Getting the most out of your professional RISC-V compiler and debugger

Ryan Sheng / 盛磊, ryan.sheng@iar.com, 021-63758658
IAR Systems (China)
2019.11.13

Highlight

- Meet the demand of quality & time-to-market for your RISC-V project
  - Easy code reuse and widest customers base from IAR Embedded Workbench, the complete IDE toolchain
  - 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

IAR Embedded Workbench
Complete C/C++ compiler and debugger toolchain

Most widely used development tools for embedded applications

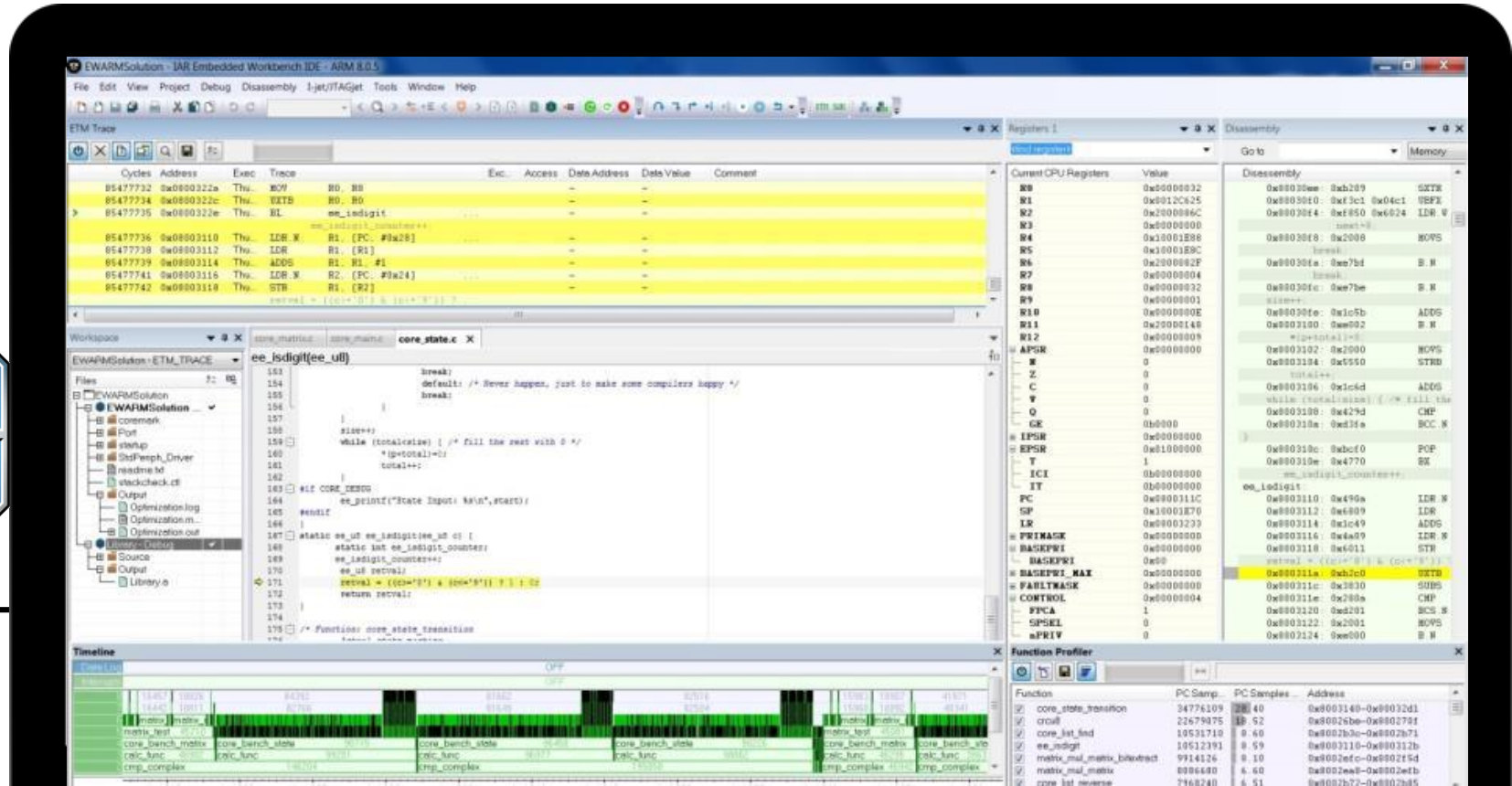
User-friendly IDE features and broad ecosystem integration

Industry leading code optimization technology

Comprehensive debugger

Integrated code analysis tools

ISO/ANSI C/C++ compliance with C18 and C++17



Support for 12,000+ devices
Different architecture, One solution

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

Cortex-M0
Cortex-M0+
Cortex-M1
Cortex-M3
Cortex-M4
Cortex-M7
Cortex-M23
Cortex-M33
Cortex-R4
Cortex-R5
Cortex-R52
Cortex-R7

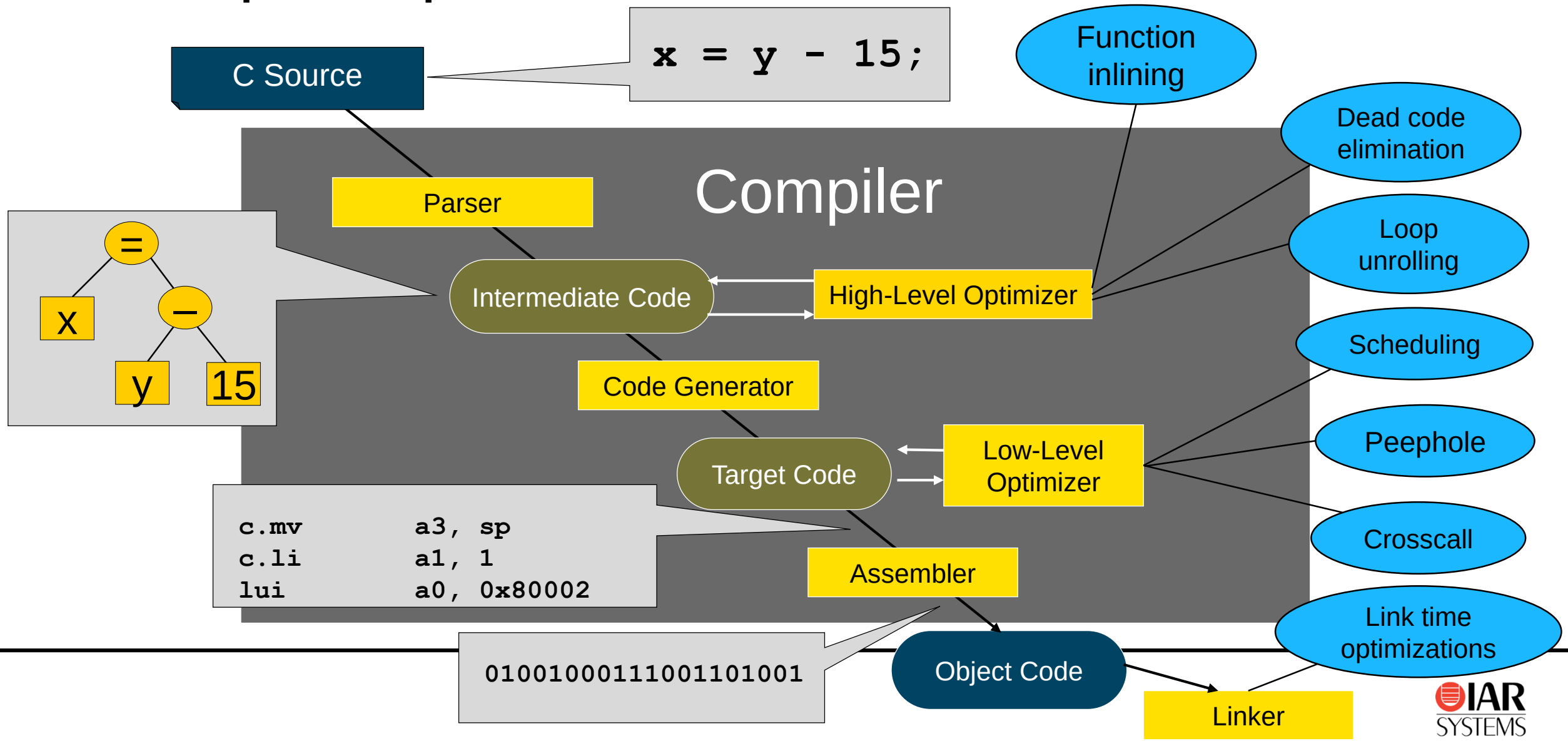
Cortex-R8
Cortex-A5
Cortex-A7
Cortex-A8
Cortex-A9
Cortex-A15
ARM11
ARM9
ARM7
SecurCore
8051
MSP430

AVR
AVR32
RX
RL78
RH850
78K
SuperH
V850
R32C
M32C
M16C
R8C

H8
STM8
ColdFire
HCS12
S08
MAXQ
CR16C
SAM8

RISC-V

Compiler optimizations



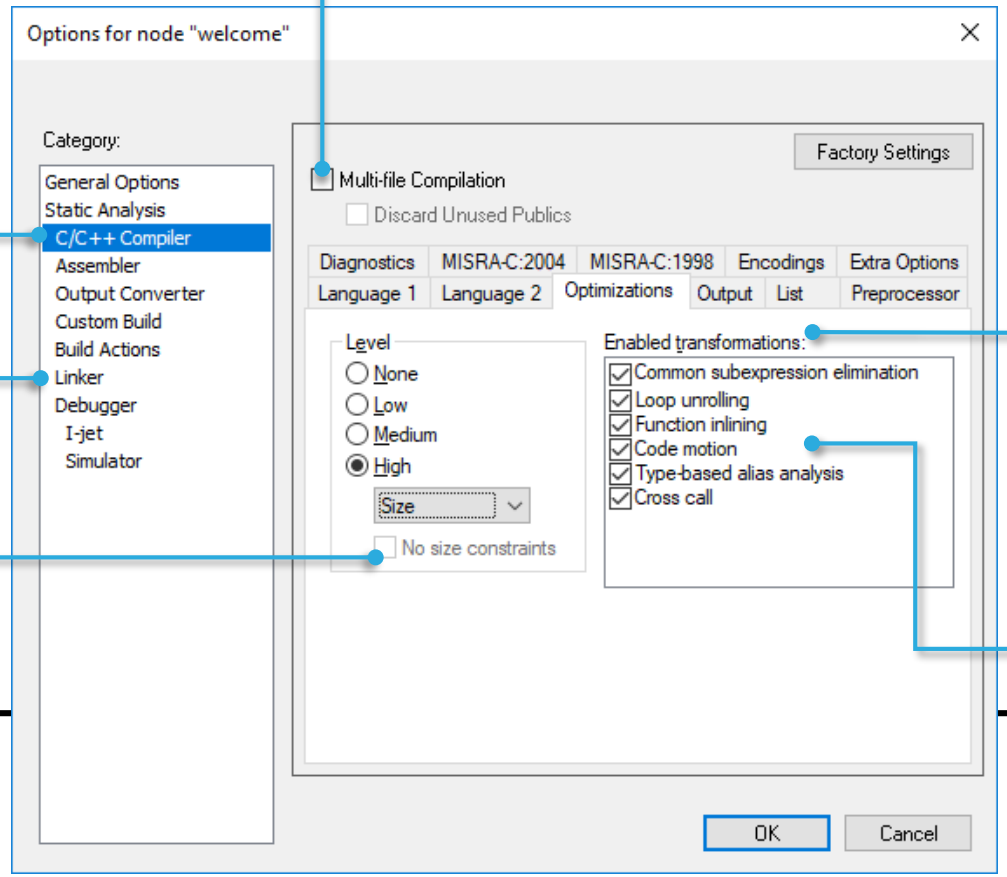
Controlling optimizations

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

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

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 →
Cross call	Speed ↓	Size ↓

Challenges on optimization

- **Size**
  - Compared to more complex instruction sets, RISC-V have some 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 are other examples
- **Speed**
  - When it comes to speed, RISC-V is relatively competitive
  - More speed optimizations come in future releases

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

GCC attributes

- In extended language mode, the 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 can be used to inline assembly code in programs written in C and C++
- 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)

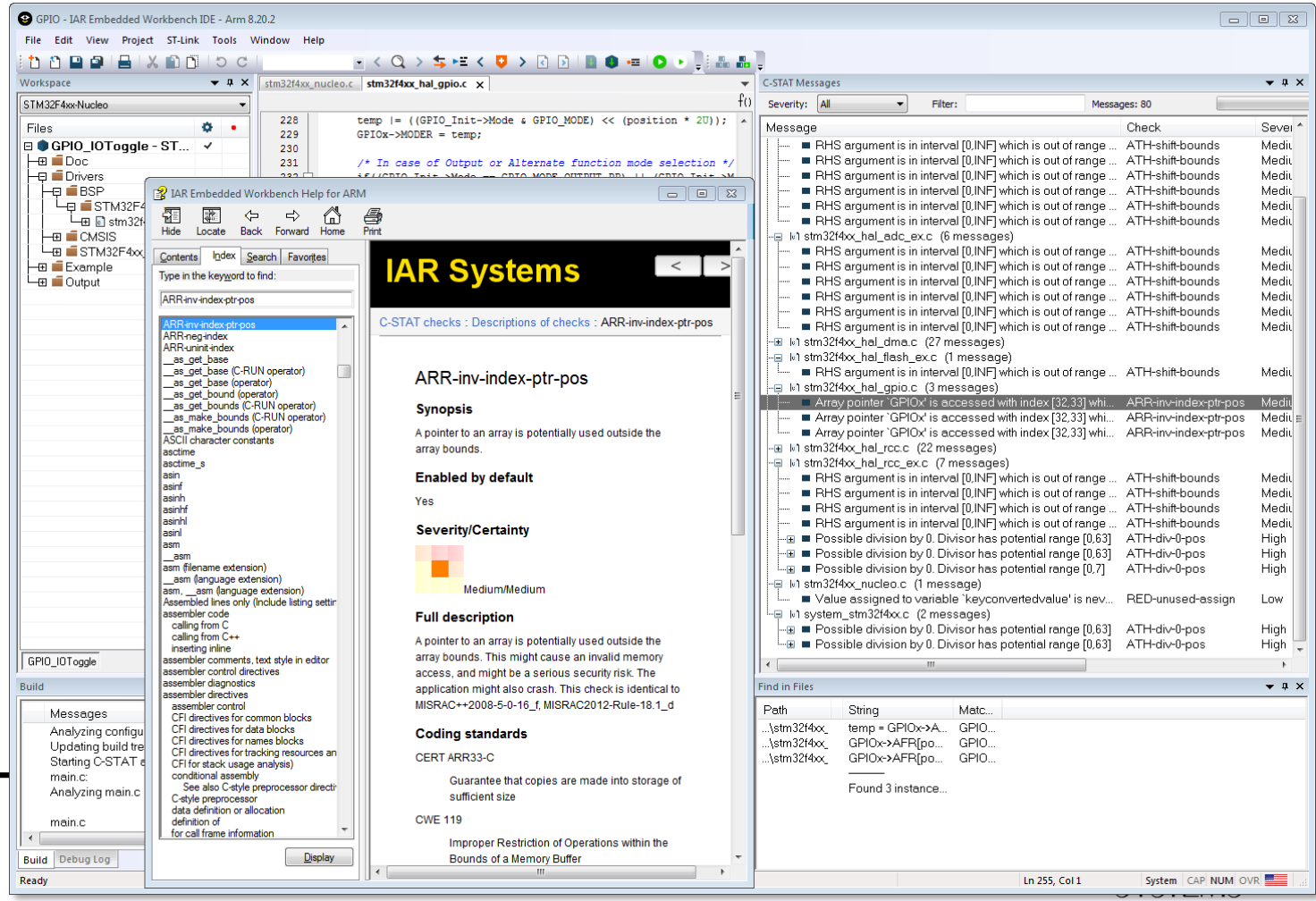
Rd', rsN'
compact instruction reg. field
integer (x8-x15) or FP (f8-f15)

expr: immediate expression

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

Code quality: C-STAT static analysis

- 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

Debug & Trace probes

	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	3.73MByte/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	48MByte/s	~380MByte/s	~380MByte/s
Power sampling resolution	~160 µA	~160µA	~160 µA
Power sampling rate	200 ksps	200 ksps	200 ksps

IAR SYSTEMS

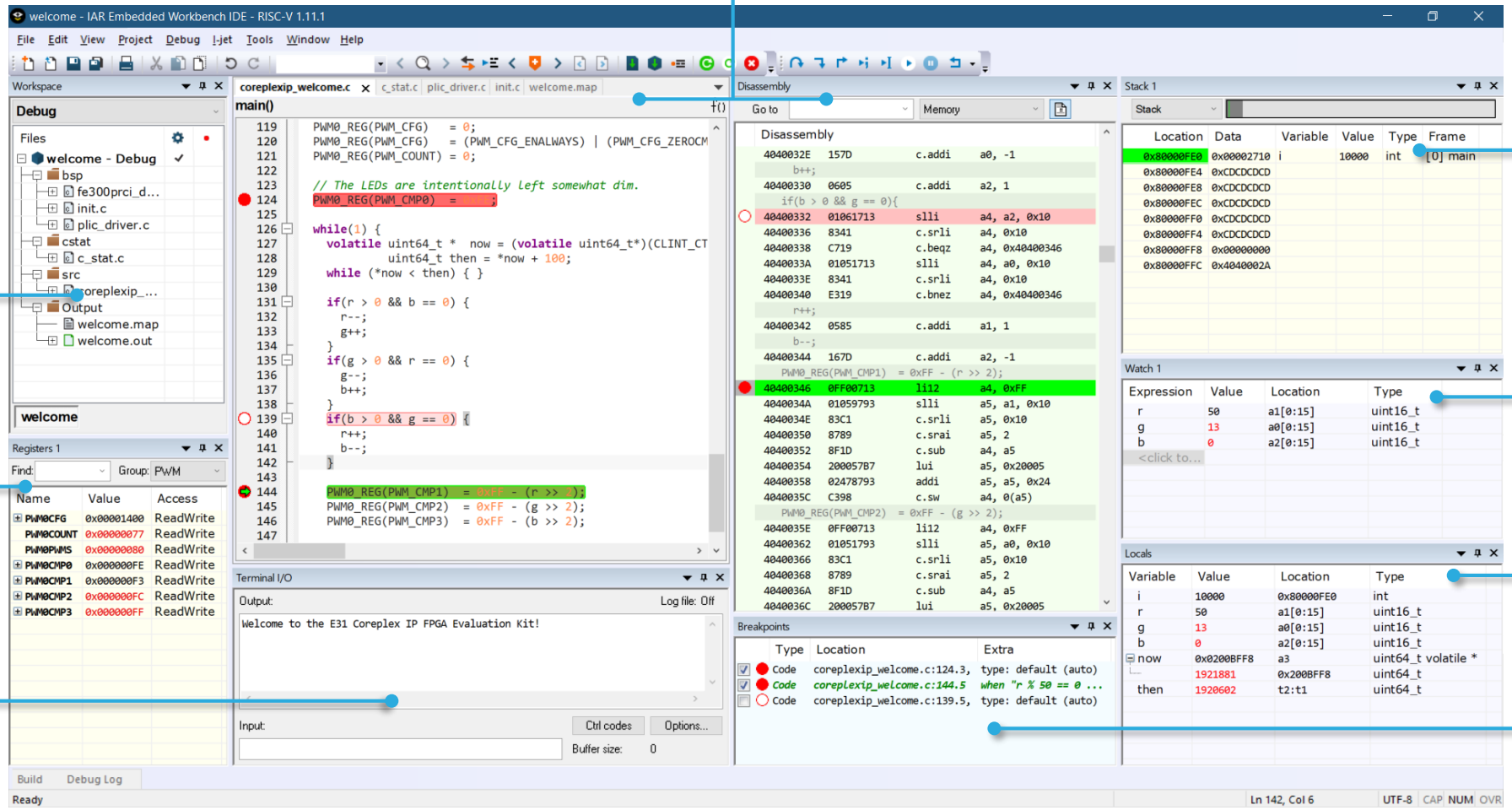
RISC-V debugging

- 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 available HW breakpoints, supported extensions etc. is read on connection
- Interrupt and exception catching
  - Configure interrupt and exception catching
  - Distinguish between different priority levels and exception types
- Set different types of breakpoints
  - Watchpoints, set breakpoints on data
  - Set conditional breakpoints
- Single step both on C/C++ and assembler level
- Full low-level access to all registers, memories and resources on RISC-V SoC
  - Low level powerful SoC-oriented debugger on roadmap
- Script/macro execution capabilities

Debugger

Source and disassembly level debugging

- C-like macro system
- Built-in Simulator
- RTOS awareness
- Trace



Dockable windows and tab groups

Registers

Semihosted Terminal I/O

Stack usage

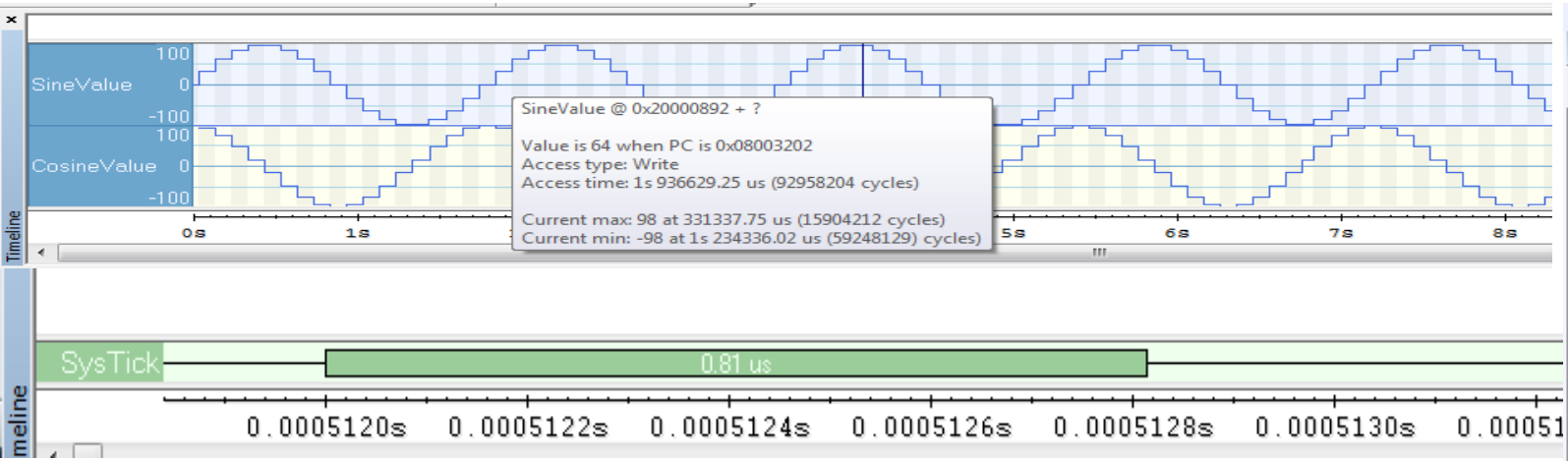
Expressions Monitoring

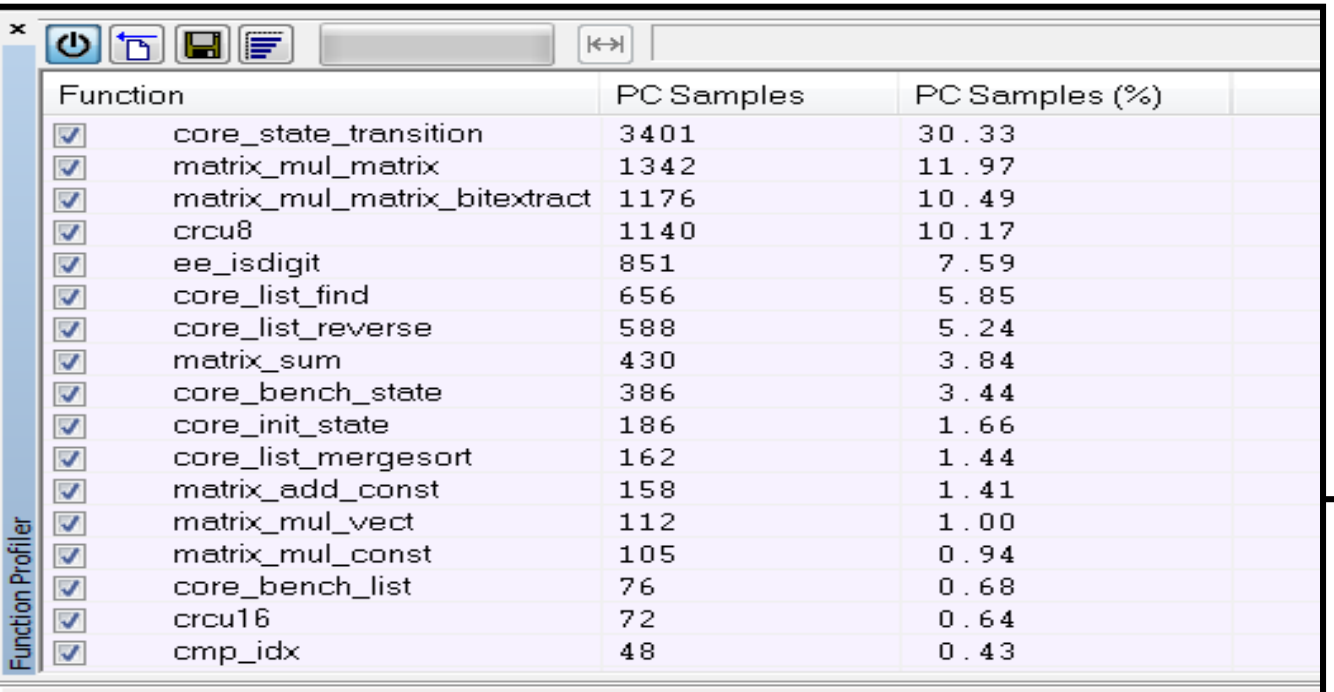
Variables Monitoring

Code & data breakpoints

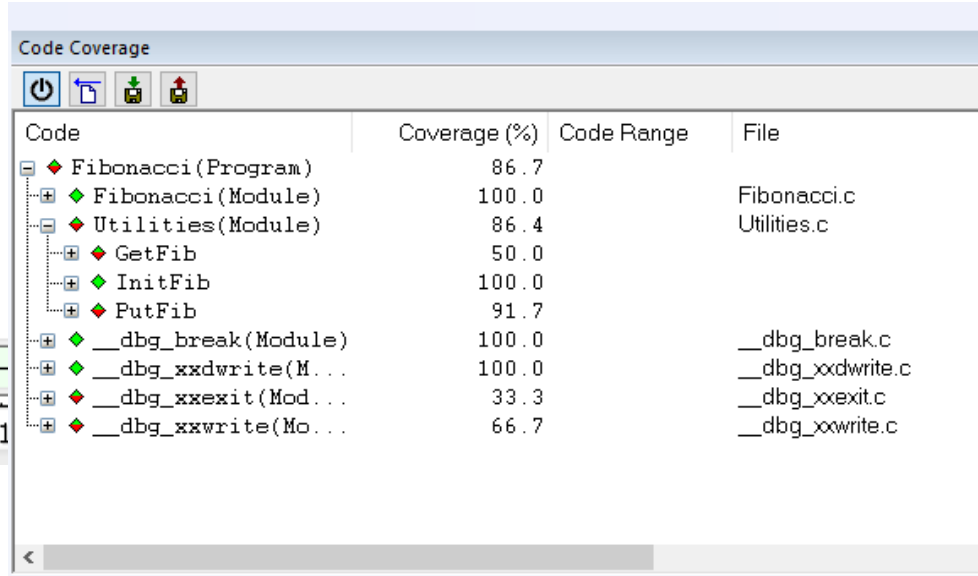
IAR SYSTEMS

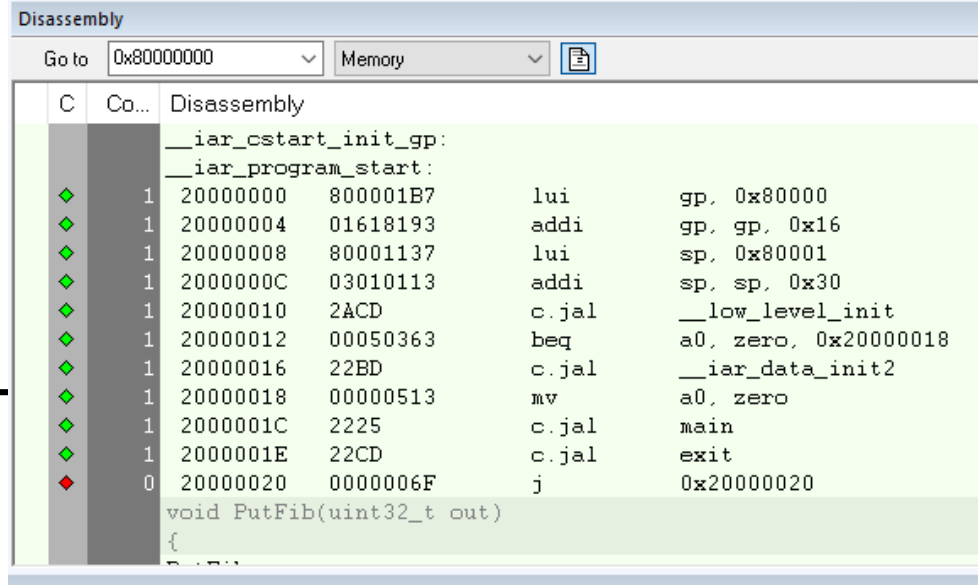
More features will come:
Data log, Interrupt log, Function profiler, Code coverage, etc.



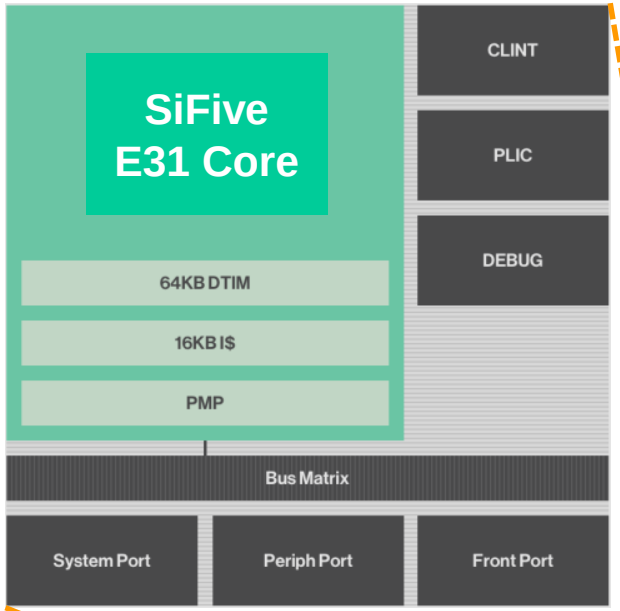


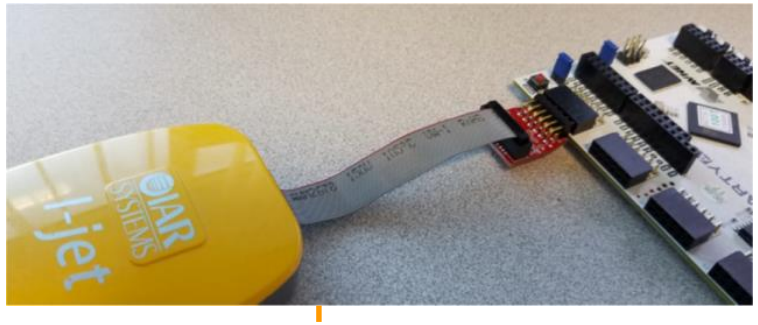
Function	PC Samples	PC Samples (%)
core_state_transition	3401	30.33
matrix_mul_matrix	1342	11.97
matrix_mul_matrix_bitextract	1176	10.49
crcu8	1140	10.17
ee_isdigit	851	7.59
core_list_find	656	5.85
core_list_reverse	588	5.24
matrix_sum	430	3.84
core_bench_state	386	3.44
core_init_state	186	1.66
core_list_mergesort	162	1.44
matrix_add_const	158	1.41
matrix_mul_vect	112	1.00
matrix_mul_const	105	0.94
core_bench_list	76	0.68
crcu16	72	0.64
cmp_idx	48	0.43





Demonstration



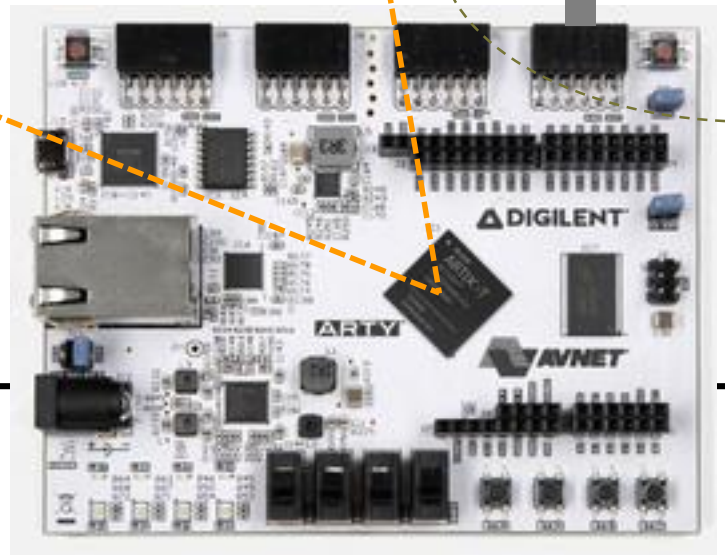


JTAG

I-jet

USB

编译/链接/下载/调试



Xilinx Artix-7
Arty FPGA Kit

Thanks for your attention!

www.iar.com/riscv

IAR SYSTEMS