enjoy Development

KMC
Member of eSOL

SOLID

Version 5.0

Real Time OS

Integrated with Developing Environment

京都マイクロコンピュータ株式会社

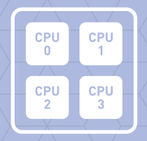
https://solid.kmckk.com/SOLID/

リアルタイムOS SOLIDとは？

SOLIDは、
開発効率と品質向上を極めた
リアルタイムOSです

開発対象機器の高機能化に伴い、RTOSシステムの規模や複雑度が増しています。
それに伴うソフトウェア開発の課題や不安を解決するのがSOLIDです。
SOLIDはバグやデバッグ工数を削減する機能を充実させた、
開発効率と品質向上を極めたリアルタイムOSです。

リアルタイムOS SOLIDをおすすめしたいポイント



ITRON仕様のカーネルです

- ITRON仕様準拠のTOPPERSカーネル
- 32bit/64bitに対応
- マルチプロセッサに対応

大規模ソフトウェアのチーム開発を支援します

- 100人規模でも安全に分割開発
- 分割単位でビルド・デバッグ可能

安全に効率よく開発できます

- プロセッサの複雑なメモリ管理(MMU)はSOLIDが自動設定
- 実行時メモリアクセスバグを自動検出
- メモリ安全性にかかわるバグをつくらせないRust 言語にも対応

導入しやすい価格で提供します

- IDE、コンパイラ、リアルタイムOS、デバッガ全てをセットで提供
- 1ユーザ・1年間利用で250,000円
- リアルタイムOSのロイヤリティが不要

品質に対する取り組み

■実績と高機能を備えた高品質な TOPPERS カーネルを採用

TOPPERS

- 国内の採用実績多数
- 32bit、64bit、マルチコアにも対応した高機能カーネル
- TOPPERSプロジェクトメンバーであるKMCがきっちりサポート

■コンパイラの品質管理・品質評価

SuperTest

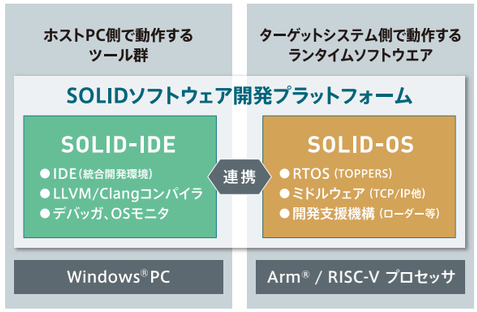
LLVM/Clang コンパイラの開発段階では、オランダ Solid Sand 社のコンパイラテストスイートである **SuperTest Aelbert Cuyp 5.0** を使用し、品質管理を行いました。

JNOVEL

本製品に含まれる LLVM/Clang コンパイラは、日本ノーベル株式会社の**第三者検証コンパイラ評価サービス**にて品質評価を実施、問題となるような不具合が無い事を検証いただいています。

SOLIDは開発環境連携で、新しい価値を提供します

安全に効率よく開発するために、進化した開発環境はここが違います



■SOLIDひとつで、全てのツールがそろう

- コーディングから実機デバッグまで、全てのツールをVS Code 互換 Theia ベースの統合開発環境(SOLID-IDE)で操作
- 「IDE」「コンパイラ」「リアルタイムOS」「デバッガ」を1ライセンスで提供
- コンパイラは先進の LLVM/Clang を同梱※
- C/C++に加えRust言語にも対応

■ツール間の連携による、SOLID独自の便利な機能

- メモリ消費量情報表示、メモリマップエディタ
- 自動バグ検出機能、タスク遷移表示機能

※ 以前の SOLID バージョン付属の GCC と LLVM/Clang コンパイラも SOLID 5.0 以降でご利用いただけます

POINT

1ユーザ、1年間の利用料250,000円と導入しやすい価格で提供します

全てのツールがセットになったサブスクリプションライセンス

- 1ライセンスで、Arm® および RISC-V プロセッサで利用可
  - Arm® Cortex®-A/R/M プロセッサ、32bit/64bit (AArch64、AArch32)
  - RISC-V プロセッサ、RV32/RV64
- シングルコア/マルチコアのいずれのリアルタイムOSも利用可
- 利用者数に応じたボリュームディスカウント契約を用意

リアルタイムOSは量産時のロイヤリティが不要

- TOPPERSは、商用利用・量産時のロイヤリティが不要

TOPPERSプロジェクトへの報告をお願いします
https://www.toppers.jp/report.html

SOLIDユーザの声

不毛なデバッグとはもうお別れです

スタックオーバーフローは、リアルタイムOSのように単一空間で動作するソフトウェアの普遍的なバグであり、単純ですが見つけるのに苦労する、こんなデバッグは不毛だけど、仕方ないと思っていました。
ところがSOLIDではスタック領域を超えてアクセスした瞬間に、デバッガがスタックオーバーフローが起きたことを教えてくれるので、原因となった箇所が即座に特定でき大変助かりました。多人数で開発していると、自分のプログラムで発生したスタックオーバーフローが原因で、他の人の担当箇所の不具合のように現れてしまい、延々と合同デバッグをした結果、実は自分のプログラムが原因だったということがあります。他人にバグを指摘される前に自分で不具合をつぶせるのは人間関係面でもメリット大です。

プロセッサのマニュアルを読むのが大変でした

Arm プロセッサのMMUは設定がかなり複雑です。そのため、キャッシュを使うためだけにMMUを有効にし、あとは論理アドレス＝物理アドレスのまま使っていました。それでもMMUの設定を間違えると、デバッガの応答もなくなってしまい、万事休す。物理メモリを効率よく使う以前の苦労が多かったです。今回採用したSOLIDでは、MMUの設定は自動でやってくれるので、マニュアルを読まなくても安全にMMUが使えるところが気に入りました。しかも、論理アドレスと物理アドレスの設定は分かりやすい表形式で入力するだけでよく、特に指定しなければSOLIDが物理メモリ上に効率よくプログラム・データを配置してくれます。MMUが苦手な私でも「簡単な設定」で「メモリの安全な割り当てがビジュアルに確認」でき、さらに「メモリ使用効率」が良いシステム設計ができたことに、とても満足しています。

Rust言語の安全性やデバッグのしやすさを実感しています

C++ 言語のプログラム歴 XX 年の私が、「Rust は C/C++ の反省の結果、作られている」という言葉に触発され Rust を使ってみました。Rust はメモリの利用に対して空間的・時間的安全性に厳密な言語です。例えば、範囲外メモリアクセスのような危険なコードは、C/C++ で書けたとしても、Rust ではコンパイラがエラーと判定します。またメモリ範囲を超えるアクセス時には、そこで処理が中断するコードを Rust コンパイラが生成するため、メモリ破壊が原因で動作不安定になることを避けられ、デバッグもしやすくなります。SOLID では C/C++ と Rust で各々作成されたソースコードを、混在して使用することができるため、プログラムの中で特に安全性を求められる部分(コンポーネント)だけでも Rust で作成することが可能です。このように、プログラムの安全性やデバッグのしやすさという点で、Rust の良さを実感しています。

価格面で上司を説得できました

新規プロジェクト着手にあたり、開発ツールを新しく揃えることになり、ツールベンダー各社の価格を比較しました。
SOLIDは全てのツールが揃っているので、他のツールをそろえるのに比べてコストメリットがありました。また、サブスクリプションライセンスなので、社内の資産管理が不要となり、更にサポート契約も不要ということで、管理部門からも好評でした。
SOLIDの機能だけでなく、価格面でもメリットが大きかったので、導入にあたって上司の説得がしやすかったです。

GOOD DEAL

プロジェクト管理、コーディング、ビルドから実機デバッグまでの全開発工程は、VS Code と互換性のある Theia をベースにしたSOLID-IDE ひとつで操作できます。SOLIDには標準的な開発環境の機能に加え、独自の便利な機能を多数揃えています。

SOLID-OS を支える、SOLID-IDE の様々な開発・デバッグ支援機能

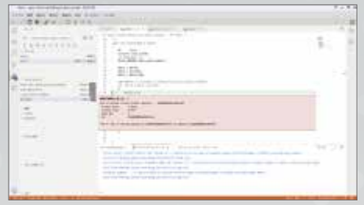
IDEとOSが連携することで、開発・設計から実行時検証までをスムーズに

THEIA

- SOLID ver.5.0 は IDE として Theia を採用しました。Theia は Visual Studio Code と互換性のあるオープンソースの IDE フレームワークです。
- VS Code の拡張機能用 API と互換性のある API を保持しているので、AI エージェント機能等を含む VS Code 拡張機能を利用することで、IDE をカスタマイズし進化・強化させることが可能です。

アドレスサニタイザ機能

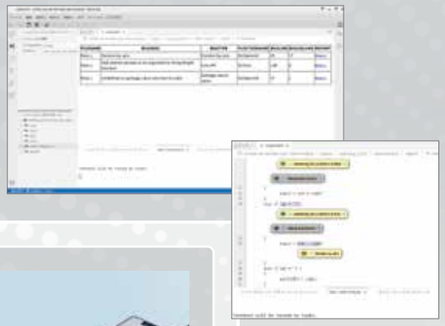
- 実行時のローカル変数、グローバル変数の、オーバーランや未定義領域アクセスを自動検出します。
- ソースを変更することなく、アドレスサニタイザモードでビルドするだけで、簡単にアドレスサニタイザ機能が使用できます。



詳しくはこちらをご覧ください

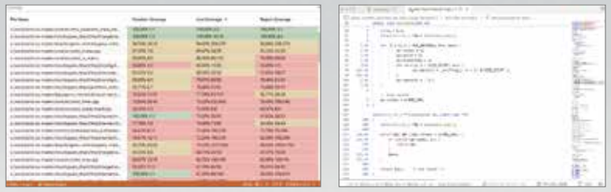
1クリックで静的解析・構文解析機能

- 実行前にソースコードの分析を行い、論理的な問題を検出する静的解析機能を標準搭載しています。
- 関数呼び出しや分岐条件を加味し、未初期化ポインタアクセスやゼロ除算など、様々な問題を指摘します。



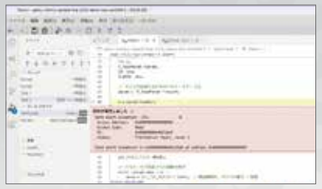
実行時カバレッジ表示機能

- ソースコード中のベーシックブロック単位ごとに、実行された回数を記録し、ブレーク時に表示します。
- カバレッジ結果はllvm-profdataなどの汎用解析ツールに取り込めるので、レポート作成にも活用できます。



メモリアクセス例外・リンカスクリプト連携

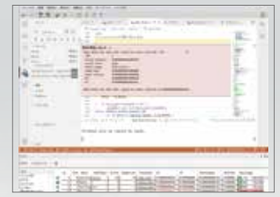
- コンパイラが出力するメモリ属性情報に加えて、ユーザが指定したメモリ属性情報もSOLIDが自動的にMMUに設定します。
- メモリアクセス例外が発生した瞬間にデバッガが実行停止(ブレーク)しアラーム表示により警告します。



PARTNER-Jet3で実機デバッグ

- SOLID-IDEからKMCのJTAGデバッガであるPARTNER-Jet3を使って、実機デバッグをします。
  (PARTNER-Jet2もご使用いただけます)

スタックフェンス機能



- スタックオーバーフローが起きた瞬間に、デバッガが実行停止(ブレーク)しアラート表示により警告します。
- スタックオーバーフローによるメモリ破壊を未然に防ぎ、デバッグ効率を大幅に向上します。

ELF マップ表示

- ELFファイル内にあるシンボル情報をもとに、変数や関数のアドレスを分かりやすく表示する機能です。
- 簡単な操作でシンボル名でのソートや検索ができます。

関数トレース機能

- 指定した関数の実行履歴をタイムスタンプや引数情報とともに確認できる機能です。
- ターゲットメモリにトレース情報を格納する方式なので、トレース端子のないSoCでも利用できます。

スタックサイズ予測

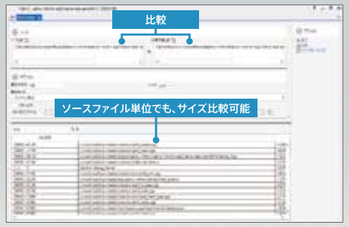
- プログラムの静的解析機能により、スタックメモリの最大使用量を予測します。
- 呼び出し先の関数単位でスタック予測表示するため、スタックサイズ分析に便利です。

メモリマップデザイナー

- 設定が複雑なARM Cortex-AのMMUを、グラフィカルインタフェースで簡単に設定し、メモリを安全に使えます。
- ビルド機能と連携し、自動的にセクションに応じたプロテクションを設定します。
- リンカスクリプトとも連携しメモリを設定します。
- Cortex-R の MPU にも対応しています。

サイズプロファイラー

- コードサイズやワークメモリサイズを、コンパイル単位やセクション単位でGUI表示・確認できます。
- ファイル比較機能を使って、バージョンアップ前後のサイズ変更の確認ができます。



関数呼び出し表示

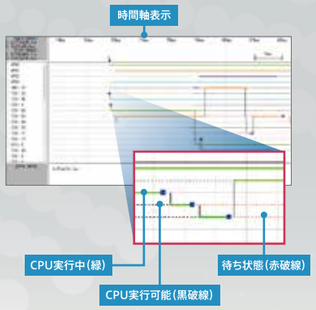
- 関数単位で、呼び出し元・呼び出し先の数を解析しリスト表示します。
- 指定した関数に対して、呼び出し関係をグラフィカルに分かりやすく表示します。

RTOSビューアー・タスクコールスタック表示

- ブレーク時に RTOS のタスク単位で、タスクの待ち状態や優先度の状態、スタック使用量と使用割合を表示します。
- タスクでの関数コールスタックをグラフィカル表示します。

イベントトラッカー

- タスク、割込み、周期ハンドラなどのモジュール名単位で実行遷移を時間軸表示します。
- タスクやOSの待ち状態、プロセッサ単位での実行状況などが確認できるので、システム全体の効率解析に有効です。



ツール開発者から

世界の知恵が結集した最先端のツールを届けたい

統合開発環境である Theia や LLVM/Clangコンパイラ等のオープンソースソフトウェア(OSS)を
SOLIDでどのように利用しているか、ツール開発チームに話を聞きました。

■ 最新のOSSをどのようにSOLIDに取り込んでいるのですか？

僕たちの場合は**GitHub Trendingを毎日のように見て**、新しい機能があったらとにかく順番に使ってみます。どんどん使ってみて、「いいな」と思うものがあれば試作して評価してみるという作業をしています。GitHub上にはソースコードだけではなく、アルゴリズムや考え方も公開されているので、まさに知恵の宝庫です。例えばSOLIDのアドレスサニタイザ機能は、考え方が公開されていなければ自力で作るのは難しかったかもしれません。今の時代、自力でゼロから全てのコードを書いて機能を作成するのではなく、**「OSSを組み合わせる力」**があればできることはどんどん増えていくと考えています。OSSの利用については、ライセンスやセキュリティの要件をしっかりと確認し、それを積極的にソフトウェア開発ツールに取り込んでいます。そのようにして、開発者により新しいツール機能を体験していただけるようにしています。

■ コンパイラやツールチェインで考慮した点は何ですか？

コードを生成するコンパイラ、また標準ライブラリを含むツールチェインですから、**安心と品質を最も重視**しました。具体的にはコンパイラにはLLVM/Clang を、ライブラリには NetBSD のものを採用しています。SOLID Version 5.0 以降のツールチェインは、ライセンス面でより安心していただくために、コンパイラやリンカなどは全て LLVM プロジェクトで構成しています。品質については、今まで利用してきたコンパイラテストプログラム群だけでなく、コンパイラの品質検査ツールである SuperTest Compiler Test (Solid Sand 社) を導入しました。KMC でツールチェイン化するときには常にSuperTest の実行により品質確認をし、マイナーな問題なども修正することで品質向上に活用してきました。またその他に、日本ノーベル株式会社が行っているコンパイラ品質評価サービスも活用し、非常に良い結果を得ており、品質については自信をもって提供できるようになっています。また、品質だけでなく、SOLID-OSでの開発では、リンカスクリプトなどは作成しなくても簡単に使えるよう、IDE と連携して利便性を向上するようにしました。

■ IDE の特徴は何ですか？

今、プログラマー用のエディタで最も人気がある **Visual Studio Code と互換**になったことです。SOLID Version 5.0 では、オープンソースの IDE フレームワークである Theia を採用しました。Theia は VS Code の拡張機能用 API と互換性のある API を保持しているので、SOLID-IDE においてもこれらの拡張機能を利用することが可能です。**VS Code 拡張機能は Web サイト上で公開されている**ので、開発スタイルに合わせて**ユーザ自らが開発環境をカスタマイズできる**点が大きなメリットになっています。拡張機能としては、強力なインテリセンスでコーディングを支援してくれる clangd 拡張機能や、ソースコード管理の利便性を向上させる Git Graph や GitLens などに加えて、**AI エージェント**によりインタラクティブなチャット機能を使ったコード作成ツールの利用等、コーディング時の開発サポート機能を充実させることができます。まさに次世代の開発環境といえるのではないかと思います。

■ これから組込みツールはどのように変化していくと思いますか？

開発ツールそのものを作る言語に関しては、既に C#、Java、Go など多様化しているので、僕たちも用途に最適な (得意な) 言語を使って効率よく開発しています。現在、C/C++ が組込みソフトウェア開発の主流ですが、AI の分野ではアプリ側を Python で作成する事も増えてきました。コンパイル言語とインタプリンタ言語が各々使いやすいように進化していくにつれ、今後は組込みでも OS のような基盤部分とアプリ側を開発する言語が分かれてくることが考えられます。さらに、C 言語と同様にシステム開発に使える**Rust という新しいプログラミング言語**の活用も増えてきており、**SOLID でも 2021 年のリリースから対応しました**。すでに一部のお客さまでも、既存のソフトウェアにセキュリティを堅固にしなければならない部分を優先して Rust 言語で作成する、などの取り組みをされています。GitHubTrending では Baibu や Alibaba など中国発のプロジェクトが次々とポストされて活気があり、活発なソフトウェア開発を肌で感じます。僕たちも、その活気に負けないようソフトウェア開発に取り組み、SOLID を進化させ続けていこうと思います。

技術解説 1　SOLIDのアドレスバグ自動検出機能で安全にメモリを使う

SOLIDには、MMUによるメモリ保護機能を利用したアドレスバグ自動検出があります。これは、従来のデバッグ手法のように、バグのありそうなところに当たりをつけてブレークポイントを張りながらバグを追い込んでいくという手法ではありません。またSOLIDがアドレスバグを検出するのは、不当アクセスが発生した瞬間です。たとえそのバグが他のプログラムに何も影響を与えていなくてもバグが検出できるため、結合テスト時のバグ発生箇所の絞り込みが効率よく行えます。

□ MMU を味方に！ Cortex-A を使った快適なuITRON システム開発

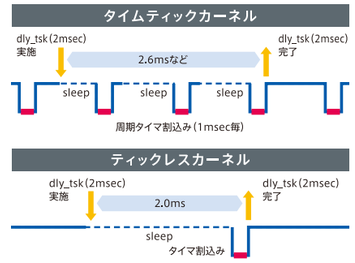
スタックオーバーフロー検出、NULLポインタアクセス検出などの便利なデバッグ機能や、RTOSで使えるDLL機能など、SOLIDがどのように実現しているか note の連載記事で詳しく解説しています。

OS開発者から

技術解説 2　TOPPERS/ASP3、TOPPERS/FMP3 カーネルの特長と機能拡張

「コンパクトなカーネル」「ティックレスタイマ採用による高精度・省電力設計可能な時間管理(右図)」などが大きな特徴で、従来のμITRONと同様なAPI体系であり、既存のITRON/T-Kernelベースのシステムからの移植コストが小さいというメリットもあります。SOLID-OSではさらに以下のような機能を拡張しています。

- □ソフトウェアの大規模化を想定し、タスク優先度を最大256まで拡張
- □タスクなどOS資源の動的生成機能に対応、プログラム実行中にタスクの生成および削除処理が簡単に実行可能
- □優先度継承ミューテックスの追加
- □FMP3に動的資源生成機能を追加
- □メッセージバッファの動的資源生成対応
- □読み手書き手ロックの追加
- □可変長メモリの追加
- □メールボックスの追加
- □一部のAPIについて、実行可能なコンテキストを拡張



技術解説 3　SOLIDのDLLはRTOS仕様

LinuxやWindowsで利用されるDLL機能を、SOLIDではRTOS向けに独自に開発しました。このDLL機能を使うことで、規模の大きなソフトウェアを安全に分割開発できるようになります。

- ■ デバッグ対象のアプリケーション単体でビルド・ロード・デバッグ可能
- ■ 改訂の度に全てのソースコードを集めてビルドする必要がないので、作業効率が大幅に向上
- ■ ロード先の物理メモリは、アドレス重複無いようSOLIDが自動的に設定するため、リロードのオーバーヘッドが無い
- ■ 実機およびシミュレーション環境のいずれでも使用可能

経験を活かしたからこそ、SOLIDの新しさがある

SOLIDには、RTOSの起動やHWの基本的な制御をするコアサービス※という、パソコンで言えばBIOSに相当するソフトウェアがあります。このページで紹介しているSOLIDコアサービスと、RTOSカーネルを開発したベテランエンジニアの皆さんに、SOLID開発に過去の経験をどのように活かしているか、を聞いてみました。※コアサービスはKMCによる造語です。

■ SOLID 開発のきっかけは、どのようなものだったのですか？

「お客様が安心して開発できる環境を提供したい」それが一番大きなきっかけでした。プロセッサが高機能化しソフトウェアが大規模化するにつれ、お客様が本来の開発テーマ以外のところで見えないバグと戦っている現場を多く見てきました。例えばスタックオーバーフロー。よくあるバグですが、一旦作り込んでしまうと見つけるのは大変です。でもLinuxではOSがメモリ保護機能を持っているので、スタックサイズを気にせずプログラムが書けます。それなら同じような仕組みをRTOSで使えるようにすればこういったバグが簡単に見つけられると考えました。プロセッサが高機能化していくと、**プロセッサの使い方もどんどん難しくなっていくのですが、そこはお客様の競争領域ではありません。**KMCはデバッガ製品を通じて、Armプロセッサについて深い知識があるという強みを持っています。またLinuxの動きもよく知っているので、MMUを活かしたRTOSのメモリ保護機構を作ることができたのだと思います。Armプロセッサが高機能化していく中で強化された機能やLinuxでのプロセッサのふるまいを参考にしながら、隅々まで注意を払って丁寧に作ったのがコアサービスです。

■ ロバスト＆セキュアなタスク実行環境を構築

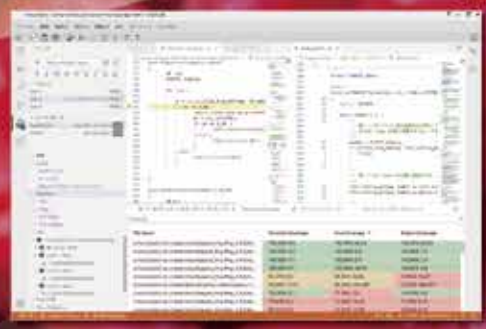
SOLID では MMU を有効活用しプロテクション機能などのメモリアクセス権を適切に設定することで、**「従来のITRON のプログラミング作法のまま安全性を高める」**を実現しています。Version 5.0 ではこの MMU の活用をさらに進化させ、カーネルや通常のタスクやハンドラとは別のアドレス空間 (制限空間) を使えるようにしました。この制限空間で動作するタスクからは、カーネルの空間や他の空間のメモリアクセスは一切できないようにしました。例えば、ネットワーク上の資源と通信するような機能を制限空間で動作させることで、カーネルやカーネルと一緒に動作する**通常のタスクやハンドラをネットワークからの攻撃から守る**ことが可能になります。この制限空間のタスクでも、多くの通常の ITRON システムコールの利用は可能で、通常のタスクやハンドラとイベントフラグやメッセージバッファ、データキューなどで通信することが可能です。この新しい制限空間の機能は、RTOSカーネルがASP3/FMP/FMP3 のまま、**既存の SOLID-OS で開発したRTOS プログラムを修正することなく利用可能**となっています。

SOLIDとRaspberry Piでリアルタイムアプリケーションを動かそう!

Linux/RTOSが同時に動作するラズパイ４用デュアルOSイメージと、Rust言語も使えるRTOS開発環境を無料で提供中!　https://solid.kmckk.com/SOLID/solid4rpi4 >>

ラズパイ4とWindowsPCがあれば、すぐに使い始められます。

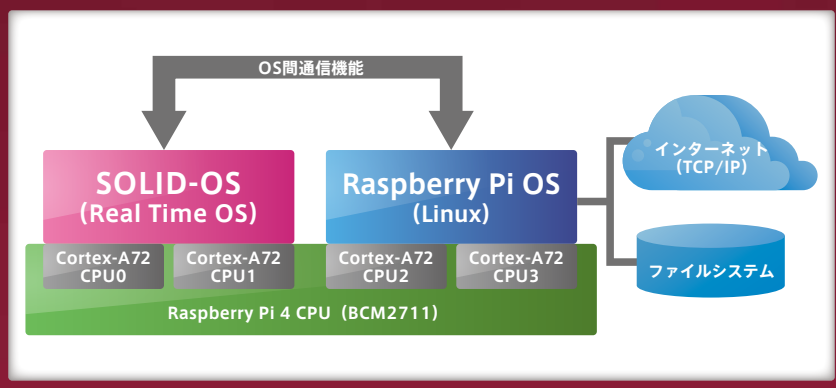
SOLID-IDEで、
Rust/C/C++ 言語をデバッグ



Windows上のIDEで、リアルタイムアプリを開発・デバッグ
Linuxアプリとリアルタイムアプリが同時に動作します!!

技術解説４　Raspberry Pi 4 を使って Linux とリアルタイムOS 共存を体験

組込み機器にUSBやネットワークなどが搭載されるようになり、OSにLinuxを採用する場合であっても 1msec 以下の正確な時間での処理が求められる事があります。ハイパーバイザーを使うことなく、LinuxとRTOS を共存する方法を、Raspberry Pi 4を使ってｎｏｔｅ の連載記事で詳しく説明しています。



KMC Member of eSOL
京都マイクロコンピュータ株式会社
本　　社:〒610-1104 京都市西京区大枝中山町2-44　Tel.075-335-1050
東京オフィス:〒170-0013 東京都豊島区東池袋 2-5-9 加瀬ビル 122 4F　Tel.03-5957-1001
https://www.kmckk.co.jp/
<お問い合わせメールアドレス>
jp-info@kmckk.co.jp

2025.10