# Getting the most out of RISC-V development tools
# 发挥专业RISC-V开发工具的最大效能

*Ryan Sheng / 盛磊*
*IAR Systems (China)*

# IAR Systems

- 业内领先的嵌入式软件开发工具供应商
- 220+员工，专业的研发、销售和技术支持团队
- 总部位于瑞典，纳斯达克-斯德哥尔摩证交所上市公司

**37 years in the industry**
**11 offices worldwide**

乌普萨拉
慕尼黑
巴黎
剑桥
东京
首尔
上海
达拉斯
波士顿
旧金山
洛杉矶

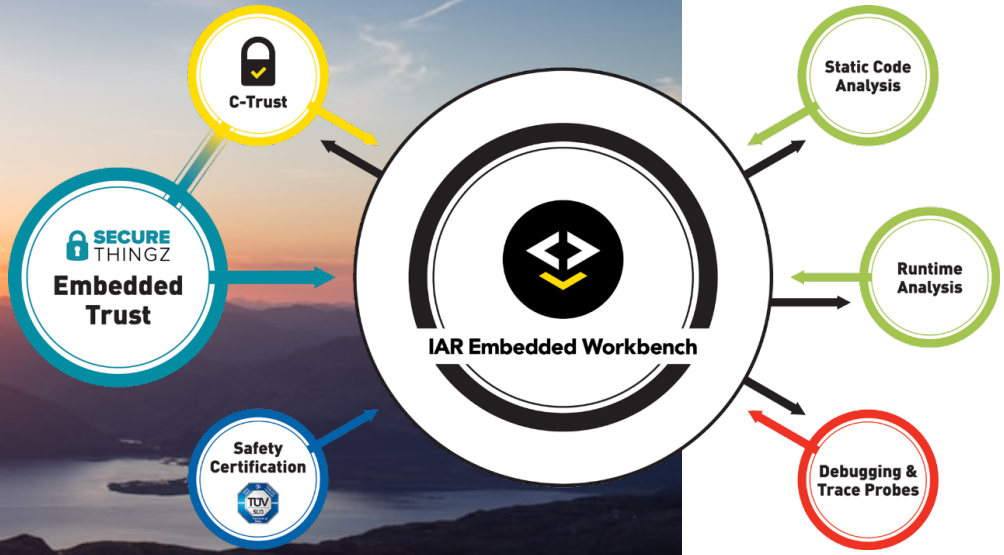
Distributor representation in 40+ countries

2019

- Sales SEK 405.6M
- Operating profit SEK 108.4M
- Cash flow SEK 105.7M

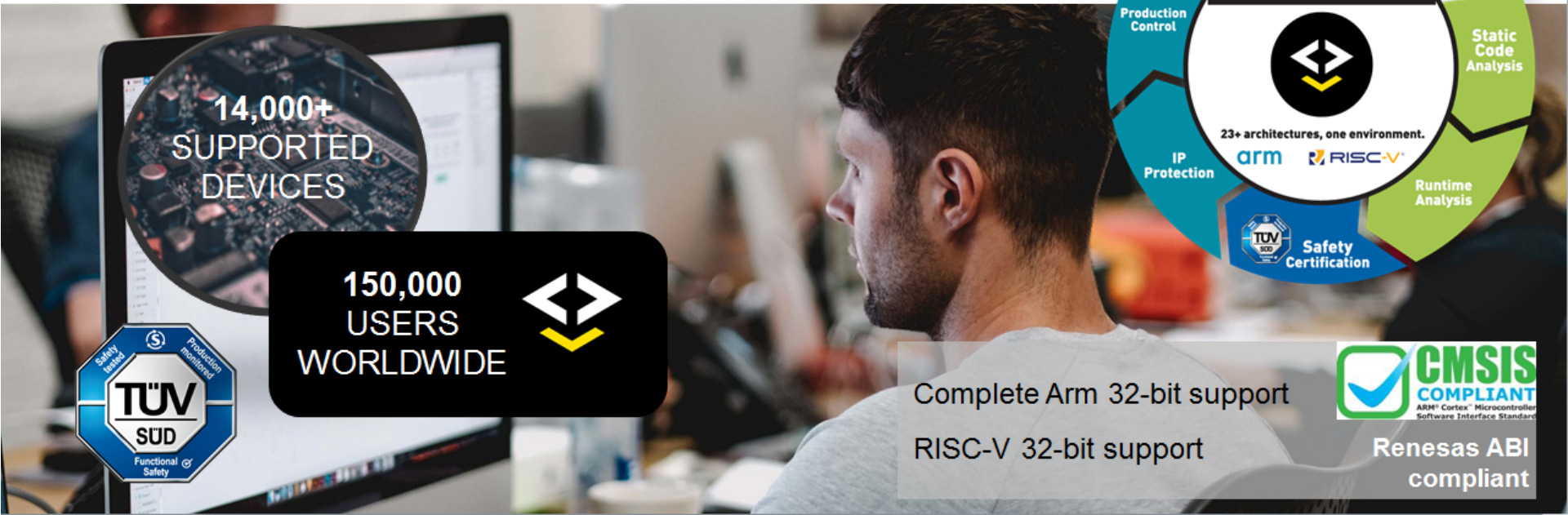
# More than an ordinary toolbox

- ✓ 代码质量（Code quality）
- ✓ 功能安全（Safety）
- ✓ 信息安全（Security）



# IAR Embedded Workbench

- ✓ 杰出的编译优化技术
- ✓ 丰富的代码调试功能
- ✓ 专业的本地技术支持



# Support for 14,000+ devices

# Different architecture, One solution

**All available 8-,16- and 32-bit MCUs**

| | | | |
|---|---|---|---|
| Cortex-M0 | Cortex-R8 | AVR | H8 |
| Cortex-M0+ | Cortex-A5 | AVR32 | STM8 |
| Cortex-M1 | Cortex-A7 | RX | ColdFire |
| Cortex-M3 | Cortex-A8 | RL78 | HCS12 |
| Cortex-M4 | Cortex-A9 | RH850 | S08 |
| Cortex-M7 | Cortex-A15 | 78K | MAXQ |
| Cortex-M23 | ARM11 | SuperH | CR16C |
| Cortex-M33 | ARM9 | V850 | SAM8 |
| Cortex-R4 | ARM7 | R32C | **RISC-V** |
| Cortex-R5 | SecurCore | M32C | |
| Cortex-R52 | 8051 | M16C | |
| Cortex-R7 | MSP430 | R8C | |

# Partnership with SiFive

Company News

**Date:** December 3, 2018

## IAR Systems and SiFive partner to meet customers' demands for professional solutions for RISC-V

**Establish partnership for delivering increased possibilities for powerful RISC-V implementations**

RISC-V Summit, Santa Clara, California—December 3, 2018—IAR Systems®, the future-proof supplier of software tools and services for embedded development, and SiFive, the leading provider of commercial RISC-V processor IP, announce that they have formed a partnership in order to deliver increased possibilities for powerful RISC-V implementations with compact code and high performance.

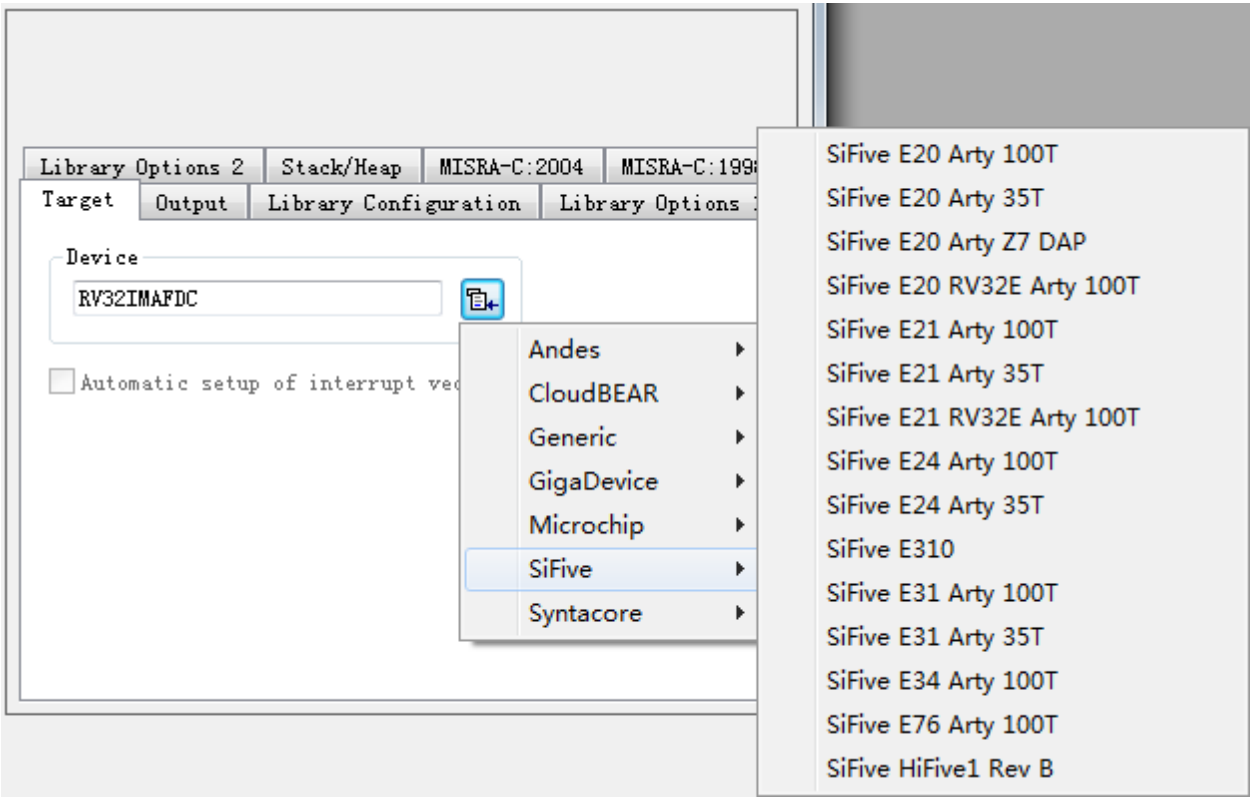
# IAR Embedded Workbench

## RISC-V core and device support in version 1.30

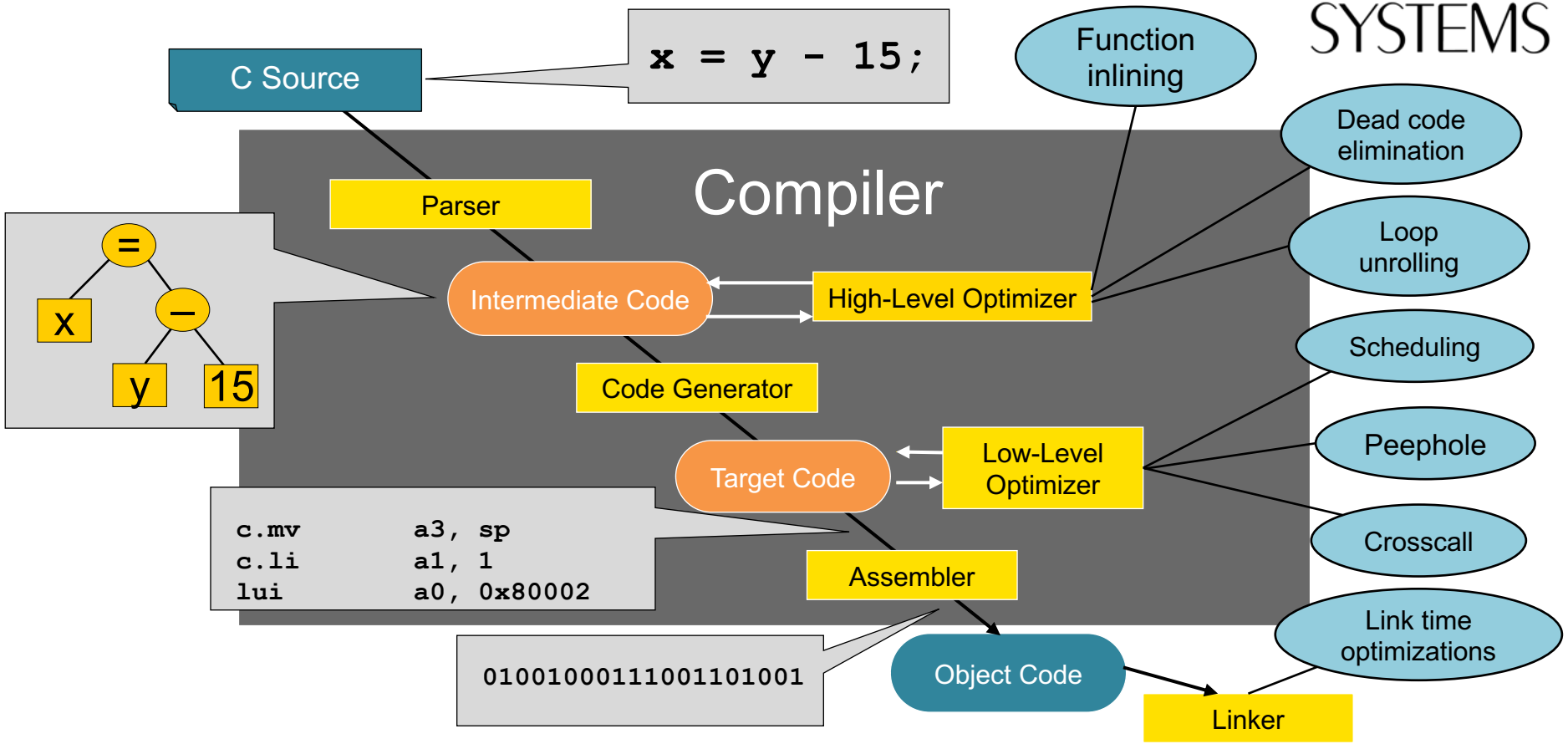
| RISC-V ISA (32-bit) | |
|---|---|
| RV32I | 基本整数指令集 |
| RV32E | 嵌入式整数指令集 |

| Standard extensions | |
|---|---|
| M | 整数乘除指令 |
| A | 原子指令 |
| F | 单精度浮点运算指令 |
| D | 双精度浮点运算指令 |
| C | 压缩指令 |

| Vendor |
|---|
| SiFive, Andes, CloudBEAR, Syntacore |
| GigaDevice GD32VF103xxxx |



# 编译过程和代码优化


C Source
x = y - 15;
Function inlining
Dead code elimination
Compiler
Parser
=
x
–
y
15
Loop unrolling
Intermediate Code
High-Level Optimizer
Scheduling
Code Generator
Peephole
Target Code
Low-Level Optimizer
c.mv a3, sp
c.li a1, 1
lui a0, 0x80002
Crosscall
Assembler
Link time optimizations
0100100011100110100 1
Object Code
Linker

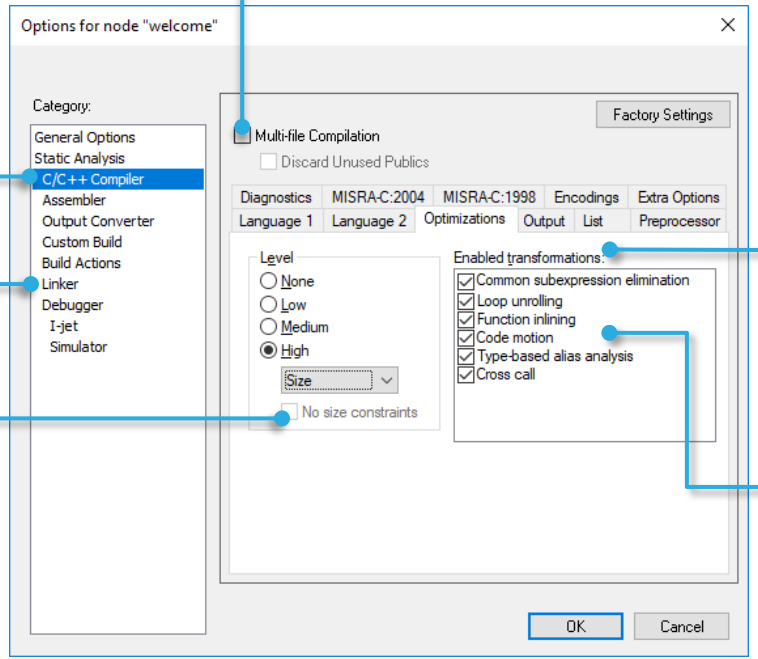
# 编译器功能设置

Multiple optimization levels for code size and execution speed

The linker can remove unused code

Option to maximize speed with no size constraints

Multi-file compilation allows the optimizer to operate on a larger set of code




## Language standards

- ISO/IEC 14882:2015 (C++14, C++17)
- ISO/IEC 9899:2018 (C18)
- ANSI X3.159-1989 (C89)

- IEEE 754 standard for floating-point arithmetic

Major features of the optimizer can be controlled individually

Balance between size and speed by setting different optimizations for different parts of the code

## Well-tested

Commercial test suites

- Plum-Hall Validation test suite
- Perennial EC++VS
- Dinkum C++ Proofer

In-house developed test suite >500,000 lines of C/C++ test code run multiple times

- Processor modes
- Memory models
- Optimization levels

# 自定义指令

- `.insn` directive generates custom instructions which are not directly supported by the assembler.
- `.insn` directive can be used to inline assembly code in programs written in C and C++.
- `.insn` directive generates instructions on all RISC-V instruction formats.

**Intrinsic-like function example**

```
long __insn_example(int lhs, int rhs) {
    long res;
    /* Generates AND r,r,r */
    asm (".insn r 0x33, 0x7, 0x0, %0, %1, %2" \
         : "=r" (res) \
         : "r" (lhs), "r" (rhs) );
    return res;
}
```

**Instruction example**

```
/* equivalent to sltiu a0, a1, 0x40 */
.insn i 0x13,  0x3, a0, a1, 0x40

/* equivalent to sb a0, 4(a1) */
.insn s 0x23,  0,   a0, 4(a1)

/* also equivalent to sb a0, 4(a1) */
.insn s STORE, 0,   a0, 4(a1)
```

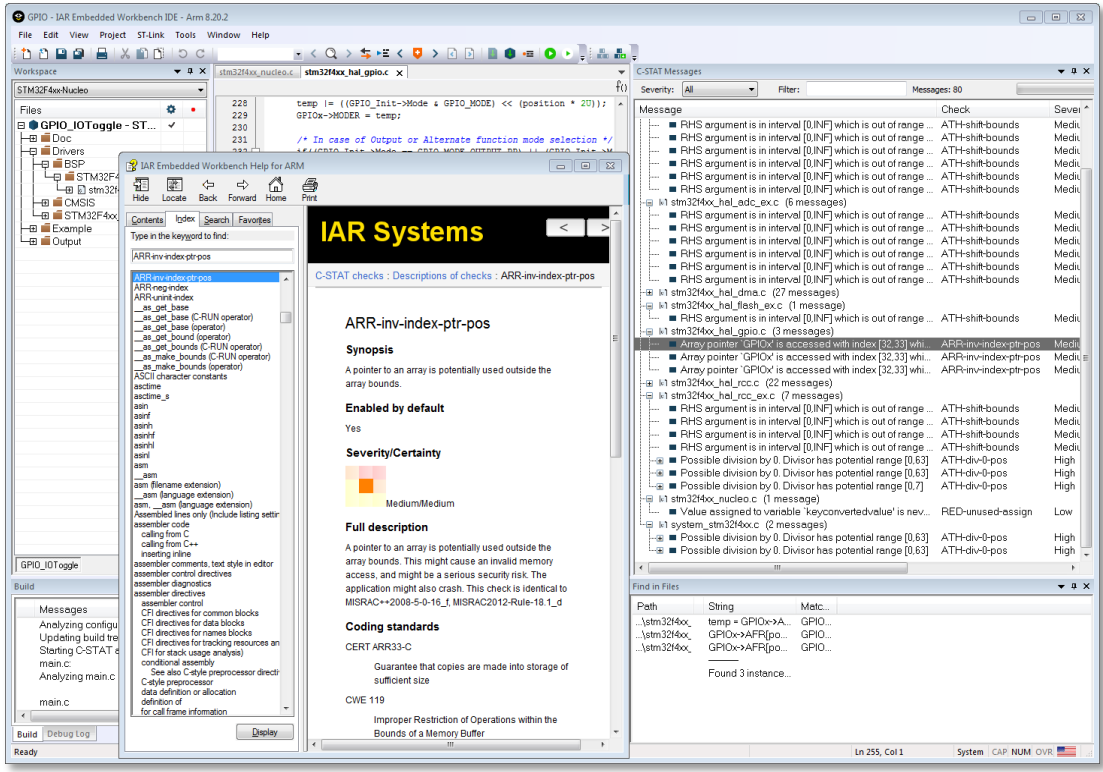
**.insn directives**

```
.insn r    op7, f3, f7, rd, rs1, rs2
.insn r    op7, f3, f7, rd, rs1, rs2, rs3
.insn r4   op7, f3, f2, rd, rs1, rs2, rs3
.insn i    op7, f3, rd, rs1, expr
.insn i    op7, f3, rd, rs1, expr (rs1)
.insn s    op7, f3, rd, rs1, expr (rs1)
.insn sb   op7, f3, rd, rs1, expr
.insn sb   op7, f3, rd, expr(rs1)
.insn b    op7, f3, rd, rs1, expr
.insn u    op7, f3, rd, expr
.insn uj   op2, rd, expr
.insn cr   op2, f4, rd, rs1
.insn ci   op2, f2, rd, expr
.insn ciw  op2, f3, rd', expr
.insn ca   op2, f6, f2, rd', rs2'
.insn cb   op2, f3, rs1', expr
.insn cj   op2, f3, expr
.insn cs   op2, f3, rs1', rs2', expr
```

# 借助分析工具C-STAT提升代码质量

- Advanced analysis of C/C++ code
- Fully integrated within IAR Embedded Workbench for RISC-V
- Check compliance with **MISRA C:2004**, **MISRA C++:2008** and **MISRA C:2012**
- Include ~250 checks mapping to hundreds of issues covered by **CWE** and **CERT C/C++**
- Intuitive and easy-to-use settings with flexible rule selection
- Support for command line execution
- Extensive and detailed documentation

CWE (Common Weakness Enumeration): http://cwe.mitre.org
CERT (Computer Emergency Response Team): http://www.cert.org



# 调试和跟踪工具

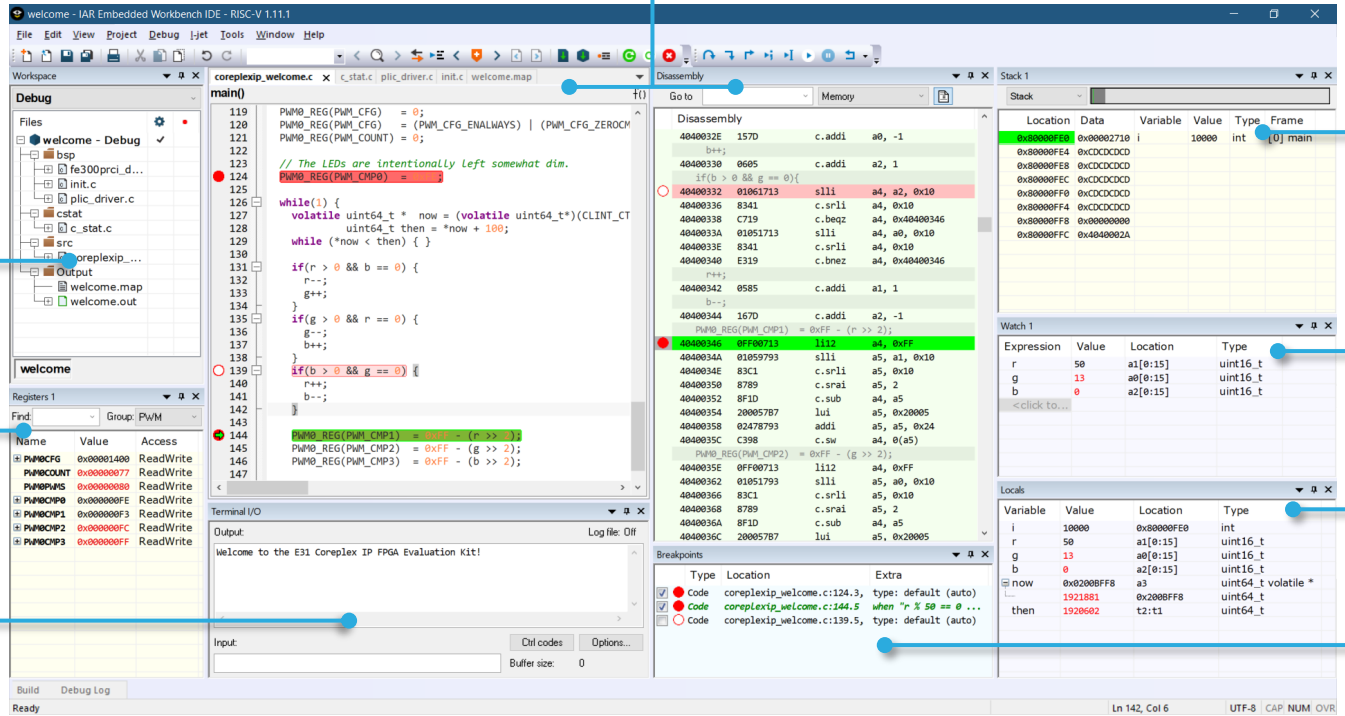
| | I-jet | I-jet Trace (4-bit MIPI20 model) | I-jet Trace (16-bit Mictor/MIPI20 model) |
|---|---|---|---|
| JTAG/SWD speed | 48 MHz | 100 MHz | 100 MHz |
| Download speed (RAM) | 1.89 MByte/s | 3.73 MByte/s | 3.73 MByte/s |
| SWO max. bandwidth | ~30 Mbps | ~60 Mbps | ~60 Mbps |
| Available trace memory | - | 64M or 256M bytes | 256M or 1G bytes |
| Trace max. bandwidth | - | 1.2 Gbps | 1.2 Gbps |
| Max streaming speed | 48 MByte/s | ~380 MByte/s | ~380 MByte/s |
| Power sampling resolution | ~160 µA | ~160 µA | ~160 µA |
| Power sampling rate | 200 ksps | 200 ksps | 200 ksps |



# 传统调试功能

- 类C的宏语言
- 内置模拟器
- RTOS内核识别
- 代码跟踪

源代码和反汇编代码窗口




源文件/工程和工作区管理

寄存器

半主机模式I/O窗口

栈使用情况分析

表达式

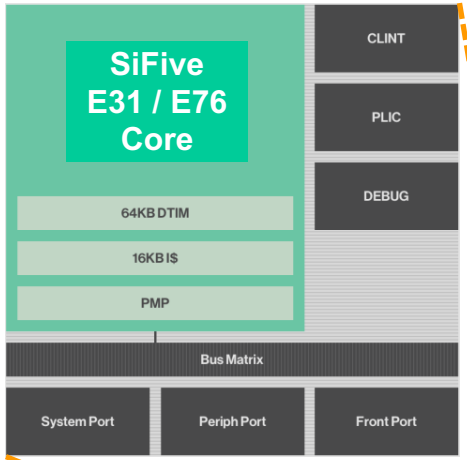
变量

代码和数据断点

# Trace调试功能

- 什么是Trace
  - 与常见的基于断点和单步的调试技术不同，Trace可以在不打断程序正常执行的情况下记录CPU执行的指令流（或数据流）
  - 根据实现方式的不同，Trace缓存可以位于：
    - Trace仿真器中（例如I-jet Trace）：　需要布置Trace总线；缓存容量大。
    - 目标处理器的Memory中：　需要处理器支持；缓存容量小。
- Trace的用途
  - 追溯代码执行的历史轨迹
  - 发现程序中隐藏的问题，例如异常的跳转，或者累积到一定条件时才触发的bug
  - 分析程序的性能和覆盖率，找出占用时间较长或者在测试过程中未能充分执行的函数
- IAR Embedded Workbench for RISC-V现已实现基于Nexus IEEE-ISTO 5001™协议的Trace调试环境，支持SiFive Insight解决方案。

# 演示环境



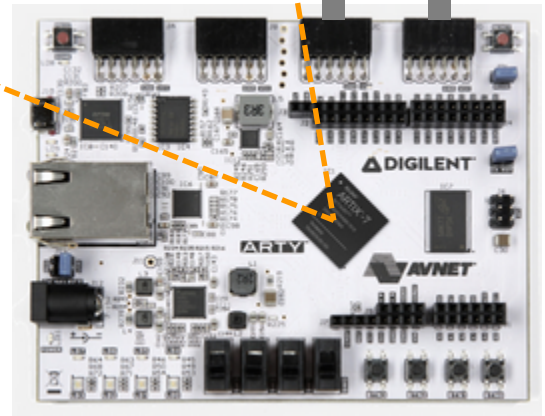


JTAG & Trace

I-jet Trace

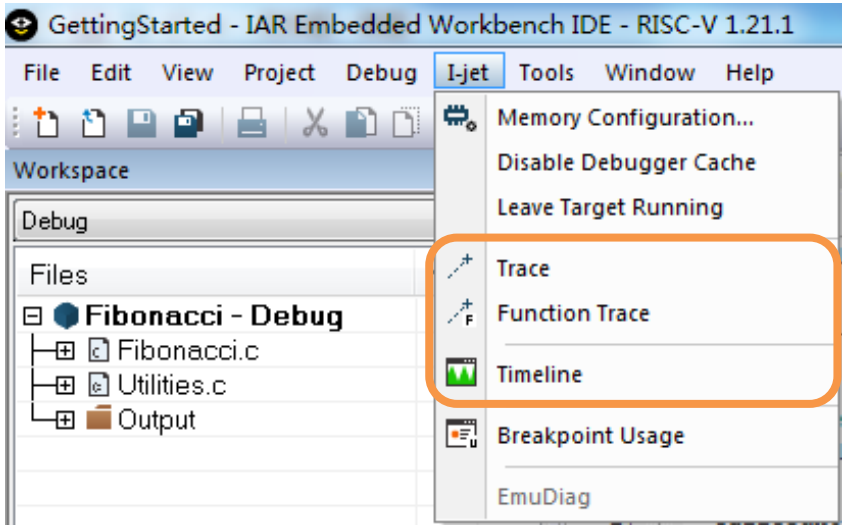
USB

编译/链接/下载/调试

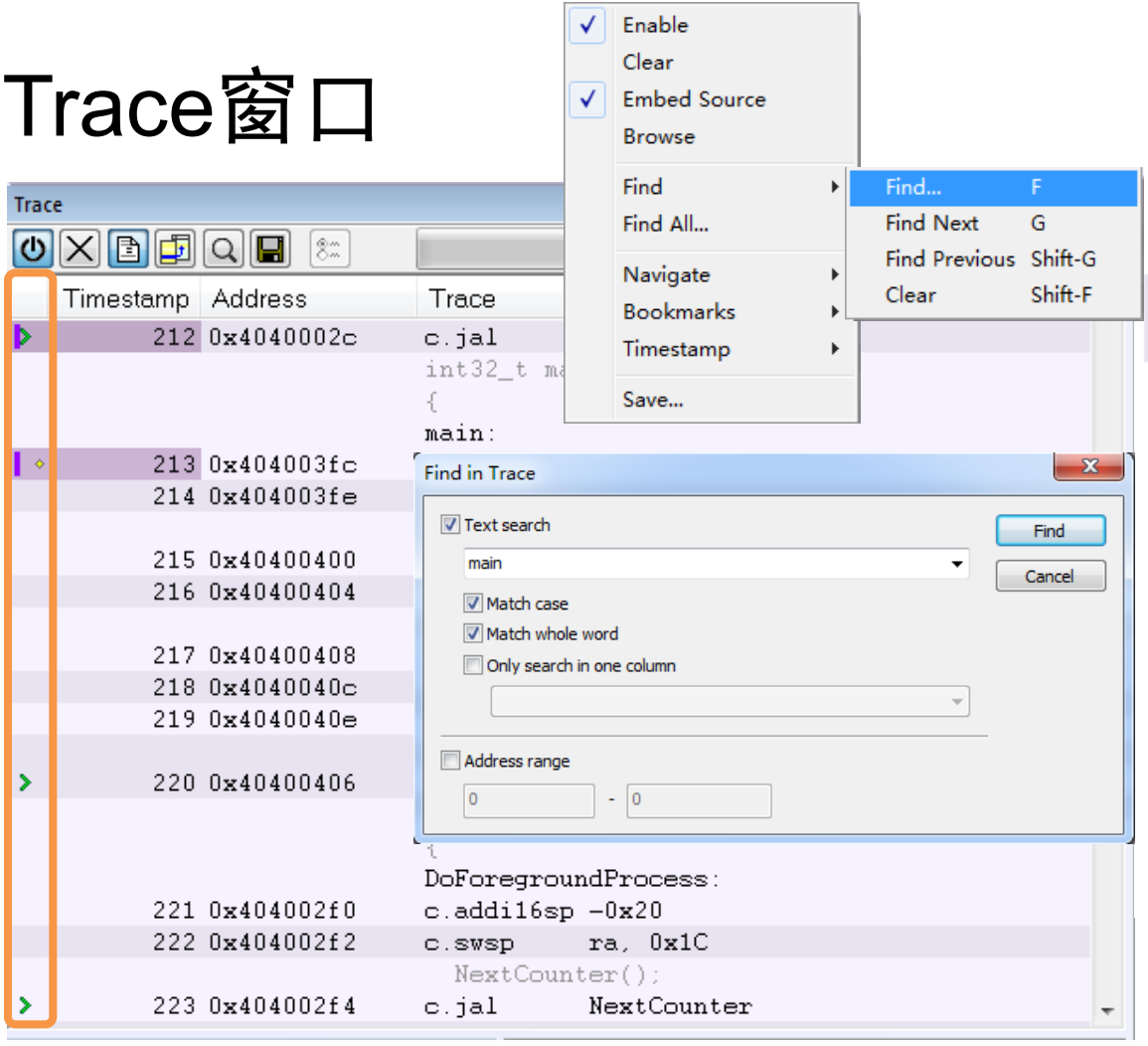

Artix-7 100T
FPGA Board

# Trace菜单




- Trace窗口
  - 显示Trace缓存中记录的指令流
  - 可以混合C/C++源代码显示
- 函数级Trace窗口
  - 从Trace缓存中提取与函数调用和返回相关的指令
  - 只显示函数级别的调用和返回信息
- 时间轴窗口
  - 以时间线为横坐标，用图形化的方式展示Trace内容
  - 函数调用栈（Call Stack）：显示函数调用的层级和时序关系

# Trace窗口




Trace和程序执行的起点

函数调用

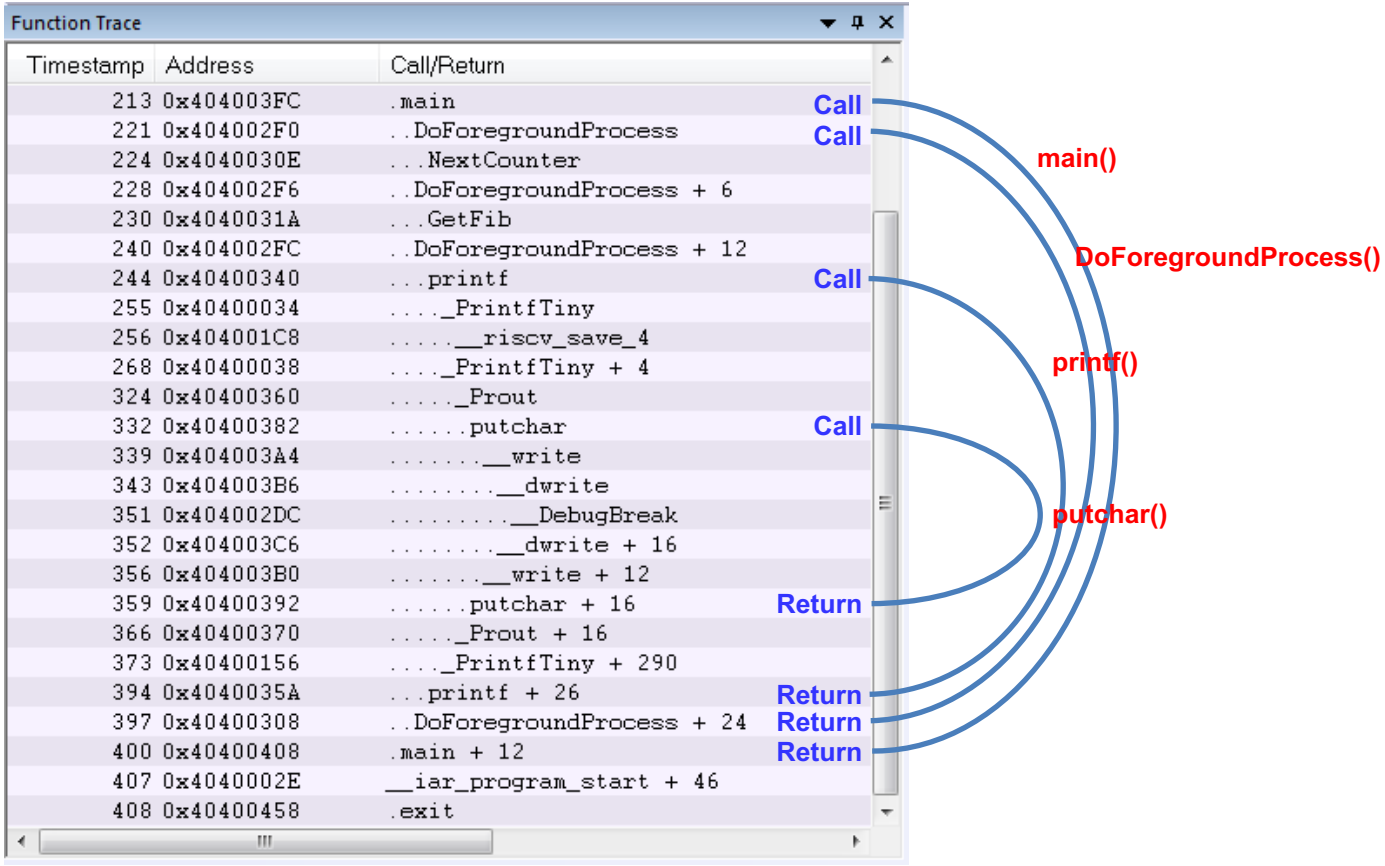
函数返回

导航书签

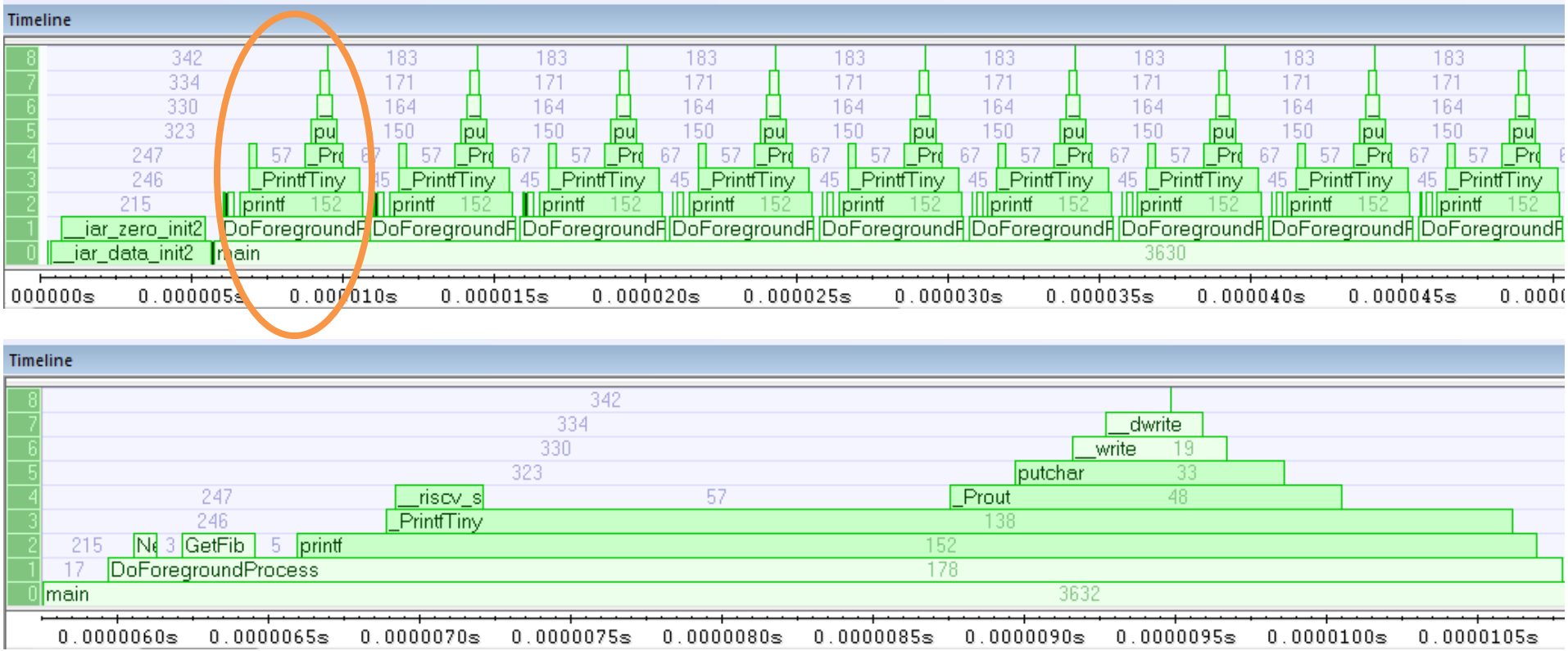
中断

搜索的结果

# 函数级Trace窗口

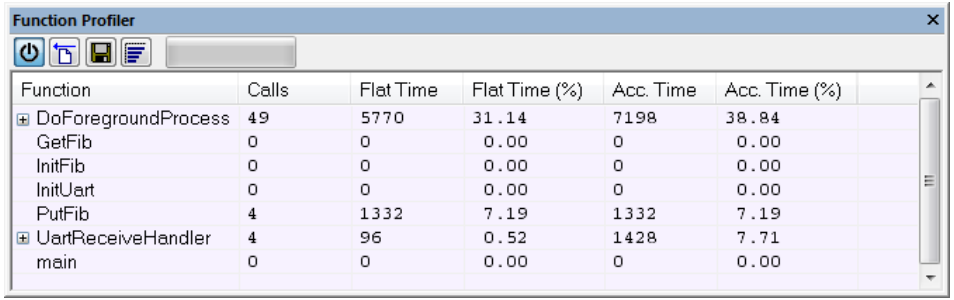

Function Trace
Timestamp Address Call/Return
213 0x404003FC .main
221 0x404002F0 ..DoForegroundProcess
224 0x4040030E ...NextCounter
228 0x404002F6 ..DoForegroundProcess + 6
230 0x4040031A ...GetFib
240 0x404002FC ..DoForegroundProcess + 12
244 0x40400340 ...printf
255 0x40400034 ....__PrintfTiny
256 0x404001C8 .....__riscv_save_4
268 0x40400038 ....__PrintfTiny + 4
324 0x40400360 .....__Prout
332 0x40400382 ......putchar
339 0x404003A4 .......__write
343 0x404003B6 ........__dwrite
351 0x404002DC .........__DebugBreak
352 0x404003C6 ........__dwrite + 16
356 0x404003B0 .......__write + 12
359 0x40400392 ......putchar + 16
366 0x40400370 .....__Prout + 16
373 0x40400156 ....__PrintfTiny + 290
394 0x4040035A ...printf + 26
397 0x40400308 ..DoForegroundProcess + 24
400 0x40400408 .main + 12
407 0x4040002E __iar_program_start + 46
408 0x40400458 .exit
Call
Call
Call
Call
Return
Return
Return
Return
main()
DoForegroundProcess()
printf()
putchar()

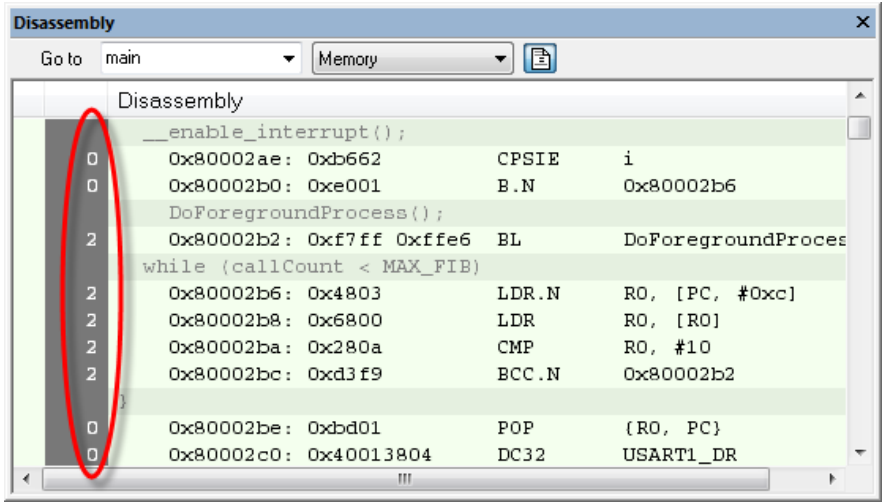
# 时间轴窗口 – 函数调用栈


Timeline
__iar_zero_init2
__iar_data_init2
DoForegroundP
main
_PrintfTiny
printf
3630
0.000000s
0.000005s
0.000010s
0.000015s
0.000020s
0.000025s
0.000030s
0.000035s
0.000040s
0.000045s
Timeline
main
DoForegroundProcess
GetFib
printf
_PrintfTiny
__riscv_s
_Prout
putchar
_write
__dwrite
3632
0.0000060s
0.0000065s
0.0000070s
0.0000075s
0.0000080s
0.0000085s
0.0000090s
0.0000095s
0.0000100s
0.0000105s

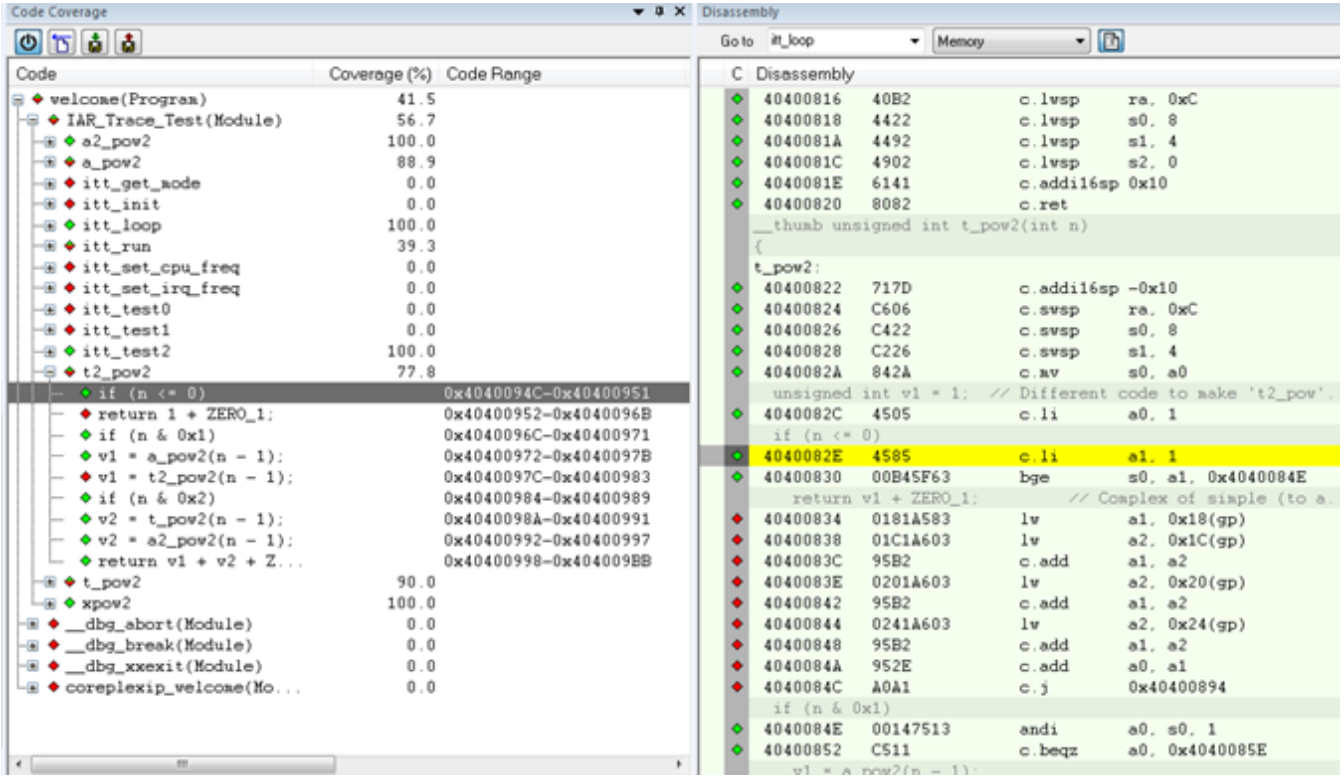
# 代码占用时间分析（Profiling）

- Function profiling
  - 在整个程序中，找出实际运行时占用CPU时间较长的函数
  - 可按调用次数或CPU占用时间进行排序
- Instruction profiling
  - 在指令级别分析程序的性能，找出被执行次数较多的指令
  - 有助于对代码进行精细的调整



# 代码覆盖率分析（Code coverage）




- 验证是否所有的代码都已经被执行过至少一次（例如，满足测试的要求）。
- 找出无法到达（没有被执行）的代码片段或指令。

# Summary

- 运用专业的开发工具，满足您的项目对代码质量和开发进度的需求
- IAR Embedded Workbench是在嵌入式软件开发领域应用最广泛的集成编译调试环境
- 借助领先的编译优化技术，达成代码体积和运行速度两方面的目标
- 在功能丰富的图形化调试环境中跟踪程序的运行，去除软件bug，还可以对程序的性能和代码覆盖率进行分析
- 通过内置的C-STAT静态分析工具，检查代码与常用C/C++编程标准的契合程度，及早发现隐藏的缺陷，有效提升代码质量

- *IAR Systems (China): 021-63758658, sales.cn@iar.com*

# IAR RISC-V免费开发套件

- IAR RISC-V Evaluation Kit
  - IAR Embedded Workbench for RISC-V, Evaluation License
  - RISC-V development board
  - I-jet Lite debug tool
- 免费提供给具有商业开发项目的企业
- 申请链接：[www.iar.com/evalkit](www.iar.com/evalkit)