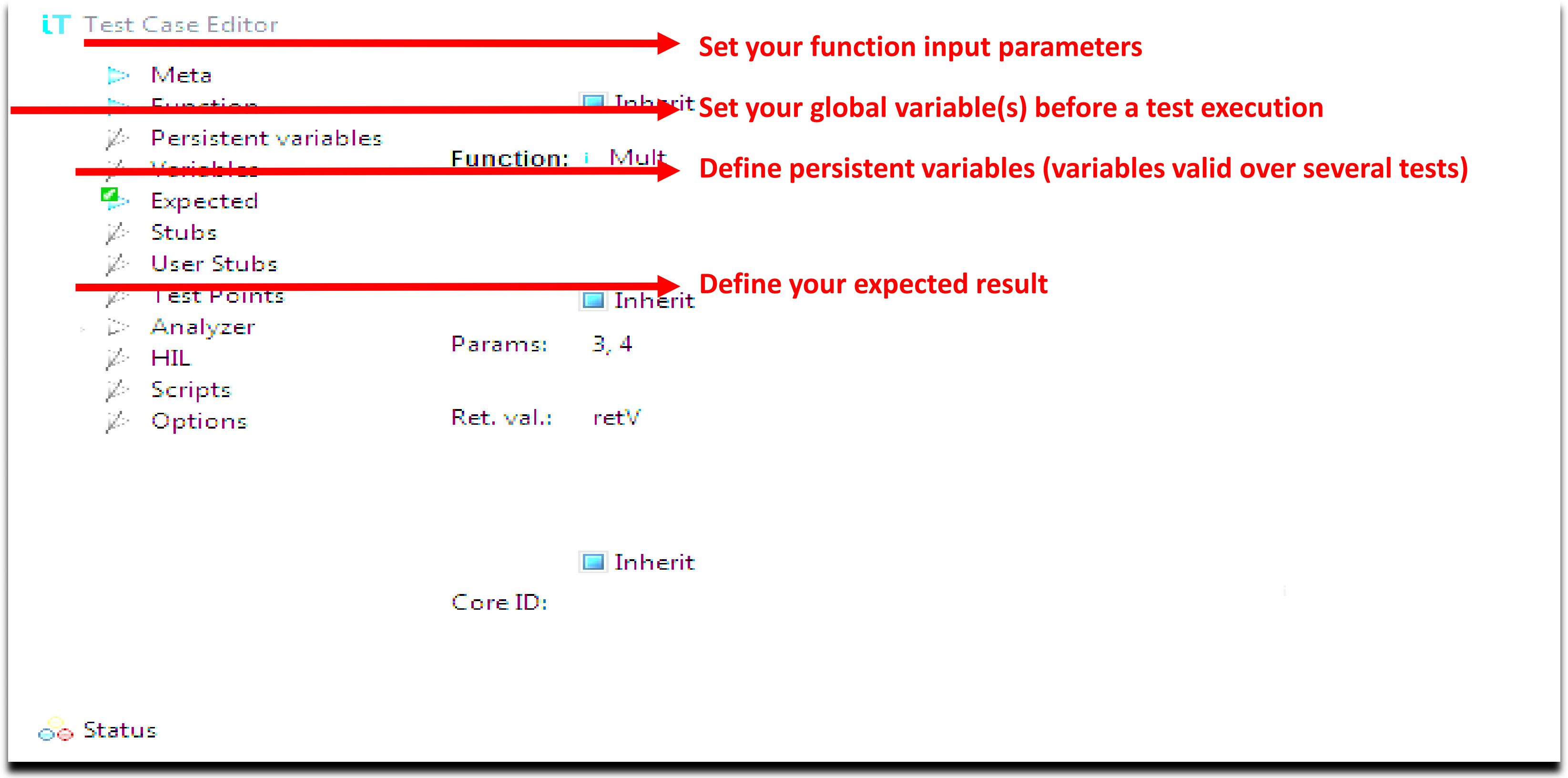
(●) Keep winIDEA setting
( ) Use hardware breakpoints
( ) Use software breakpoints
( ) Use hardware breakpoints during target init, software breakpoints during testing

Input Parameters / Expected Return Values

Input Parameters can be

- Function Parameters
- Global Variables
- Persistent Variables

Expected Return Value



Test Case Editor

Meta
Function
Persistent variables
Variables
Expected
Stubs
User Stubs
Test Points
Analyzer
HIL
Scripts
Options

Inherit
Function: Mult

Inherit
Params: 3, 4

Ret. val.: retV

Inherit
Core ID:

Status

Set your function input parameters

Set your global variable(s) before a test execution

Define persistent variables (variables valid over several tests)

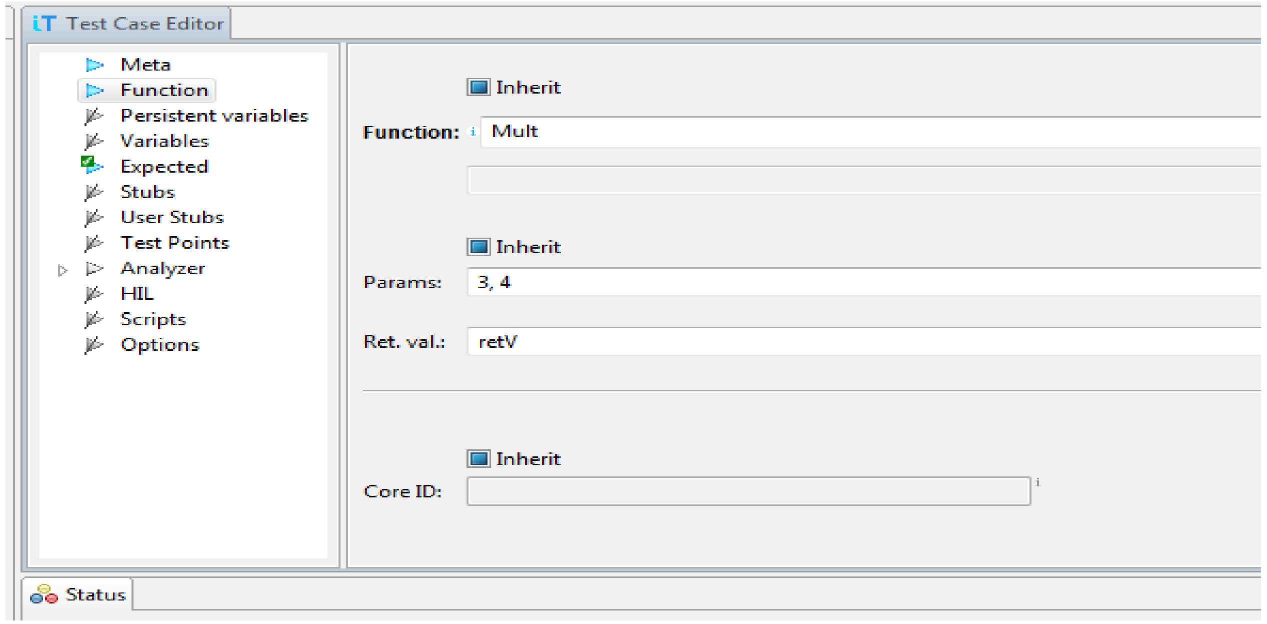
Define your expected result

Input Parameters / Expected Return Values

Input Parameters can be

- Function Parameters
- Global Variables
- Persistent Variables

Expected Return Value



Inherit

Function: Mult

Inherit

Params: 3, 4

Ret. val.: retV

Inherit

Core ID:

Test Specification as text:

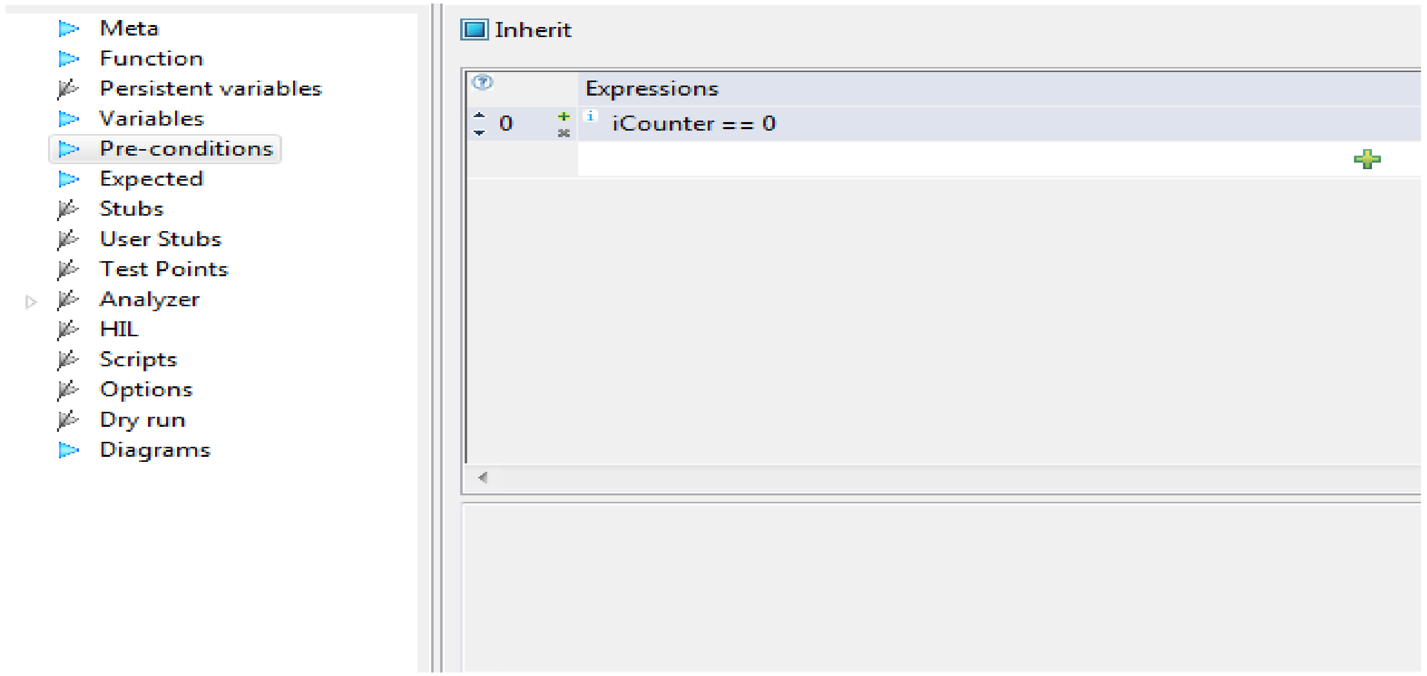
```
- id: FunctionTest
  desc: |-
    My Function Test
  init:
    iCounter: 0
  func: [Mult, [3, 4], retV]
  expect:
  - 'retV == 12'
```

Pre-Conditions

This section contains expressions, which are evaluated before test case execution is started.

This functionality can be used, when certain global variables or hardware input values must match some criteria for test to succeed.

If we detect error early, it is easier to find the reason for test failure.



0 iCounter == 0

Stubs

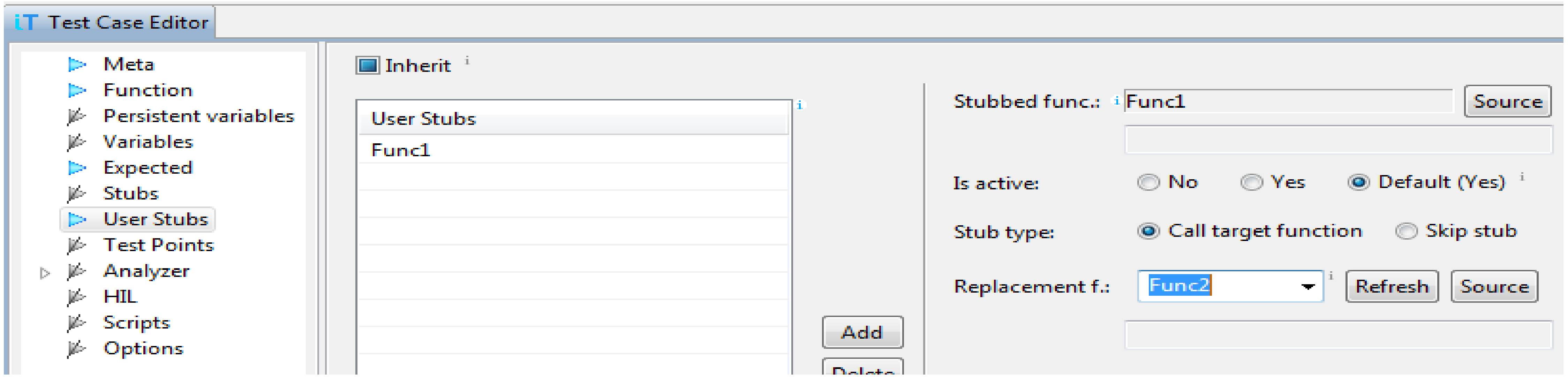
Method stub (From Wikipedia)

A method stub or simply stub in software development **is a piece of code used to stand in for some other programming functionality**. A stub may simulate the behavior of existing code (such as a procedure on a remote machine) or be a temporary substitute for yet-to-be-developed code. **Stubs are therefore most useful in** porting, distributed computing as well as general software development and **testing**.

Typical Use Cases in testIDEA:

1. Replace a function call with a return value -> the function is not called, the return value is set (not real time)
2. Do nothing – a function call is simply ignored (real time)
3. A function call is replaced with another function call (real time). The replaced function is already a part of the downloaded code.

Stubs



Inherit

User Stubs
Func1

Stubbed func.: Func1 Source

Is active: No Yes [✓ Default (Yes)]

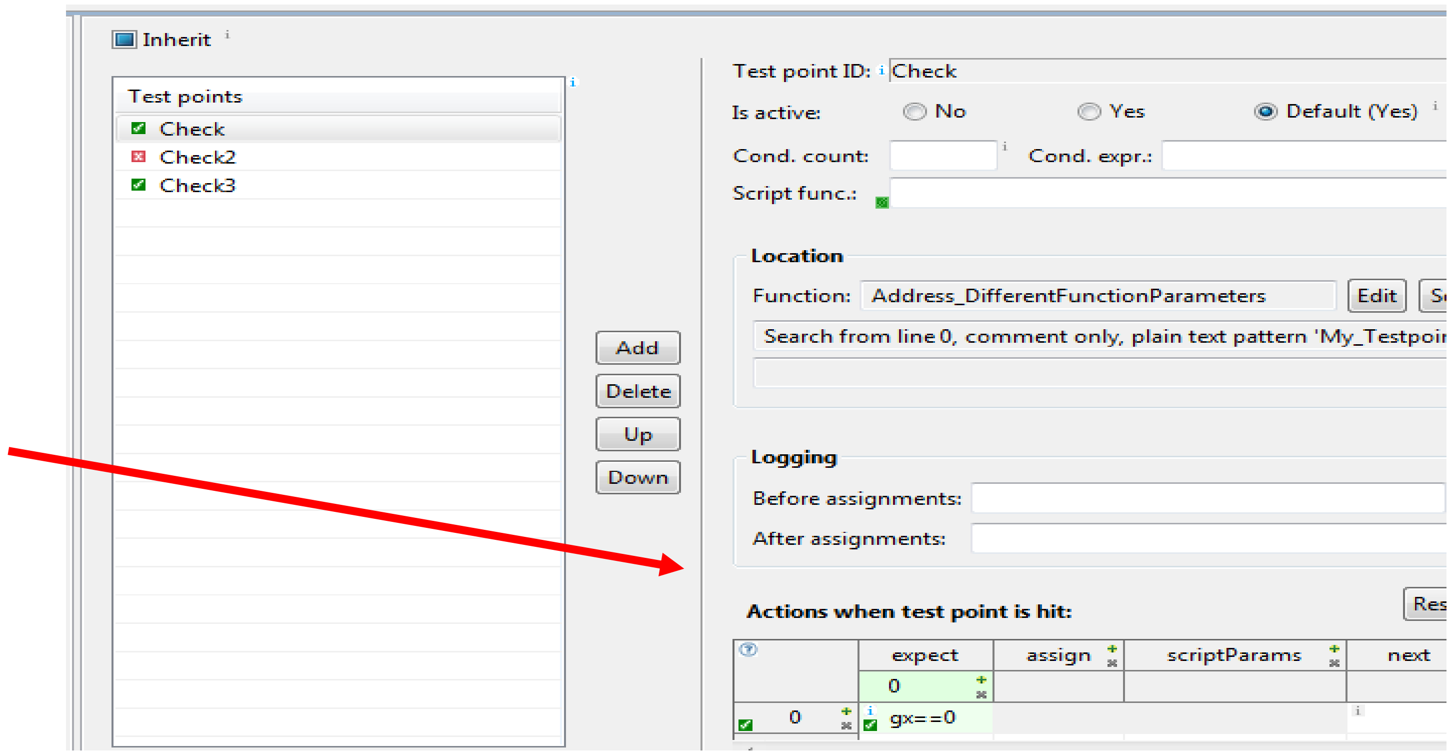
Stub type: [✓ Call target function] Skip stub

Replacement f.: Func2 Refresh Source

Test points

Test points pause test execution at any location in the tested code and perform a certain action. This could be

- Anything you can do with a debugger like read/write Memory, etc.
- Modify Variables, Function Calls (-> Fault Injection)
- Logging
- Use a script for modification



Inherit

Test points
✓ Check
✗ Check2
✓ Check3

Test point ID: Check

Is active: ○ No ○ Yes [✓ Default (Yes)]

Cond. count:

Cond. expr.:

Script func.:

Location

Function: Address_DifferentFunctionParameters

Search from line 0, comment only, plain text pattern 'My_Testpoi

Logging

Before assignments:

After assignments:

Actions when test point is hit:

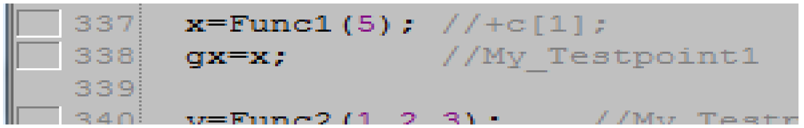
	expect	assign	scriptParams	next
	0			
0	gx==0			

Test points

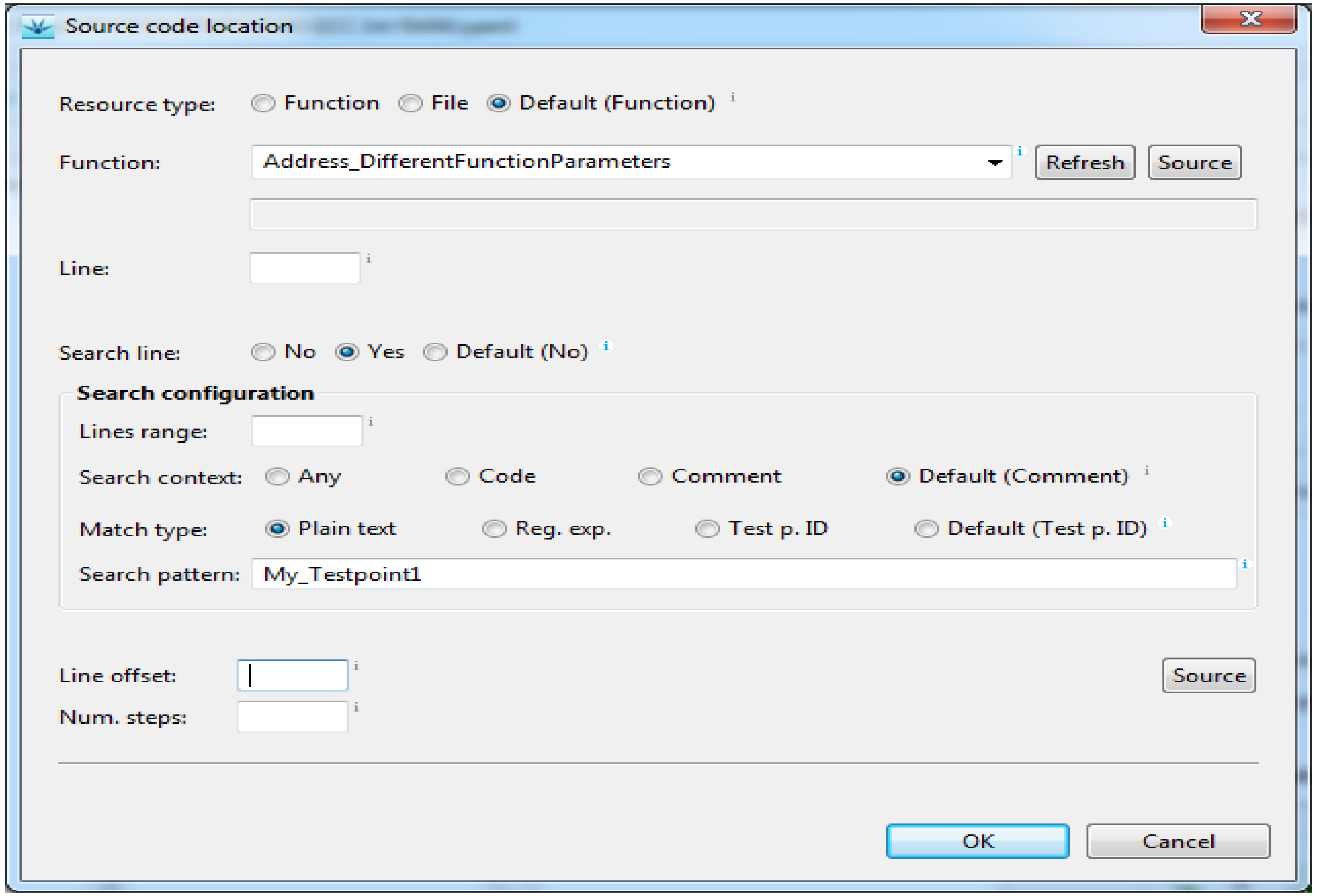
Where to stop?

Base could be a file name (module.c) or a function name.

- Line Number (not recommended because they could change after a rebuild)
- Search Pattern
- Test ID



```
x=Func1(5); //+c[1];
gx=x;       //My_Testpoint1

y=Func2(1, 2, 3);     //My_Testp
```



Source code location [illegible]

Resource type:	Function File [✓ Default (Function)]
Function:	Address_DifferentFunctionParameters
Line:	
Search line:	No [✓ Yes] Default (No)

Search configuration

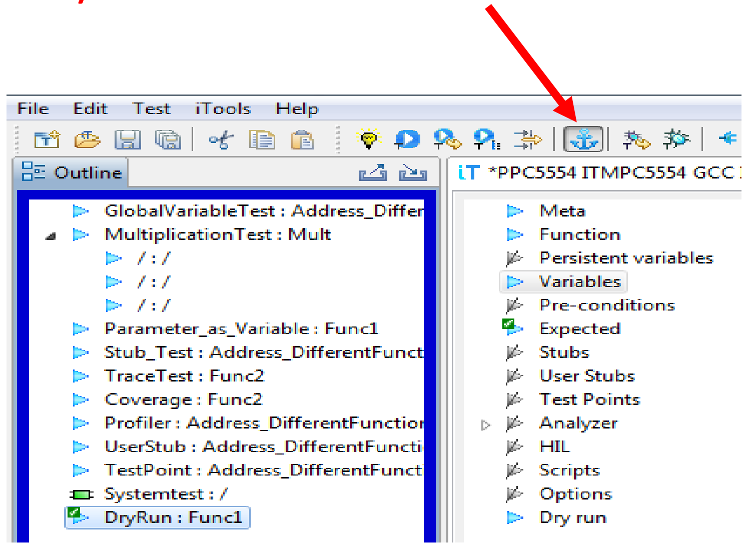
Lines range:	
Search context:	Any Code Comment [✓ Default (Comment)]
Match type:	[✓ Plain text] Reg. exp. Test p. ID Default (Test p. ID)
Search pattern:	My_Testpoint1

Line offset:	
Num. steps:	

Dry Run

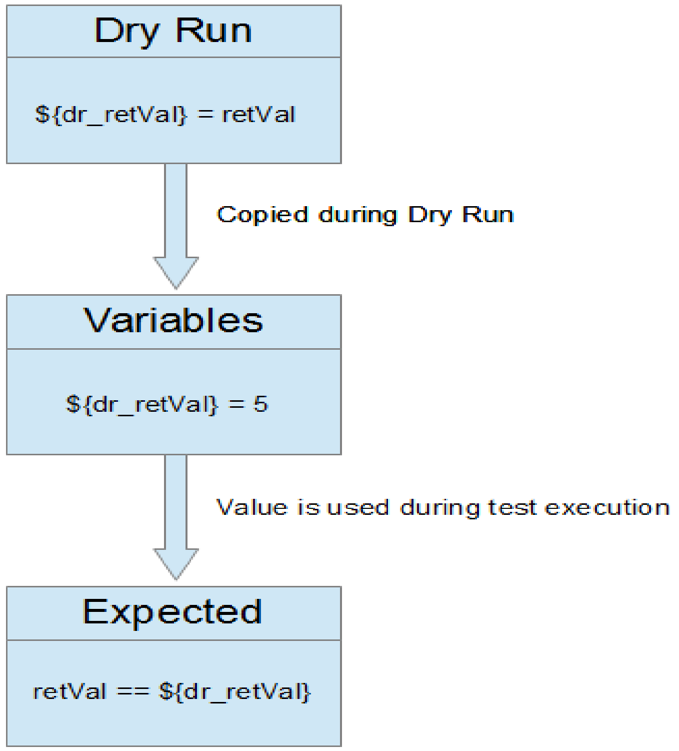
This functionality can be used to record behavior of existing and tested target code and use the result for the next test run.

Dry Run Mode



GlobalVariableTest : Address_Differ
MultiplicationTest : Mult
/ : /
/ : /
/ : /
Parameter_as_Variable : Func1
Stub_Test : Address_DifferentFunct
TraceTest : Func2
Coverage : Func2
Profiler : Address_DifferentFunctio
UserStub : Address_DifferentFuncti
TestPoint : Address_DifferentFunct
Systemtest : /
DryRun : Func1

Meta
Function
Persistent variables
Variables
Pre-conditions
Expected
Stubs
User Stubs
Test Points
Analyzer
HIL
Scripts
Options
Dry run



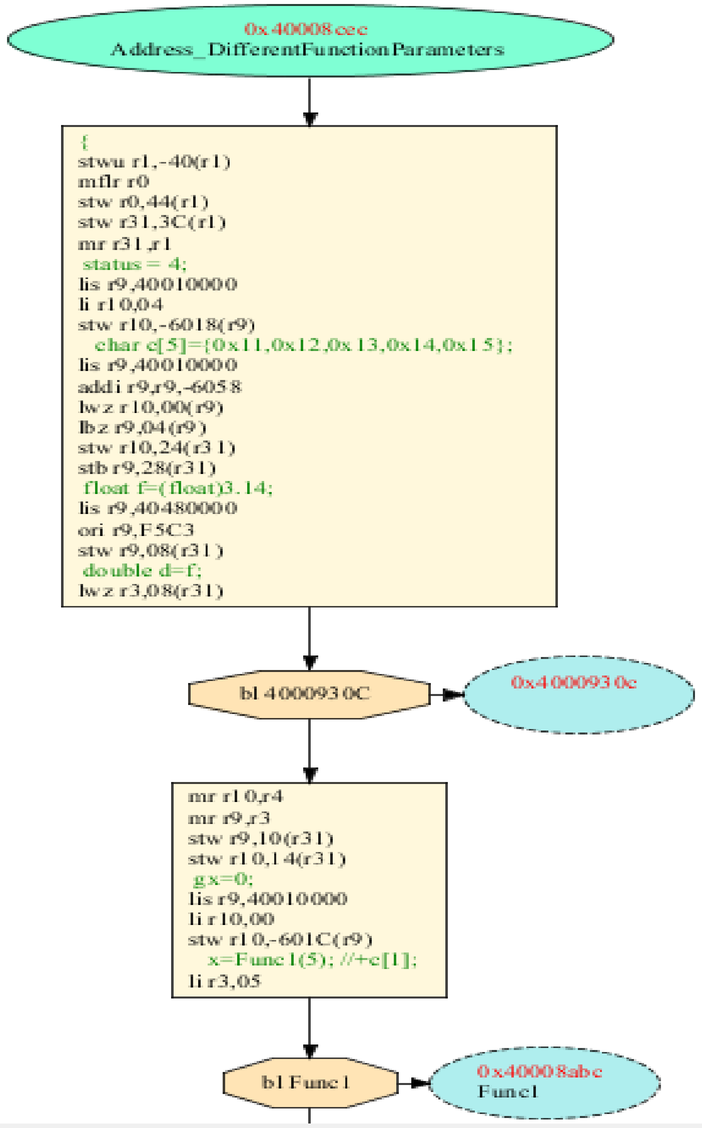
Charts

Flow Chart

A flowchart is a type of diagram that represents the program flow of the function under test.

The information is based on the downloaded code, not on the executed code.

In general flowcharts are used in analyzing, designing, documenting or managing a process or program in various fields.



```
{
stwu r1,-40(r1)
mflr r0
stw r0,44(r1)
stw r31,3C(r1)
mr r31,r1
 status = 4;
lis r9,40010000
li r10,04
stw r10,-6018(r9)
  char c[5]={0x11,0x12,0x13,0x14,0x15};
lis r9,40010000
addi r9,r9,-6058
lwz r10,00(r9)
lbz r9,04(r9)
stw r10,24(r31)
stb r9,28(r31)
 float f=(float)3.14;
lis r9,40480000
ori r9,F5C3
stw r9,08(r31)
 double d=f;
lwz r3,08(r31)
```

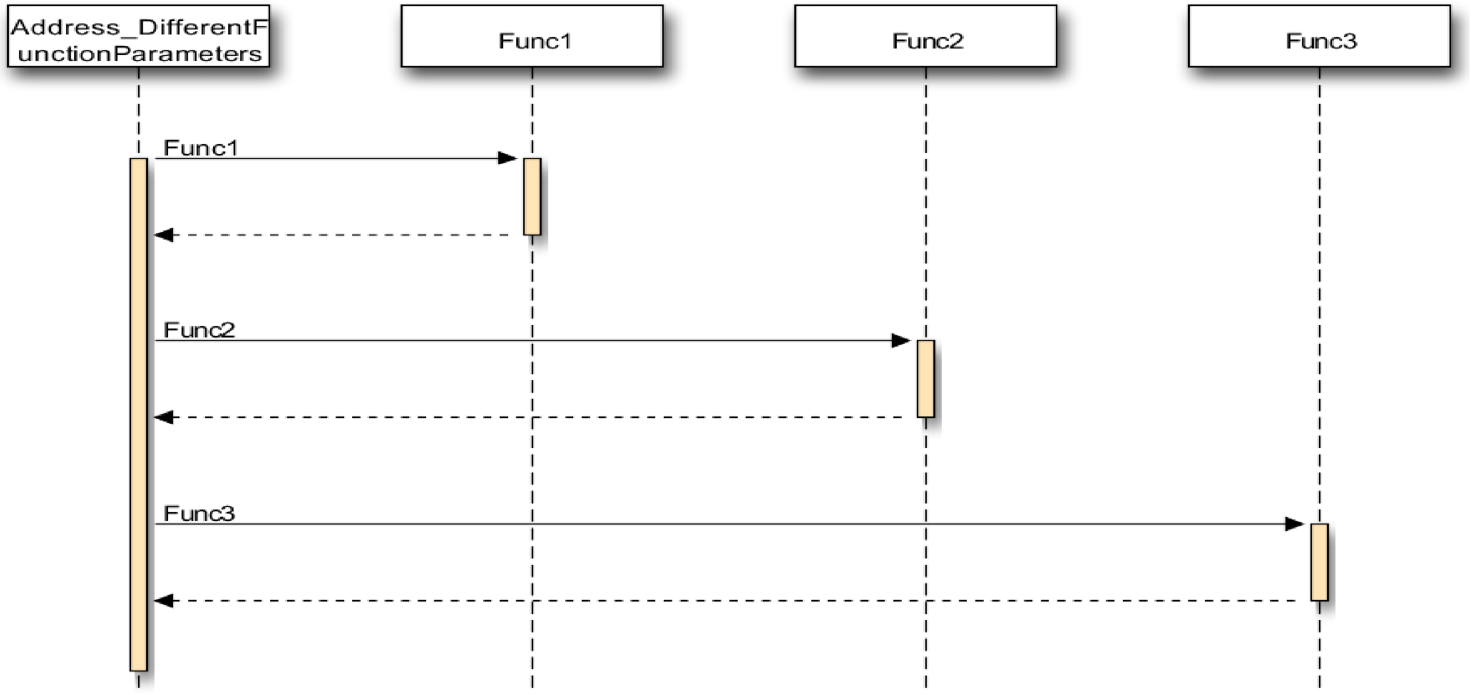
```
mr r10,r4
mr r9,r3
stw r9,10(r31)
stw r10,14(r31)
 gx=0;
lis r9,40010000
li r10,00
stw r10,-601C(r9)
  x=Func1(5); //+c[1];
li r3,05
```

Charts

Sequence Diagram

A Sequence diagram is an interaction diagram that shows how processes operate with one another and what is their order. A sequence diagram shows object interactions arranged in time sequence.

The information is based on the profiled data(time measurement) and shows what was executed and when during the test execution.



Sequence diagrams are typically associated with use case realizations in the Logical View of the system under development. Sequence diagrams are sometimes called event diagrams or event scenarios.

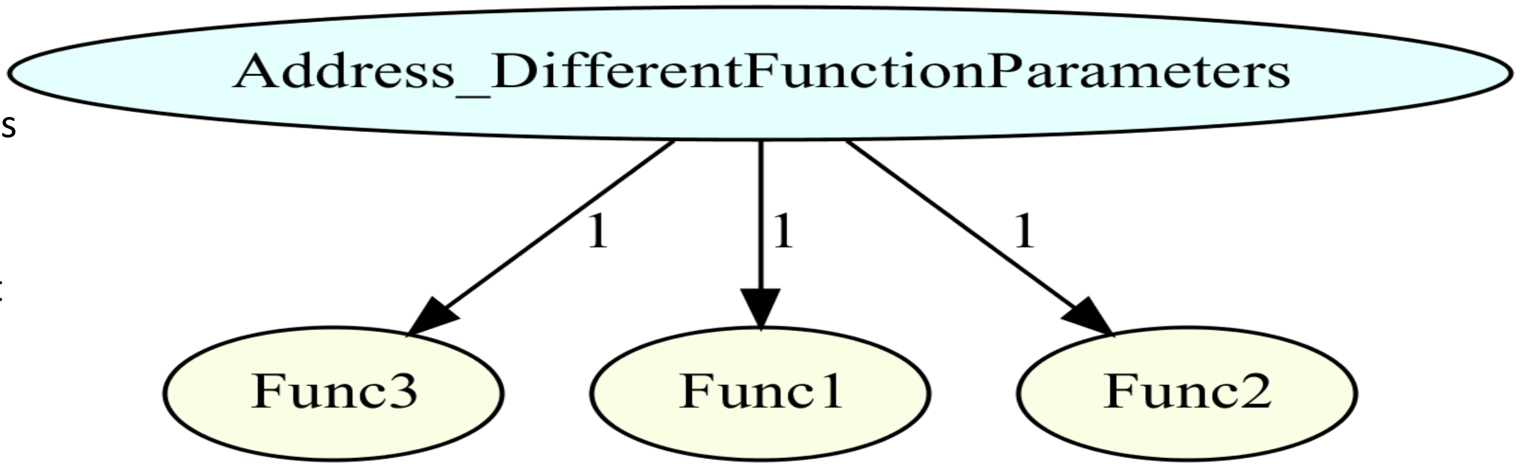
Charts

Call Graph

A call graph is a directed graph that represents calling relationships between subroutines in a computer program.

Call graphs are a basic program analysis result that can be used for human understanding of programs, or as a basis for further analyses, such as an analysis that tracks the flow of values between procedures.

One simple application of call graphs is finding procedures that are never called.



Call graphs can be dynamic or static. A dynamic call graph is a record of an execution of the program, e.g., as output by a profiler. Thus, a dynamic call graph can be exact, but only describes one run of the program.

In our case the information is based on the profiled data / executed code.

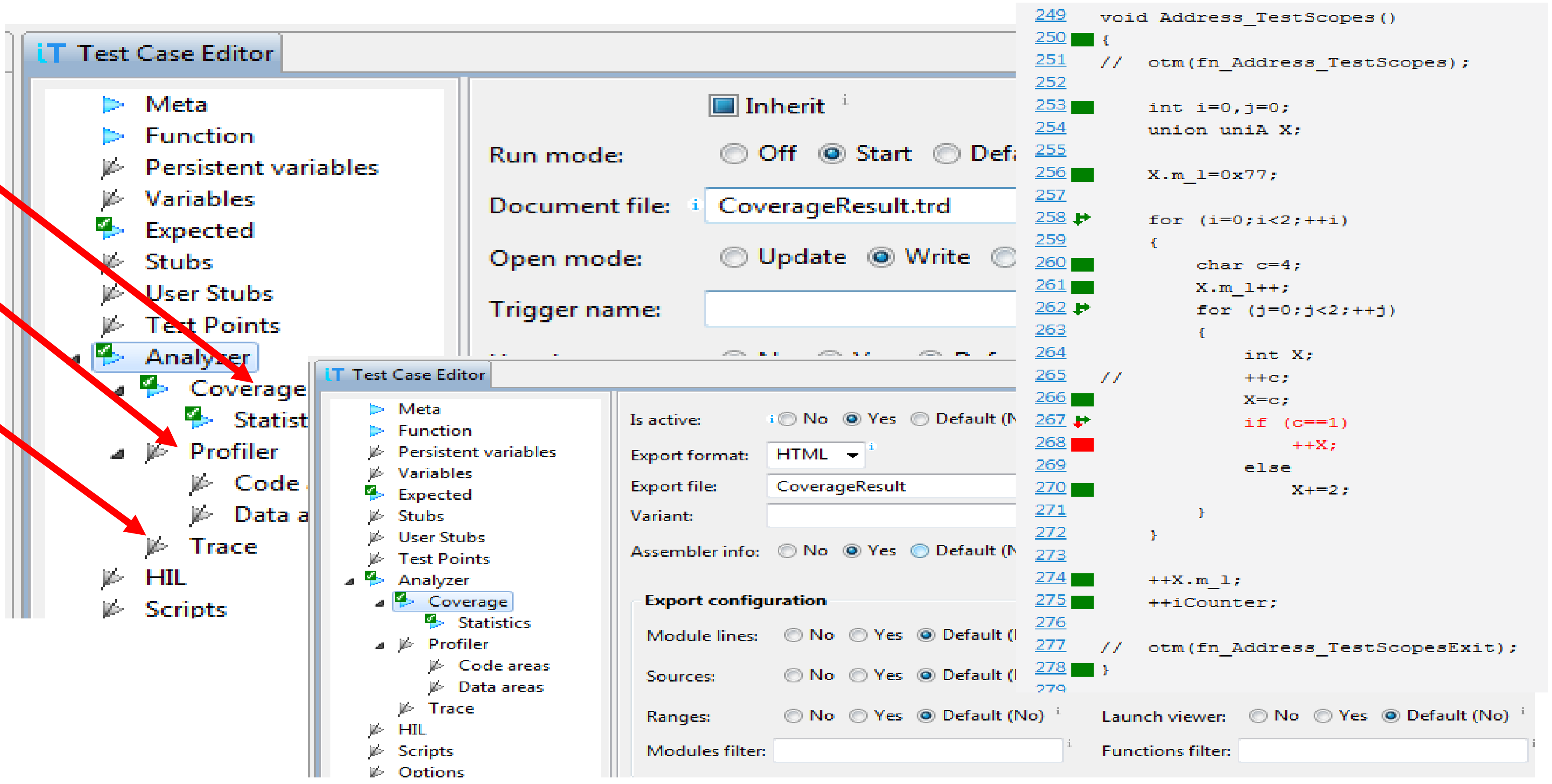
Analyzer

Any test action in testIDEA can be combined with analyzer functionality:

Coverage

Profiler

Trace



Inherit

Run mode: Off [✓ Start] Defa

Document file: CoverageResult.trd

Open mode: Update [✓ Write]

Trigger name:

Is active: No [✓ Yes] Default (N

Export format: HTML

Export file: CoverageResult

Variant:

Assembler info: No [✓ Yes] Default (N

Export configuration

Module lines: No Yes [✓ Default (]

Sources: No Yes [✓ Default (]

Ranges: No Yes [✓ Default (No)]

Launch viewer: No Yes [✓ Default (No)]

Modules filter:

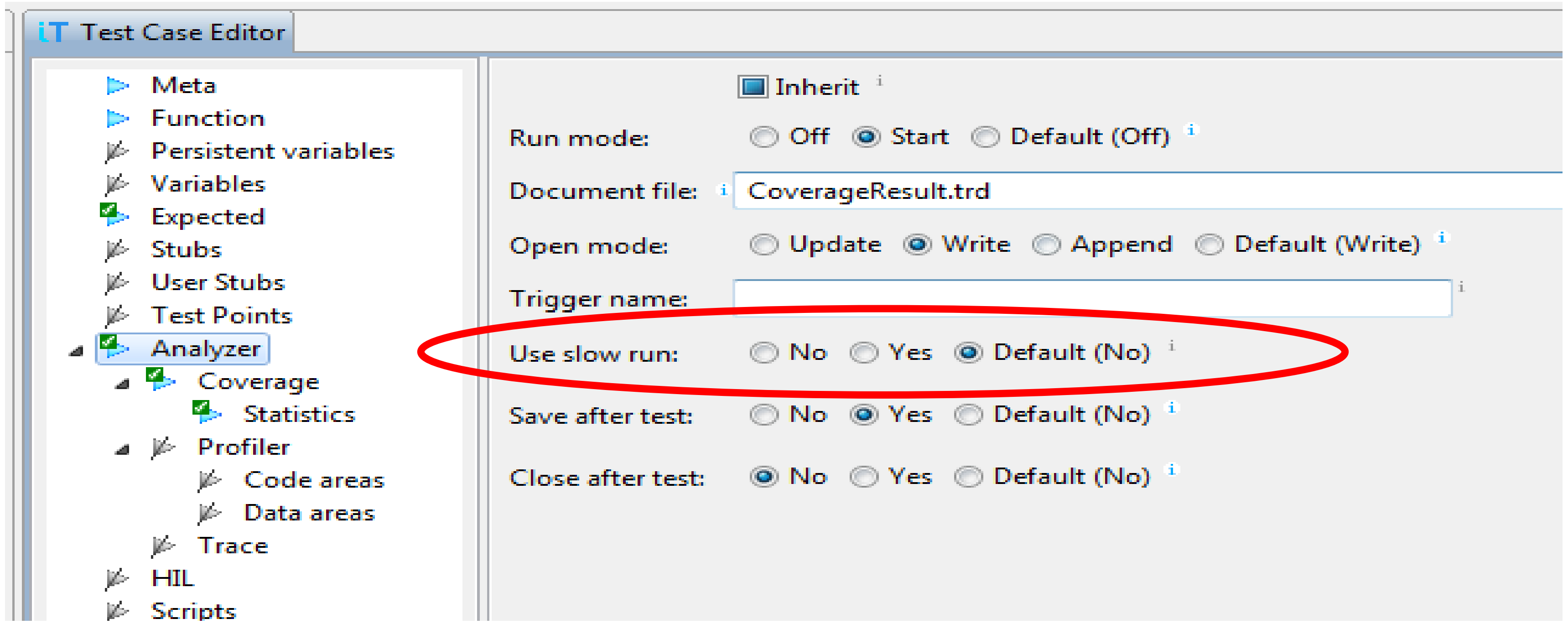
Functions filter:

```
void Address_TestScopes()
{
//   otm(fn_Address_TestScopes);

     int i=0,j=0;
     union uniA X;

     X.m_l=0x77;

     for (i=0;i<2;++i)
     {
          char c=4;
          X.m_l++;
          for (j=0;j<2;++j)
          {
               int X;
//             ++c;
               X=c;
               if (c==1)
                    ++X;
               else
                    X+=2;
          }
     }

     ++X.m_l;
     ++iCounter;

//   otm(fn_Address_TestScopesExit);
}
```

e.g. Nexus @ Power Architecture, ETM @ ARM, ICE

Analyzer

Even without trace possibility on the microcontroller, you still have analyzer possibilities:



Inherit

Run mode: Off [✓ Start] Default (Off)

Document file: CoverageResult.trd

Open mode: Update [✓ Write] Append Default (Write)

Trigger name:

Use slow run: No Yes [✓ Default (No)]

Save after test: No [✓ Yes] Default (No)

Close after test: [✓ No] Yes Default (No)

e.g. S08, S12, ..

indes
The choice of professionals

Code Coverage Functionality

iSYSTEM
EMBEDDED TOOLS FOR SINGLE AND MULTI-CORE
Software Development, Analysis, Test Automation and Certification

Today

On-Chip Trace

- ETB / OTB = Embedded / Onchip Trace Buffer
  - Limited buffer size (4KB to 1 MB)
  - Limited length of recording
- Trace Port (ETM, Nexus, Aurora)
  - Limitation depends on external hardware
  - Long term recording
  - Bandwidth (4-32Bit, LVDS signaling)

Sampling Information in the Trace Buffer I

- A good trace system requires
  - An (emulation/umbrella) controller that provides visibility of the internal memory bus to the outside world (via dedicated trace bus or message port)
  - A trace buffer onchip or inside the debug system that stores the information from the controller and other sources in real-time and for a long time
  - A trigger that starts recording at dedicated conditions
  - A qualifier / filter that starts / stops recording in certain memory areas
  - Store the information in a file for documentation capabilities

Sampling Information in the Trace Buffer II

Same technique and buffer is used for different purposes

- Trace: Record complete instruction flow
- Profiling: Record complete instruction flow and show function timing
  mixed with data
- Coverage: Record complete instruction flow and check
  against source code and determine for every instruction if executed or not

What is Code Coverage?

Code Coverage is a measure used in software testing.

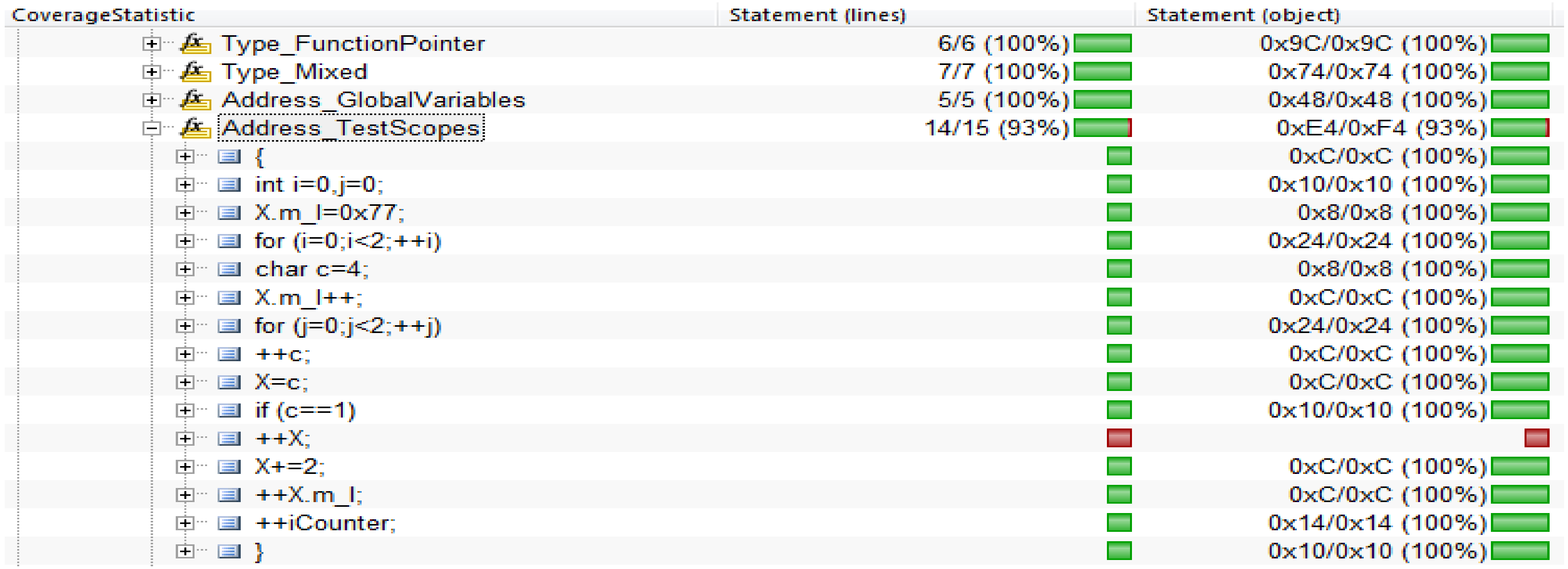
You measure

- The quality of your code
- The size of executed code by a set of test cases
- The quality of your test cases

All Coverage Results at iSYSTEM are based on executed object code!

Types of Coverage

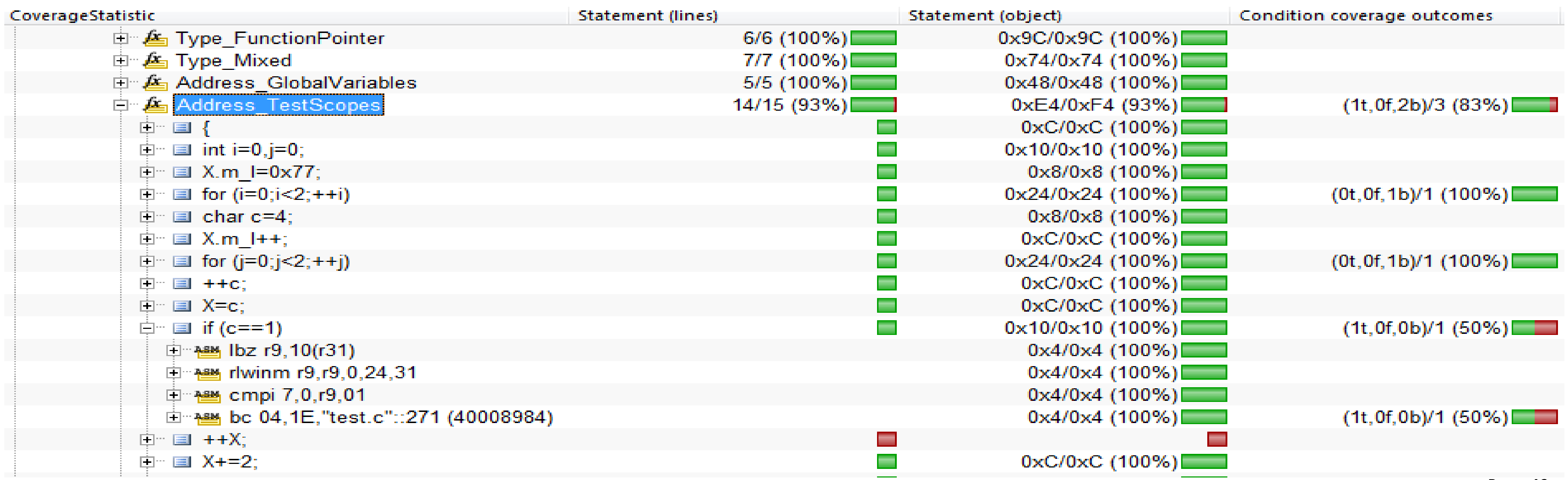
- Statement Coverage
  - marks statements or instructions as executed or not executed based on lines and based on object code size



CoverageStatistic	Statement (lines)	Statement (object)
Type_FunctionPointer	6/6 (100%) 🟩	0x9C/0x9C (100%) 🟩
Type_Mixed	7/7 (100%) 🟩	0x74/0x74 (100%) 🟩
Address_GlobalVariables	5/5 (100%) 🟩	0x48/0x48 (100%) 🟩
Address_TestScopes	14/15 (93%) 🟩	0xE4/0xF4 (93%) 🟩
{	🟩	0xC/0xC (100%) 🟩
int i=0,j=0;	🟩	0x10/0x10 (100%) 🟩
X.m_l=0x77;	🟩	0x8/0x8 (100%) 🟩
for (i=0;i<2;++i)	🟩	0x24/0x24 (100%) 🟩
char c=4;	🟩	0x8/0x8 (100%) 🟩
X.m_l++;	🟩	0xC/0xC (100%) 🟩
for (j=0;j<2;++j)	🟩	0x24/0x24 (100%) 🟩
++c;	🟩	0xC/0xC (100%) 🟩
X=c;	🟩	0xC/0xC (100%) 🟩
if (c==1)	🟩	0x10/0x10 (100%) 🟩
++X;	🟥	🟥
X+=2;	🟩	0xC/0xC (100%) 🟩
++X.m_l;	🟩	0xC/0xC (100%) 🟩
++iCounter;	🟩	0x14/0x14 (100%) 🟩
}	🟩	0x10/0x10 (100%) 🟩

Types of Coverage

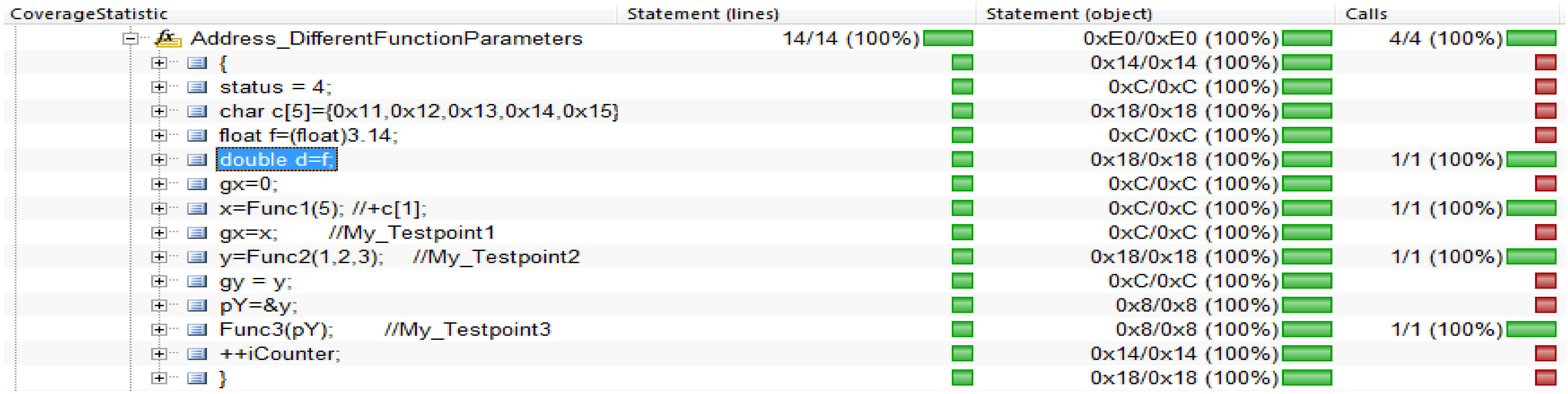
- Condition Coverage
  - Marks conditional branch instructions or conditional statements as
    - Executed
    - branch taken
    - branch not taken
    - both paths covered



CoverageStatistic	Statement (lines)	Statement (object)	Condition coverage outcomes
Type_FunctionPointer	6/6 (100%) 🟩	0x9C/0x9C (100%) 🟩	
Type_Mixed	7/7 (100%) 🟩	0x74/0x74 (100%) 🟩	
Address_GlobalVariables	5/5 (100%) 🟩	0x48/0x48 (100%) 🟩	
Address_TestScopes	14/15 (93%) 🟩	0xE4/0xF4 (93%) 🟩	(1t,0f,2b)/3 (83%) 🟩
{	🟩	0xC/0xC (100%) 🟩	
int i=0,j=0;	🟩	0x10/0x10 (100%) 🟩	
X.m_l=0x77;	🟩	0x8/0x8 (100%) 🟩	
for (i=0;i<2;++i)	🟩	0x24/0x24 (100%) 🟩	(0t,0f,1b)/1 (100%) 🟩
char c=4;	🟩	0x8/0x8 (100%) 🟩	
X.m_l++;	🟩	0xC/0xC (100%) 🟩	
for (j=0;j<2;++j)	🟩	0x24/0x24 (100%) 🟩	(0t,0f,1b)/1 (100%) 🟩
++c;	🟩	0xC/0xC (100%) 🟩	
X=c;	🟩	0xC/0xC (100%) 🟩	
if (c==1)	🟩	0x10/0x10 (100%) 🟩	(1t,0f,0b)/1 (50%) 🟩🟥
lbz r9,10(r31)		0x4/0x4 (100%) 🟩	
rlwinm r9,r9,0,24,31		0x4/0x4 (100%) 🟩	
cmpi 7,0,r9,01		0x4/0x4 (100%) 🟩	
bc 04,1E,"test.c"::271 (40008984)		0x4/0x4 (100%) 🟩	(1t,0f,0b)/1 (50%) 🟩🟥
++X;	🟥	🟥	
X+=2;	🟩	0xC/0xC (100%) 🟩	

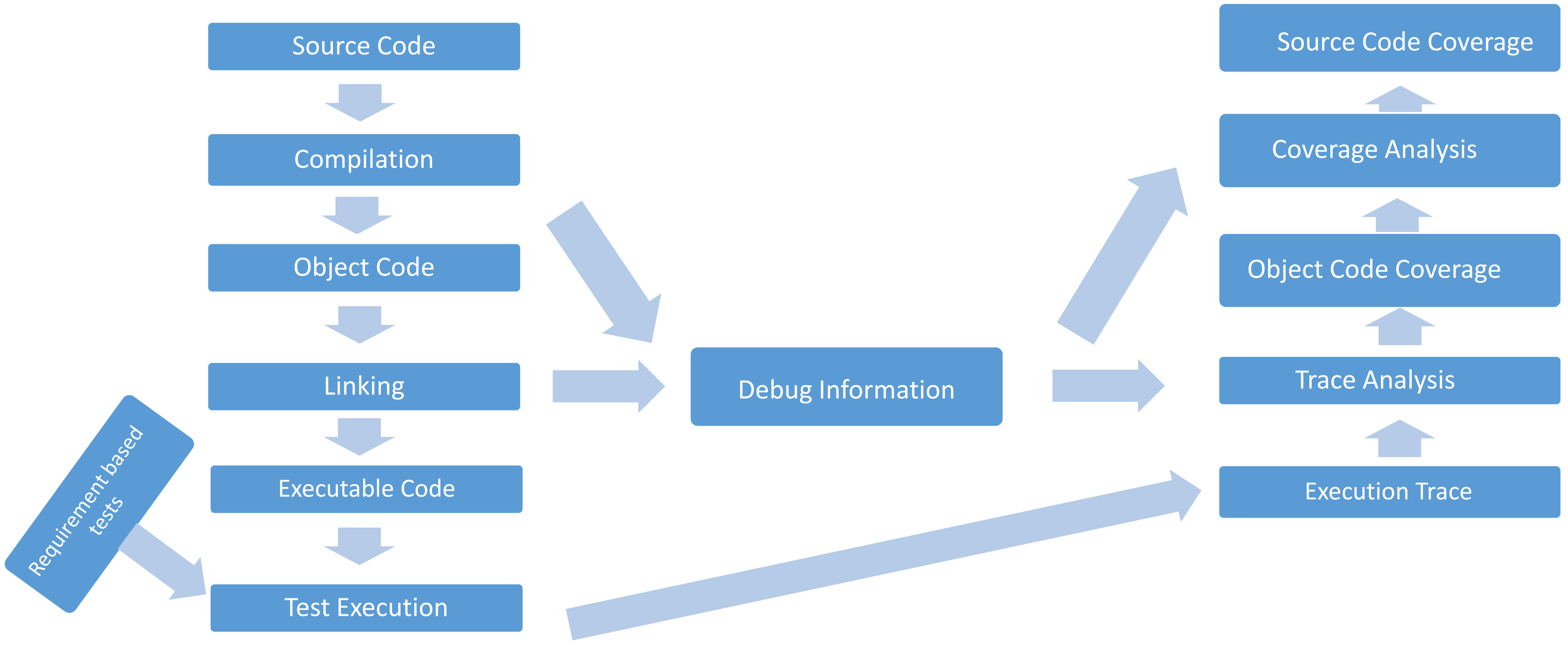
Types of Coverage

- Call Coverage
  - Each (function) call is reported



CoverageStatistic	Statement (lines)	Statement (object)	Calls
Address_DifferentFunctionParameters	14/14 (100%) 🟩	0xE0/0xE0 (100%) 🟩	4/4 (100%) 🟩
{	🟩	0x14/0x14 (100%) 🟩	🟥
status = 4;	🟩	0xC/0xC (100%) 🟩	🟥
char c[5]={0x11,0x12,0x13,0x14,0x15}	🟩	0x18/0x18 (100%) 🟩	🟥
float f=(float)3.14;	🟩	0xC/0xC (100%) 🟩	🟥
double d=f;	🟩	0x18/0x18 (100%) 🟩	1/1 (100%) 🟩
gx=0;	🟩	0xC/0xC (100%) 🟩	🟥
x=Func1(5); //+c[1];	🟩	0xC/0xC (100%) 🟩	1/1 (100%) 🟩
gx=x; //My_Testpoint1	🟩	0xC/0xC (100%) 🟩	🟥
y=Func2(1,2,3); //My_Testpoint2	🟩	0x18/0x18 (100%) 🟩	1/1 (100%) 🟩
gy = y;	🟩	0xC/0xC (100%) 🟩	🟥
pY=&y;	🟩	0x8/0x8 (100%) 🟩	🟥
Func3(pY); //My_Testpoint3	🟩	0x8/0x8 (100%) 🟩	1/1 (100%) 🟩
++iCounter;	🟩	0x14/0x14 (100%) 🟩	🟥
}	🟩	0x18/0x18 (100%) 🟩	🟥

How is Code Coverage executed?



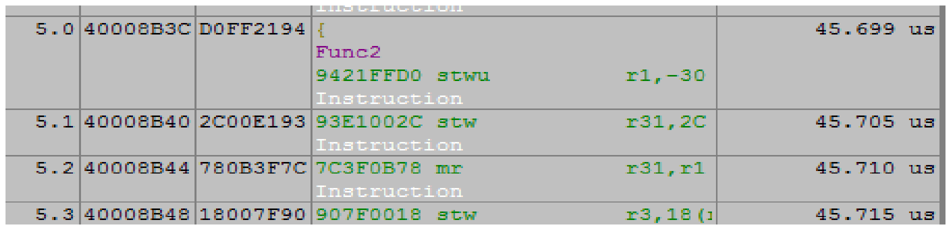
How is Code Coverage executed?

In Words:

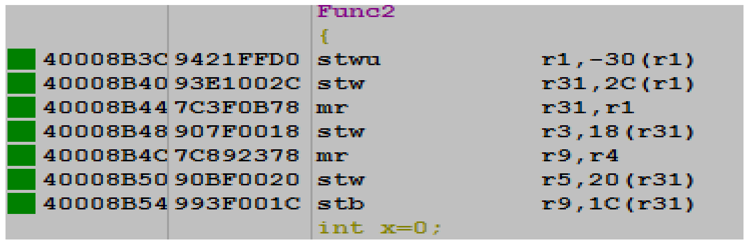
program execution + iSYSTEM tools
= program trace

program trace + disassembler information
= object code coverage

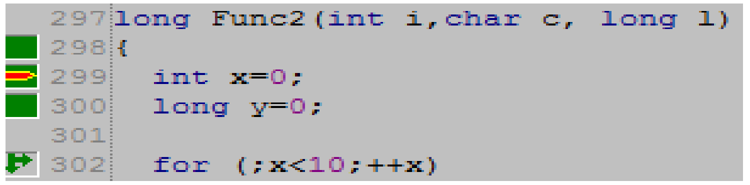
object code coverage + debug information
= source code coverage



```
5.0 40008B3C D0FF2194 {                                45.699 us
                      Func2
                      9421FFD0 stwu      r1,-30
                      Instruction
5.1 40008B40 2C00E193 93E1002C stw       r31,2C        45.705 us
                      Instruction
5.2 40008B44 780B3F7C 7C3F0B78 mr        r31,r1        45.710 us
                      Instruction
5.3 40008B48 18007F90 907F0018 stw       r3,18(        45.715 us
```



```
                    Func2
                    {
40008B3C 9421FFD0   stwu       r1,-30(r1)
40008B40 93E1002C   stw        r31,2C(r1)
40008B44 7C3F0B78   mr         r31,r1
40008B48 907F0018   stw        r3,18(r31)
40008B4C 7C892378   mr         r9,r4
40008B50 90BF0020   stw        r5,20(r31)
40008B54 993F001C   stb        r9,1C(r31)
                    int x=0;
```



```
long Func2(int i,char c, long l)
{
  int x=0;
  long y=0;

  for (;x<10;++x)
```

Code Coverage in practice

- The amount of time a coverage session can run depends on
  - The trace buffer size
  - Trace port or onchip trace buffer
  - The upload speed to the PC
  - Upload while sampling possibility
  - CPU speed / Trace Port Speed

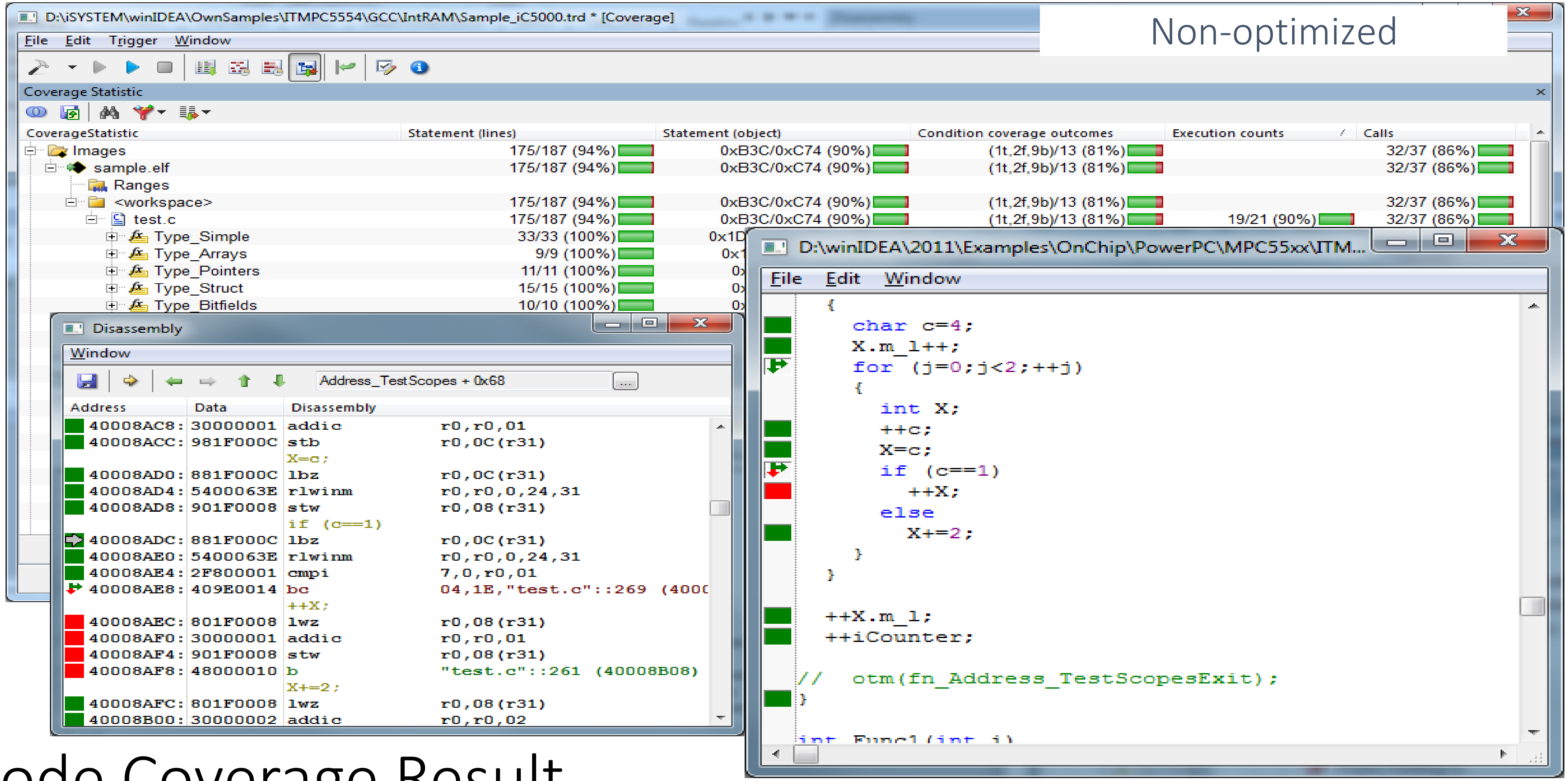
- Object code level results are mostly conclusive, except
  - Arithmetic op-codes are used instead of logical ones

    -> the conditional outcomes are undetectable (at least in real-time trace)

- Possible Limitations of Source Code Coverage

  Object code does not correspond to the source code because of
  - Compiler optimization
  - Complex conditions
  - Libraries
  - Conversion

Non-optimized



D:\iSYSTEM\winIDEA\OwnSamples\ITMPC5554\GCC\IntRAM\Sample_iC5000.trd * [Coverage]

Coverage Statistic

CoverageStatistic	Statement (lines)	Statement (object)	Condition coverage outcomes	Execution counts	Calls
Images	175/187 (94%)	0xB3C/0xC74 (90%)	(1t,2f,9b)/13 (81%)		32/37 (86%)
sample.elf	175/187 (94%)	0xB3C/0xC74 (90%)	(1t,2f,9b)/13 (81%)		32/37 (86%)
Ranges					
<workspace>	175/187 (94%)	0xB3C/0xC74 (90%)	(1t,2f,9b)/13 (81%)		32/37 (86%)
test.c	175/187 (94%)	0xB3C/0xC74 (90%)	(1t,2f,9b)/13 (81%)	19/21 (90%)	32/37 (86%)
Type_Simple	33/33 (100%)	0x1D			
Type_Arrays	9/9 (100%)	0x1			
Type_Pointers	11/11 (100%)	0x			
Type_Struct	15/15 (100%)	0x			
Type_Bitfields	10/10 (100%)	0x			

Disassembly

Address_TestScopes + 0x68

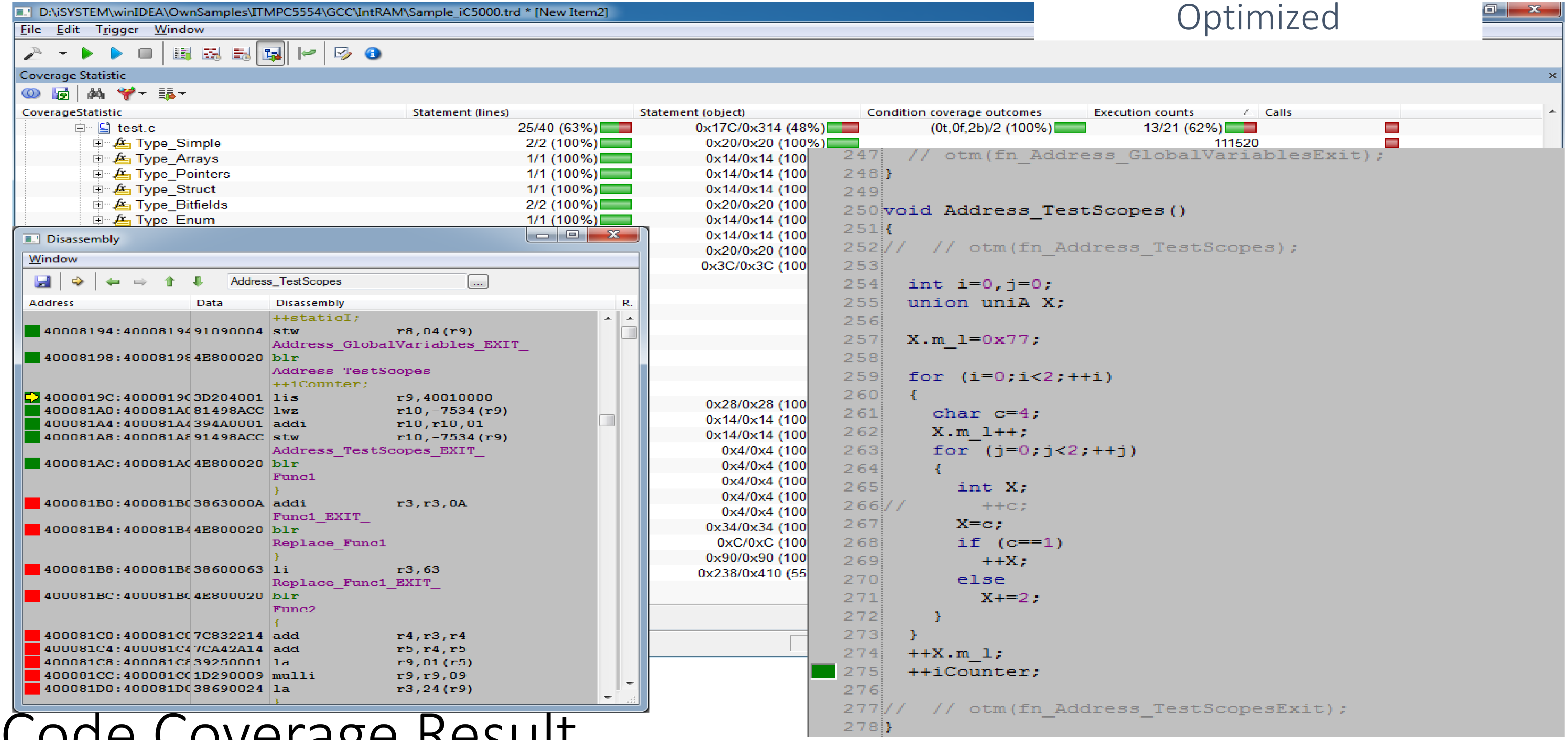
```
Address     Data      Disassembly
40008AC8: 30000001 addic        r0,r0,01
40008ACC: 981F000C stb          r0,0C(r31)
                   X=c;
40008AD0: 881F000C lbz          r0,0C(r31)
40008AD4: 5400063E rlwinm       r0,r0,0,24,31
40008AD8: 901F0008 stw          r0,08(r31)
                   if (c==1)
40008ADC: 881F000C lbz          r0,0C(r31)
40008AE0: 5400063E rlwinm       r0,r0,0,24,31
40008AE4: 2F800001 cmpi         7,0,r0,01
40008AE8: 409E0014 bc           04,1E,"test.c"::269 (4000
                   ++X;
40008AEC: 801F0008 lwz          r0,08(r31)
40008AF0: 30000001 addic        r0,r0,01
40008AF4: 901F0008 stw          r0,08(r31)
40008AF8: 48000010 b            "test.c"::261 (40008B08)
                   X+=2;
40008AFC: 801F0008 lwz          r0,08(r31)
40008B00: 30000002 addic        r0,r0,02
```

D:\winIDEA\2011\Examples\OnChip\PowerPC\MPC55xx\ITM...

```
  {
    char c=4;
    X.m_l++;
    for (j=0;j<2;++j)
    {
      int X;
      ++c;
      X=c;
      if (c==1)
        ++X;
      else
        X+=2;
    }
  }

  ++X.m_l;
  ++iCounter;

//  otm(fn_Address_TestScopesExit);
}

int Func1(int i)
```

Code Coverage Result

Optimized



D:\iSYSTEM\winIDEA\OwnSamples\ITMPC5554\GCC\IntRAM\Sample_iC5000.trd * [New Item2]
File Edit Trigger Window
Coverage Statistic
Disassembly
Window
Address_TestScopes

CoverageStatistic	Statement (lines)	Statement (object)	Condition coverage outcomes	Execution counts	Calls
test.c	25/40 (63%)	0x17C/0x314 (48%)	(0t,0f,2b)/2 (100%)	13/21 (62%)	🟥
Type_Simple	2/2 (100%)	0x20/0x20 (100%)		111520	🟥
Type_Arrays	1/1 (100%)	0x14/0x14 (100			
Type_Pointers	1/1 (100%)	0x14/0x14 (100			
Type_Struct	1/1 (100%)	0x14/0x14 (100			
Type_Bitfields	2/2 (100%)	0x20/0x20 (100			
Type_Enum	1/1 (100%)	0x14/0x14 (100			
		0x14/0x14 (100			
		0x20/0x20 (100			
		0x3C/0x3C (100			
		0x28/0x28 (100			
		0x14/0x14 (100			
		0x14/0x14 (100			
		0x4/0x4 (100			
		0x4/0x4 (100			
		0x4/0x4 (100			
		0x4/0x4 (100			
		0x4/0x4 (100			
		0x34/0x34 (100			
		0xC/0xC (100			
		0x90/0x90 (100			
		0x238/0x410 (55			

```
Address            Data      Disassembly
                             ++staticI;
40008194:40008194  91090004  stw      r8,04(r9)
                             Address_GlobalVariables_EXIT_
40008198:40008198  4E800020  blr
                             Address_TestScopes
                             ++iCounter;
4000819C:4000819C  3D204001  lis      r9,40010000
400081A0:400081A0  81498ACC  lwz      r10,-7534(r9)
400081A4:400081A4  394A0001  addi     r10,r10,01
400081A8:400081A8  91498ACC  stw      r10,-7534(r9)
                             Address_TestScopes_EXIT_
400081AC:400081AC  4E800020  blr
                             Func1
                             }
400081B0:400081B0  3863000A  addi     r3,r3,0A
                             Func1_EXIT_
400081B4:400081B4  4E800020  blr
                             Replace_Func1
                             }
400081B8:400081B8  38600063  li       r3,63
                             Replace_Func1_EXIT_
400081BC:400081BC  4E800020  blr
                             Func2
                             {
400081C0:400081C0  7C832214  add      r4,r3,r4
400081C4:400081C4  7CA42A14  add      r5,r4,r5
400081C8:400081C8  39250001  la       r9,01(r5)
400081CC:400081CC  1D290009  mulli    r9,r9,09
400081D0:400081D0  38690024  la       r3,24(r9)
                             }
```

```
  // otm(fn_Address_GlobalVariablesExit);
}

void Address_TestScopes()
{
//  // otm(fn_Address_TestScopes);

  int i=0,j=0;
  union uniA X;

  X.m_l=0x77;

  for (i=0;i<2;++i)
  {
    char c=4;
    X.m_l++;
    for (j=0;j<2;++j)
    {
      int X;
//      ++c;
      X=c;
      if (c==1)
        ++X;
      else
        X+=2;
    }
  }
  ++X.m_l;
  ++iCounter;

//  // otm(fn_Address_TestScopesExit);
}
```

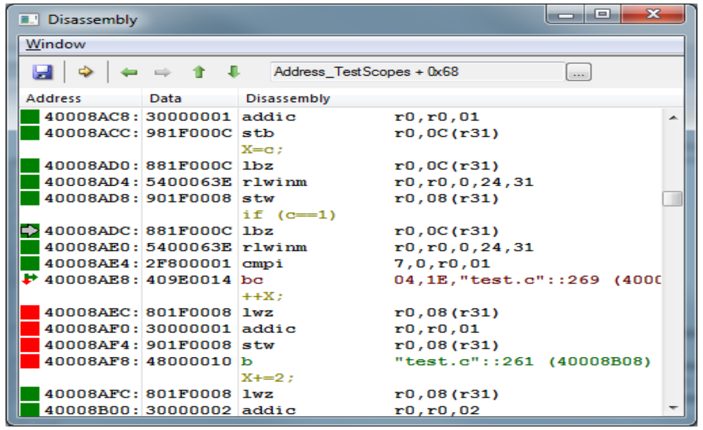
Code Coverage Result

Code Coverage Execution

- Setup is a typical debug scenario
  - winIDEA (debug IDE)
  - Blue Box for debuging
  - Target
- Trace is mandatory
  - Needed for coverage

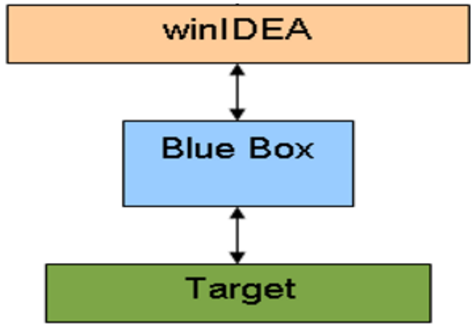
Code Coverage based on executed Op-Code

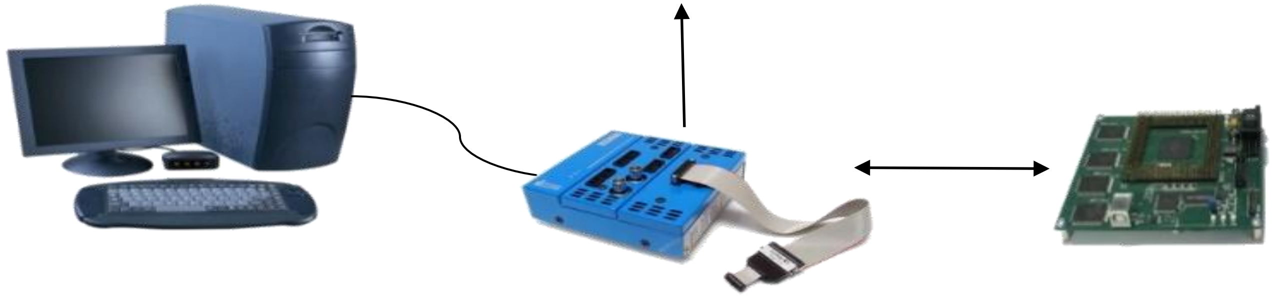
Source Code Coverage



```
Address     Data      Disassembly
🟩 40008AC8: 30000001 addic          r0,r0,01
🟩 40008ACC: 981F000C stb            r0,0C(r31)
                      X=c;
🟩 40008AD0: 881F000C lbz            r0,0C(r31)
🟩 40008AD4: 5400063E rlwinm         r0,r0,0,24,31
🟩 40008AD8: 901F0008 stw            r0,08(r31)
                      if (c==1)
   40008ADC: 881F000C lbz            r0,0C(r31)
🟩 40008AE0: 5400063E rlwinm         r0,r0,0,24,31
🟩 40008AE4: 2F800001 cmpi           7,0,r0,01
   40008AE8: 409E0014 bc             04,1E,"test.c"::269 (4000
                      ++X;
🟥 40008AEC: 801F0008 lwz            r0,08(r31)
🟥 40008AF0: 30000001 addic          r0,r0,01
🟥 40008AF4: 901F0008 stw            r0,08(r31)
🟥 40008AF8: 48000010 b              "test.c"::261 (40008B08)
                      X+=2;
🟩 40008AFC: 801F0008 lwz            r0,08(r31)
🟩 40008B00: 30000002 addic          r0,r0,02
```

Debug Information





Summary

Code Coverage

- Features
  - Analysis of the trace buffer and comparison to the source code, specifically for all addresses executed while application is running (execution coverage)
  - Creates reports to be used for certification documents
- Usage
  - Statement coverage (executed / not executed)
  - Condition coverage (analysis of conditional branch instructions and statements)
  - Call Coverage (analysis of call instructions)
  - Off-line operation mode (analysis after recording)
- Benefits
  - Coverage of NON-INSTRUMENTED and optimized code over a long period of time
  - Identify “dead” code
  - Save, and RESTART or CONTINUE coverage sessions
  - Report/Export format: HTML, XML, text
  - Remote control and automate coverage sessions with iSYSTEM’s API

indes
The choice of professionals

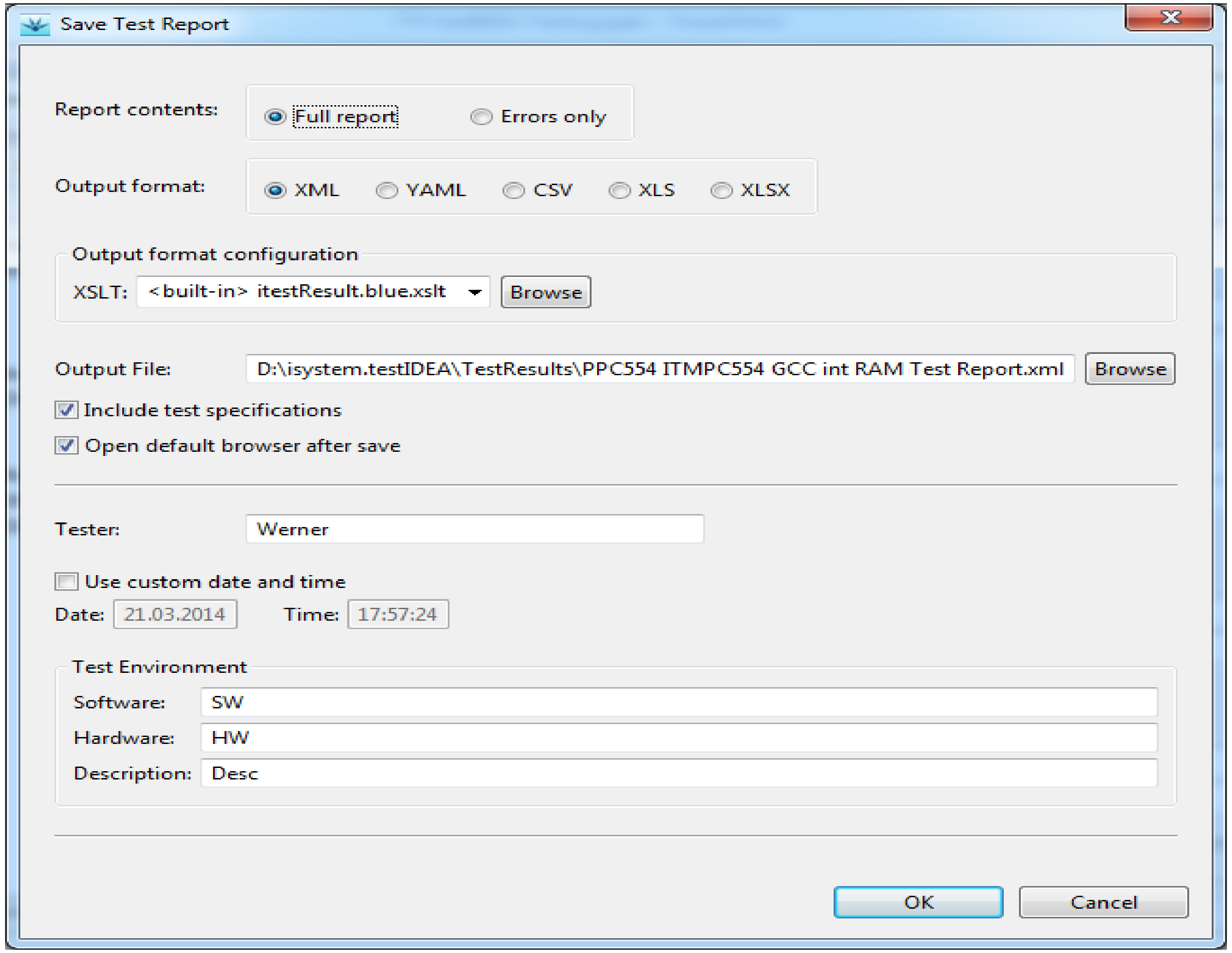
Test Report Generation

iSYSTEM

EMBEDDED TOOLS FOR SINGLE AND MULTI-CORE

Software Development, Analysis, Test Automation and Certification

Test Report Generation


Save Test Report

Report contents: [✓ Full report] Errors only

Output format: [✓ XML] YAML CSV XLS XLSX

Output format configuration

XSLT: <built-in> itestResult.blue.xslt Browse

Output File: D:\isystem.testIDEA\TestResults\PPC554 ITMPC554 GCC int RAM Test Report.xml Browse

[✓] Include test specifications

[✓] Open default browser after save

Tester: Werner

[ ] Use custom date and time

Date: 21.03.2014 Time: 17:57:24

Test Environment

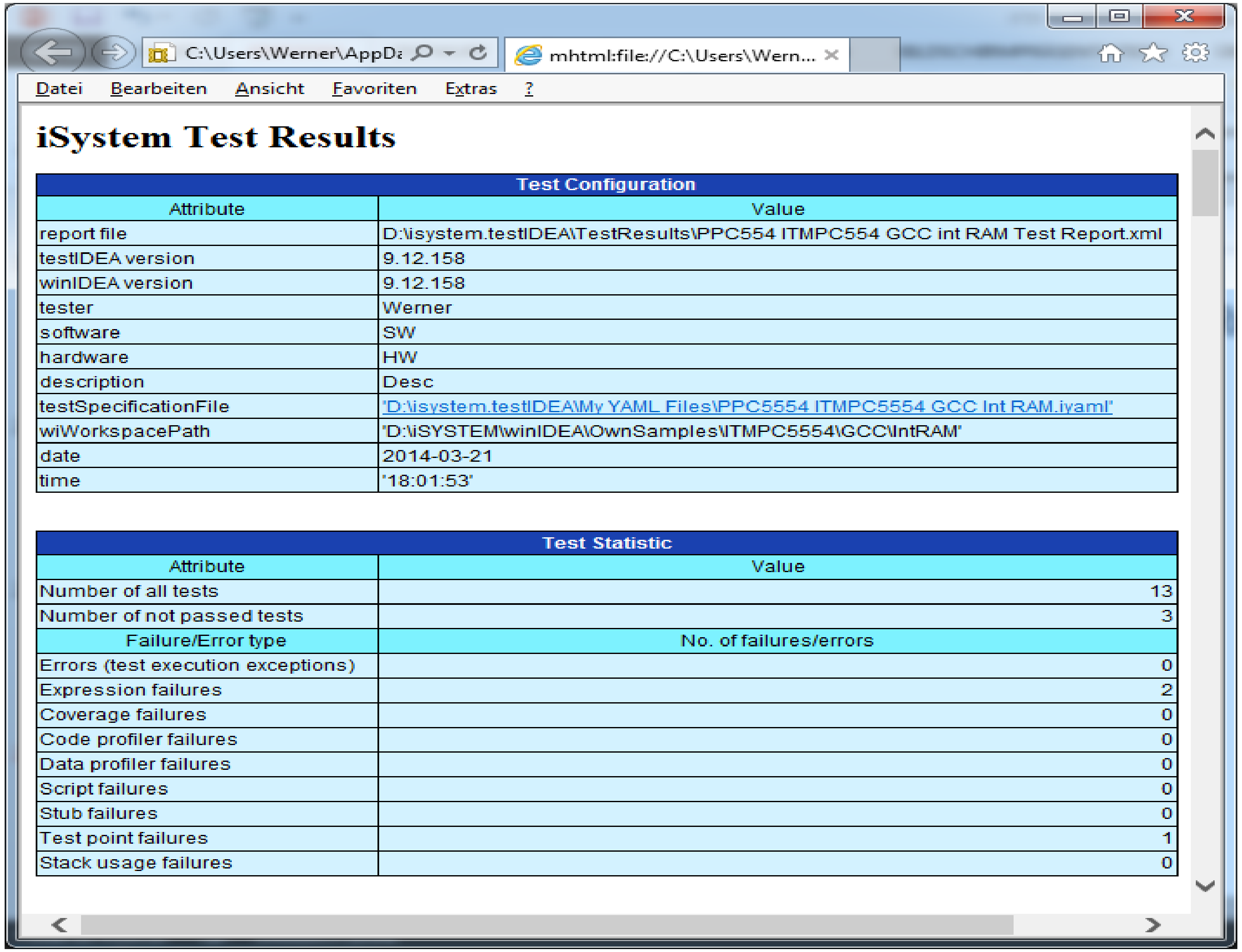
Software: SW

Hardware: HW

Description: Desc

OK Cancel

Test Report - Overview

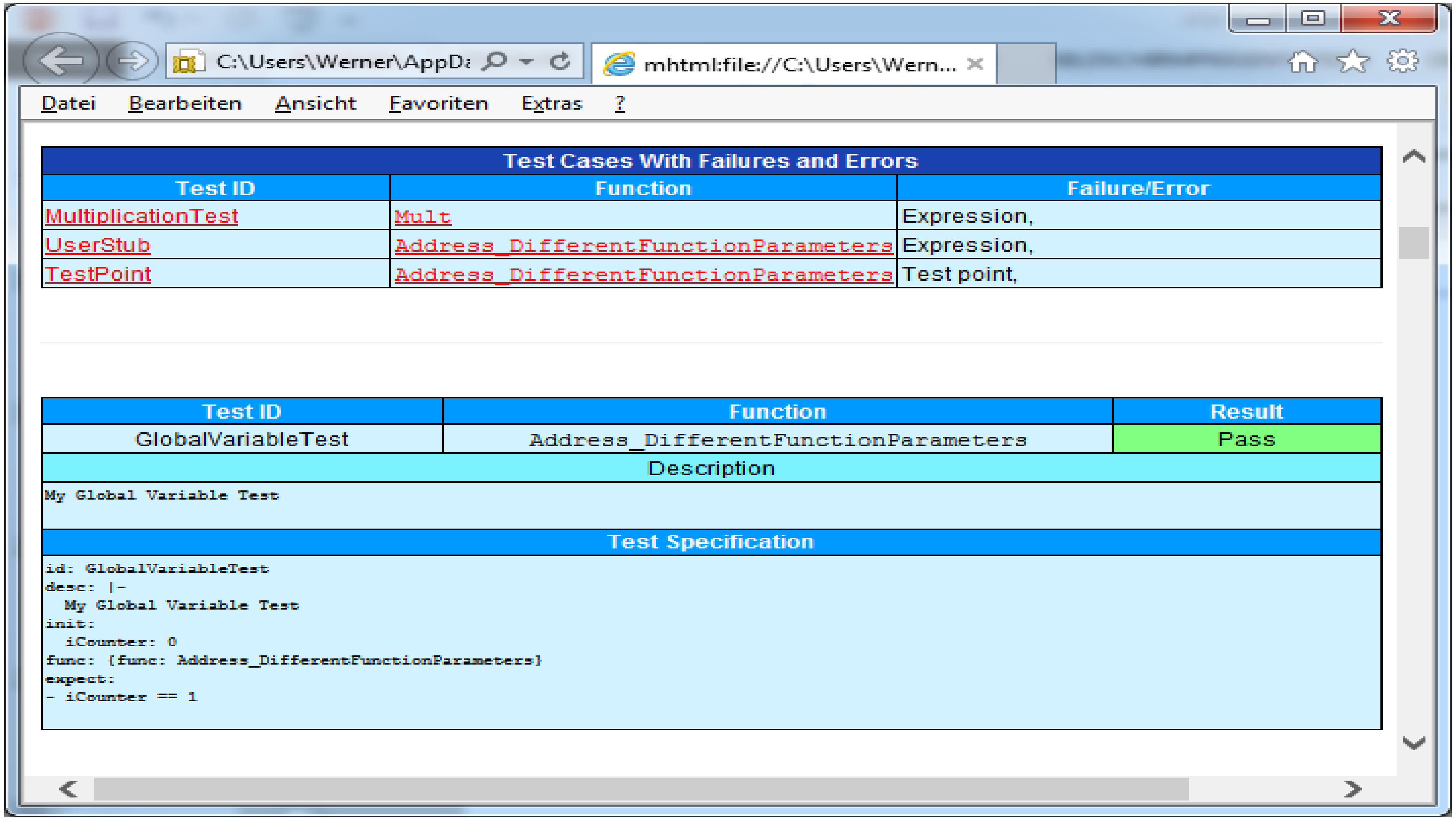


iSystem Test Results

Test Configuration	
Attribute	Value
report file	D:\isystem.testIDEA\TestResults\PPC554 ITMPC554 GCC int RAM Test Report.xml
testIDEA version	9.12.158
winIDEA version	9.12.158
tester	Werner
software	SW
hardware	HW
description	Desc
testSpecificationFile	'D:\isystem.testIDEA\My YAML Files\PPC5554 ITMPC5554 GCC Int RAM.iyaml'
wiWorkspacePath	'D:\iSYSTEM\winIDEA\OwnSamples\ITMPC5554\GCC\IntRAM'
date	2014-03-21
time	'18:01:53'

Test Statistic	
Attribute	Value
Number of all tests	13
Number of not passed tests	3
Failure/Error type	No. of failures/errors
Errors (test execution exceptions)	0
Expression failures	2
Coverage failures	0
Code profiler failures	0
Data profiler failures	0
Script failures	0
Stub failures	0
Test point failures	1
Stack usage failures	0

Test Report – Individual Results


C:\Users\Werner\AppD
mhtml:file://C:\Users\Wern...
Datei Bearbeiten Ansicht Favoriten Extras ?

Test Cases With Failures and Errors		
Test ID	**Function**	**Failure/Error**
MultiplicationTest	Mult	Expression,
UserStub	Address_DifferentFunctionParameters	Expression,
TestPoint	Address_DifferentFunctionParameters	Test point,

Test ID	Function	Result
GlobalVariableTest	Address_DifferentFunctionParameters	Pass

Description

```
My Global Variable Test
```

Test Specification

```
id: GlobalVariableTest
desc: |-
  My Global Variable Test
init:
  iCounter: 0
func: {func: Address_DifferentFunctionParameters}
expect:
- iCounter == 1
```

iSYSTEM

EMBEDDED TOOLS FOR SINGLE AND MULTI-CORE

Software Development, Analysis, Test Automation and Certification

Do not hestitate to contact us for more information.

Unit Test with Code Coverage

indes

The choice of professionals

Tel: +31 – 345-545.535 / gerard@indes.com / www.indes.com