



Stay ahead with the latest advances
in RISC-V development tools

Shawn A. Prestridge
US FAE Team Leader, IAR Systems

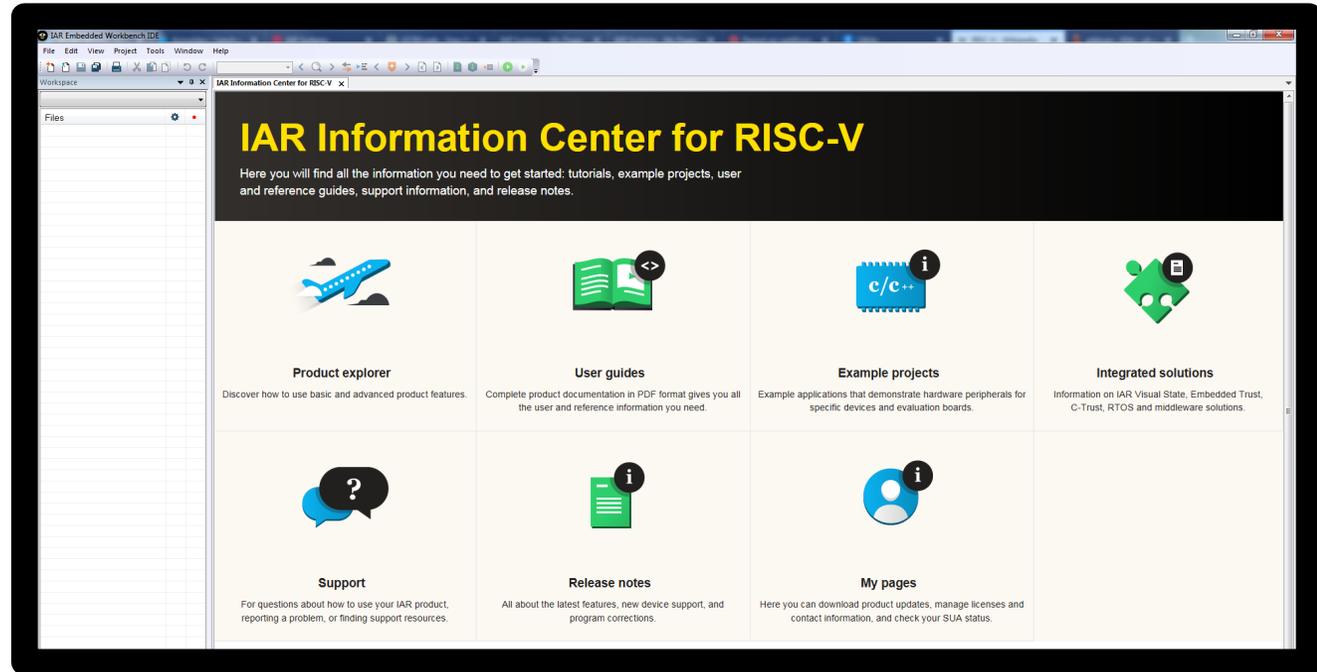
Agenda

- Development tools for RISC-V
- Compiler and optimizations
- Debugging
- Trace
- Code analysis
- Demo

Development tools for RISC-V

IAR Embedded Workbench for RISC-V

- Complete build and debug toolchain for RISC-V
- User-friendly IDE features and broad ecosystem integration
- Outstanding performance through sophisticated optimization technology
- Comprehensive debugger
- ISO/ANSI C/C++ compliance with support for C18 and C++17
- Integrated static analysis



Device support

Supported base instruction sets:

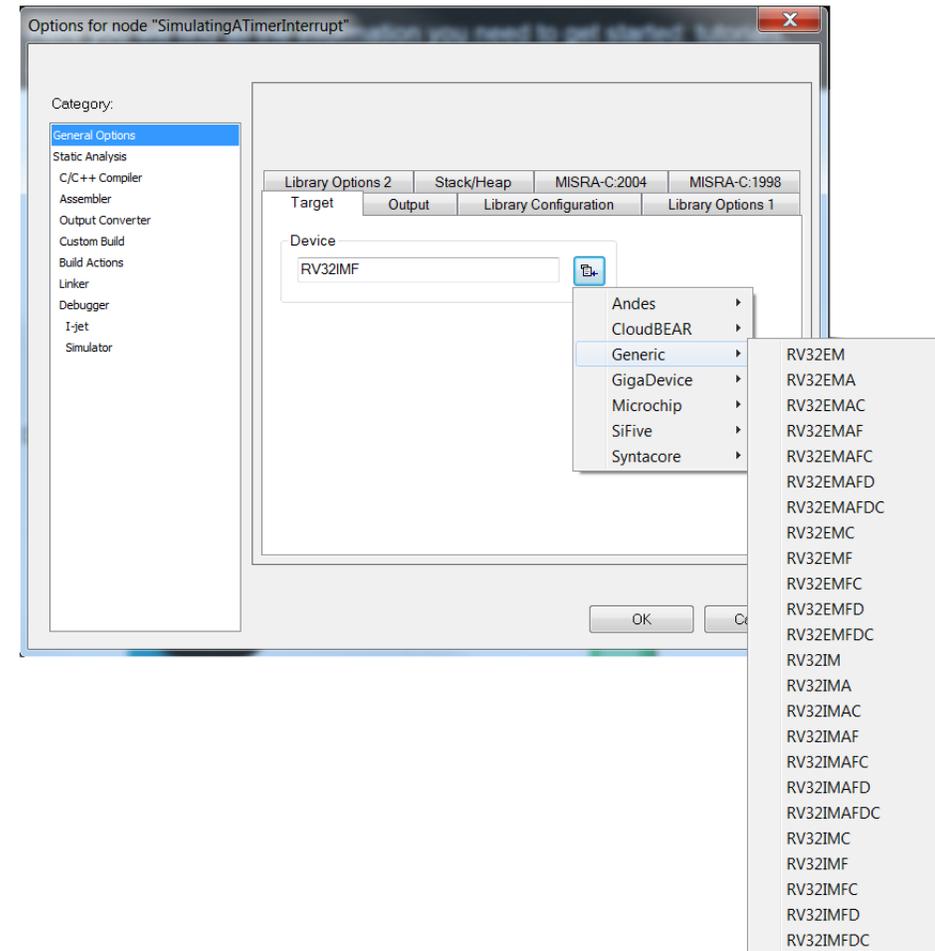
- RV32I Base Int instruction set
- RV32E Base Int instruction set (embedded)

Supported extensions:

- M integer mul & div
- A atomic instructions
- F single precision float
- D double precision float
- C compressed instructions
- P DSP and Packed SIMD

Device support for RISC-V devices from:

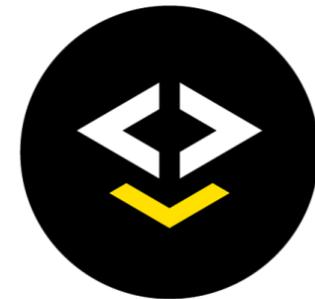
- Andes
- CloudBEAR
- Microchip
- SiFive
- Syntacore
- GigaDevice



Compiler and optimizations

Compiler design

- Proprietary design based on over 37 years of experience
- Based on a platform that is common among different targets to handle global optimizations, etc.
- Target unique backend for specific adaptations and optimizations
- RISC-V specifics:
 - Primary focus will be on adding standard extensions
 - Initial prioritization is on code size
 - Canary stack protection



Compiler technology

Multiple optimizations 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

Major features of the optimizer can be controlled individually

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

Language standards

- ISO/IEC 14882:2015 (C++14, C++17)
- ISO/IEC 9899:2012 (C11, C18)
- ANSI X3.159-1989 (C89)
- IEEE 754 standard for floating-point arithmetic

Well-tested

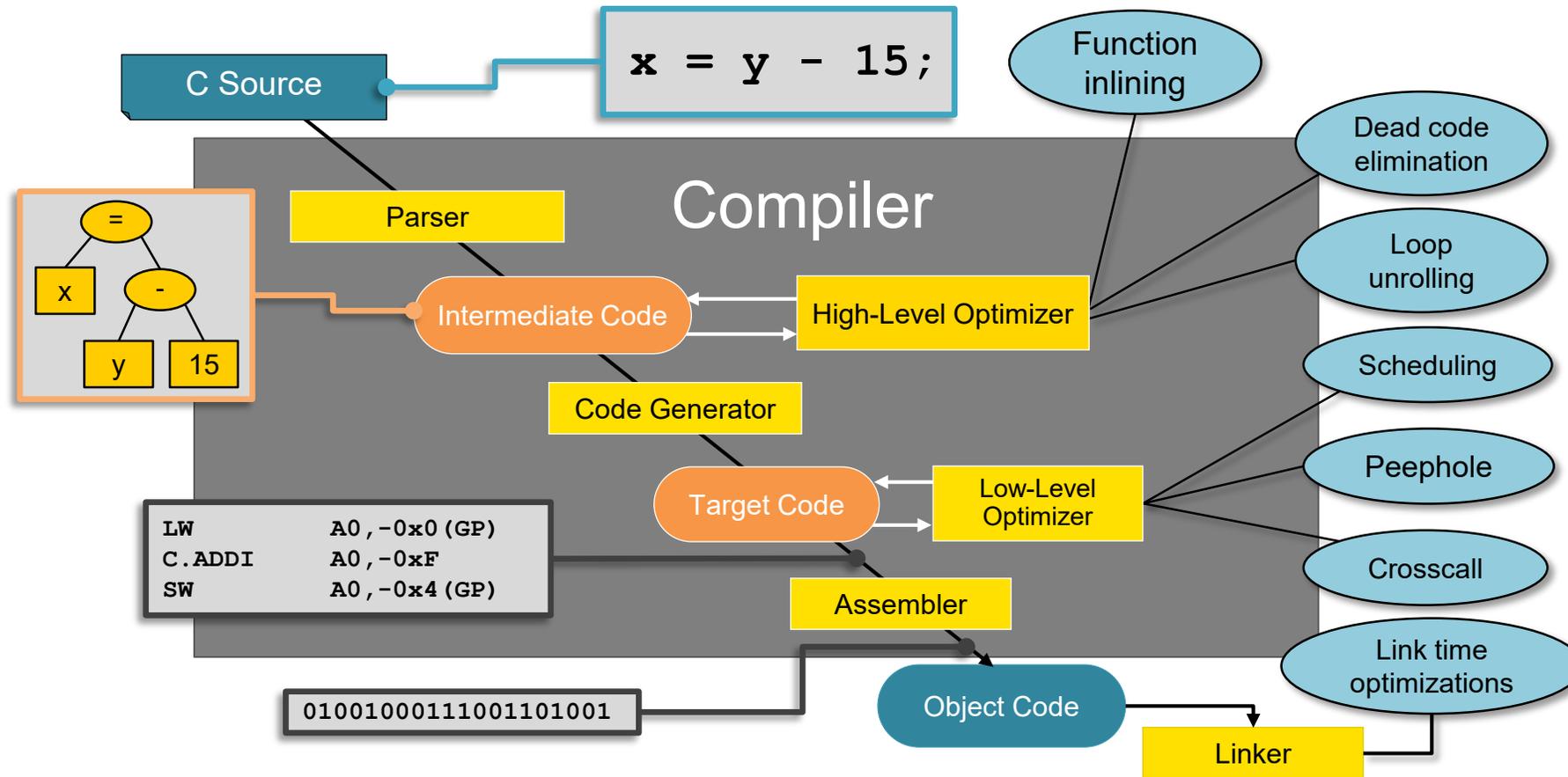
Commercial test suites

- 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

Compiler optimizations



Debugging

Debugging

The IAR Embedded Workbench for RISC-V offers a fully integrated debug solution

- Built on the same C-SPY debugger interface that is used in the IAR Embedded Workbench for different architectures
- Built in Simulator
- RTOS awareness plugins
- Support for C-SPY debug macros that can be used to automate debug sessions
- IAR Systems I-jet debug probe support (3rd party debug probe support will be added in future releases)
- Trace support



Debugging

Integrated debugger for source and disassembly debugging

- C like macro system
- Built-in simulator
- RTOS aware
- Trace

Stack usage

Watch

Locals

Registers

Dockable windows and tab groups

Complex breakpoints

Semihosted terminal I/O

The screenshot displays the IAR Embedded Workbench IDE interface. The main window shows the source code for a C program named 'main()'. The code includes comments about generating 20 interrupts and a loop that prints the result of a calculation. The disassembly window shows the corresponding assembly instructions, including 'addi', 'c.mv', 'c.jal', 'c.srrsi', 'c.li', 'c.lw', 'c.ltu', 'c.disable_interrupt()', 'c.jal', 'c.printf', 'c.addi', 'c.jal', 'c.for', 'c.addi', 'c.andi', 'c.swp', 'c.addi', 'c.jal', 'c.for', 'c.andi', 'c.li', 'c.bit', 'c.lwsp', 'c.lwsp', 'c.lwsp', 'c.lwsp', 'c.addi16sp', and 'c.ret'. The stack window shows the current stack frame with variables 'var1' and 'var2'. The watch window shows the expression 'counter < 20' with a value of 20. The locals window shows the local variables 'var1' and 'var2'. The registers window shows the CPU registers (ra, sp, gp, tp, s0/fp, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10) and their values. The terminal I/O window shows the output of the program, including the start and end of the interrupt service routine.

Debugging

IAR Systems I-jet debug probe

- Supports RISC-V and Arm cores
- Hi-speed USB 2.0 interface (480Mbps)
- Target power of up to 400mA can be supplied from I-jet with overload protection
- Target power consumption can be measured with $\sim 200\mu\text{A}$ resolution at 200kHz
- JTAG and Serial Wire Debug (SWD) clocks up to 32MHz (no limit on the MCU clock speed)
- Support for SWO speeds of up to 60MHz
- Unlimited flash breakpoints (*to be added for RISC-V)



Trace

Advanced debugging and trace capabilities

The screenshot displays a comprehensive debugging environment with the following components:

- Source Code:** Shows C code for a Fibonacci sequence simulation with interrupt handling. Key lines include: `/* Increment the current number in the Fibonacci sequence. */ ++callCount;` and `while (callcount < MAX_FIB) { DoForegroundProcess(); _disable_interrupt(); }`
- Disassembly:** Shows assembly instructions for the `DoForegroundProcess` function, such as `FEA1A503 lw a0, -0x16(gp)` and `FEB56CE3 bltu a0, a1, 0x800002F2`.
- Trace Table:** Lists execution events with columns for Timestamp, Trace, Read Addr, Read Data, Write Addr, and Write Data. It shows multiple calls to `DoForegroundProcess` at address `0x800002F2`.
- Watch:** Monitors the `callCount` variable, showing its value as `10` at memory location `0x80002000`.
- Registers:** Displays CPU registers (ABI) with values for `ra`, `sp`, `gp`, `tp`, `s0/fp`, `s1`, `s2`, `s3`, `s4`, and `fp`.
- Code Coverage:** Shows coverage percentages for various code blocks, such as `SimulatingSerialPort...` at 86.0% and `SerialPortInterrupt...` at 100.0%.
- Timeline:** A Gantt-style chart showing the execution flow of `DoForegroundProcess`, `UartReceiveHandler`, and `PutFib` over time, with a timer interrupt at `4.76 us`.
- Profiler:** Provides performance metrics for different functions, including `DoForegroundProcess` with 1293 calls and 96718 total execution time.
- Interrupt Log:** Details the timer interrupt event, showing it was triggered at `0x80000258` and took `4.76 us` to execute.
- Terminal I/O:** Shows the output of the program, displaying the Fibonacci sequence: `205.....`, `205.....`, `205.....`.

*Nexus IEEE-ISTO 5001™ compatible trace

Trace

IAR Systems I-jet Trace debug probe

- Supports RISC-V and Arm cores
- SuperSpeed USB 3.0 interface (5 Gbps) Fully compatible with USB 2.0 (480 Mbps)
- JTAG and Serial Wire Debug (SWD) clocks up to 100MHz (no limit on the MCU clock speed)
- Up to 150MHz trace clock
- 64-bit time stamp with CPU cycle accuracy for timing analysis
- Support for SWO using Manchester and UART, at up to 200 Mbps

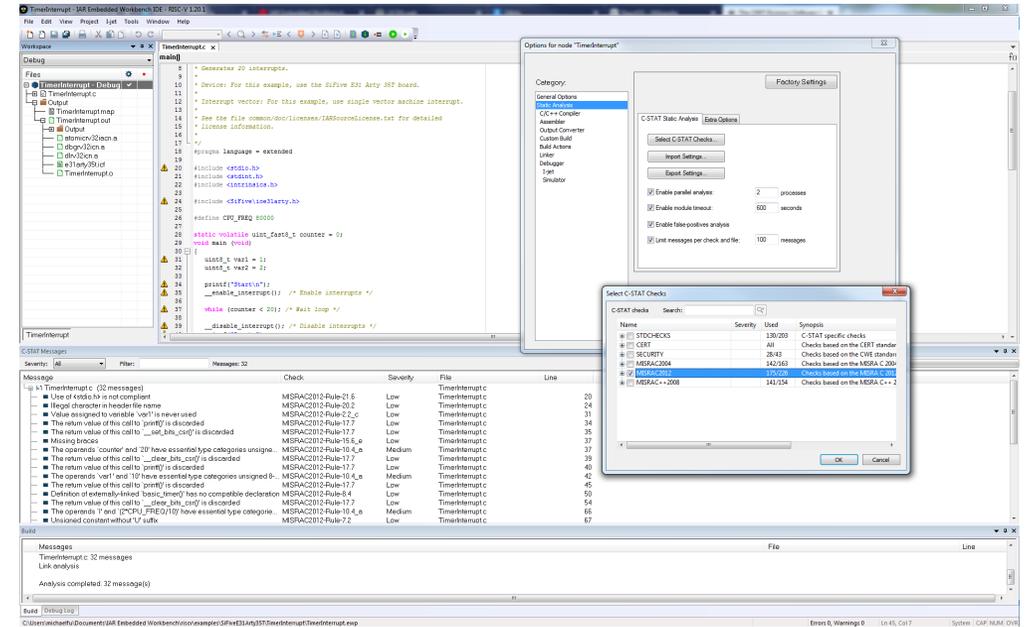


Code analysis

Code analysis

IAR Systems C-STAT static code analysis

- Complete static analysis tool fully integrated in IAR Embedded Workbench
- Intuitive and easy-to-use settings with flexible rule selection
- Support for export/import of selected checks
- Support for command line execution
- Extensive and detailed documentation
- Checks compliance with MISRA C:2004, MISRA C++:2008 and MISRA C:2012
- Includes ~250 checks mapping to hundreds of issues covered by CWE and CERT C/C++



<http://cwe.mitre.org/>

<https://www.sei.cmu.edu/about/divisions/cert/index.cfm>

Demo



Summary

- Advanced toolchain for RISC-V
- Sophisticated compiler technology
- Comprehensive C-SPY debugger including Simulator
- Built in static code analysis C-STAT
- More than an ordinary toolbox