Professional development tools for RISC-V

Ryan Sheng, ryan.sheng@iar.com, 021-63758658
IAR Systems (China)
2019.12.20

IAR SYSTEMS

ETCC

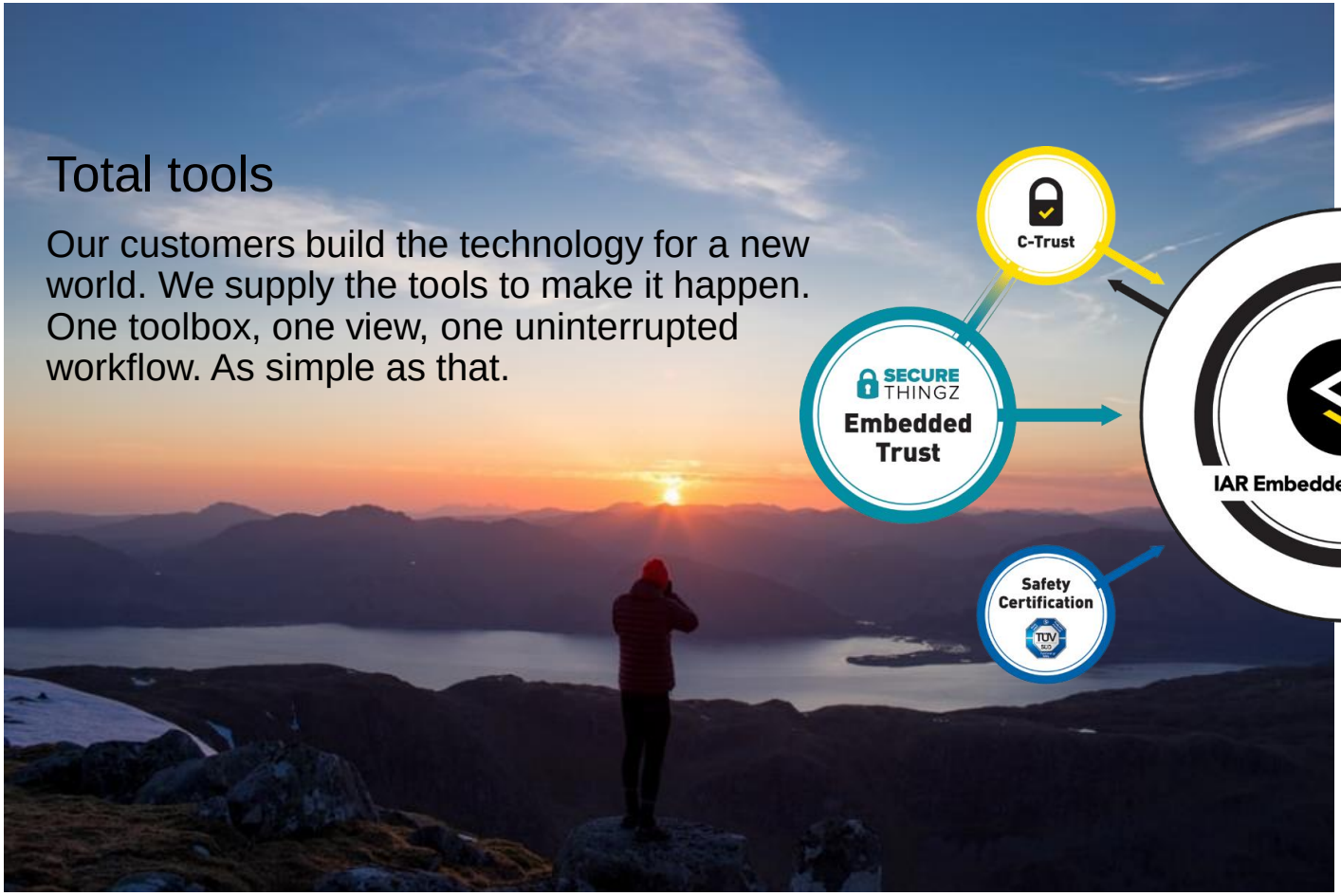
2019年中国嵌入式技术大会
EMBEDDED TECHNOLOGY
Conference China 2019

IAR Systems

- World-leading embedded development tools vendor
- Established in 1983
- Headquartered in Uppsala, Sweden
- 200 employees, 11 offices in EMEA / APAC / US
- Listed on Stockholm/NASDAQ-OMX

IAR SYSTEMS

Total quality • Total safety • Total security



Total tools

Our customers build the technology for a new world. We supply the tools to make it happen. One toolbox, one view, one uninterrupted workflow. As simple as that.

IAR SYSTEMS

IAR Embedded Workbench C/C++ Compiler and Debugger IDE

Most widely used embedded software development tools

User-friendly IDE features and broad ecosystem integration

Industry leading optimization for code size and speed

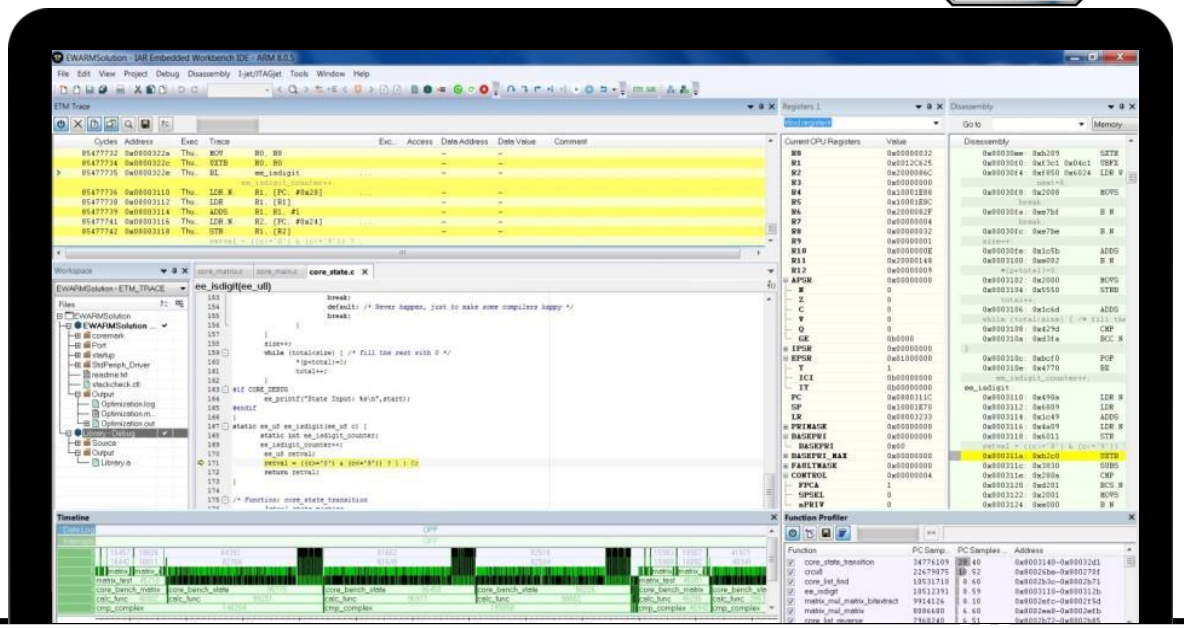
ISO/ANSI compliance with C18 and C++17

Comprehensive graphical debugger interface

Integrated code analysis add-ons

Functional safety certified

Global support & service



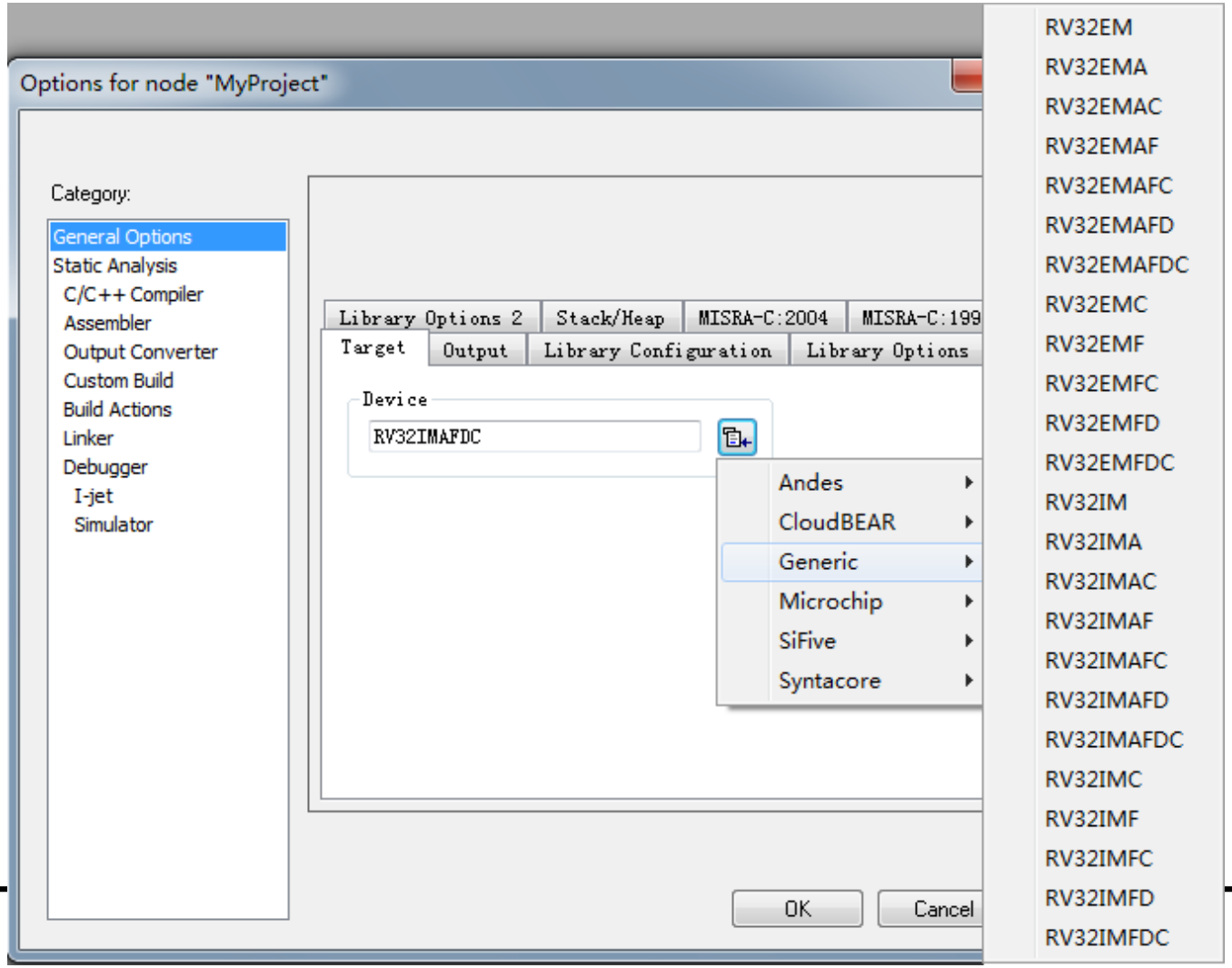
IAR SYSTEMS

Support for 13,000+ 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	

Device support for RISC-V



RV32I
Base Integer Instruction Set

RV32E
Base Integer Instruction Set (embedded, 16 registers)

Supported extensions

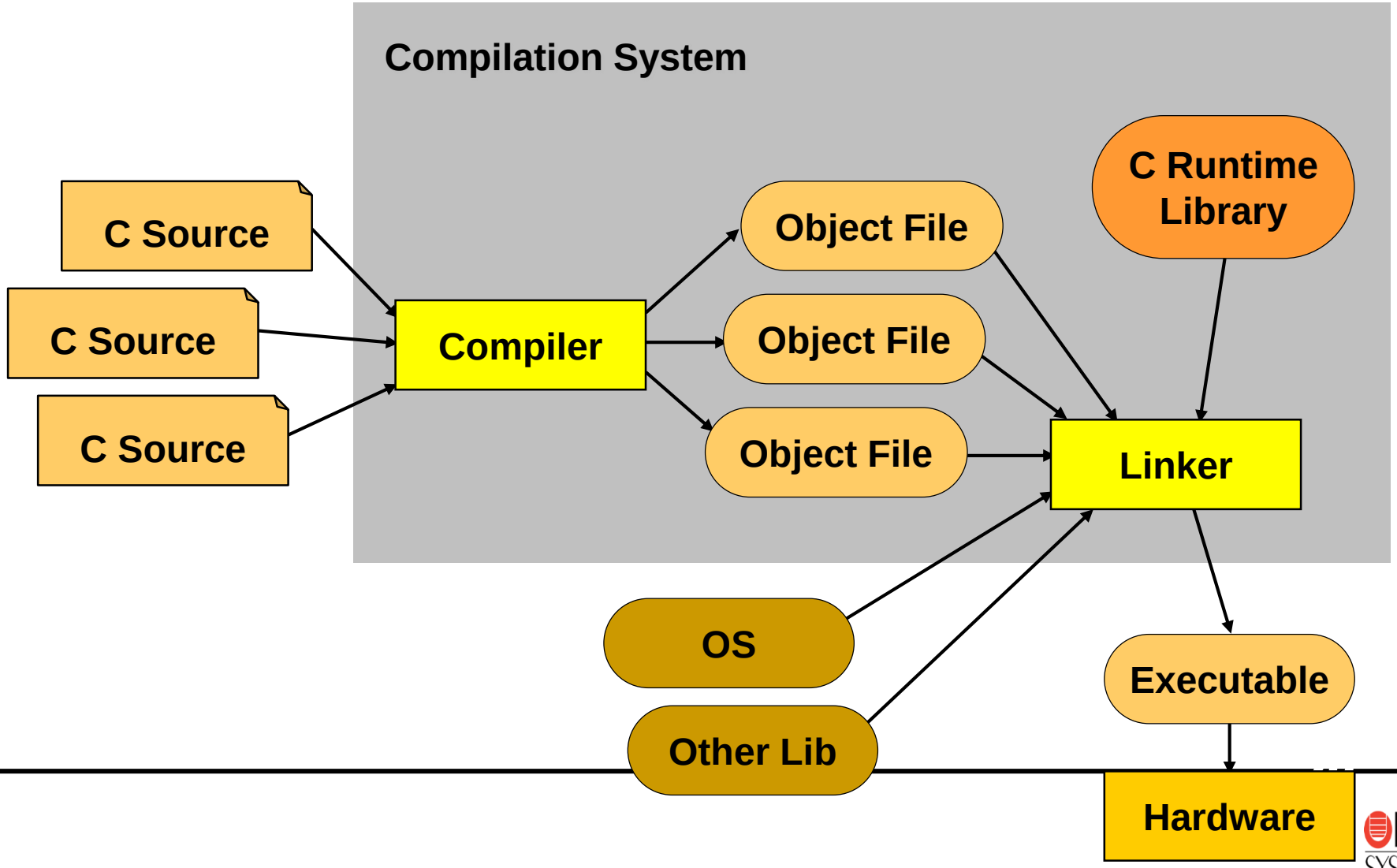
M	integer mul & div
A	atomic
F	single precision float
D	double precision float
C	compressed

Supported IP vendors

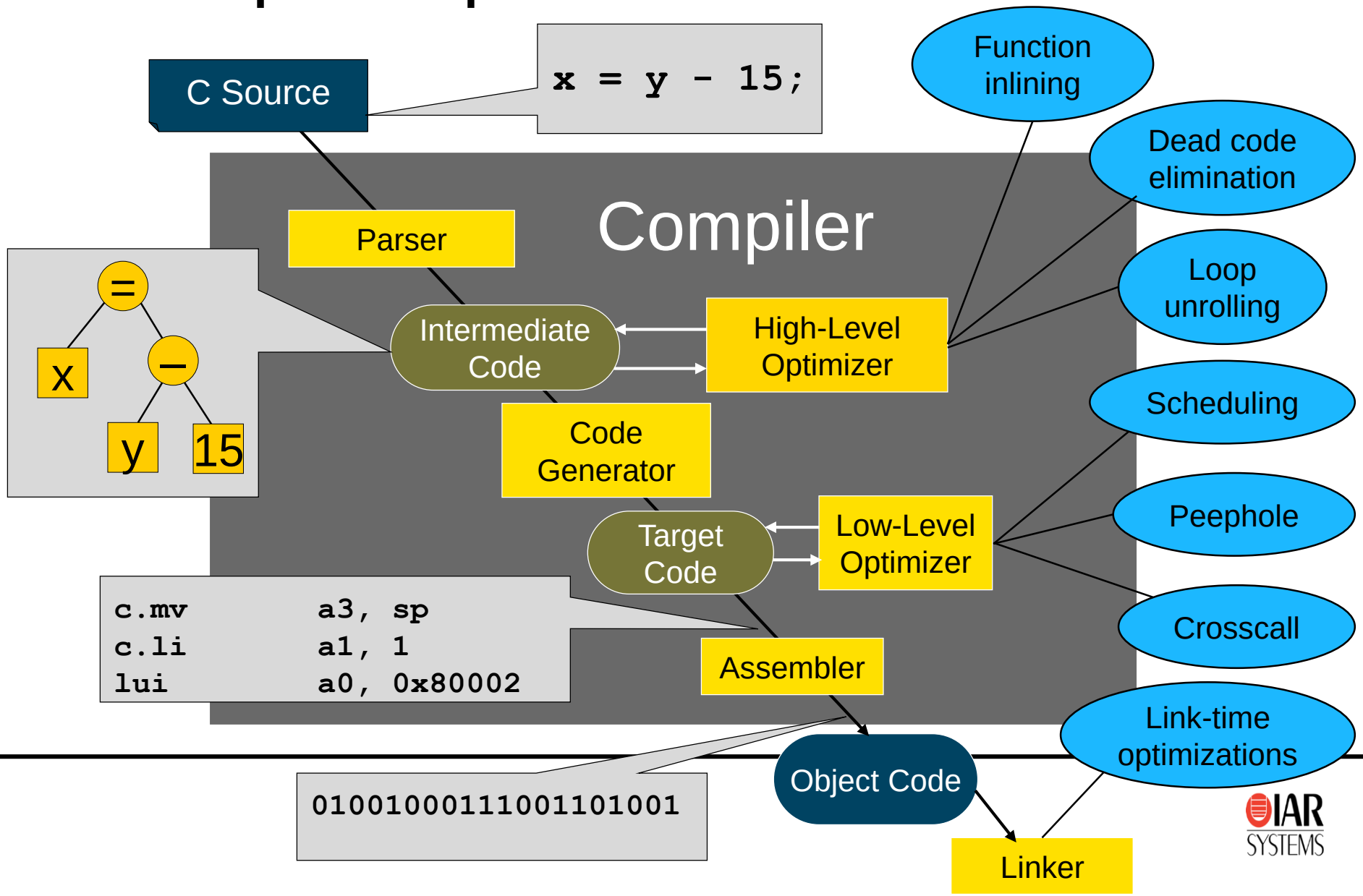
Andes
CloudBEAR
Microchip
SiFive
Syntacore
……

IAR SYSTEMS

Compilation system



Compiler optimizations



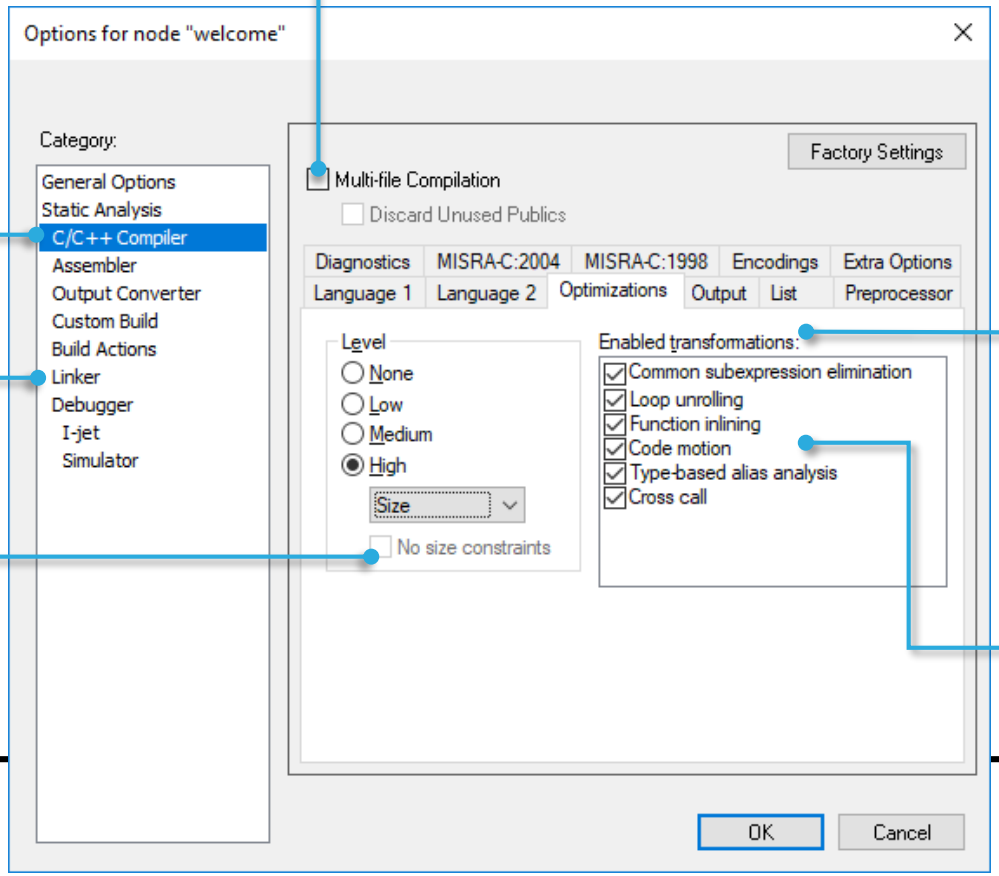
Controlling optimizations

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)

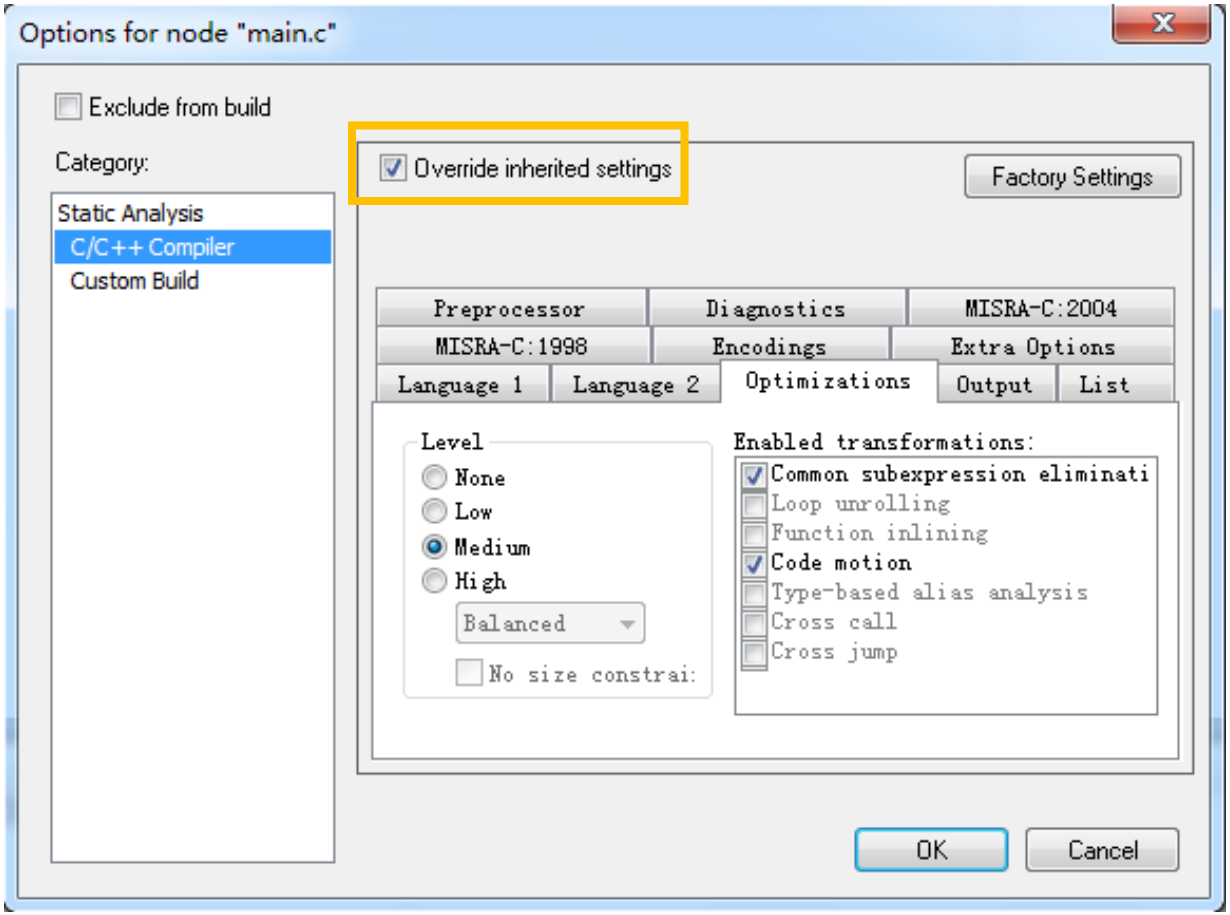
- IEEE 754 standard for floating-point arithmetic

Major features of the optimizer can be controlled individually

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

IAR SYSTEMS

Controlling optimizations




```
#pragma optimize=high
unsigned int GetFib(int n)
{
  if ((n > 0) && (n <= MAX_FIB))
  {
    return (Fib[n-1]);
  }
  else
  {
    return 0;
  }
}
```

IAR SYSTEMS

Speed, size or both?

Optimization	Effect	
Common sub-expressions	Speed ↑	Size ↓
Loop unrolling	Speed ↑	Size ↑
Function inlining	Speed ↑	Size ↑
Code motion	Speed ↑	Size →
Dead code elimination	Speed →	Size ↓
Static clustering	Speed ↑	Size ↓
Instruction scheduling	Speed ↑	Size →
Peephole	Speed ↑	Size ↓
Cross call	Speed ↓	Size ↓

Challenges on optimization

- **Size**
  - Compared to more complex instruction sets, RISC-V have challenges especially when it comes to code size
  - Arithmetic with higher resolution than the natural data size yields larger code
  - Absence of carry flags and instructions to save and restore multiple registers
- **Speed**
  - When it comes to speed, RISC-V is competitive
  - More speed optimizations in future releases

Our initial target will be reducing code size for small embedded systems. Our main focus has always been to supply the best code size and speed on the market.

GCC attributes

- In the extended language mode, IAR C/C++ compiler supports a selection of commonly used GCC-style attributes
- Use the `__attribute__ ((attribute-list))` syntax for these attributes
- The following attributes are supported in part or in whole

alias	aligned	always_inline	constructor
deprecated	noinline	noreturn	packed
pcs	section	target	transparent_ union
unused	used	volatile	weak

Custom instructions

- The `.insn` directive generates custom instructions which are not directly supported by the assembler
- The `.insn` directive generates instructions on all RISC-V instruction formats

`.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`

op2, op7
unsigned immediate
2 or 7-bit opcode

fN
unsigned immediate for function code
2-7 bits wide

rd, rsN
register field
integer (x0-x31) or FP (f0-f31)

rd', rsN'
compact instruction register field
integer (x8-x15) or FP (f8-f15)

expr
immediate expression

* Please refer to the RISC-V ISA specification sections 2.3 and 12.2 for details on bit-layout

Custom instructions: Example

- The `.insn` directive can be used to inline assembly code in programs written in C and C++
- Built-in constants are available when generating a custom instruction

Intrinsic-like function example

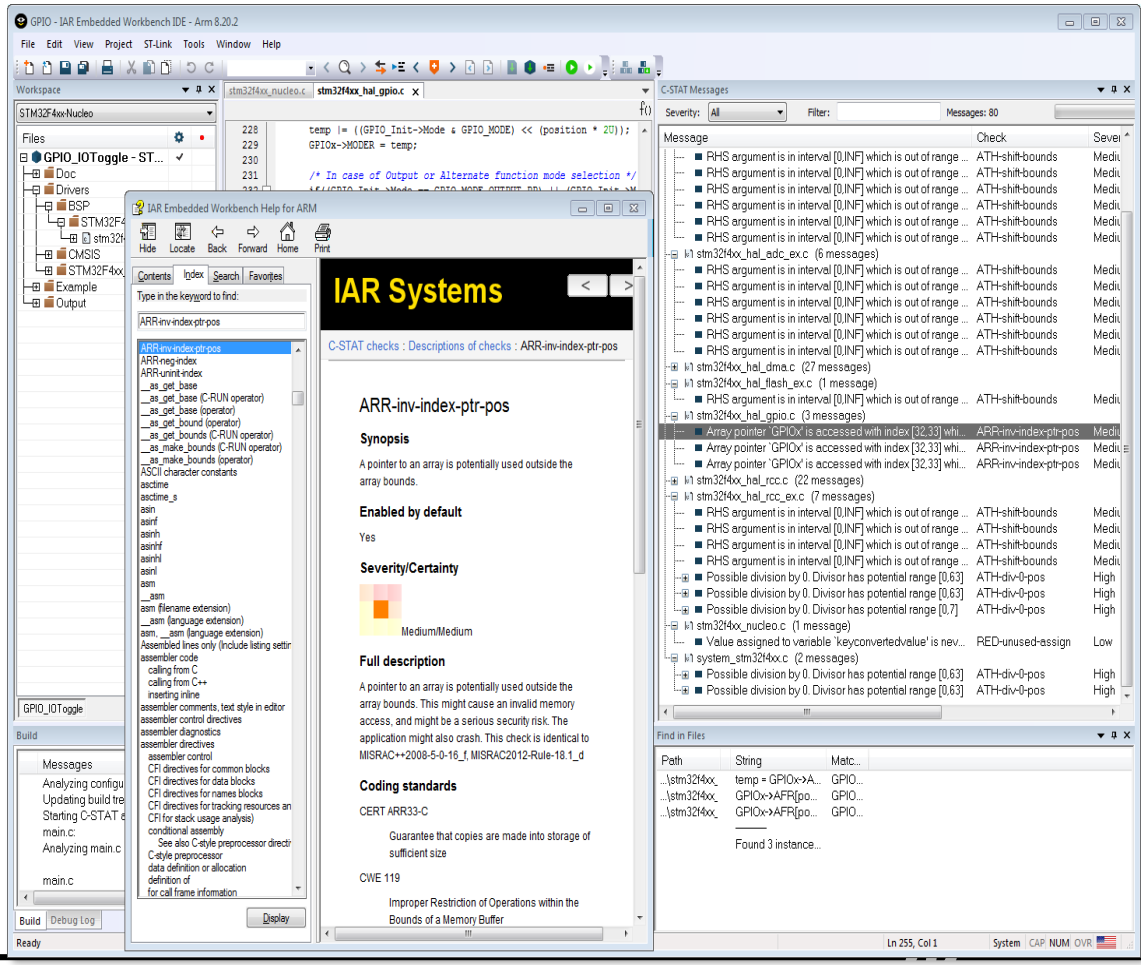
```
long __insn_example(int lhs, int rhs) {
    long res;
    /* Generates AND r,r,r */
    asm (".insn r 0x33, 0x7, 0x0, %0, %1, %2" \
         : "=r" (res) \
         : "r" (lhs), "r" (rhs) );
    return res;
}
```

Intrinsic-like macro example

```
/* Generates AND r,r,r */
#define __insn_example(lhs, rhs) ({ \
    int __lhs = (lhs), __rhs = (rhs), __res; \
    asm (".insn r 0x33, 0x7, 0x0, %0, %1, %2" \
         : "=r" (__res) \
         : "r" (__lhs), "r" (__rhs)); \
    __res; \
  })
```

C-STAT: static code analysis

- Advanced C/C++ code analysis
- Fully integrated within IAR Embedded Workbench
- Check the compliance with **MISRA C:2004**, **MISRA C++:2008** and **MISRA C:2012**
- 250+ checks mapping to hundreds of issues covered by **CWE** and **CERT C/C++**
- Intuitive and easy-to-use settings
- Flexible rules selection
- Extensive and detailed documentation



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

IAR SYSTEMS

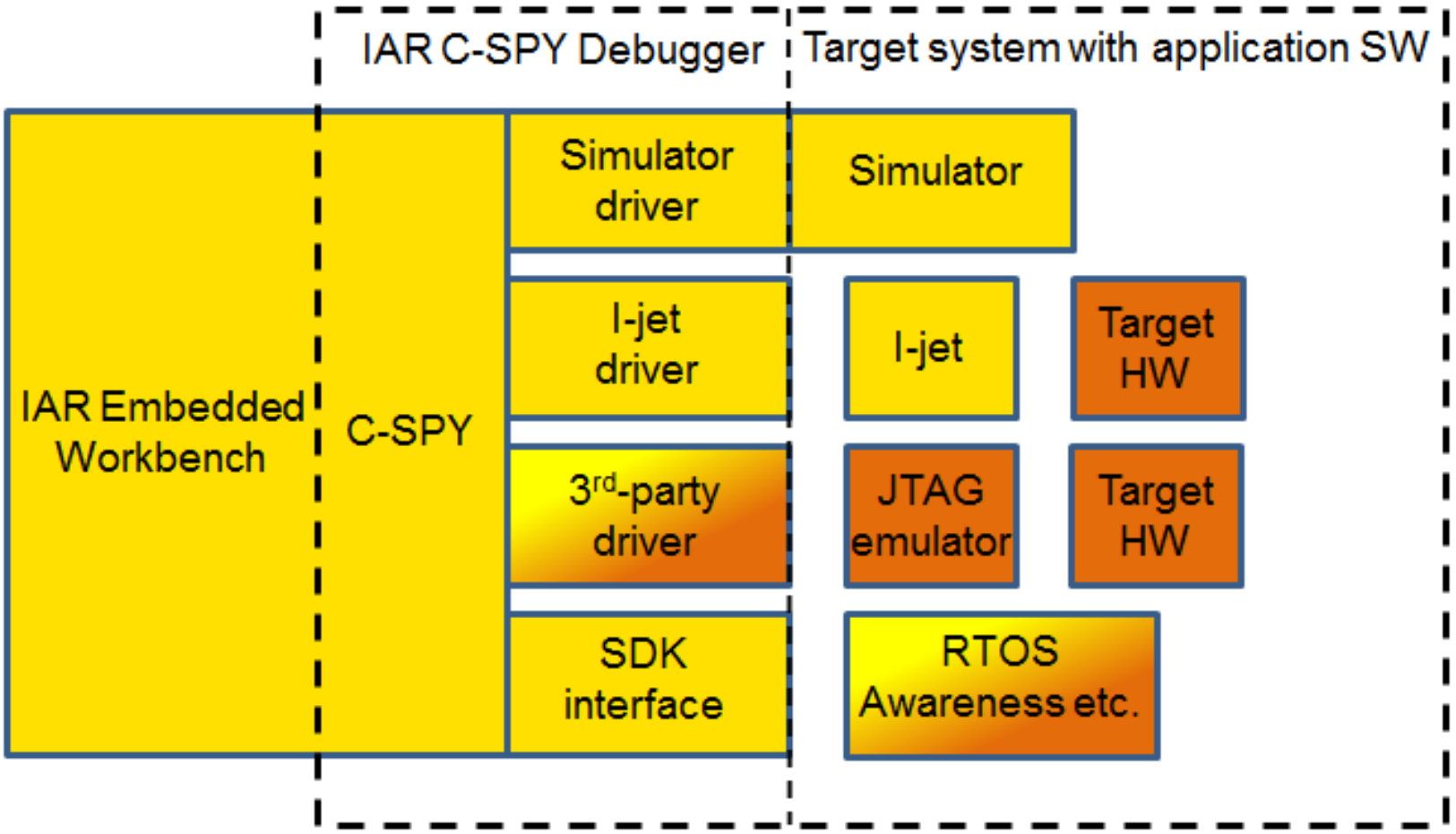
RISC-V debugging

- IAR supports the latest complete RISC-V debug spec, currently v0.13
  - Any additional updates will continuously be supported
- Automated discovery of implemented debug features in a MCU or SoC
  - Implemented debug features like h/w breakpoints, supported extensions etc. are automatically read on connection
- Interrupt and exception catching
  - Distinguish between different priority levels and exception types
- Set different types of breakpoints
  - Code, data, log, trace start and stop, etc.
- Single step on both C/C++ and assembler level
- Full low-level access to all registers, memories and other resources on the MCU or SoC
- Script/macro execution capabilities

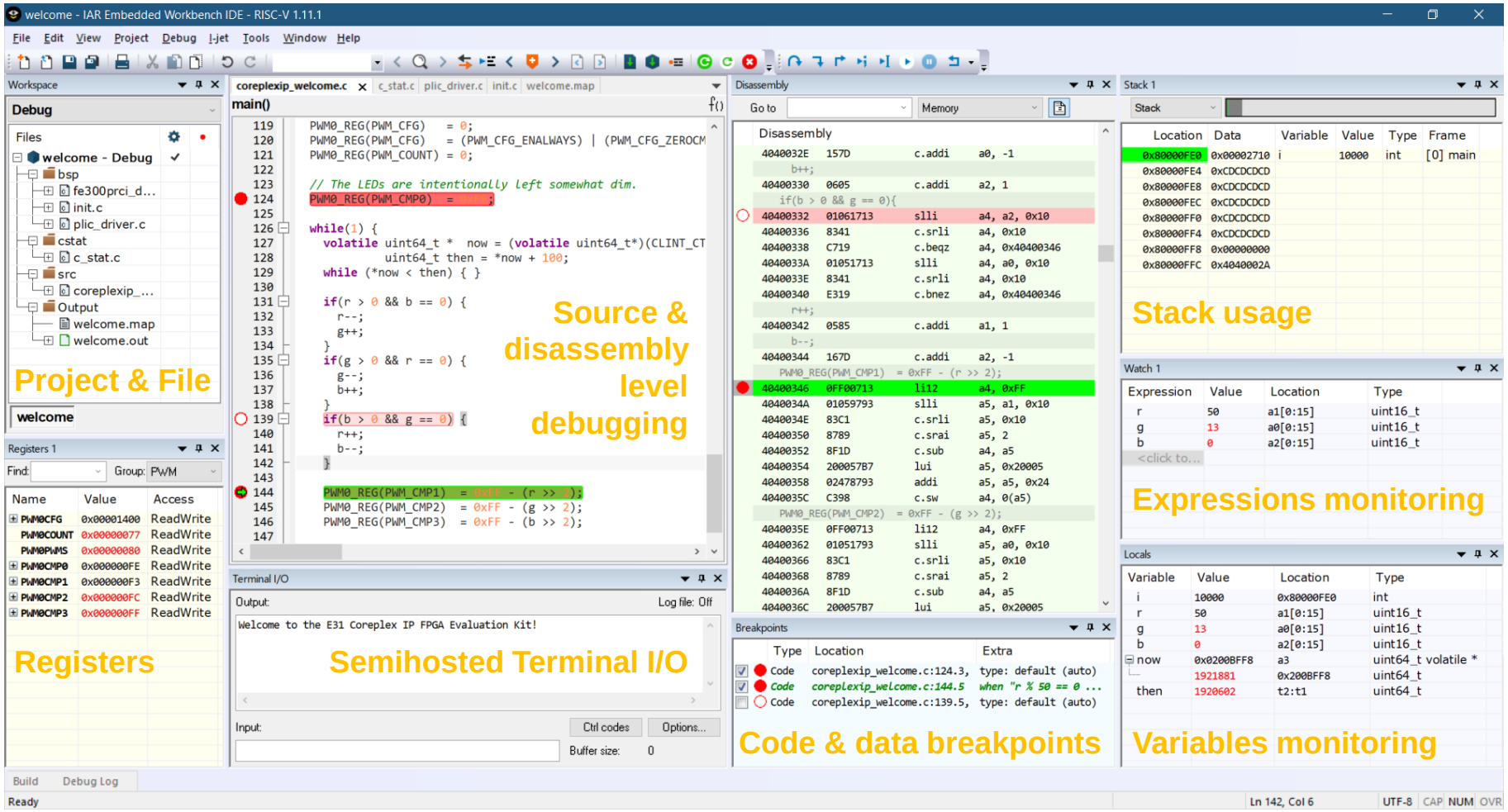
Debug & Trace probes

	I-jet	I-jet Trace (4-bit model)	I-jet Trace(16-bit 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	11.2 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

C-SPY debugger overview

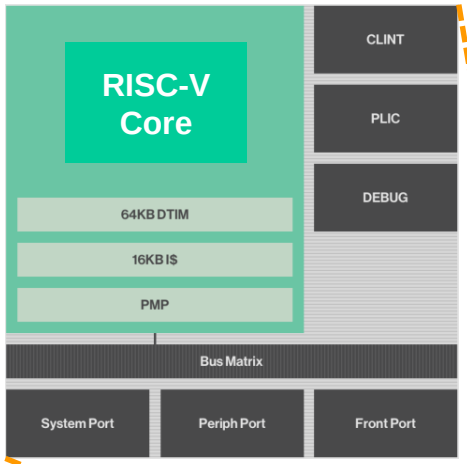


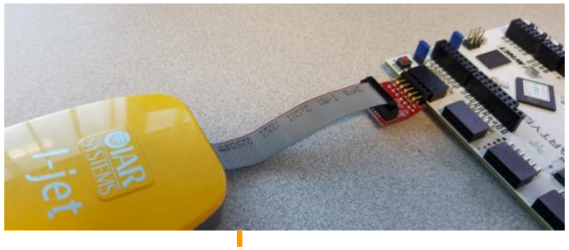
C-SPY debugger implementation



IAR SYSTEMS

Full development environment



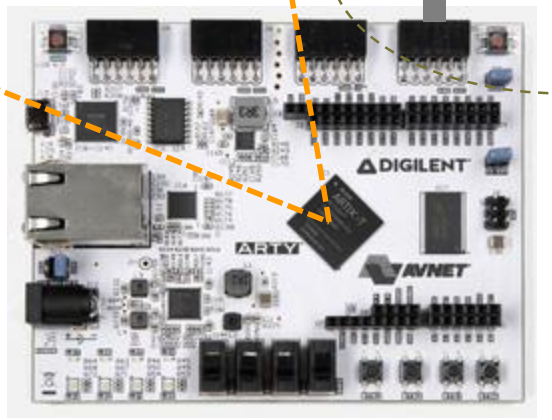


JTAG

I-jet

USB

编译/链接/下载/调试



FPGA Kit

IAR SYSTEMS

Trace on RISC-V (coming …)

- What is Trace ?
  - In contrast to traditional debugging, trace is non-intrusively observing your application
  - Capture the full PC flow
  - Go back in time and see how you arrived at the current point
  - Quickly isolate exceptions and hard faults
  - Find bugs that are rare and dependent on the order-of-execution
  - Performance and coverage monitoring, e.g. find where your application is spending its time, isolate the dead code, show test deficiencies, etc.

- RISC-V trace specifics
  - Specifications of standard RISC-V trace are still under development
  - Processor Trace TG defines trace encoder packets and the core→encoder interface
  - More work is needed to make all aspects of trace standard (e.g. control & export)
  - Goal is to get on par with what is already existing on more mature architectures

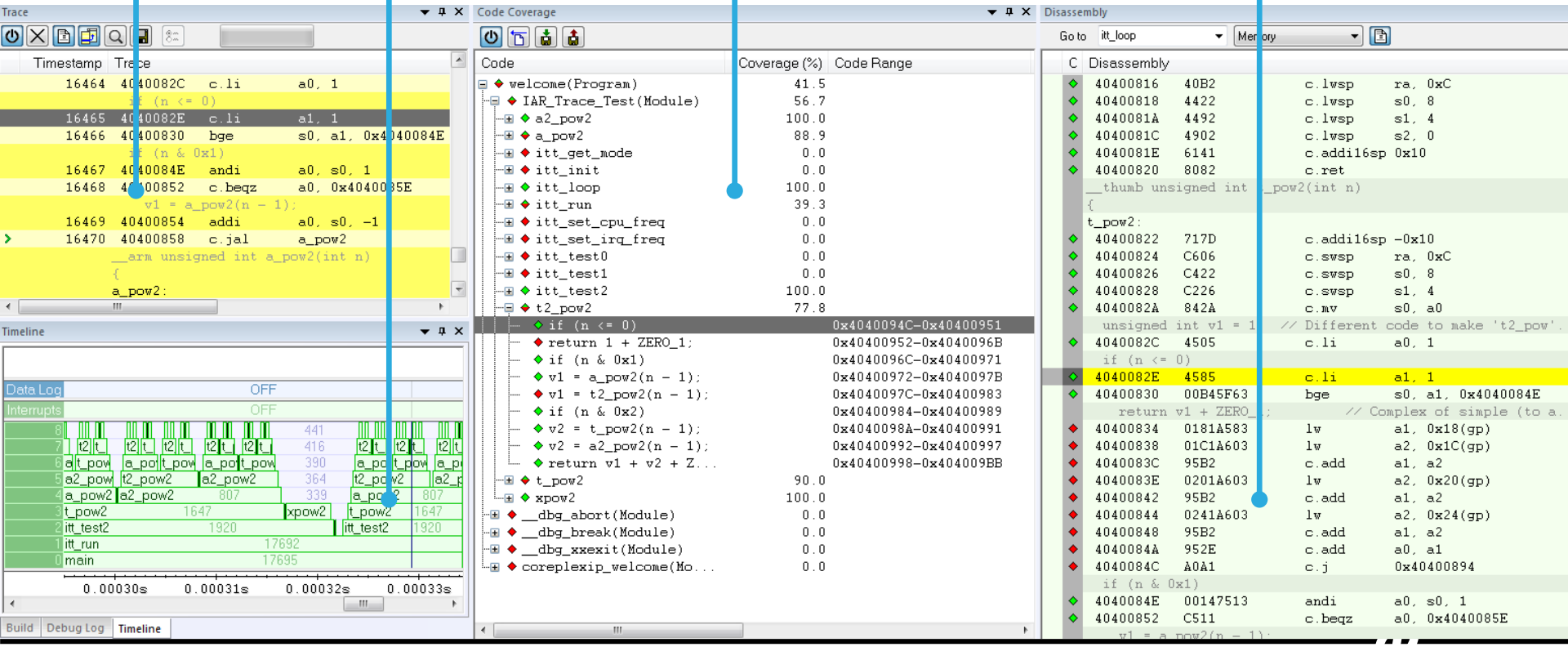
Trace on RISC-V (coming …)

Instruction Trace

Timeline

Code Coverage

Instruction Coverage



Summary

- Meet your demand of quality & time-to-market
  - Easy code reuse and widest customers base from IAR Embedded Workbench
  - Fit the needs of both memory size and necessary performance by the outstanding C/C++ compiler
  - Improve the code quality and find potential issues earlier by the integrated C-STAT analysis
  - Identify low level bugs and provide graphical visibility to all SoC resource by the powerful debugger

IAR SYSTEMS

Thanks for your attention !

www.iar.com/riscv