

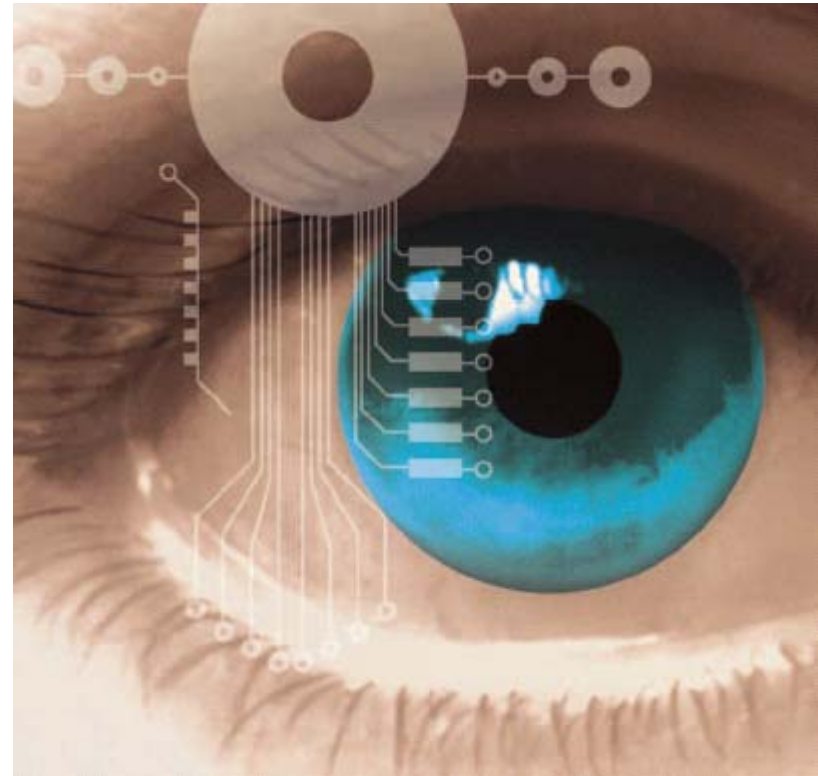


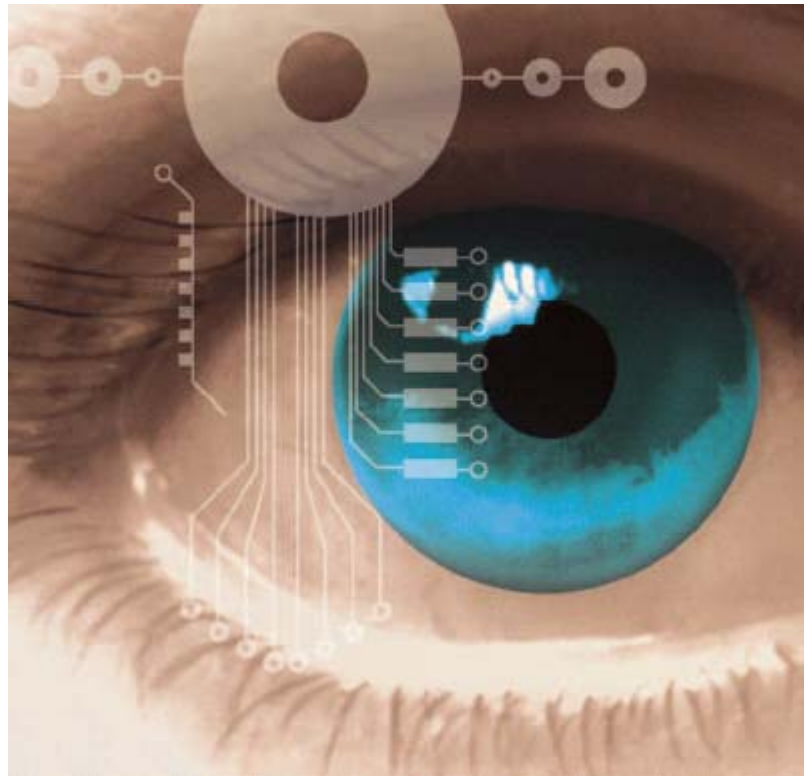
IAR WORKBENCH FOR 8051
PART3

CONTENTS
Creating Project
Profiling
Interrupt Simulation

Advance debugging

1. Profiling
2. Code coverage
3. Breakpoints
4. Tracing
5. Simulation of interrupts

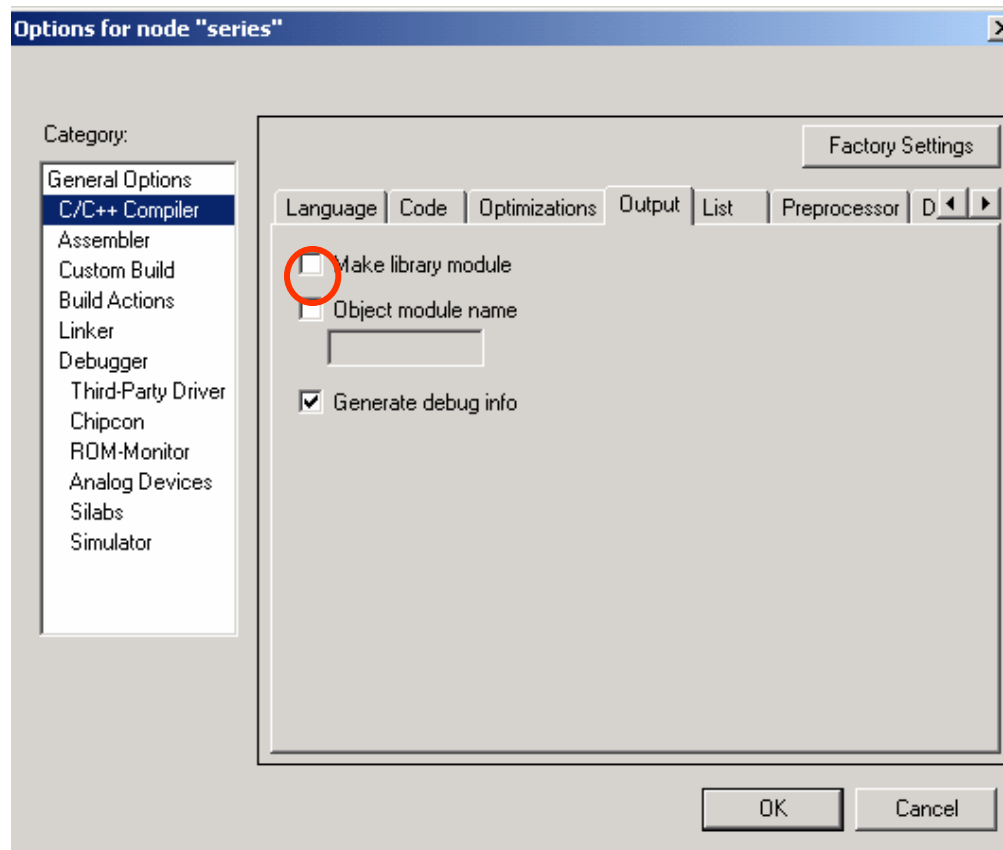




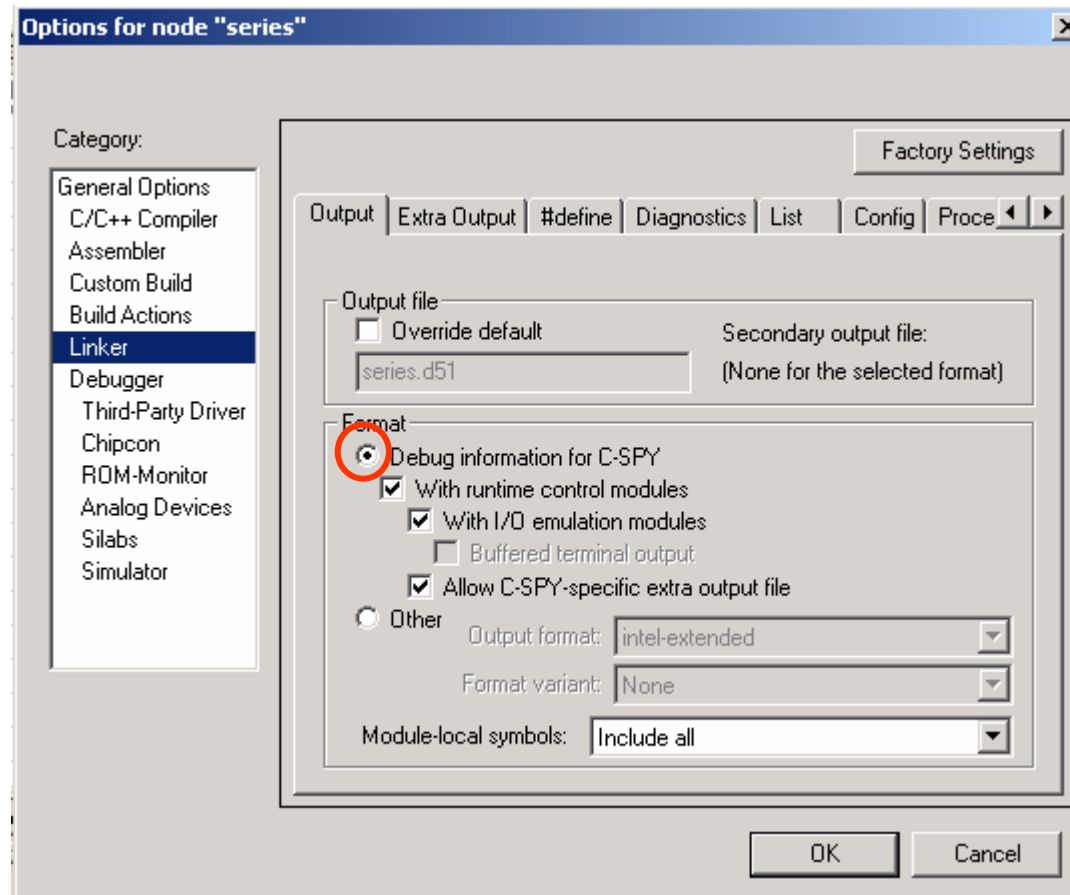
C- SPY debugger

Must Remember 1/2

Output file should have debug information



Linker Format: debugging information for debugging



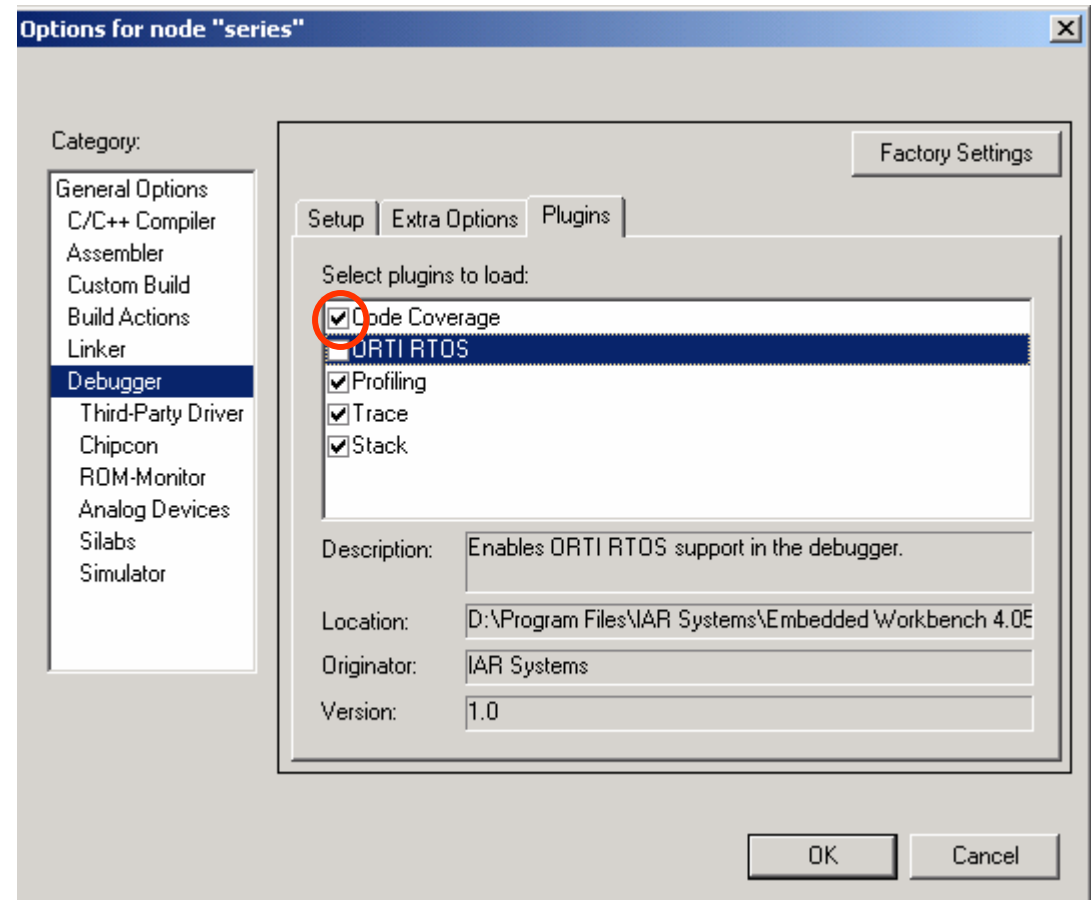
Profiling

- Display time consumed in each function
- So that user can check, which function is taking more CPU time and identify bottle neck problem,
- This will help to make code fast and

Function	Calls	Flat Tim...	Flat Time (%)	Accumu...	Accumulated Time (%)
Outside main	0	0		0	
__low_level_init	0	0		0	
do_foreground_process	1	83		296	
get_fib	1	26		26	
init_fib	0	0		0	
main	0	2		298	
next_counter	1	10		10	
put_fib	1	169		177	
putchar	2	8		8	

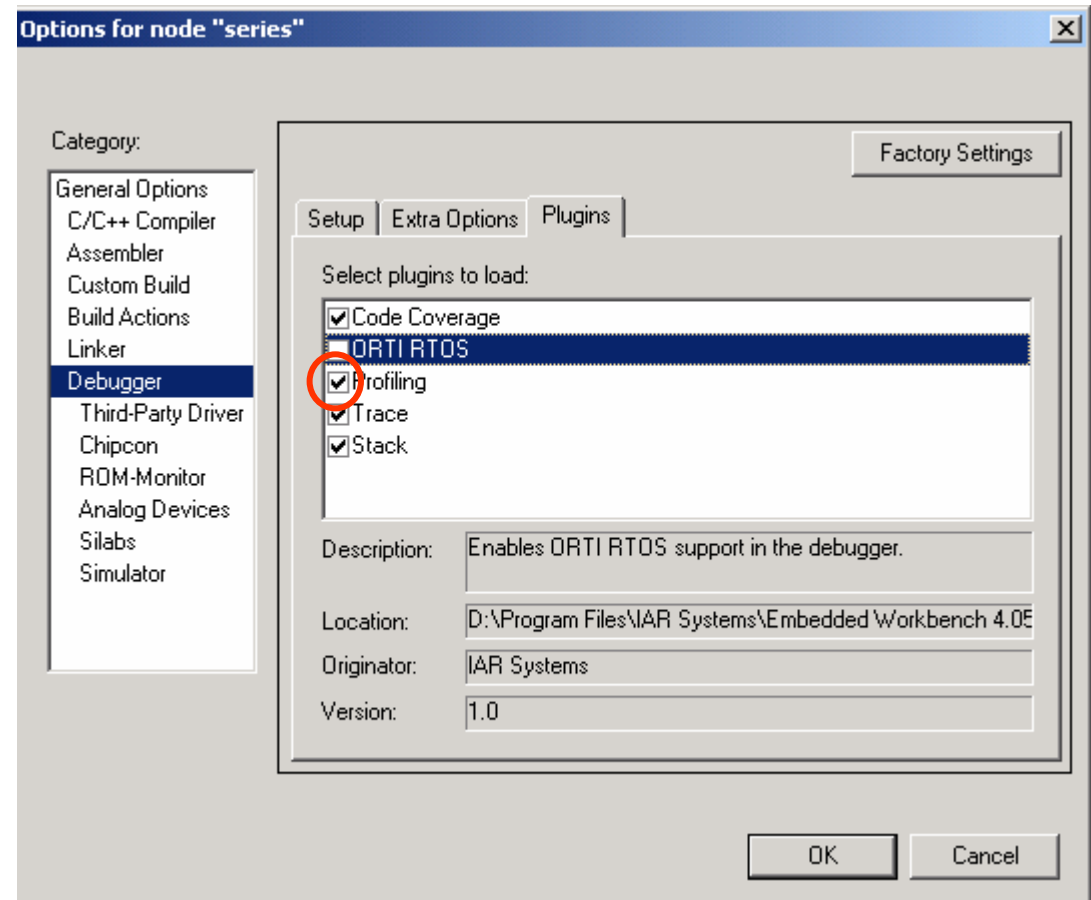
Profiling/Code Coverage/Trace

- These are to be loaded as plugins
- So enable them from
- General options > Debugger > Plugins



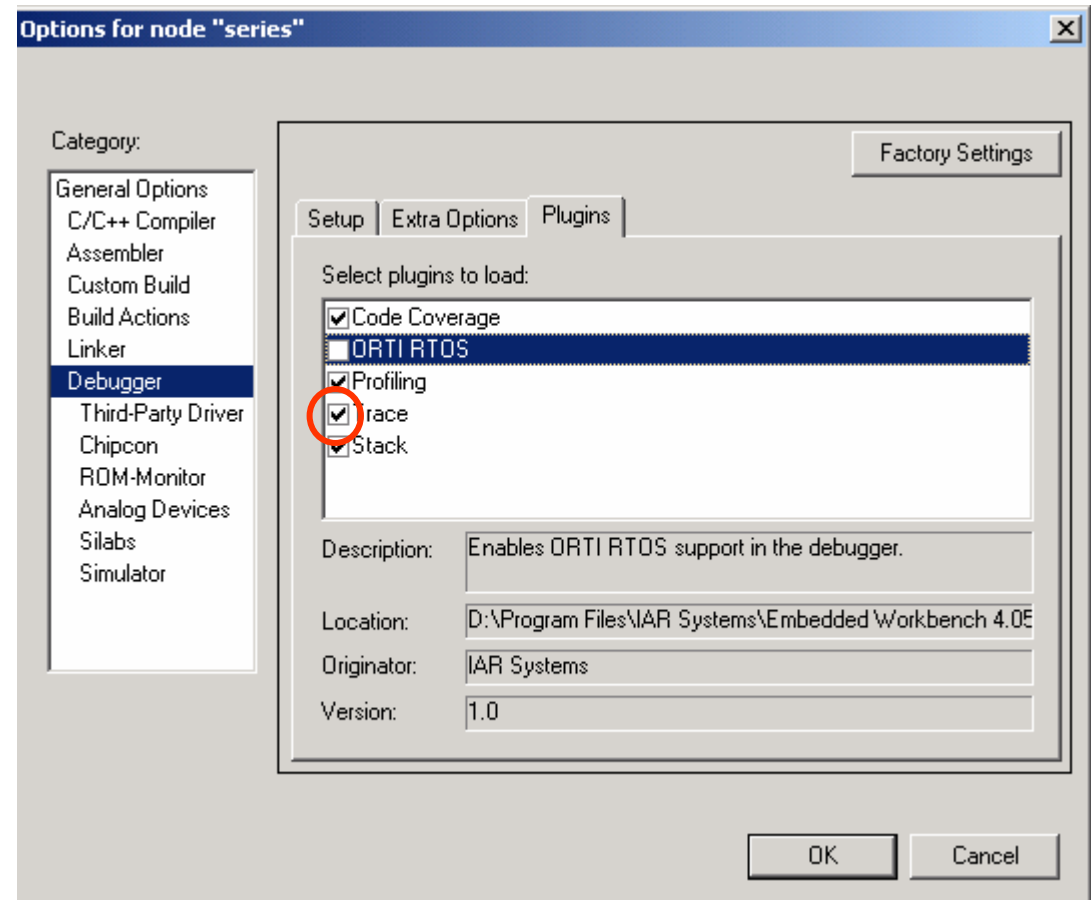
Profiling/Code Coverage/Trace

- These are to be loaded as plugins
- So enable them from
- General options > Debugger > Plugins

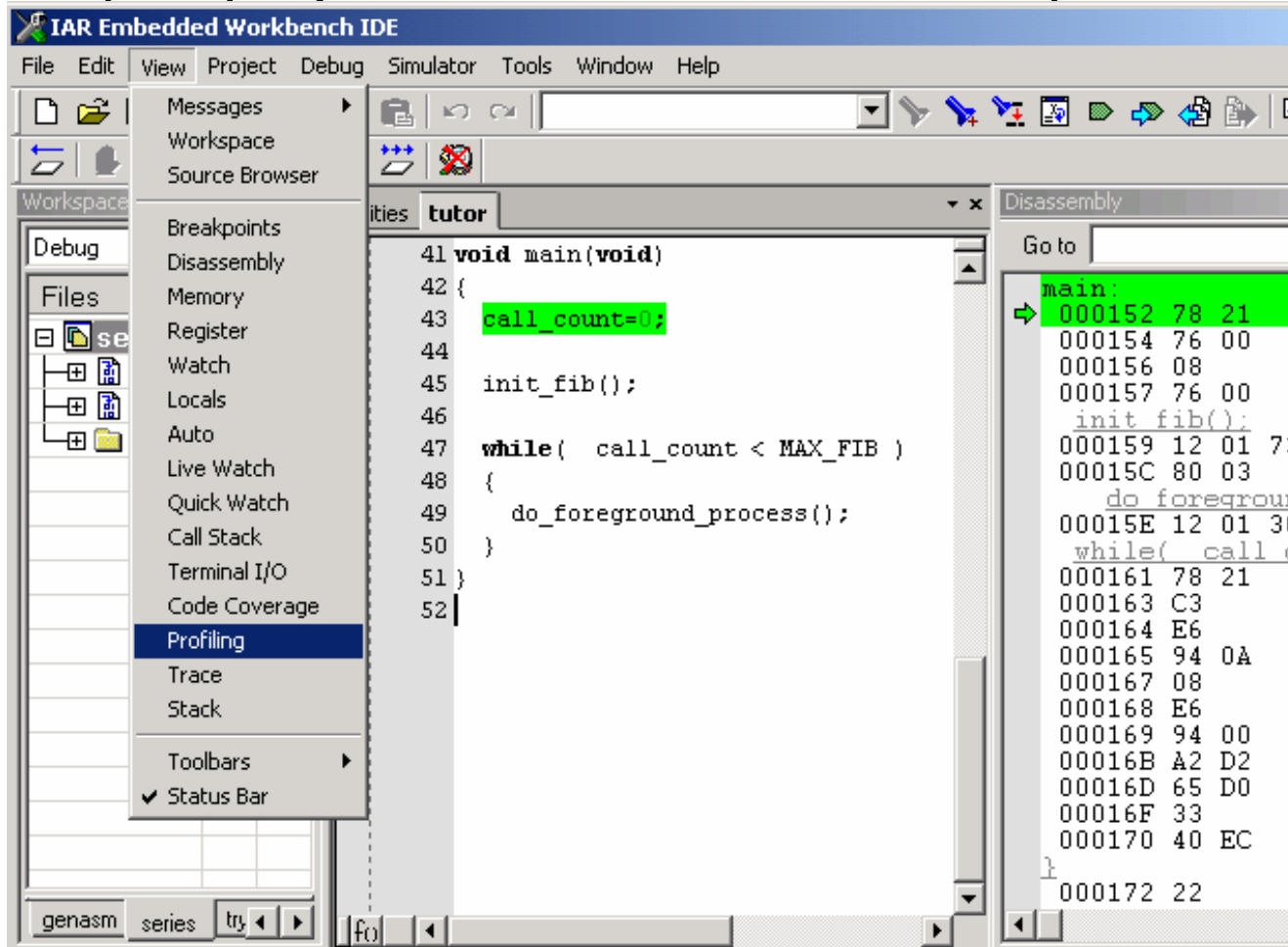


Profiling/Code Coverage/Trace

- These are to be loaded as plugins
- So enable them from
- General options > Debugger > Plugins

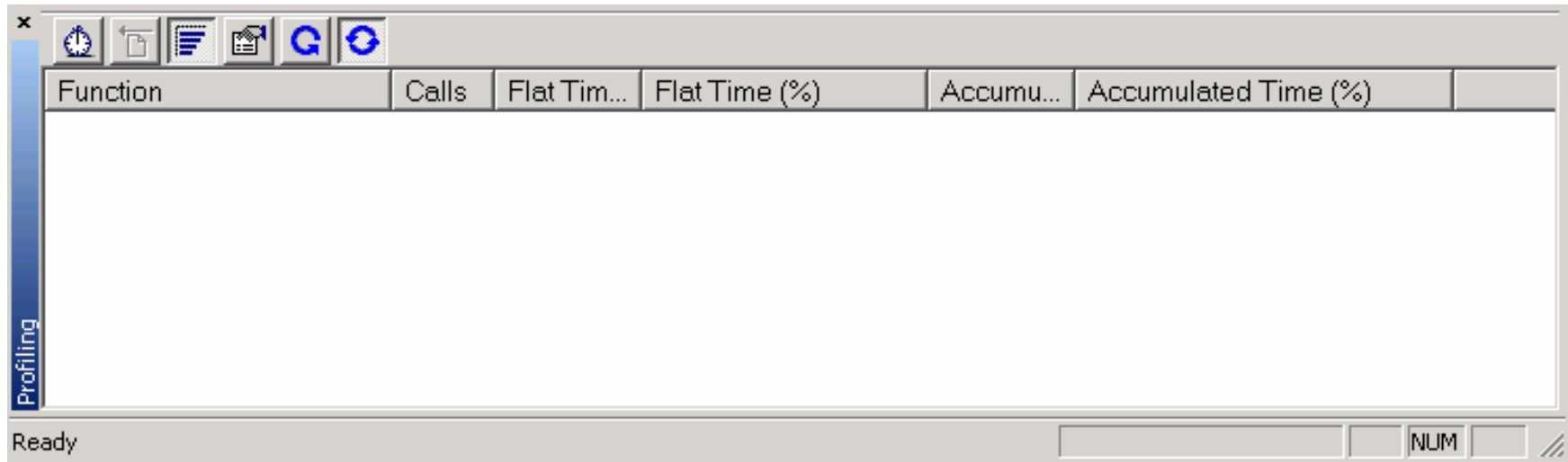


Open project **series** >> rebuild >> open Debugger



Profiling: Enabling profile

Enable Profile



Profiling: Enabling profile

Auto Refresh

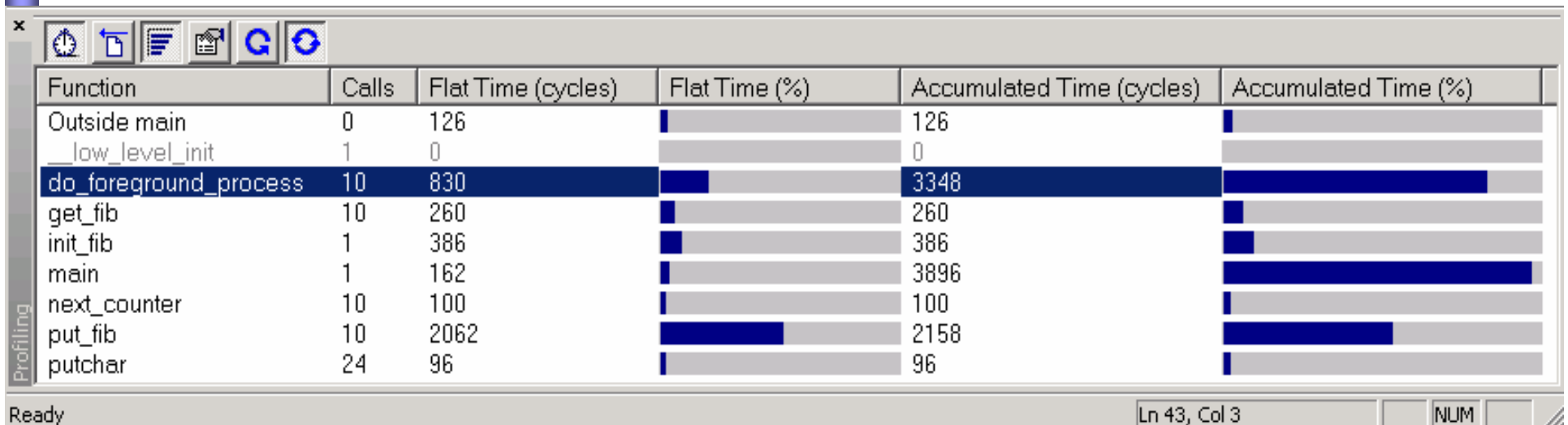


The screenshot shows a window titled 'Profiling' with a toolbar containing icons for refresh, copy, print, and zoom. Below the toolbar is a table with the following columns: Function, Calls, Flat Tim..., Flat Time (%), Accumu..., and Accumulated Time (%). The table lists several functions with zero values for all metrics. The status bar at the bottom shows 'Ready', 'Ln 52, Col 1', and 'NUM'.

Function	Calls	Flat Tim...	Flat Time (%)	Accumu...	Accumulated Time (%)
Outside main	0	0		0	
__low_level_init	0	0		0	
do_foreground_process	0	0		0	
get_fib	0	0		0	
init_fib	0	0		0	
main	0	0		0	
next_counter	0	0		0	
put_fib	0	0		0	
nutchar	0	0		0	

Profiling: Collecting profiling information

Reset and Run the code either upto any breakpoint or full
And profile information will be displayed

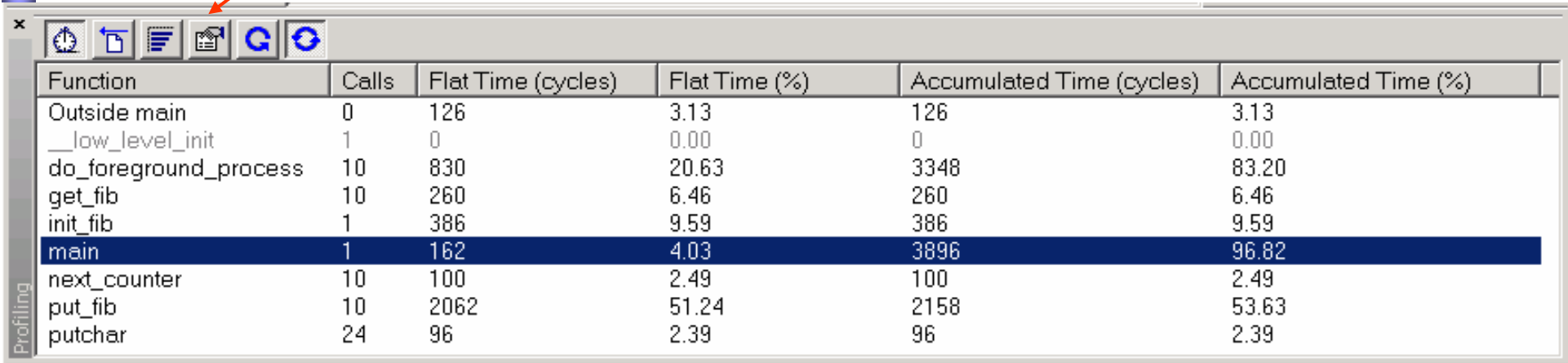


Function	Calls	Flat Time (cycles)	Flat Time (%)	Accumulated Time (cycles)	Accumulated Time (%)
Outside main	0	126		126	
__low_level_init	1	0		0	
do_foreground_process	10	830		3348	
get_fib	10	260		260	
init_fib	1	386		386	
main	1	162		3896	
next_counter	10	100		100	
put_fib	10	2062		2158	
putchar	24	96		96	

Ready Ln 43, Col 3 NUM

Profiling: Detail of any function 1/2

Select function
and
click on detail



Function	Calls	Flat Time (cycles)	Flat Time (%)	Accumulated Time (cycles)	Accumulated Time (%)
Outside main	0	126	3.13	126	3.13
__low_level_init	1	0	0.00	0	0.00
do_foreground_process	10	830	20.63	3348	83.20
get_fib	10	260	6.46	260	6.46
init_fib	1	386	9.59	386	9.59
main	1	162	4.03	3896	96.82
next_counter	10	100	2.49	100	2.49
put_fib	10	2062	51.24	2158	53.63
putchar	24	96	2.39	96	2.39

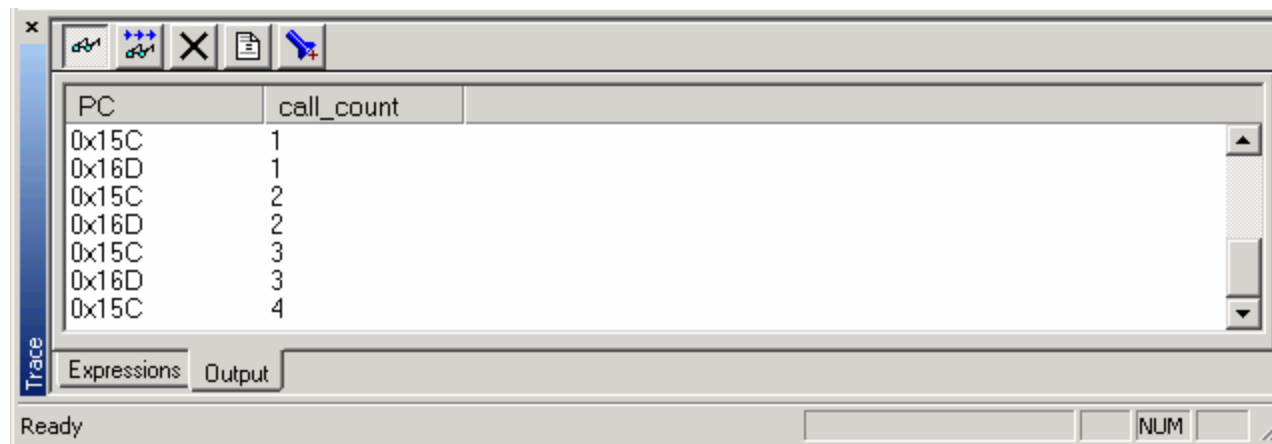
Profiling: Detail of any function 2/2

The screenshot shows a software interface for profiling. The top part is a table with columns: Function, Calls, Flat Time (cycles), Flat Time (%), Accumulated Time (cycles), and Accumulated Time (%). The 'main' function is highlighted in blue. Below the table is a detailed view for the 'main' function, showing its flat and accumulated times, callers, and callees.

Function	Calls	Flat Time (cycles)	Flat Time (%)	Accumulated Time (cycles)	Accumulated Time (%)
Outside main	0	126	3.13	126	3.13
__low_level_init	1	0	0.00	0	0.00
do_foreground_process	10	830	20.63	3348	83.20
get_fib	10	260	6.46	260	6.46
init_fib	1	386	9.59	386	9.59
main	1	162	4.03	3896	96.82
next_counter	10	100	2.49	100	2.49
put_fib	10	2062	51.24	2158	53.63
putchar	24	96	2.39	96	2.39

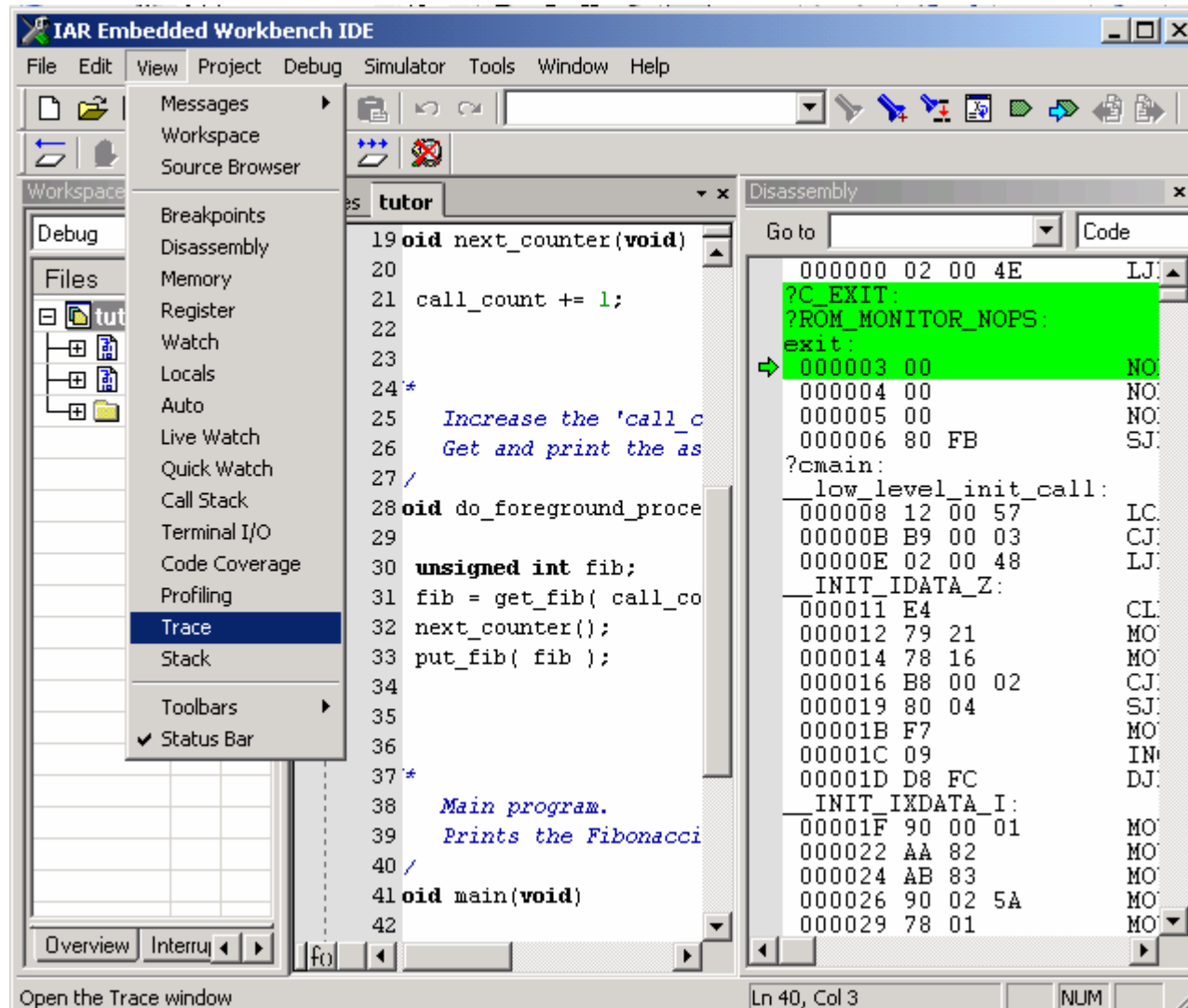
Function: main
Flat time 162 cycles, Accumulated time 3896 cycles.
Callers:
Total: 1
Count Function
1 Unknown caller(s)
Callees:
Count Function

Display the execution of code, step by step,
can trace the value of any variable, after each line of execution

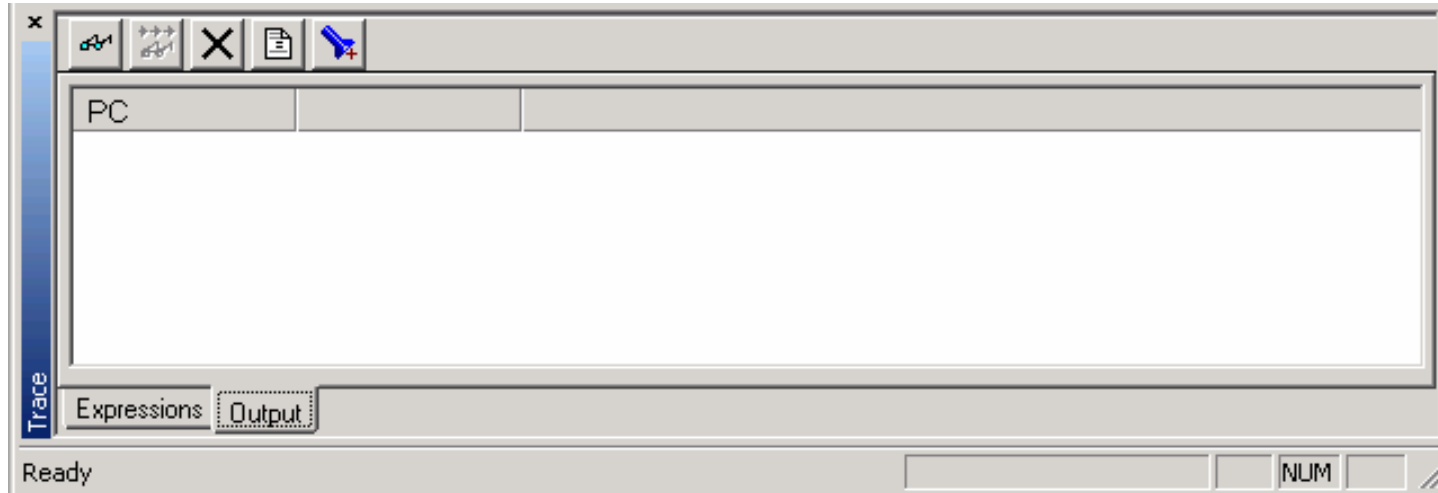


Trace: setting 1

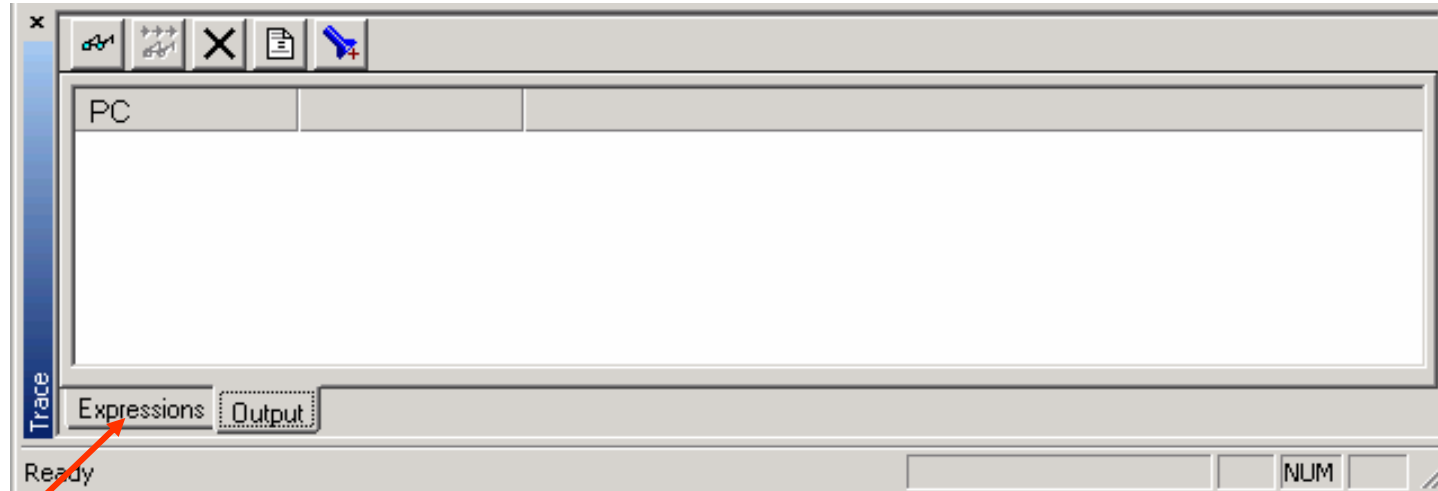
Open series project >> compile >> debug



Watch window

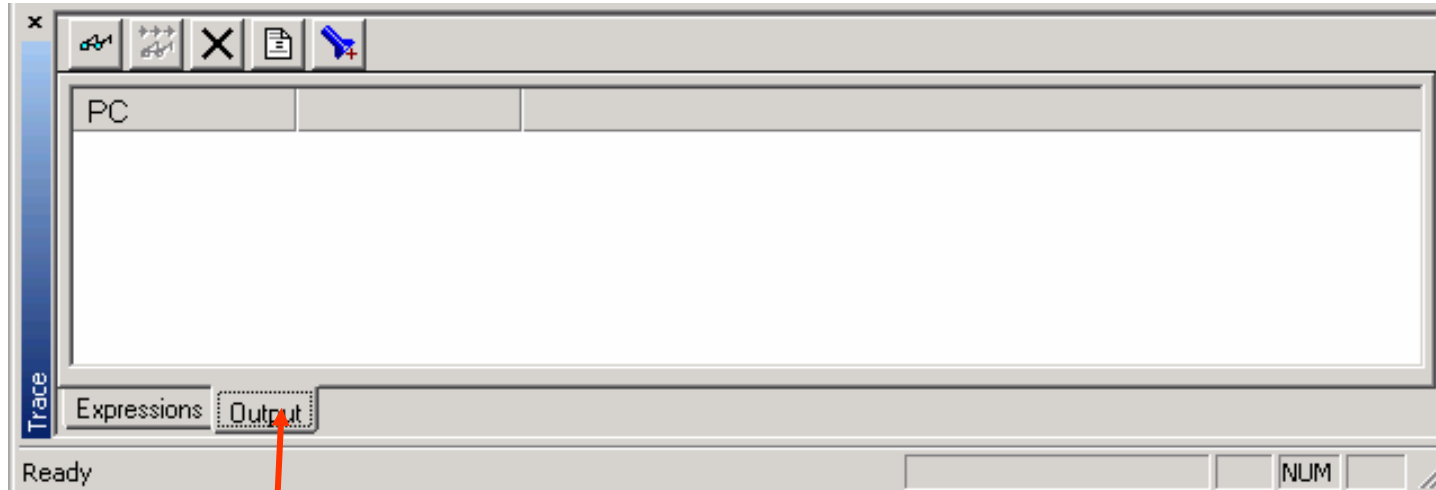


Two Tabs



Enter the value of variable, want to examine

Two Tabs



Display the output

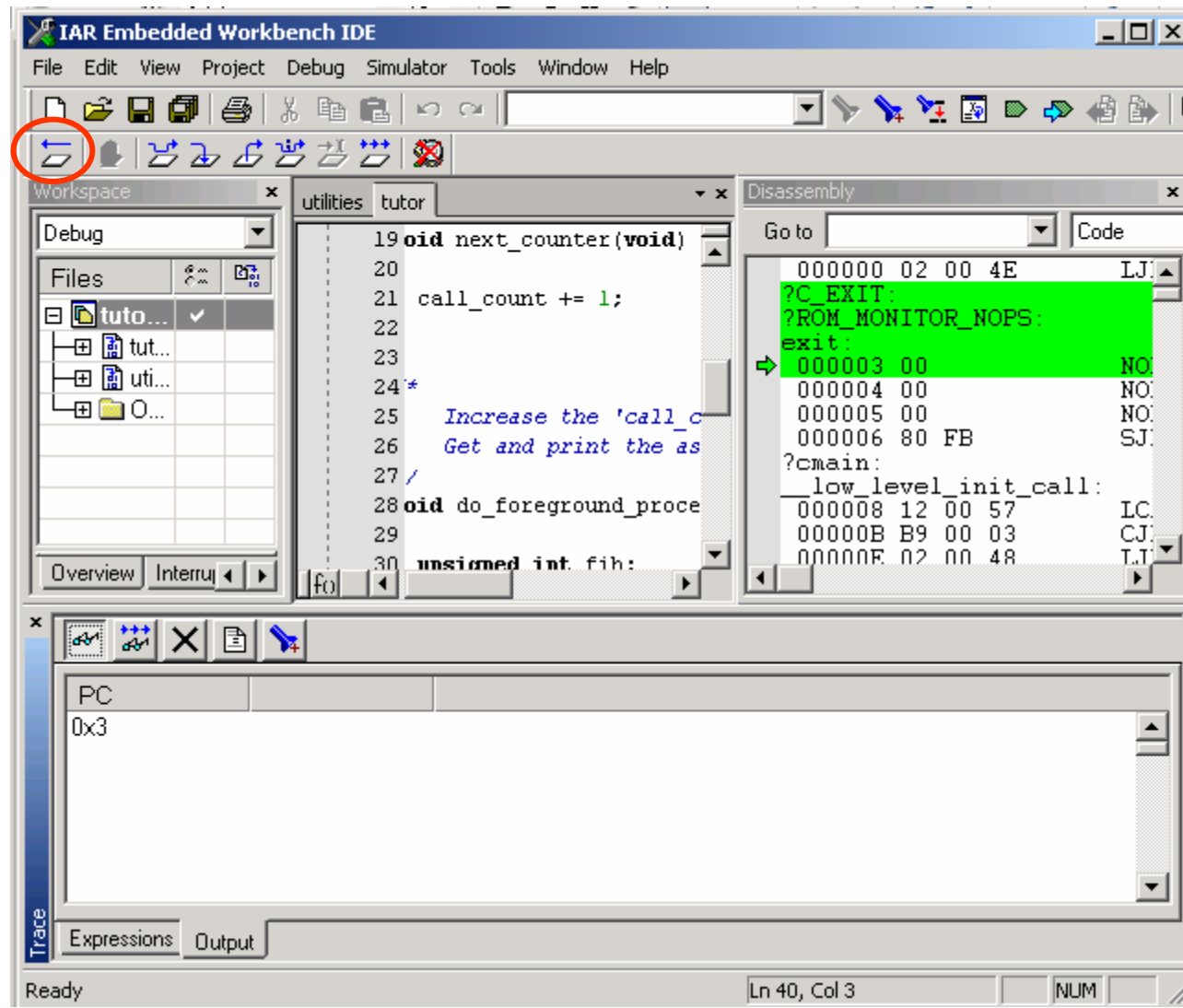
Enable trace

Enable Trace



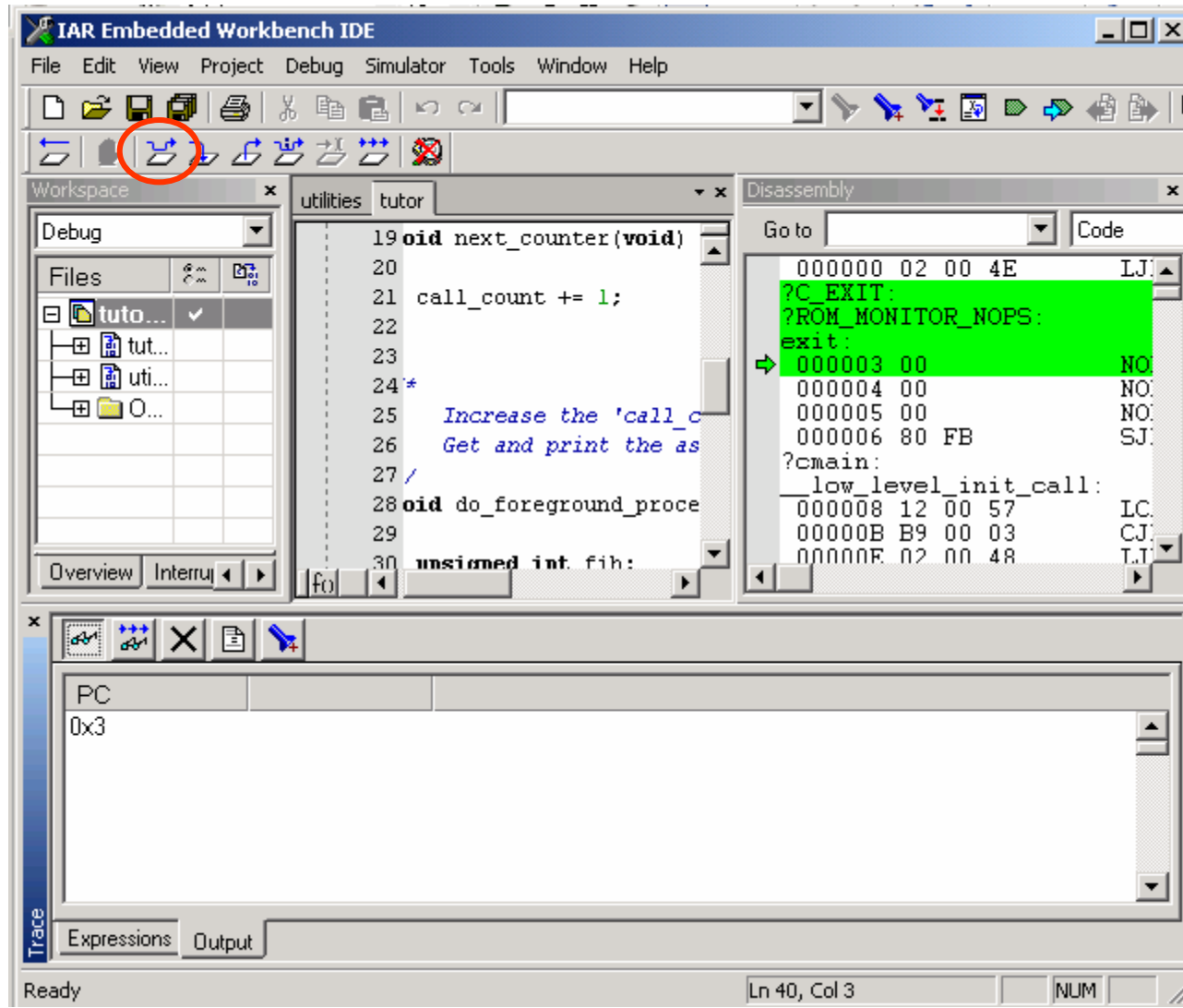
Trace: setting 5

Reset



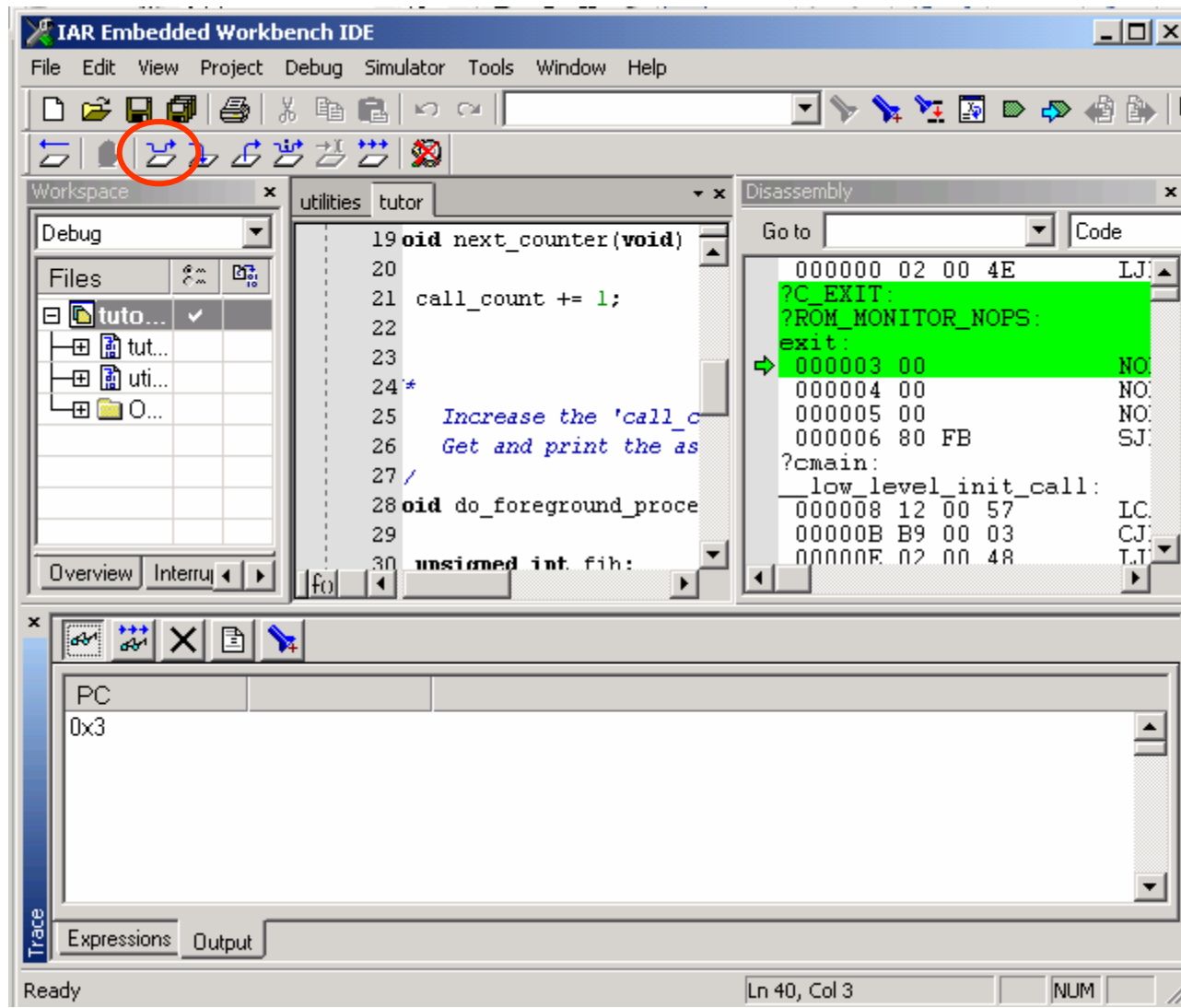
Trace: setting 6

Execute Code
step by step



Trace: setting 7

Execute Code
step by step



Trace: setting 8

The screenshot displays a debugger interface with three main panels:

- Workspace:** Shows a file tree with 'tuto...' selected.
- Source Code:** Displays C code for a Fibonacci program. Line 47, `while(call_count < M`, is highlighted in green. A green arrow points to this line.
- Disassembly:** Shows assembly instructions. Address `00015C 78 21 MO` is highlighted in green, with a green arrow pointing to it.
- Trace Window:** Shows a list of memory addresses: `PC`, `0x3`, `0x3`, `Reset` (highlighted in red), `0x152`, `0x159`, and `0x15C`.

The status bar at the bottom indicates 'Ready' and 'NUM'.

Trace: Value of particular variable 9

The screenshot displays the IAR Embedded Workbench IDE interface. The main window shows a C program with the following code:

```
38  Main program.  
39  Prints the Fibonacci  
40  /  
41  void main(void)  
42  /  
43  call_count=0;  
44  /  
45  init_fib();  
46  /  
47  while( call_count < M  
48  {  
49  do_foreground_proces
```

The 'Disassembly' window shows the following assembly code:

```
Go to Code  
000152 78 21 MO  
000154 76 00 MO  
000156 08 IN  
000157 76 00 MO  
000159 12 01 73 IC  
00015C 78 21 MO  
00015E C3 CL  
00015F E6 MO  
000160 94 0A SU  
000162 08 IN  
000163 E6 MO
```

The 'Trace' window is open at the bottom, showing the 'Expressions' tab with the following table:

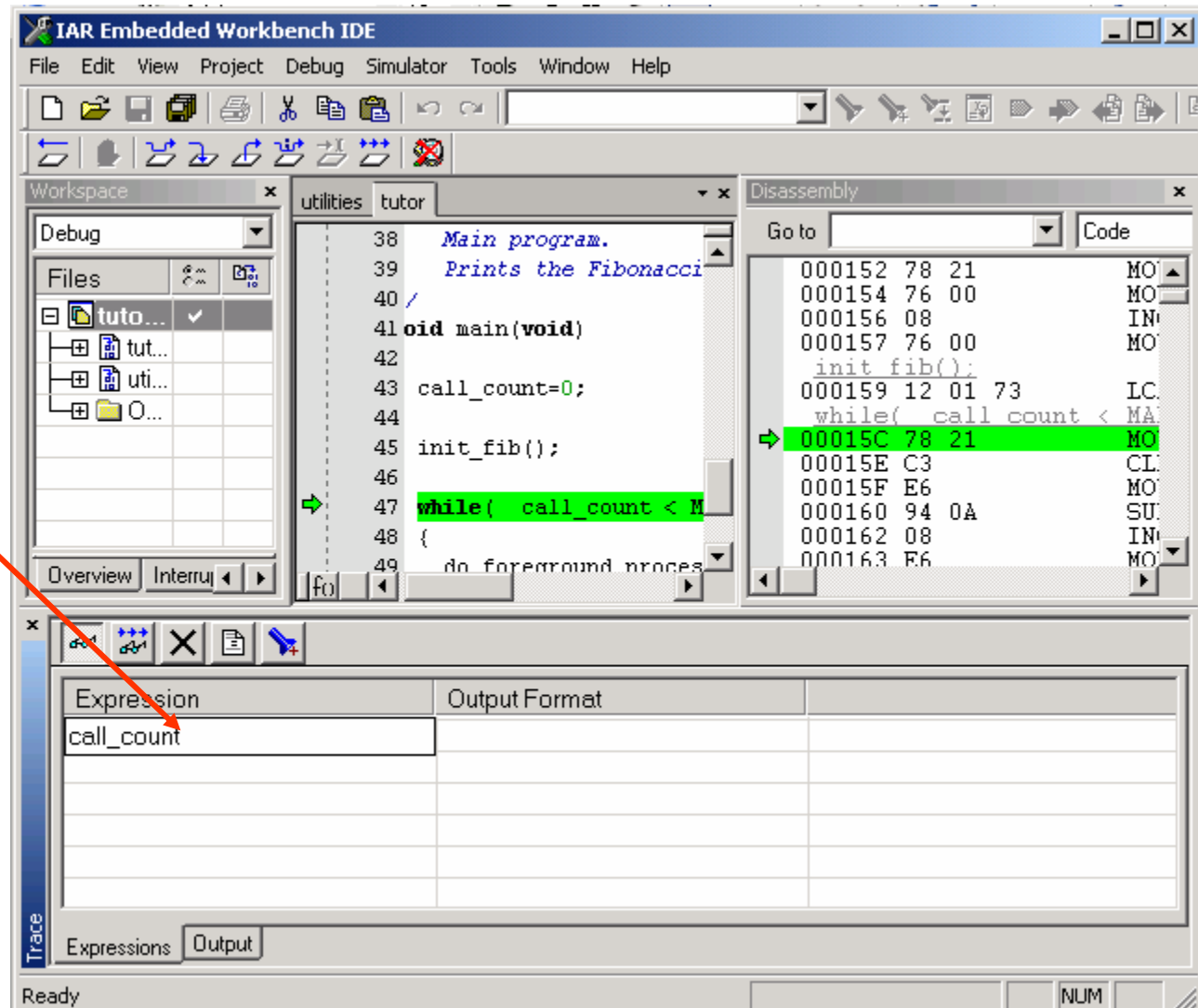
Expression	Output Format
call_count	

An arrow points from the text 'Select expression tab' to the 'Expressions' tab in the Trace window.

Select expression tab

Trace: Value of particular variable 10

Enter variable



Trace: Value of particular variable 11

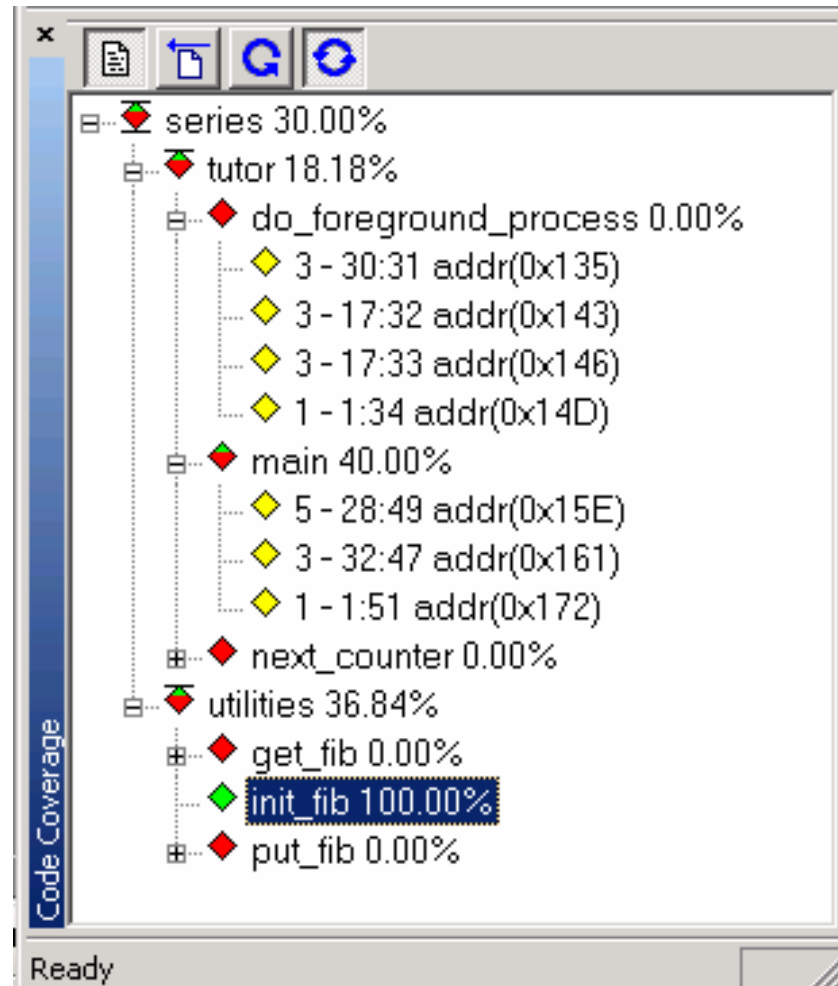
The screenshot shows the IAR Embedded Workbench IDE interface. The main window is divided into several panes:

- Workspace:** Shows a file tree with folders for 'tuto...' and 'uti...'. The 'Debug' dropdown is set to 'Debug'.
- Source Code:** Displays C code for a function. Line 47, `while(call_count < M`, is highlighted in green. A green arrow points to this line.
- Disassembly:** Shows assembly code. Line `MO 00015C 78 21` is highlighted in green, with a green arrow pointing to it.
- Trace:** A table at the bottom shows the execution trace for the variable `call_count`. The table has two columns: 'PC' and 'call_count'. The data is as follows:

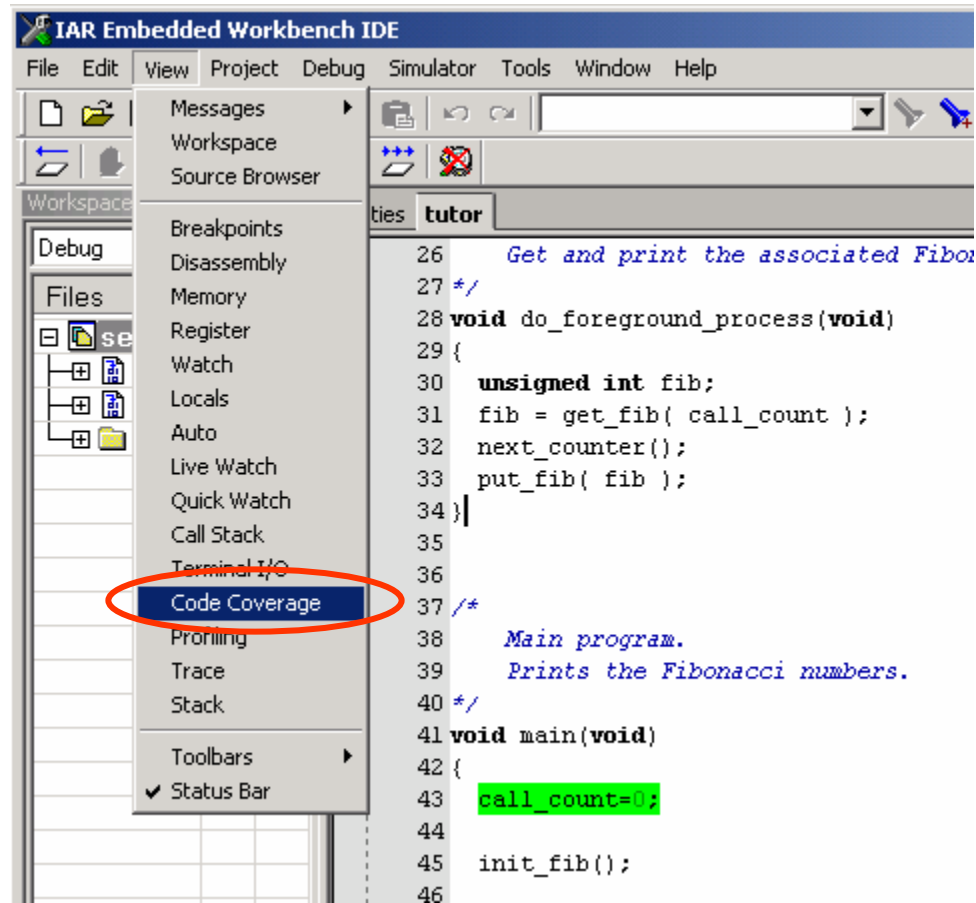
PC	call_count
0x15C	1
0x16D	1
0x15C	2
0x16D	2
0x15C	3
0x16D	3
0x15C	4

The status bar at the bottom indicates 'Ready' and 'NUM'.

Shows which part of the code is executed

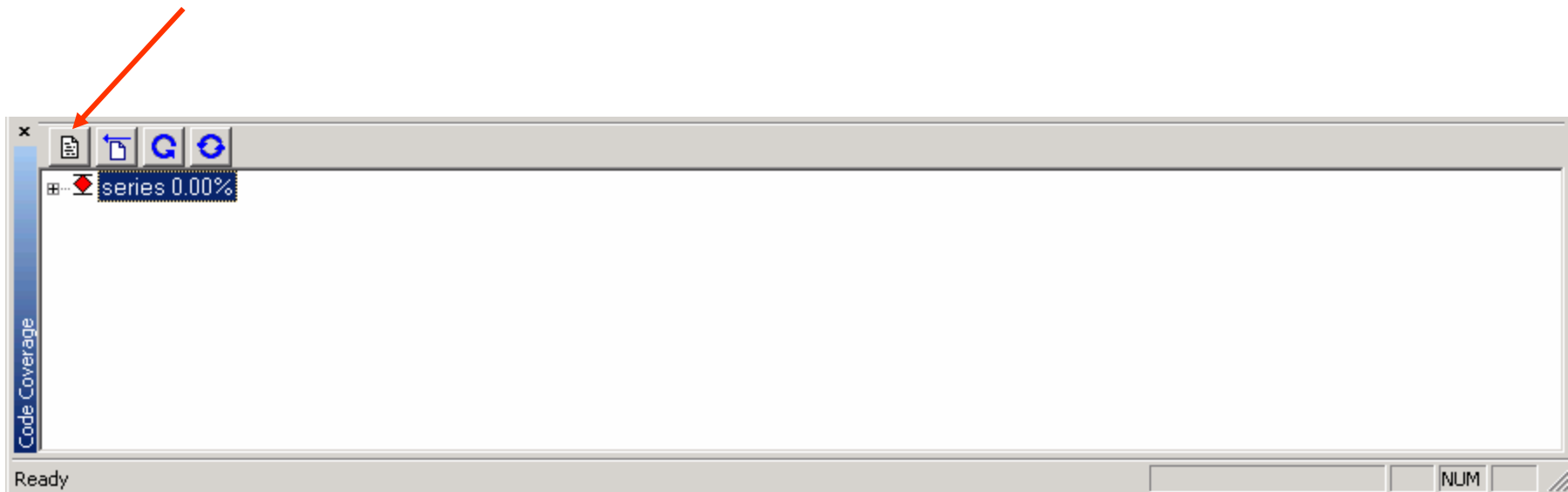


To open Code Coverage window



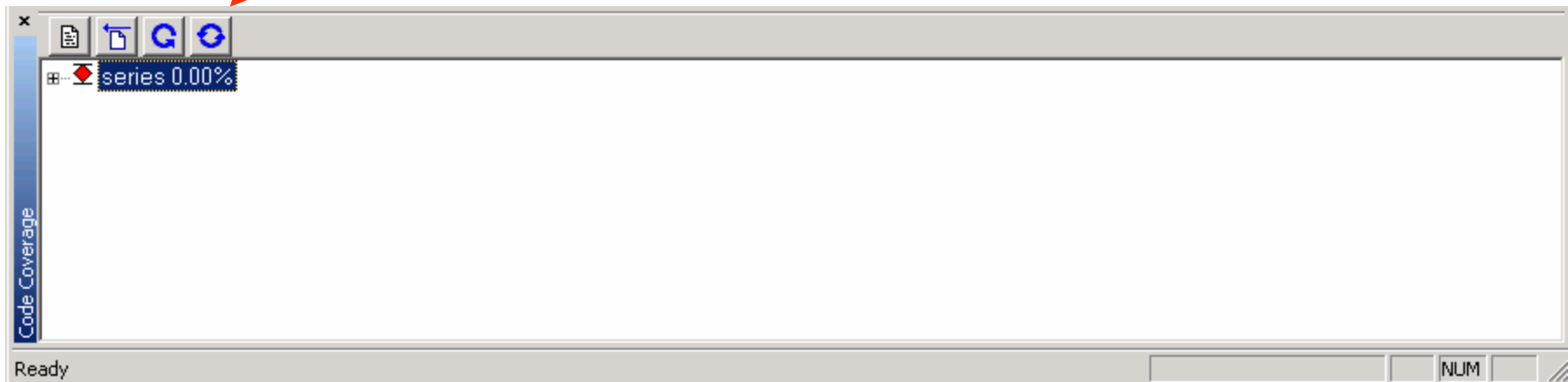
Steps to start code coverage

1. Active window

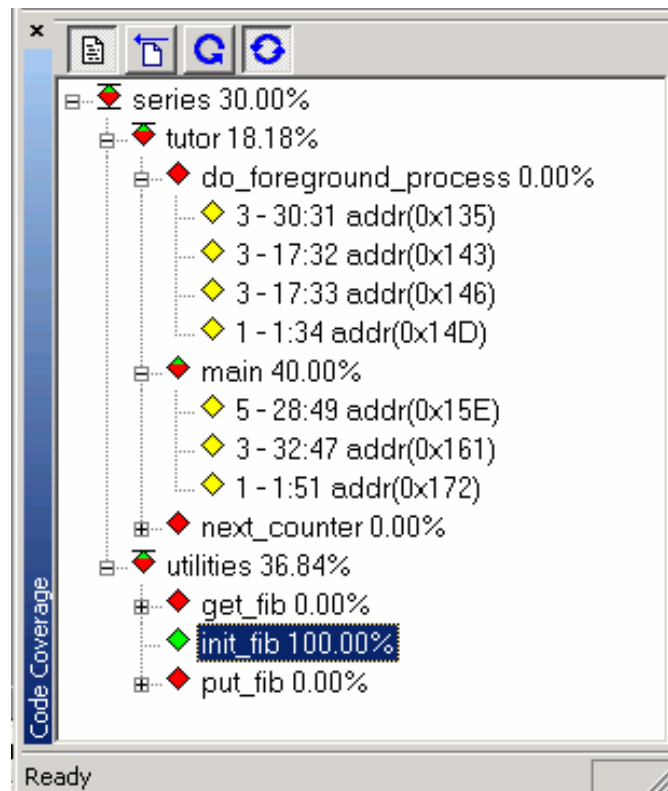


Steps to start code coverage

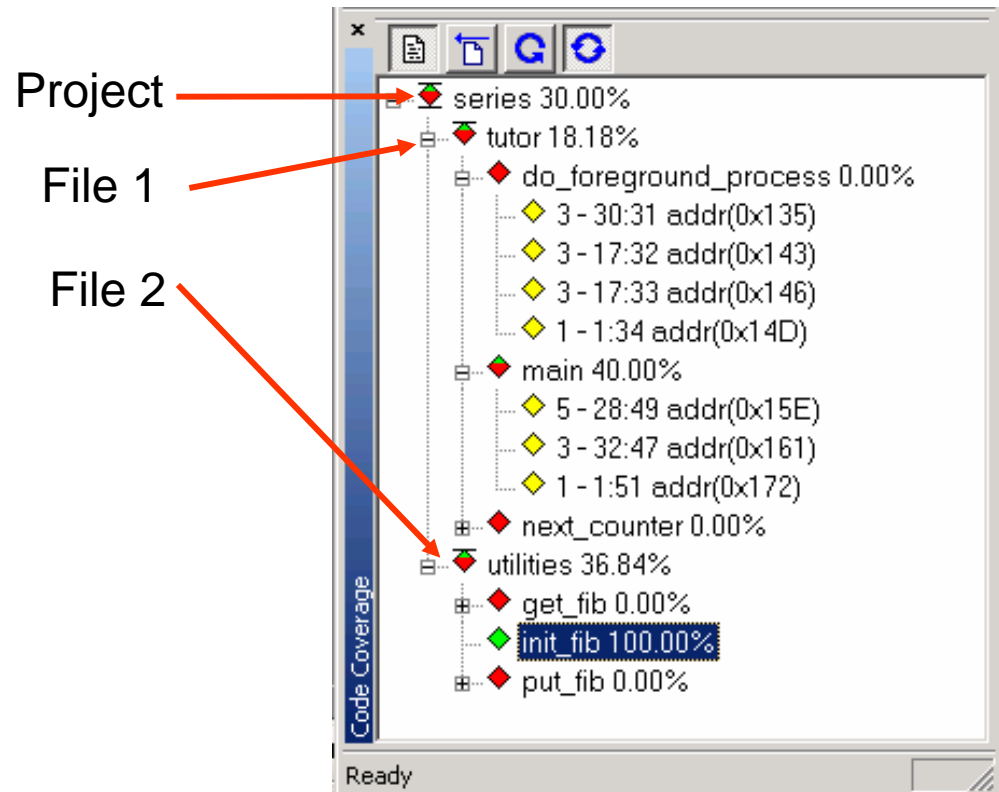
2. Auto fresh



To see code coverage execute the code step by step or execute completely

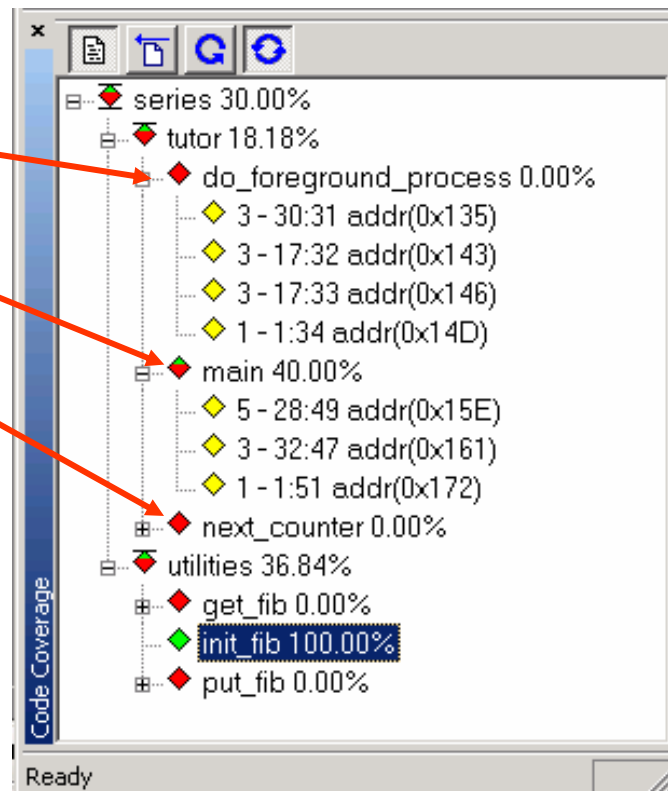


Details displayed in code coverage window

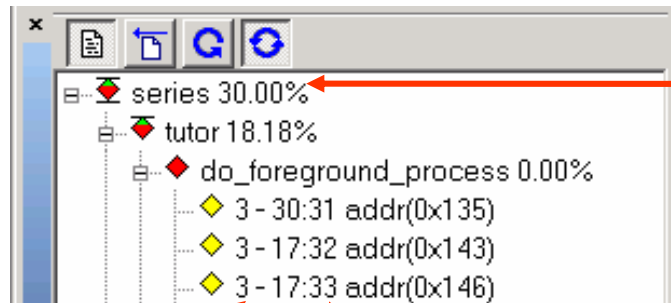


Details displayed in code coverage window

Function of file



Details displayed in code coverage window



Percentage of code is execute

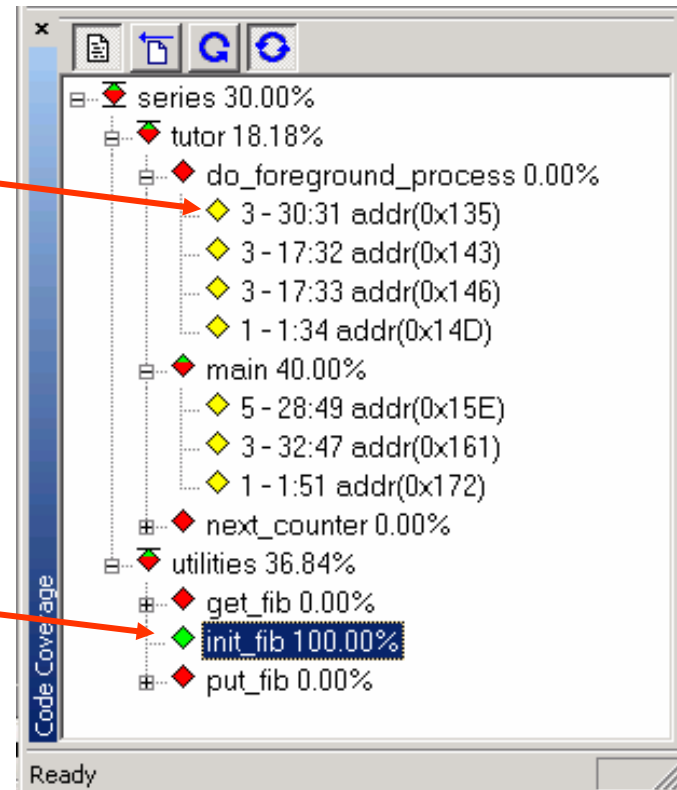
Address of block

Start column
End column
Row No

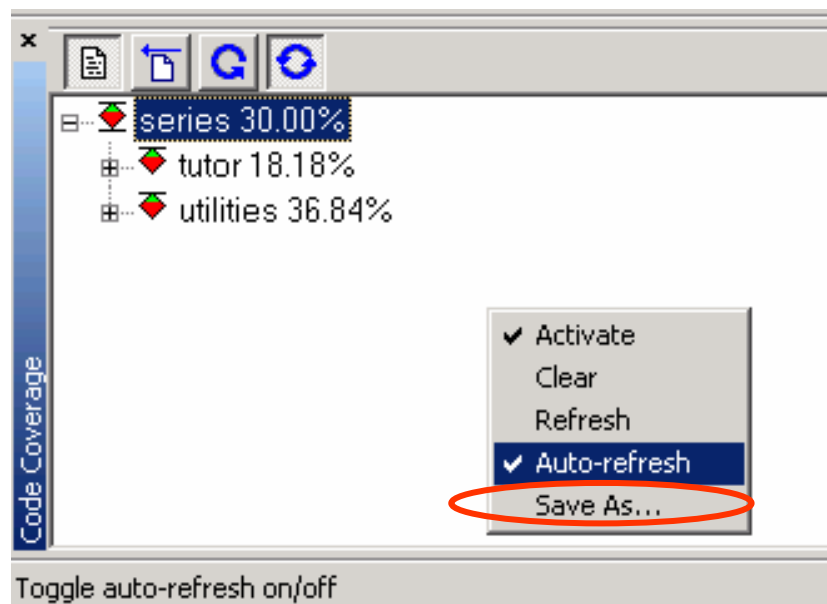
Color Scheme

Code which to be execute

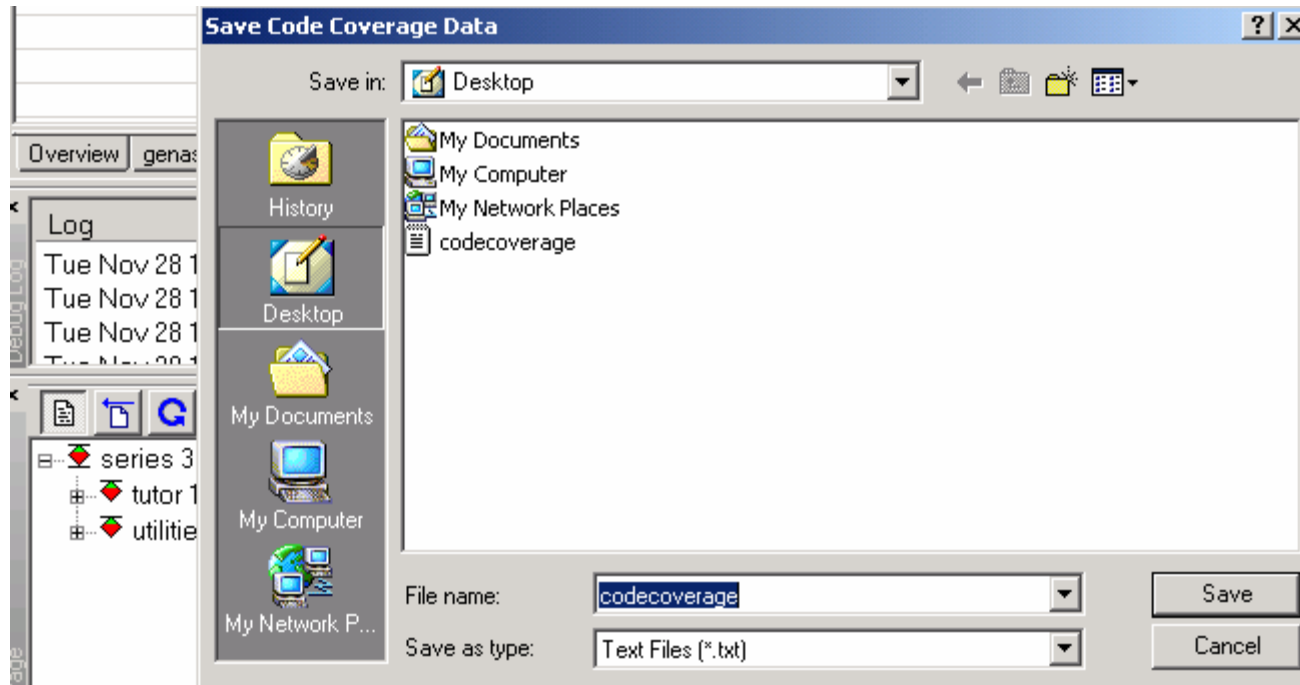
Function executed completely



To generate a report right click in the window



Save the file (file is in the text format)



<http://www.embeddedcraft.org>

EmbeddedCraft
crafting of intelligent systems

