RISC-Vプロセッサの導入を加速する
商用開発環境IAR Embedded Workbench

IAR Systems
FAE, Shinji Tonoshita

IAR SYSTEMS


Agenda
–IARシステムズ紹介
–IAR Embedded Workbench for RISC-V
–コンパイラ
–カスタム命令
–デバッグ
–コード品質と機能安全

世界で最も広く使われている組込み開発ツール！

- ✓ 優れた最適化技術
- ✓ 包括的なデバッガ機能
- ✓ 充実したテクニカルサポート

IAR SYSTEMS

IAR Embedded Workbench for RISC-V

IAR SYSTEMS

IAR Embedded Workbench for RISC-V

- RISC-Vの唯一の商用ツールベンダ
- コードサイズと実行速度に優れた最適化
- 静的コード解析ツールC-STATを統合
- デバッグプローブを使用した実機デバッグとシミュレーション機能を搭載したC-SPYデバッガ
- 充実したテクニカルサポート
- 将来のリリースには、機能安全認証とセキュリティソリューションが含まれます

IAR SYSTEMS

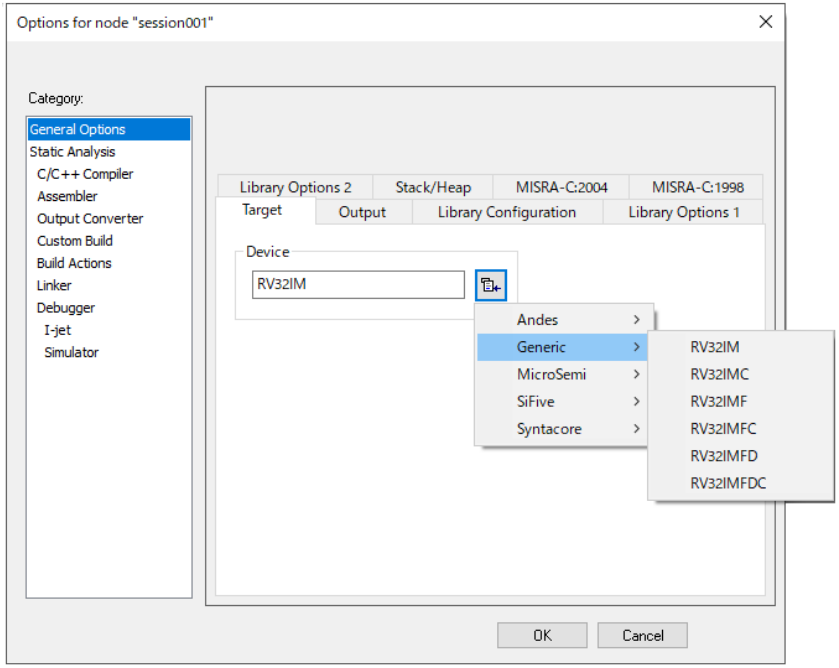
デバイスサポート

- IAR Embedded Workbench for RISC-V Ver1.11.1デバイスサポート状況

基本命令セット	
RV32I	Integer Instruction Set, 32-bit

拡張命令セット	
M	Integer Multiplication and Division
F	Single-Precision Floating-Point
D	Double-Precision Floating-Point
C	Compressed Instructions
X	Non-Standard Instructions

メーカー	サポートコア
Andes	A25, N25, N25F
MicroSemi	MicroSemi AHB,MicroSemi AHBF,MicroSemi AXI
SiFive	E20, E21, E24, E31, E34, E76
Syntacore	SCR1



さあ、開発を始めるぞ！

4ステップで、動作するプログラムが出来ます。

1. ワークスペースとプロジェクトを作り
2. 使用するデバイスを選択
3. main()関数を含むソースファイルを追加
4. デバッグ環境を設定！（シミュレータまたはターゲットを選択）

main()関数からデバッグを開始

IAR SYSTEMS

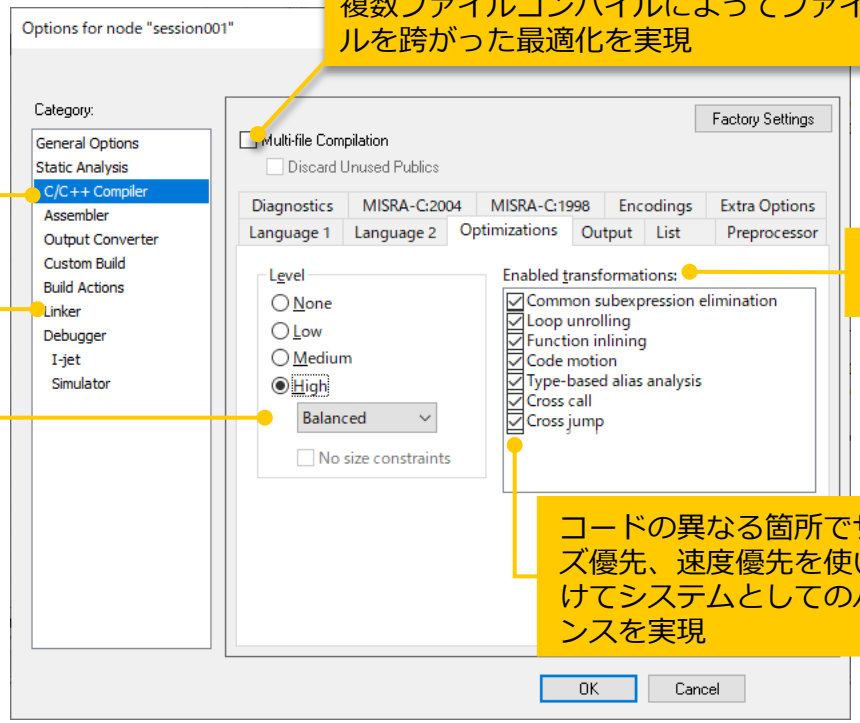
コンパイラ

EWRISC-V C/C++コンパイラ概要

コードサイズあるいは実行速度にフォーカスした複数レベルの最適化

リンカで不使用コードを削除

オプションとしてサイズ制限なく実行速度を最大化



複数ファイルコンパイルによってファイルを跨がった最適化を実現

主要な最適化機能は個別で制御可能

コードの異なる箇所でサイズ優先、速度優先を使い分けてシステムとしてのバランスを実現

言語標準

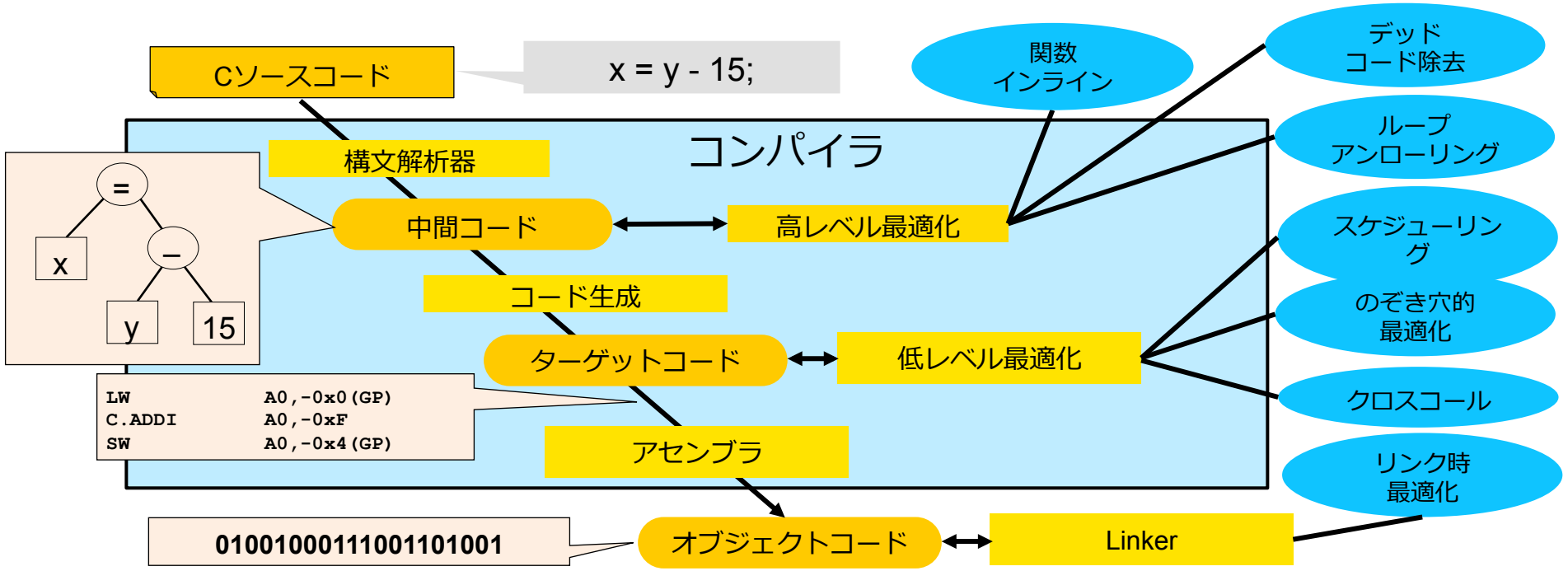
- ISO/IEC 14882:2015 (C++14, C++17)
- ISO/IEC 9899:2018 (C18)
- ANSI X3.159-1989 (C89)
- IEEE 754標準浮動小数点演算

十分なテスト検証

- 商用テストセット
  - Plum-Hall Validation test suite
  - Perennial EC++VS
  - Dinkum C++ Proofer
- 社内で500,000ラインのC/C++ テストを繰り返し実施
  - プロセッサモード
  - メモリモデル
  - 最適化レベル

IAR SYSTEMS

コンパイラ最適化動作


Cソースコード
x = y - 15;
コンパイラ
構文解析器
=
x
–
y
15
中間コード
高レベル最適化
コード生成
ターゲットコード
低レベル最適化
LW A0,-0x0(GP)
C.ADDI A0,-0xF
SW A0,-0x4(GP)
アセンブラ
0100100011100110100 1
オブジェクトコード
Linker
関数
インライン
デッド
コード除去
ループ
アンローリング
スケジューリン
グ
のぞき穴的
最適化
クロスコール
リンク時
最適化

最適化の種類は？

- 最適化の種類によって効果は異なる

最適化項目	最適化の効果	
共通部分式除去	↑ 速度	↓ サイズ
ループアンローリング	↑ 速度	↑ サイズ
関数インライン	↑ 速度	↑ サイズ
コード・モーション	↑ 速度	→ サイズ
デッドコード除去	→ 速度	↓ サイズ
静的クラスター	↑ 速度	↓ サイズ
スケジューリング	↑ 速度	→ サイズ
のぞき穴的最適化	↑ 速度	↓ サイズ
クロスコール	↓ 速度	↓ サイズ

カスタム命令

カスタム命令は .insnディレクティブで

- .insnディレクティブは、アセンブラでカスタム命令を生成
- .insnディレクティブを使用して、CおよびC ++で作成されたプログラムのアセンブリコードをインライン化

【AND r,r,r命令の実装例】

```
int32_t func1(int32_t lhs, int32_t rhs)
{
  int32_t res;

  asm (".insn r 0x33, 0x7, 0x0, %0, %1, %2" : "=r" (res) : "r" (lhs),
"r" (rhs) );
  return( res );
}
```

※詳細に関しては、「IAR C/C++ Development Guide」-「Part 1. Using the build tools」-「Reference information for inline assembler」-「Assembler language interface」を参照して下さい

.insnディレクティブ詳細について

- .insnディレクティブは、すべてのRISC-V命令形式で命令を生成

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

IAR SYSTEMS

カスタム命令の実際

- 組み込みの関数は、カスタム命令を生成するときに使用

【C言語ソース】

```
/* sltiu a0, a1, 0x40                */
asm (".insn i 0x13,0x3, a0, a1,
0x40");
/* sb a0, 4(a1)                      */
asm (".insn s 0x23,0,a0,4(a1)");
/* sb a0, 4(a1)                      */
asm (".insn s STORE,0,a0,4(a1)");
```

【逆アセンブラ】

```
asm (".insn i 0x13,0x3, a0, a1, 0x40");
4040007E 0405B513 sltiu      a0, a1,
0x40
asm (".insn s 0x23,0,a0,4(a1)");
40400082 00B50223 sb         a1, 4(a0)
asm (".insn s STORE,0,a0,4(a1)");
40400086 00B50223 sb         a1, 4(a0)
```

【命令詳細】

	31–25	24–20	19–15	14–12	11–7	6–0
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	imm[11:0]		rs1	funct3	rd	opcode
S-type	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode

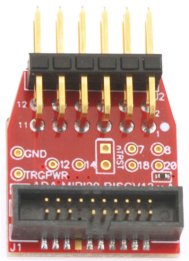
IAR SYSTEMS

デバッグ

IAR SYSTEMS

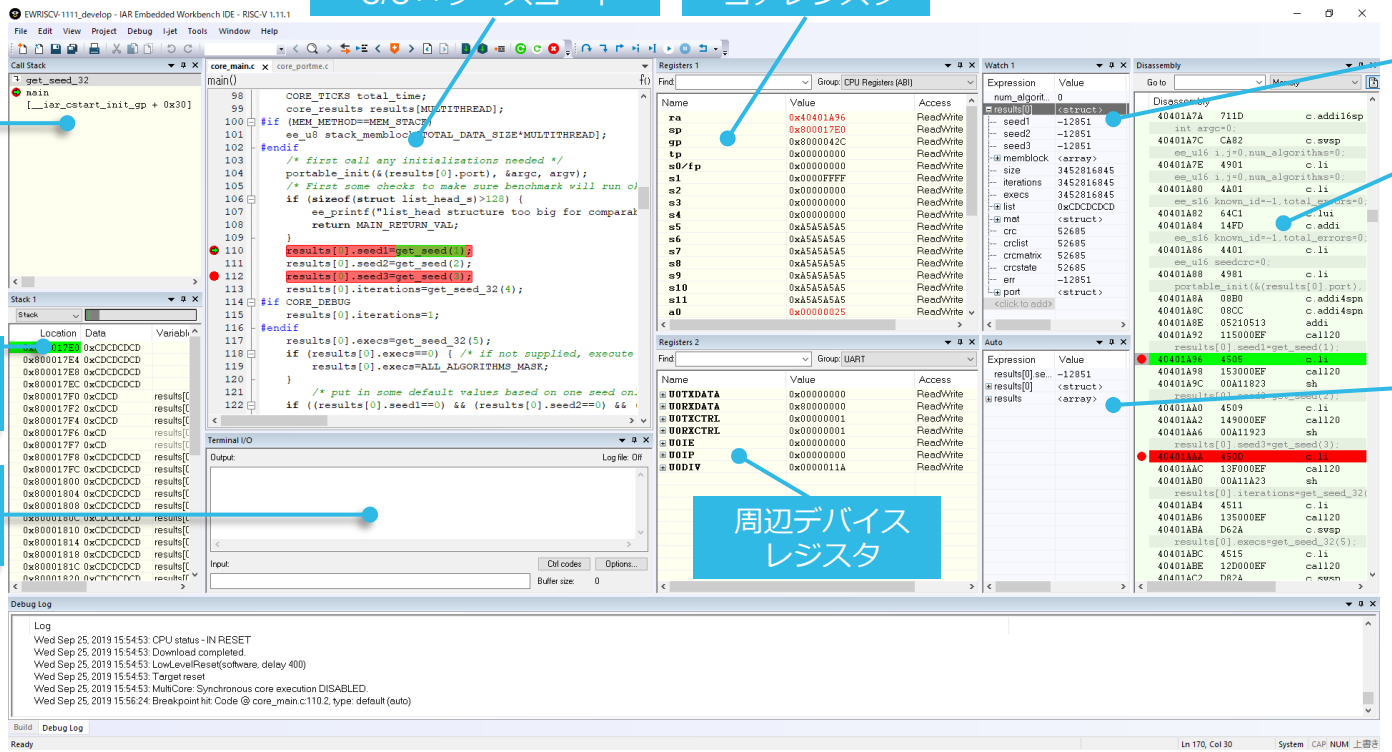
I-jet インサーキットデバッグプローブ

- RISC-VとArmコアに対応
- ハイスピード USB 2.0 インタフェース (480Mbps)
- 最大400mAまでI-jetからターゲット電源を供給可能*過電流保護回路付き
- 最大32MHzのJTAGおよびシリアルワイヤデバッグ（SWD）クロック（MCUクロック速度の制限なし）
- 最大60MHzのSWO速度のサポート
- Arty A7 boards用デバッグアダプタ (*ADA-MIPI20-RISCV12)



IAR SYSTEMS

C-SPYデバッガ

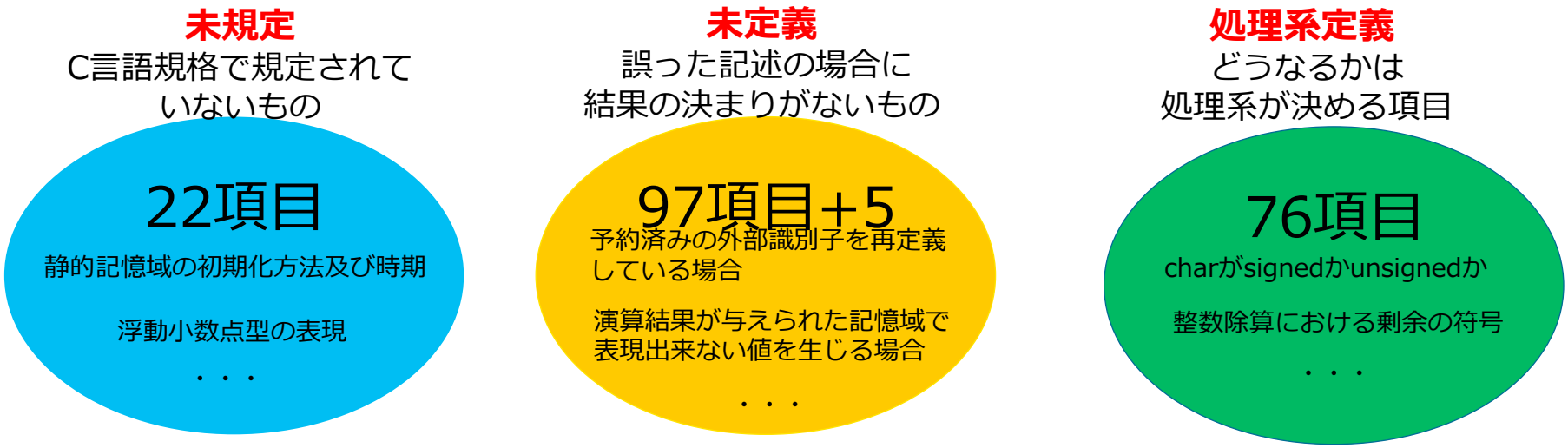

C/C++ソースコード
コアレジスタ
コールスタック
変数ウオッチ
逆アセンブラ
スタック
使用状況
セミ
ホスティング
自動
変数ウオッチ
周辺デバイス
レジスタ

コード品質と機能安全

IAR SYSTEMS

C言語の問題点

- 多くの組込み機器の開発では、C言語で開発が行われていますが、C言語は脆弱なプログラミング言語で、定義が曖昧です。

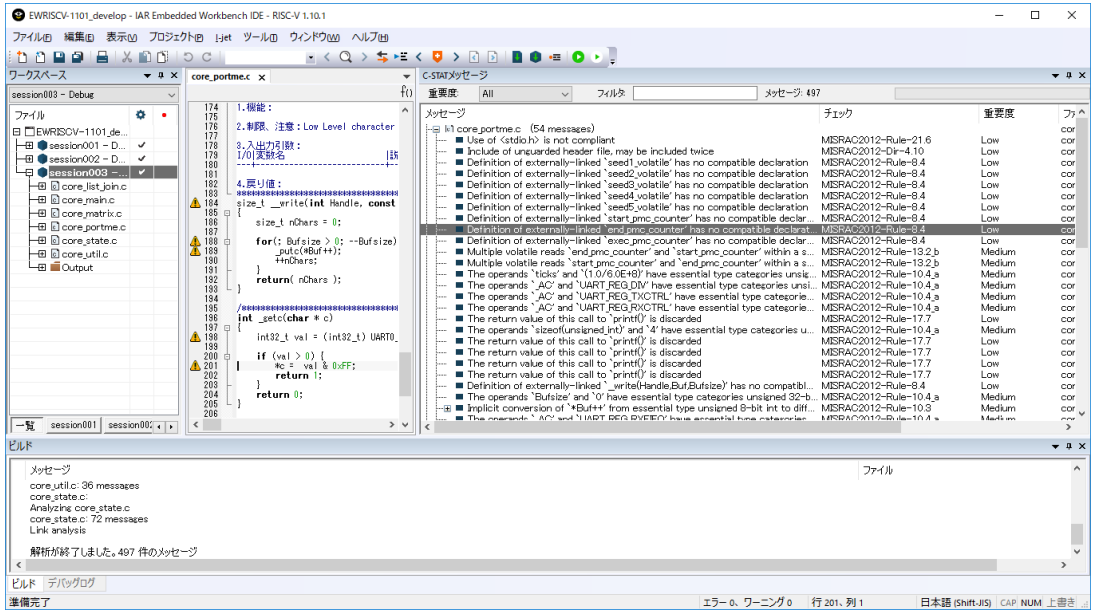


脆弱性を回避するために、まずはC言語の問題を認識する必要があります。

IAR SYSTEMS

C-STAT静的解析

- IAR Embedded Workbenchに完全に統合された静的解析ツール
  - 直感的かつ簡単に設定可能な柔軟なルール選択
  - 選択したルールのエクスポート/インポート機能
  - コマンドライン実行のサポート
  - 詳細なドキュメント
  - メッセージリストおよびHTMLレポートの保存が可能
  - MISRA C:2004, MISRA C++:2008, MISRA C:2012の準拠
  - CWEやCERT C/C++ルールにマッピングされた100以上のルールを含む、合計約250のルールチェック



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

機能安全対応が必要なアプリケーション向けソリューション

- **機能安全認証取得済みツールチェーン***
  - 機能安全版IAR Embedded Workbench

検証作業を簡単に
- TÜV SÜDの認証を取得済み
- TÜV SÜDからのセーフティレポート
- セーフティガイド

プロダクトのライフサイクルを通じて提供されるサポート
- 優先サポート
- 検証済みサービスパック
- 既知の問題についてのレポート

Validated according to:
IEC 61508
ISO 26262
EN 50128, EN 50657
IEC 62304

IAR SYSTEMS

Thank you for your attention!

www.iar.com/jp/RISCV

IAR SYSTEMS