KMC

SOLID

enjoy Development

Real Time OS

Integrated with Developing Environment

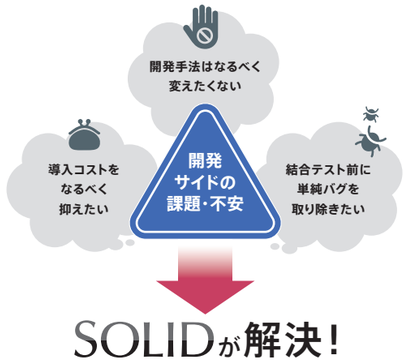
京都マイクロコンピュータ株式会社

https://solid.kmckk.com/

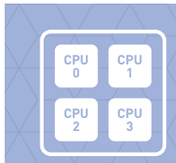
リアルタイムOS SOLIDとは？

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

- 入力補完、自動構文チェックで品質向上
- 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

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200,000円
- リアルタイムOSのロイヤリティが不要

リアルタイムOSの特長

TOPPERS

■ 実績と高機能を備えたTOPPERSカーネルを採用

- 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

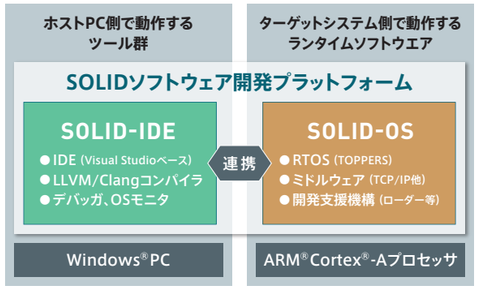
KMC ORIGINAL EXTENSION

■ KMC独自の拡張機能でOSを強化

- uITRON4.0仕様のAPIについて、Toppersで削除された一部のAPIを追加
- 独自開発のDLL機能を追加。DLL単位で単体ビルドやデバッグが可能
  DLL単位で管理できるので、開発時のソースコードアイソレーションが容易

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Visual Studio 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とGCCの2種類が利用可能

■ 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

POINT

1ユーザ、1年間の利用料200,000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、Cortex-A、R、Mプロセッサ、32bit/64bit(AArch64,AArch32両モード対応)全て利用可
  ※ Cortex-M/Rはアップデートで対応予定です
- 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、こんな不毛なデバッグも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、とても満足しました。

コンパイルエラーから解放されました！

コーディングをしていると、変数指定の間違いや、綴じ括弧の抜けなど、エディタが文法ミスを指摘してくれるんです。
入力の手がふっと止まるや否や、文法エラーの箇所に赤波線が出るので、その都度「おっと、そうだった」という感覚で修正していくだけ。そうすればコンパイルが一発で終わるので、気持ちいいですね。
というのも、コンパイル時にエラーが出ると、エラー箇所のソースコードまで戻って「えーっと、ここはどうするんだっけ？」と思い出して修正しなければいけないので、頭を切り替えるのが結構大変なんです。このエディタに慣れてしまったら、もう戻れませんね。

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。

SOLIDを使ったソフトウェア開発

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、Visual Studio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。

コーディング

インテリセンスによるコーディング支援

- 入力候補をリスト表示するコード補完型のエディタで、快適なタイピング環境を提供します。

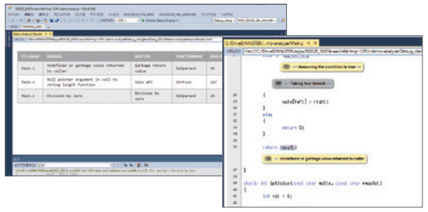
バックグラウンドコンパイルによる文法チェック

- ソースコード入力時に自動的にコンパイラエンジンが動作し、文法エラーを警告します。
- コンパイル時の文法エラー対応から解放されます。

バグを見つける便利な機能

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。

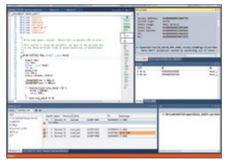


PARTNER-Jet2で実機デバッグ

- 実機デバッグでは、SOLID IDEからKMCのJTAGデバッガであるSOLID-IDEからPARTNER-Jet2を使って実機デバッグをします。

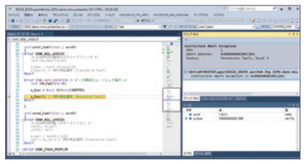
スタックフェンス機能

- 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）しアラーム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。



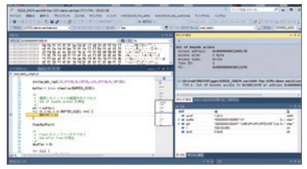
メモリアクセス例外・リンカスクリプト連携

- 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。



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。

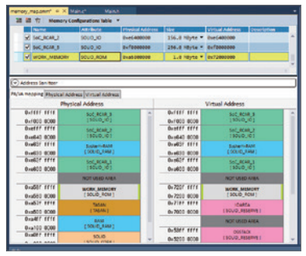


詳しくはこちらをご覧ください

机上デバッグ

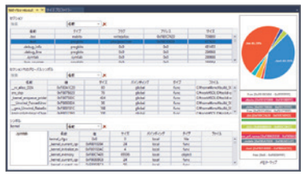
メモリマップデザイナー

- 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。



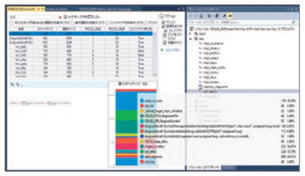
メモリマップ表示

- 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。



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。

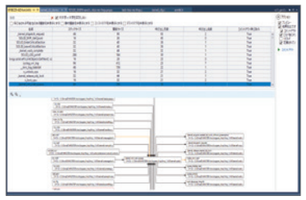


サイズプロファイラー

- 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

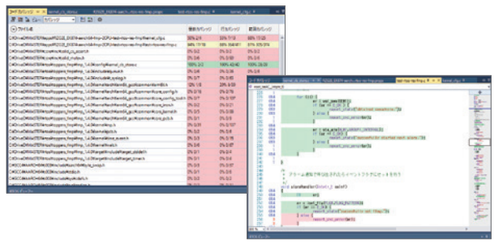
- 関数単位で、呼び出し元・呼び出し先の数を解析しリスト表示します。
- 指定した関数に対して、呼び出し関係をグラフィカルに分かりやすく表示します。



実機デバッグ

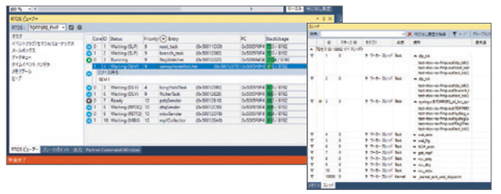
実行時カバレッジ表示機能

- 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。



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を表示します。
- タスク単位で、ブレーク時のスタック使用量と使用割合を表示します。



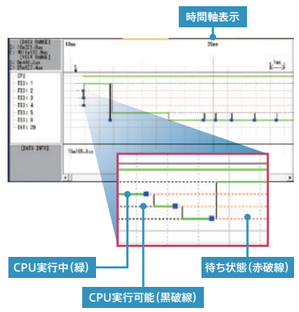
システム評価

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。

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。



システム設計・評価に便利な機能

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Visual Studio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はでなく、**「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。

■ ツールの使いやすさとは何だと思いますか？

ズバリ、**使いたい機能がすぐに探せて簡単に使えること**です。「ツールの使い方が分かりにくい」というお話を聞くことがあります。確かに今までの組込みソフトウェア開発環境は、プロセッサベンダー独自のツールも多く、操作方法や機能がバラバラでした。ここ数年、Armプロセッサへの集約が進んできた結果、プロセッサを変えた途端にツールの操作方法も変わるので、それがユーザのストレスになるのは当たり前です。SOLIDでは、組込みに限らずIDEとして世界的に広く利用されているMicrosoft社のVisual Studioのフレームワークを利用することで、**「コマンド探しで迷わない」**ようにしました。またSOLID独自の新しいデバッグ機能を有効にするための操作は、1クリックもしくは1つのダイアログボックスで全ての設定ができるようにしました。いくら便利なデバッグ機能であっても、簡単に使えなければ使っていただけないので、操作しやすさは徹底的に作り込んでいます。

■ コンパイラやツールチェーンで考慮した点は何ですか？

ひとつは**GCCとClangの2種類のコンパイラで同じソースをコンパイルできることです。**新しいデバッグ機能の多くはClangコンパイラと連携して動作するため、デバッグ時はClangコンパイラを使っていただくのですが、リリース時はGCCでコンパイルしても問題ありません。GCCで作った既存のコードも利用できるので、便利だと思っています。また全てのビルド条件をSOLID-IDE上で設定できるようにし、リンカスクリプトを書かなくて済むようにしたのも、SOLID/SOLID-OSの使いやすさの特長です。これはLinuxのアプリケーションを作成するときにリンカスクリプト無しで開発ができるのと同様の感覚でSOLID-OSの開発ができるということを意味しています。というのも、リンカスクリプトは自由に書ける一方で過去の資産を使う場合に「秘伝のたれ」のように複雑な作りになっていて、プロジェクト移植上の悩みの種だと聞くことが多く、SOLIDで何とかそこを解決したいと考えたからです。

■ 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でも今後取り組みの対象にしたいと思っています。GitHubTrendingではBaidu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。

□ スタックオーバーフロー

SOLIDでは、アドレスバグの代表的なものである、スタックオーバーフローの自動検出を標準機能として備えています。SOLIDのコアサービスが、左図のようにスタックメモリの境界に物理メモリを割り当てない領域を配置し、スタックを積み過ぎた瞬間にアクセス例外としてデバッガがバグを検出します。

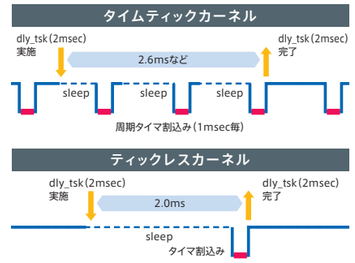
さらに詳しく

OS開発者から

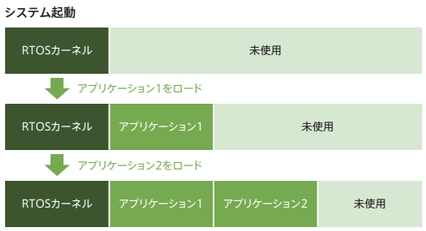
技術解説2 TOPPERS/ASP3、TOPPERS/FMP3 カーネルの特長と機能拡張

「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解放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
- FMP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の開発のきっかけは、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。

■ RTOSカーネルを開発するとき、どのような点に留意しましたか？

「移植性の良さ」「メンテナンス性の良さ」にはかなり気を使って開発しました。SOLIDの最初のターゲットは32bitのArm Cortex-Aプロセッサですが、64bitプロセッサやマルチプロセッサ構成への対応、さまざまなSoCに展開することを想定した作りになるよう心掛けました。過去、組込みプロセッサが16 -> 32bitに移行したときに、RTOS開発担当者として苦労した経験があるので、プロセッサの違いをなるべくカーネルやコアサービス側で吸収してお客様の負担を減らせるような作りにしています。とはいえプロセッサの展開を見ていると、時にはバージョンアップとは言えないような、大きな機能変更に遭遇することがあります。KMCでは、新しいプロセッサやSoCに対応するとき、まずPARTNERデバッガを接続するところからスタートしているので「いざとなったらデバッガを使って内部動作をとことん調べればいい」という気持ちで、これからもどんどん新しいSoCに対応していきます。