
Lauterbach Ltd



Many in this audience have to meet specifications such as –
DO178C / ISO26262 / IEC62304

For many years we have been able to provide analysis of your
software with :-

NO INSTRUMENTATION
CODE RUNNING IN REAL TIME

This includes :-

- Code coverage
- Best and worst case timings
- Core loading
- Cache usage

From mid 2014 we will offer a tool kit you can use to produce reports for most safety specifications :-

- A manual on how to use the tools to produce the reports
- Tool qualification details for each safety standard
- Operational requirements
- Tool verification and validation cases
- Records of test results

address	tree	coverage	executed	0%	50%	100	taken	nottaken	bytes	ok
P:4001005C--40010077	⊖ e500_init	never	0.000%				0.	0.	28.	0.
P:4001005C--40010077	⊕ usr_init	never	0.000%				0.	0.	28.	0.
P:4001039C--4001065B	⊖ __setupMMU	never	0.000%				0.	0.	704.	0.
P:4001039C--4001065B	⊕ __initElf	never	0.000%				0.	0.	704.	0.
P:4001065C--40010697	⊖ __setupExtBusInt	never	0.000%				0.	0.	60.	0.
P:4001065C--40010697	⊕ __initExtBusInterface	never	0.000%				0.	0.	60.	0.
P:40010698--400106CF	⊖ __setupETPU2	never	0.000%				0.	0.	56.	0.
P:40010698--400106CB	⊕ __initETPU1	never	0.000%				0.	0.	52.	0.
P:400106CC--400106CF	⊕ __initETPU2	never	0.000%				0.	0.	4.	0.
	none									
P:40010000--40010027	⊖ __ppc_eabi_init									
P:40010028--4001005B	⊕ __init_hardware	never	0.000%				0.	0.	40.	0.
P:40010028--4001005B	⊕ __flush_cache	never	0.000%				0.	0.	52.	0.
P:400106D0--400106EF	⊕ __init_user	ok	100.000%	████████████████████			0.	0.	32.	32.
P:400106F0--4001073B	⊕ __init_cpp	partial	73.684%	██████████████████			0.	1.	76.	56.
P:4001073C--40010787	⊕ __fini_cpp	never	0.000%				0.	0.	76.	0.
P:40010788--400107AB	⊕ exit	never	0.000%				0.	0.	36.	0.
P:400107AC--400107AF	⊕ ExitProcess	never	0.000%				0.	0.	4.	0.
P:400107B0--400119AB	⊖ sieve	partial	97.914%	██████████████████			1.	3.	4604.	4508.
P:400107B0--400107BF	⊕ func1	ok	100.000%	████████████████████			0.	0.	16.	16.
P:400107C0--40010853	⊕ func2	ok	100.000%	████████████████████			0.	0.	148.	148.
P:40010854--400108AB	⊕ func2a	ok	100.000%	████████████████████			0.	0.	88.	88.
P:400108AC--400108F7	⊕ func2b	ok	100.000%	████████████████████			0.	0.	76.	76.
P:400108F8--400109FB	⊕ func2c	ok	100.000%	████████████████████			0.	0.	260.	260.
P:400109FC--40010A57	⊕ func2d	ok	100.000%	████████████████████			0.	0.	92.	92.
P:40010A58--40010A5F	⊕ func3	ok	100.000%	████████████████████			0.	0.	8.	8.
P:40010A60--40010A97	⊕ func4	ok	100.000%	████████████████████			0.	0.	56.	56.
P:40010A98--40010AA7	⊕ func5	ok	100.000%	████████████████████			0.	0.	16.	16.
P:40010AA8--40010B33	⊕ func6	ok	100.000%	████████████████████			0.	0.	140.	140.
P:40010B34--40010B8B	⊕ func7	ok	100.000%	████████████████████			0.	0.	136.	136.
P:40010B8C--40010E2F	⊕ func8	ok	100.000%	████████████████████			0.	0.	628.	628.
P:40010E30--40010EB7	⊕ func9	ok	100.000%	████████████████████			0.	0.	136.	136.

Code coverage overview showing object, function and source line with branches taken or not taken and lines executed or not
We now extract reports in the correct format to include in your certification package

```
B::Data.List P:0x40011488 /COV

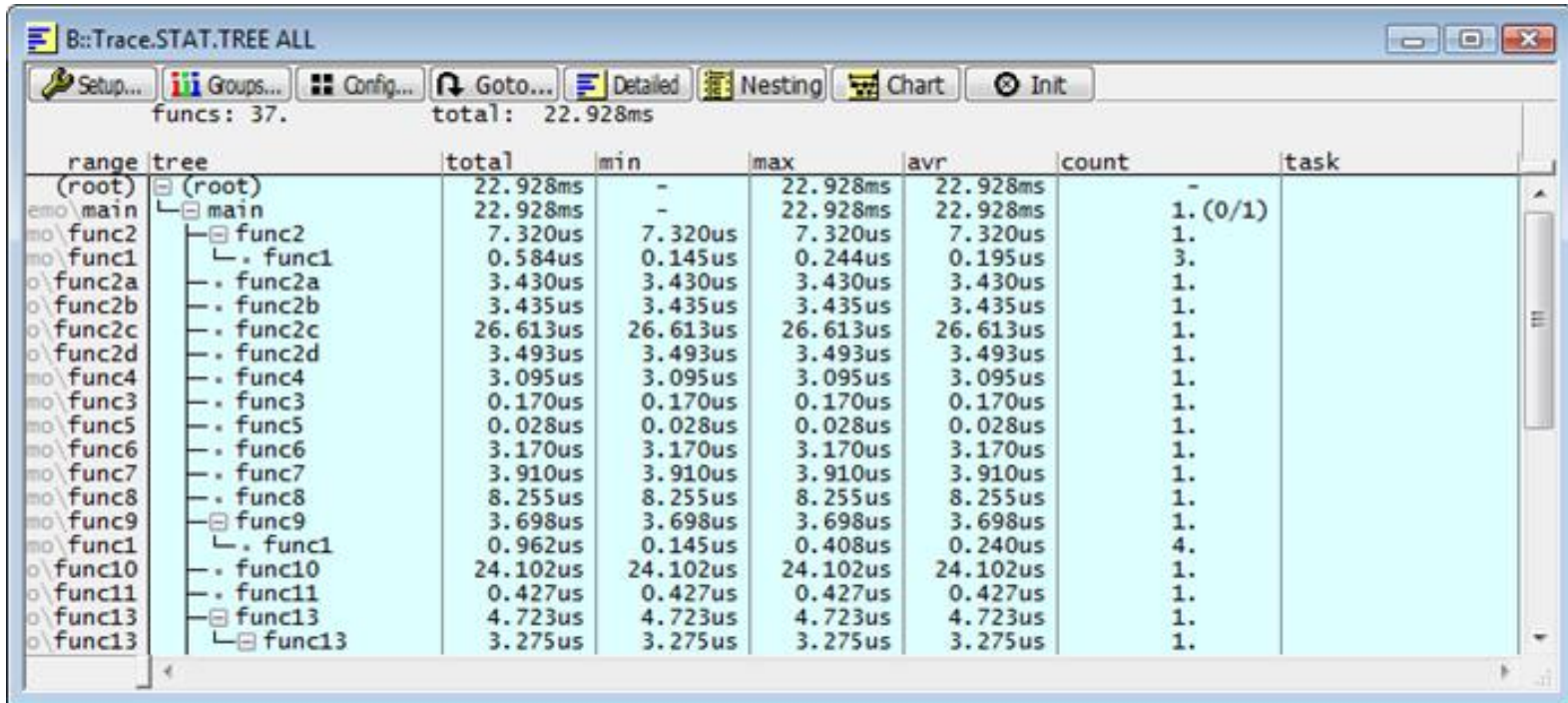
Step Over Next Return Up Go Break Mode Find:
coverage addr/line code label mnemonic comment

int func11( x ) /* multiple retu
int x;
{
    switch ( x )
    {
        func11: cmpwi r3,0x4 : x,4
        not taken SP:40011488 4182004C beq 0x400114D8 : 0x400114D
        taken SP:40011490 4080001C bge 0x400114AC : 0x400114A
        never SP:40011494 2C030002 cmpwi r3,0x2 : x,2
        never SP:40011498 41820030 beq 0x400114C8 : 0x400114C
        never SP:4001149C 40800034 bge 0x400114D0 : 0x400114D
        never SP:400114A0 2C030001 cmpwi r3,0x1 : x,1
        never SP:400114A4 40800014 bge 0x400114B8 : 0x400114B
        never SP:400114A8 48000048 b 0x400114F0
        ok SP:400114AC 2C030006 cmpwi r3,0x6 : x,6
        not taken SP:400114B0 41820038 beq 0x400114E8 : 0x400114E
        ok SP:400114B4 4800003C b 0x400114F0

        case 1:
            never 452 x = x+1;
            never SP:400114B8 38630001 addi r3,r3,0x1 : x,x,1
            never 453 x = x*2;
            never SP:400114BC 5463083C slwi r3,r3,0x1 : x,x,1
    }
}
```

The quick view whilst developing the code

Yellow code is code not executed or branches not taken – click for more detailed view



B::Trace.STAT.TREE ALL

funcs: 37. total: 22.928ms

range	tree	total	min	max	avr	count	task
(root)	(root)	22.928ms	-	22.928ms	22.928ms	-	
emo\main	└─ main	22.928ms	-	22.928ms	22.928ms	1. (0/1)	
emo\func2	└─┬─ func2	7.320us	7.320us	7.320us	7.320us	1.	
emo\func1	└─┬─┬─ func1	0.584us	0.145us	0.244us	0.195us	3.	
o\func2a	└─┬─┬─┬─ func2a	3.430us	3.430us	3.430us	3.430us	1.	
o\func2b	└─┬─┬─┬─ func2b	3.435us	3.435us	3.435us	3.435us	1.	
o\func2c	└─┬─┬─┬─ func2c	26.613us	26.613us	26.613us	26.613us	1.	
o\func2d	└─┬─┬─┬─ func2d	3.493us	3.493us	3.493us	3.493us	1.	
emo\func4	└─┬─┬─┬─ func4	3.095us	3.095us	3.095us	3.095us	1.	
emo\func3	└─┬─┬─┬─ func3	0.170us	0.170us	0.170us	0.170us	1.	
emo\func5	└─┬─┬─┬─ func5	0.028us	0.028us	0.028us	0.028us	1.	
emo\func6	└─┬─┬─┬─ func6	3.170us	3.170us	3.170us	3.170us	1.	
emo\func7	└─┬─┬─┬─ func7	3.910us	3.910us	3.910us	3.910us	1.	
emo\func8	└─┬─┬─┬─ func8	8.255us	8.255us	8.255us	8.255us	1.	
emo\func9	└─┬─┬─┬─ func9	3.698us	3.698us	3.698us	3.698us	1.	
emo\func1	└─┬─┬─┬─┬─ func1	0.962us	0.145us	0.408us	0.240us	4.	
o\func10	└─┬─┬─┬─┬─ func10	24.102us	24.102us	24.102us	24.102us	1.	
o\func11	└─┬─┬─┬─┬─ func11	0.427us	0.427us	0.427us	0.427us	1.	
o\func13	└─┬─┬─┬─┬─ func13	4.723us	4.723us	4.723us	4.723us	1.	
o\func13	└─┬─┬─┬─┬─┬─ func13	3.275us	3.275us	3.275us	3.275us	1.	

Performance analysis in numbers

Call nesting

Time in each function

Time for interrupt handlers

Time for task switching

max min and mean runtimes

The screenshot displays the Lauterbach Trace.ListNesting tool interface. The window title is "B::Trace.ListNesting". The toolbar includes buttons for Setup..., Goto..., Find..., TREE, Chart, Profile, MIPS, More, and Less. The main area shows a call stack on the left and expanded C code on the right. The call stack includes:

- record
- 02383852
- 02383851 + initADC0+0x44
- 02383851 + initPWM etpu_pwm.c\1
- 023838610 + initPWM+0x144
- 023838609 + func2 main.c\171

The expanded code for func2 is as follows:

```
void func2()
{
    int autovar;
    register int regvar;
    static int fstatic = 44; /* initialized static variable */
    static int fstatic2; /* not initialized static variable */

    autovar = regvar = fstatic;
    autovar++;
    func1( &autovar ); /* to force autovar as stack-scope */
    func1( &fstatic ); /* to force fstatic as static-scope */
    for ( regvar = 0; regvar < 5 ; regvar++ )
    for ( regvar = 0; regvar < 5 ; regvar++ )
    for ( regvar = 0; regvar < 5 ; regvar++ )
```

The timing information for each line of code is shown on the right side of the expanded code:

- 173 -023838609 . 1.833us
- 179 -023838604 . 1.153us
- 180 -023838602 . 1.167us
- 182 -023838600 . 1.847us
- 184 -02383596 . 1.660us
- 186 -02383592 . 0.170us
- 186 -02383592 . 1.657us
- 186 -02383587 . 2.180us

Expand to see high level language and timings

entTREE C:0x40000000

xs... Config... Goto... Detailed Nesting Chart

funcs: 9. total: 9.333us

range	tree	total	min	max	avr	count	total%	1%	2%	5%	10%	20%
in/func1	func1	9.333us	1.320us	1.340us	1.333us	7.	100.000%					
in/func2	└─ func2	4.020us	1.340us	1.340us	1.340us	3.	43.071%					
in/main	└─ main	4.020us	1.340us	1.340us	1.340us	3.	43.071%					
└─ start	└─ ── start	4.020us	1.340us	1.340us	1.340us	3.	43.071%					
(root)	└─ ── (root)	4.020us	1.340us	1.340us	1.340us	3.	43.071%					
in/func9	└─ func9	5.313us	1.320us	1.340us	1.328us	4.	56.928%					
in/main	└─ main	5.313us	1.320us	1.340us	1.328us	4.	56.928%					
└─ start	└─ ── start	5.313us	1.320us	1.340us	1.328us	4.	56.928%					
(root)	└─ ── (root)	5.313us	1.320us	1.340us	1.328us	4.	56.928%					

Showing the call tree for each function with timings and ratios

What do you need ? A core with off chip trace

With the right tools it can provide full code history from reset or start and record for hours or days

Most core types have variants with trace ports

ARM, MPC, Qorivva, QoriQ, Tricore, Mips, Atom, SH, etc