

# Debugging techniques for IoT applications

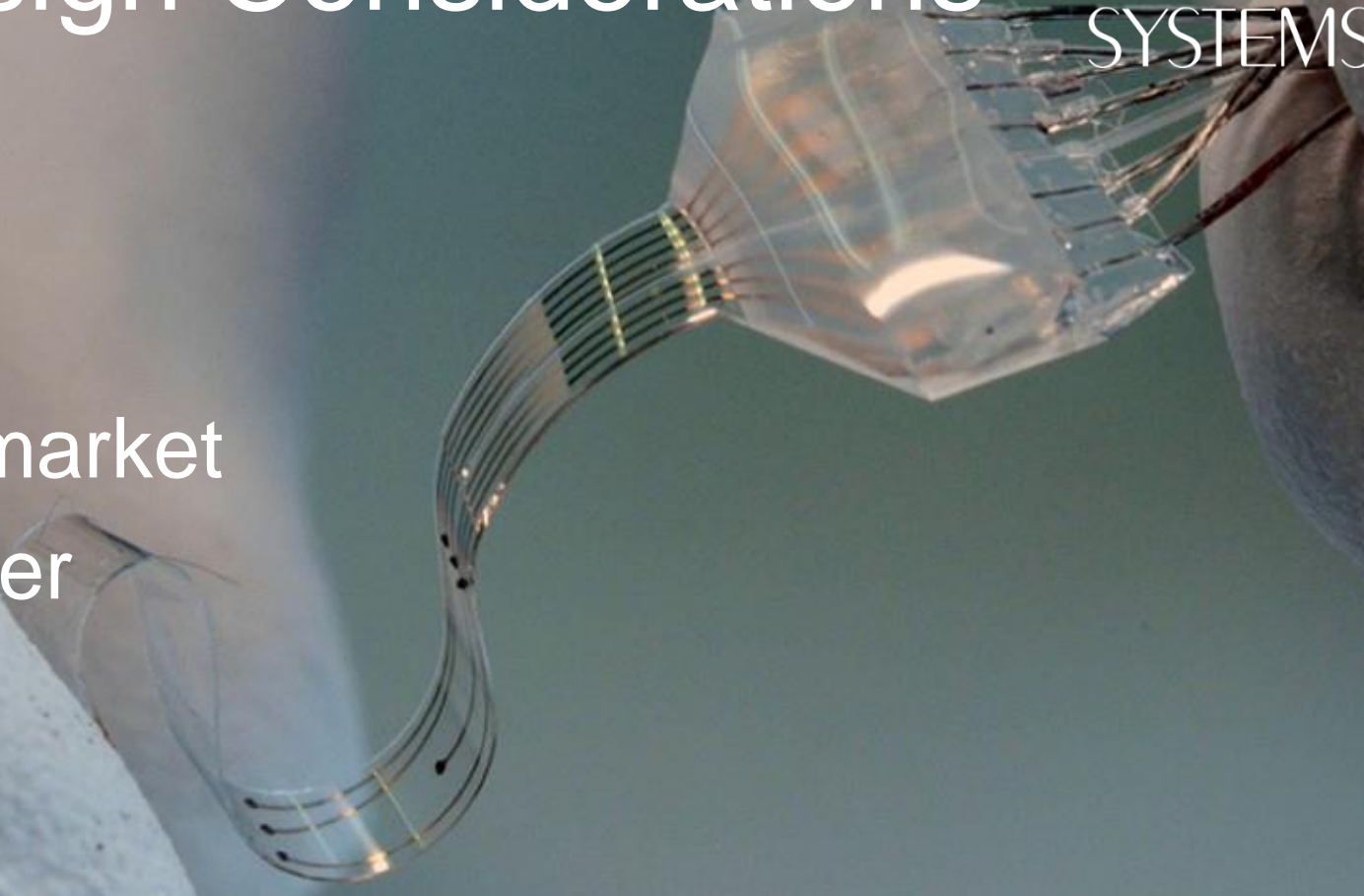


Shawn A. Prestridge

# IoT Design Considerations



- Small
- Cheap
- Time to market
- Low power



# Small & Cheap

- Small devices
  - Limited debug features
  - Develop on larger device, then make iteration on production hardware
- Cheap devices have limited peripherals



# Time to Market

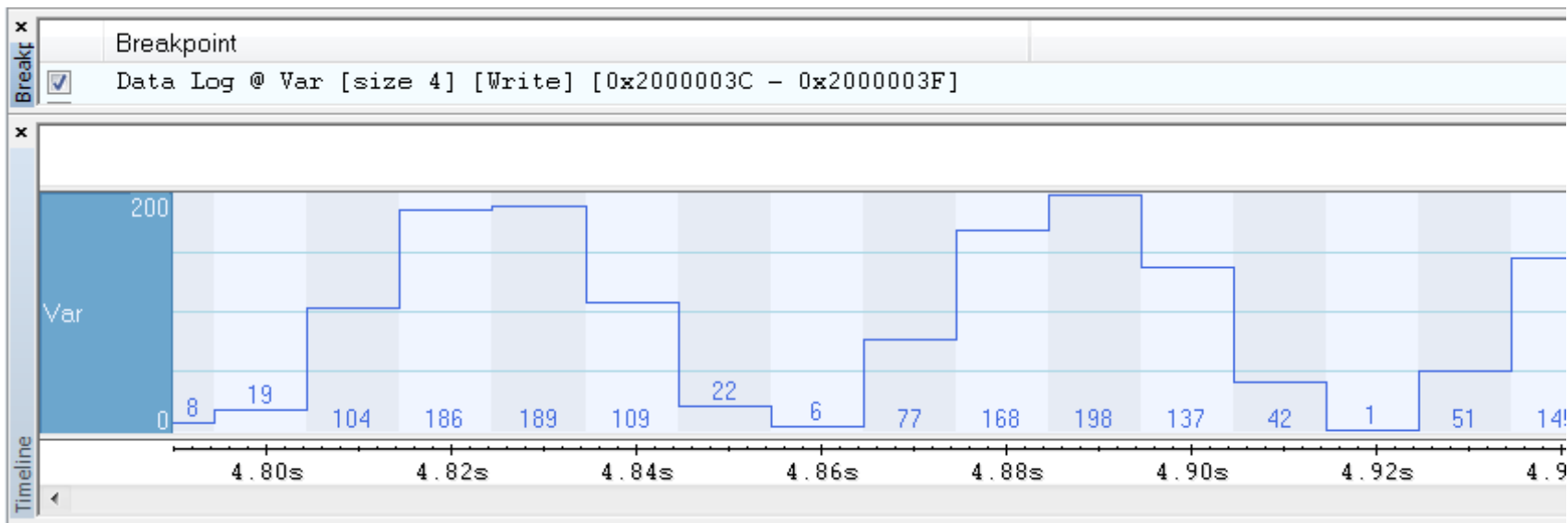
To decrease time to market, you must:

- Use all pertinent debug features
- Use code analysis to quickly identify bugs
- Analyze stack usage
- RTOS kernel-aware debugging (if using an RTOS)
- Use trace (if available) to find "million dollar" bugs

Debug features in ARM Cortex-  
M3/M4 that are not available on the  
ARM Cortex-M0

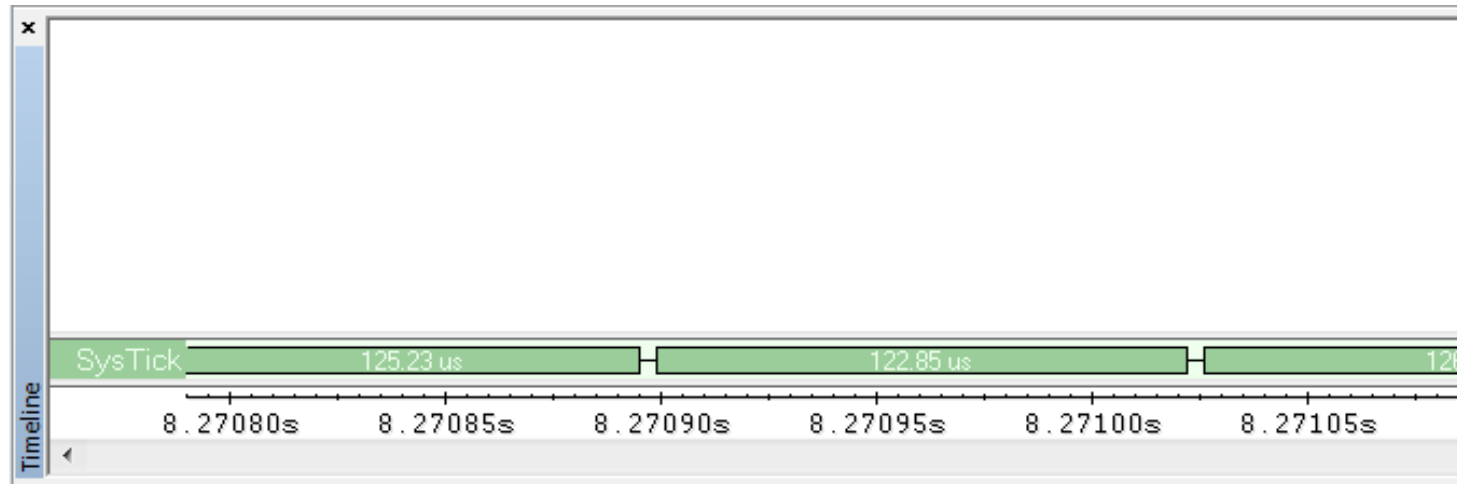
# SWO:

Using a data log breakpoint to visualize data

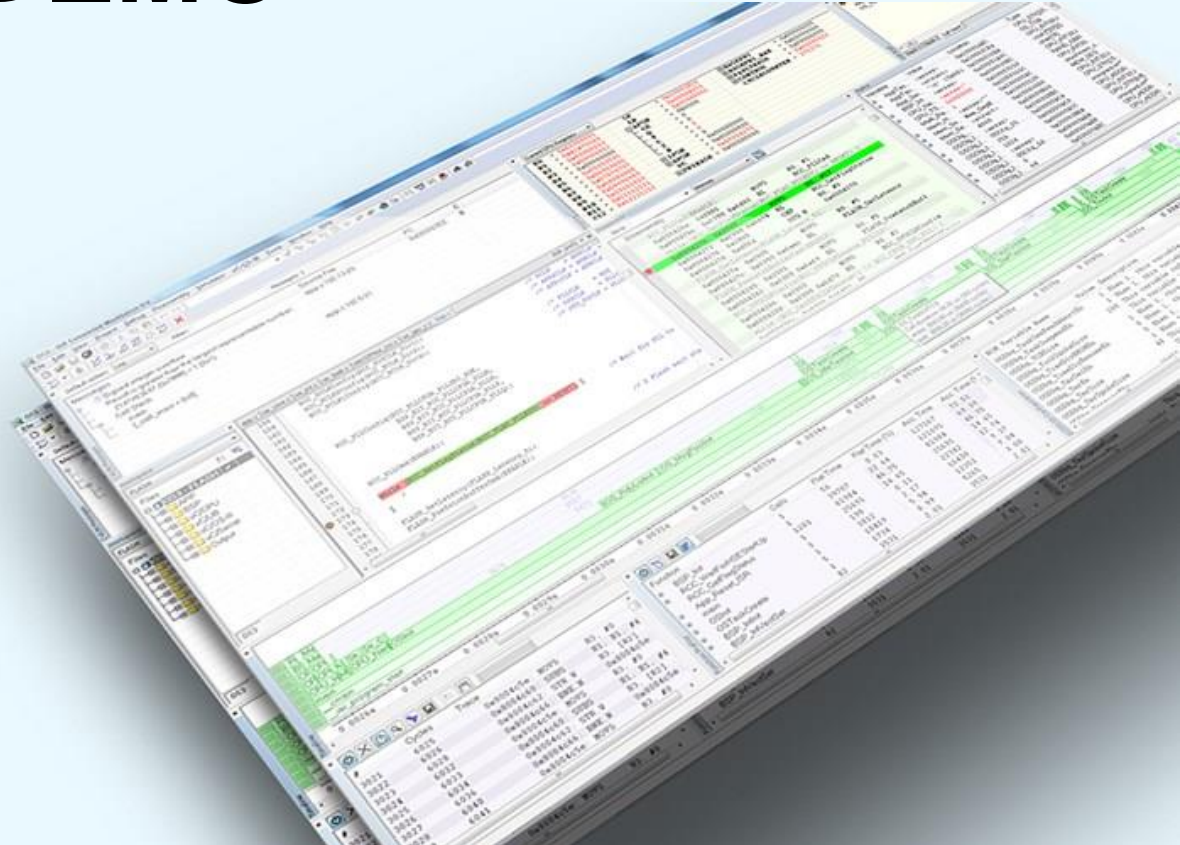


# SWO:

Using interrupt logging to validate timing

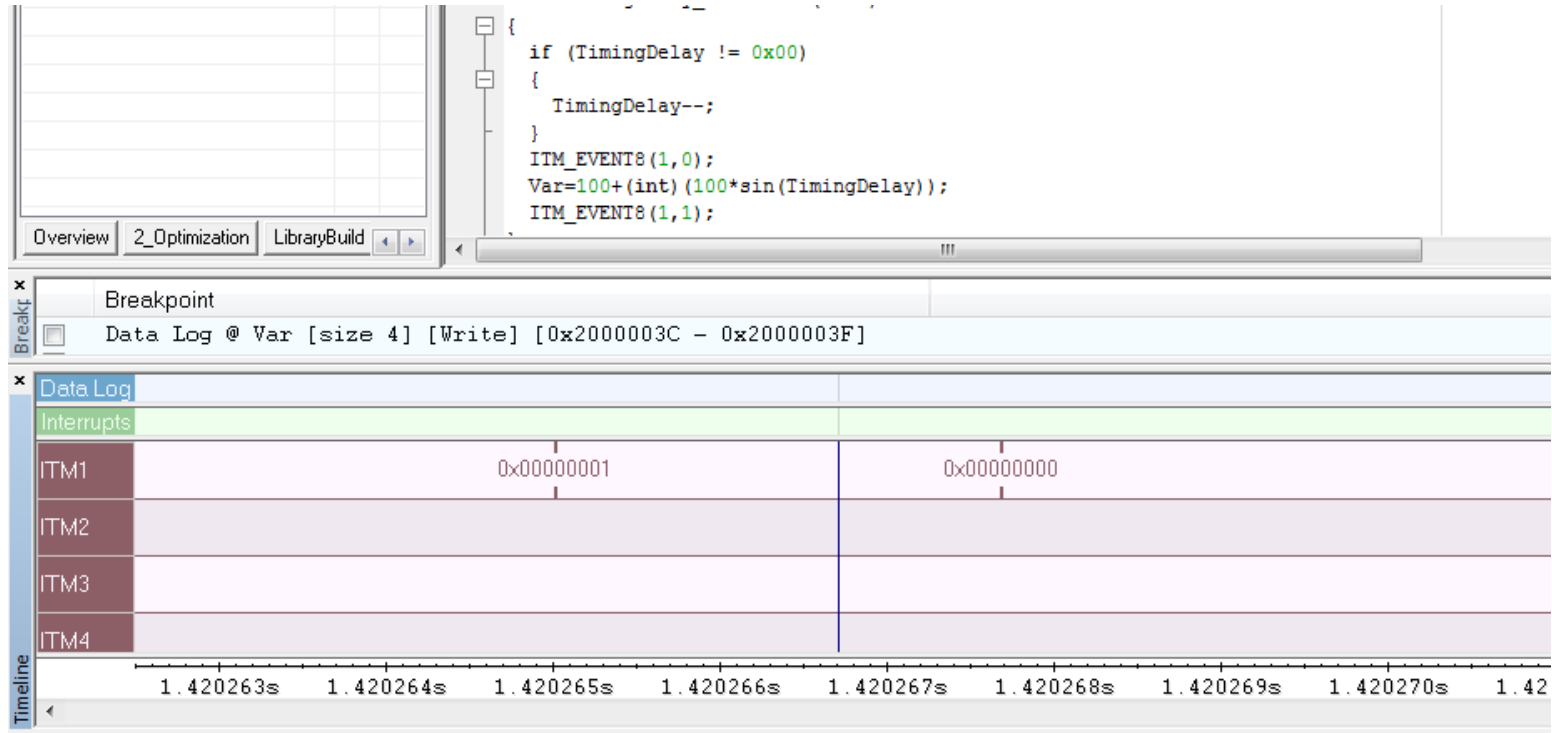


# SWO Data Log and Interrupt Log DEMO

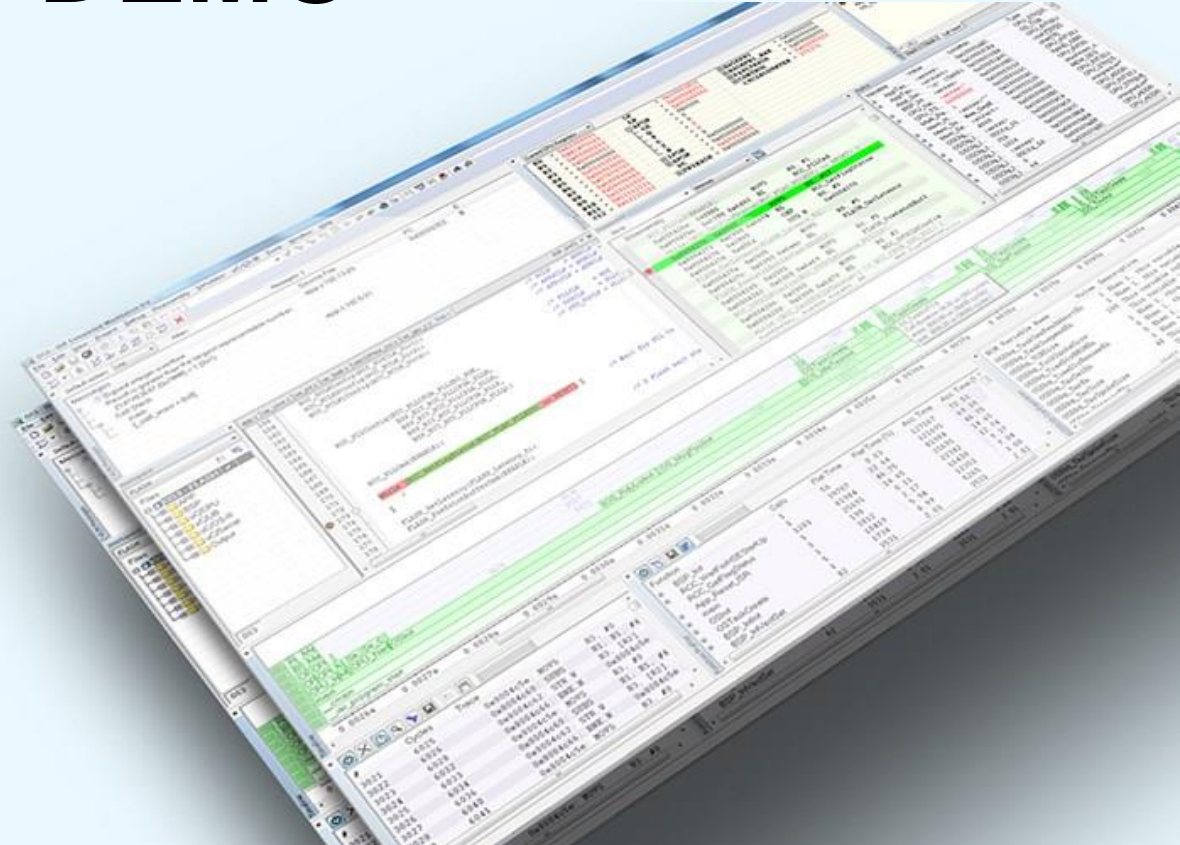




# SWO: ITM events. Track program execution



# ITM Events and Code Analysis **DEMO**



# Minimizing Power

- Work fast, sleep a lot
  - Optimization
  - Efficient code
  
- Power Debugging

# SWO: Sampled instruction trace

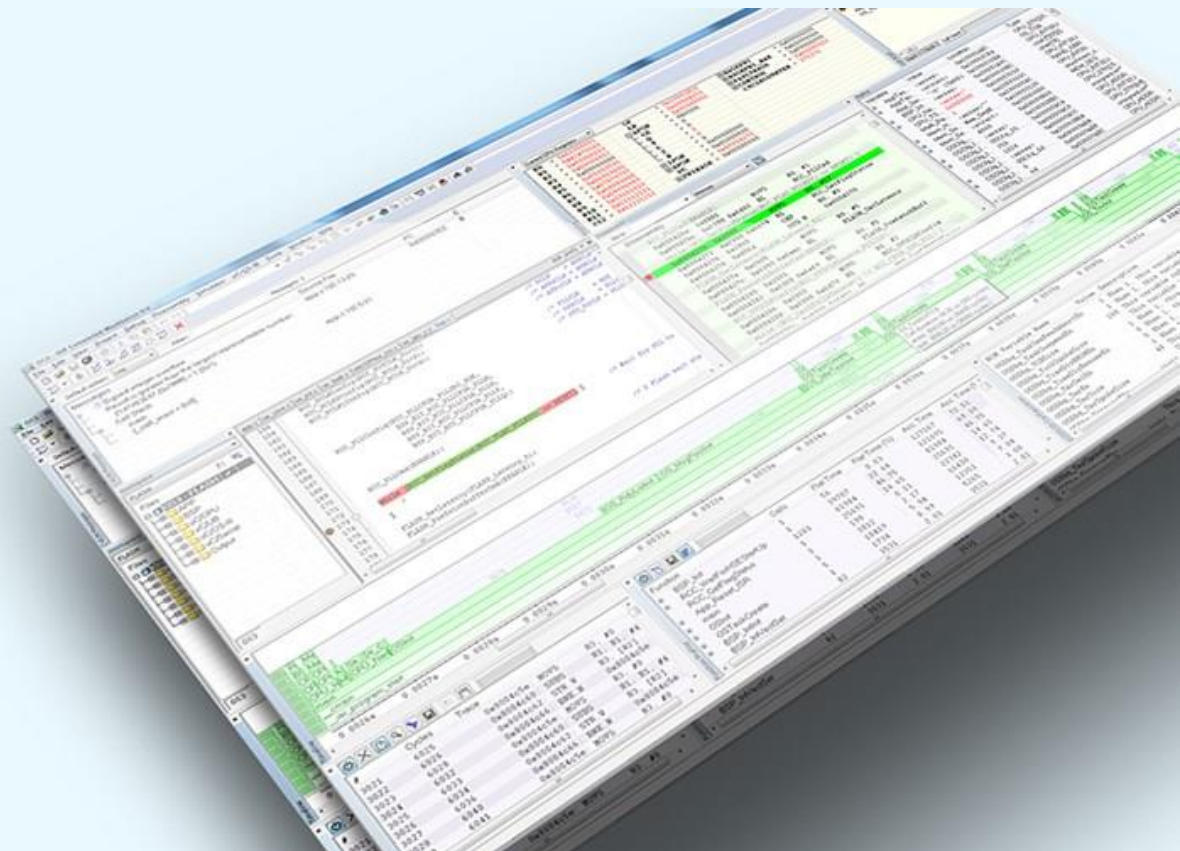


The screenshot displays two windows from the IAR Embedded Workbench. The left window is the 'Function Profiler', which shows a table of functions and their execution statistics. The right window is the 'Output' window, which displays the results of a CoreMark benchmark test.

Function	PC Samp...	PC Samples...
core_state_transition	123648	27.64
crcu8	70531	15.77
core_list_find	41136	9.20
matrix_mul_matrix_bitextract	40529	9.06
matrix_mul_matrix	34565	7.73
ee_isdigit	32912	7.36
core_list_reverse	31180	6.97
core_bench_state	14529	3.25
matrix_sum	13952	3.12
core_list_mergesort	10564	2.36
core_bench_list	5327	1.19
crcu16	4874	1.09
matrix_add_const	4664	1.04
calc_func	4618	1.03

```
*** Starting Coremark ***
CPU Clock      : 168 MHz
2K performance run pers for coremark.
CoreMark Size  : 666
Total ticks     : 2688701333
Total time (secs): 16.004175
Iterations/Sec  : 187.451092
Iterations      : 3000
Compiler version : IAR ANSI C/C++ Compiler V7.50.
Memory location  : STACK
seedcrc         : 0xe9f5
[0]crclist      : 0xe714
[0]crcmatrix    : 0x1fd7
```

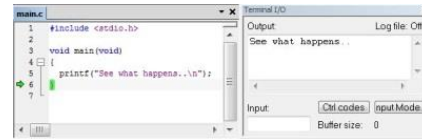
# SWO Trace DEMO



# Take control of your debug session

- Some key highlights of our C-SPY debugger

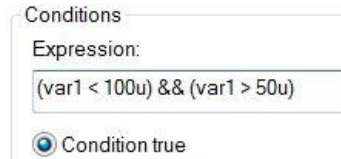
✓ C-SPY Terminal I/O `printf()`



A screenshot of the C-SPY Terminal I/O window. The left pane shows a C program with a `printf` statement. The right pane shows the output of the program, which is "See what happens...".

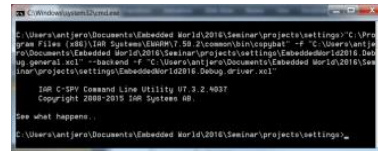
✓ C-SPY Macros 

✓ Conditional Breakpoints, Log Breakpoints, etc.



A screenshot of the C-SPY Conditions dialog box. The "Expression" field contains the code `(var1 < 100u) && (var1 > 50u)`. The "Condition true" radio button is selected.

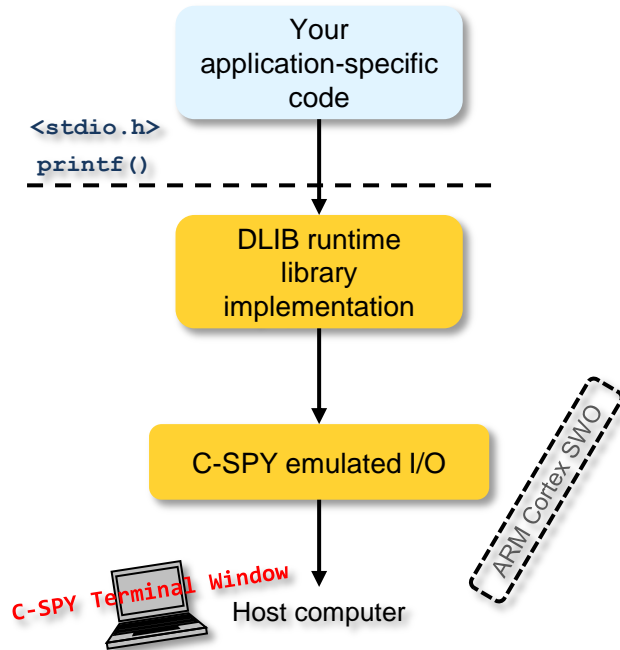
✓ Command line utility `cspybat`



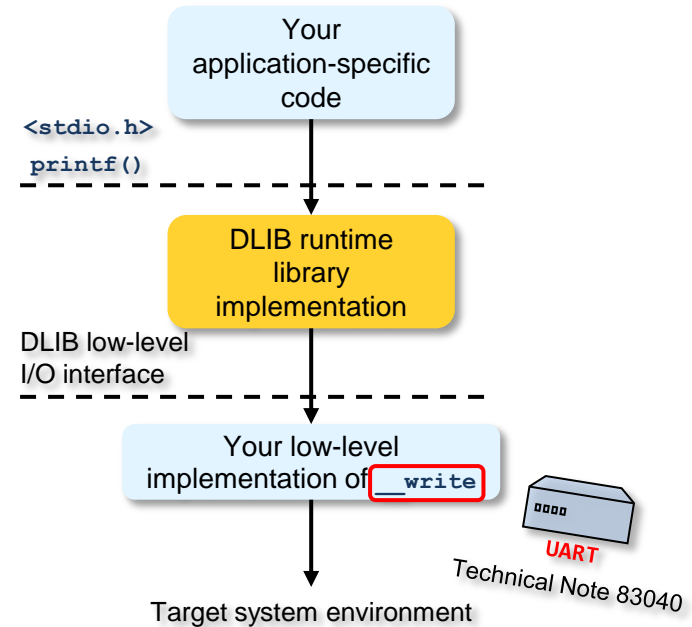
A screenshot of the C-SPY Command Line Utility window. The window title is "C-SPY Command Line Utility". The output shows the program's path and version information: "IAR C-SPY Command Line Utility 07.3.2.4037 Copyright 2008-2015 IAR Systems AB".

# printf () in the Terminal I/O

- Debug Configuration



- Release Configuration



# printf () DEMO

A screenshot of an IDE window. The left pane shows a C program named 'main.c' with the following code:

```
1 #include <stdio.h>
2
3 void main(void)
4 {
5     printf("See what happens..\n");
6 }
7
```

A green arrow points to line 6, and a green cursor is on line 5. The right pane is titled 'Terminal I/O' and shows the output 'See what happens..' in a monospaced font. Below the output area are controls for 'Input', 'Ctrl codes', 'Input Mode..', and 'Buffer size: 0'.

main.c

```
1 #include <stdio.h>
2
3 void main(void)
4 {
5     printf("See what happens..\n");
6 }
7
```

Terminal I/O

Output: Log file: Off

See what happens..

Input: Ctrl codes Input Mode..

Buffer size: 0



# C-SPY Macros

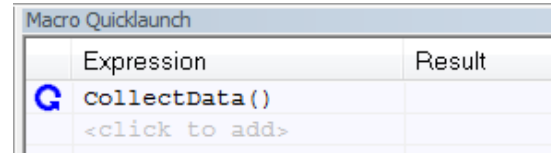
- C-SPY macros enable you to build complex debug functions like system test or peripheral simulation, suited to your needs.
- Written in simplified C-style.
- They can use functions such as:
  - File operations
  - Memory read/write
  - Breakpoint setting/clearing
- Can be executed
  - Automatically at specific times
  - manually
  - associated with breakpoints


Run on the PC!  
Not Downloaded  
to the hardware!

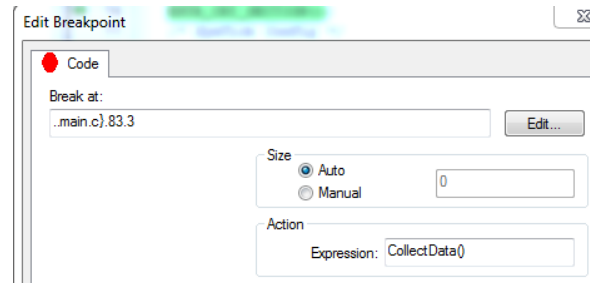
```
execUserSetup ()
{
    __message "execUserSetup() called.\n";

    /* Opens the log (text) file */
    __fHandle = __openFile("ADC.log", "w");
    ...
}
```

Setup.mac



Expression	Result
 collectData ()	
<click to add>	



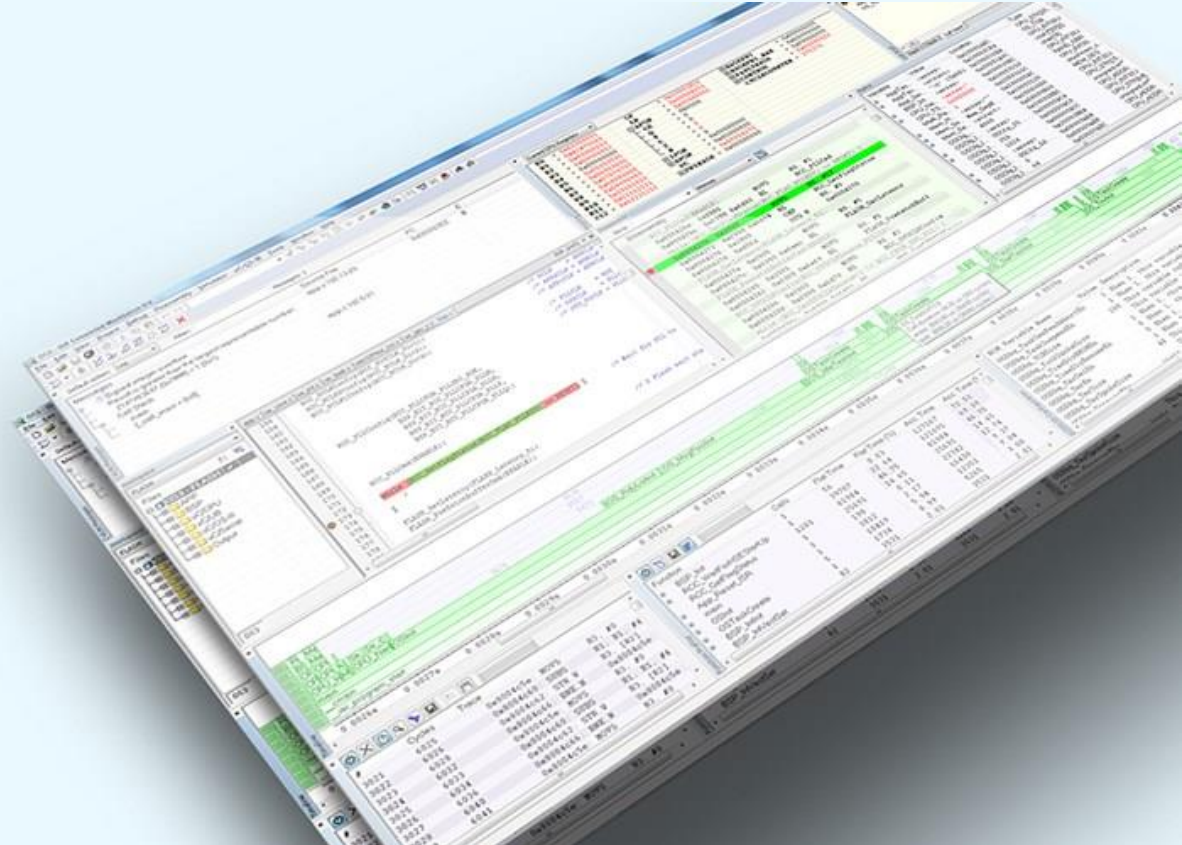
Code

Break at:

Size:  Auto  Manual

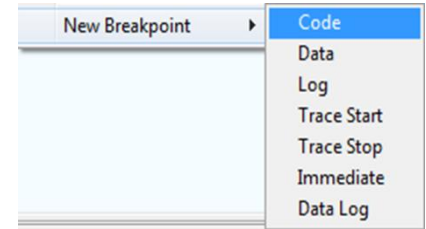
Action:

# C-SPY Macros DEMO



# Conditional Breakpoints

- Execution stops at breakpoint.
- Condition is evaluated.
- Execution is resumed if condition is false.
- Condition can be any expression including C-SPY macro functions.
- Works for the breakpoint types **Code** and **Log**.



# Conditional Breakpoints DEMO

