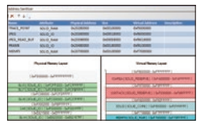
SOLIDでは「コアサービス」と名付けた一連のランタイムがIDEと密接に連携することで、見開き面の多くの機能を実現しています。RTOSでMMUを活用するための機能もコアサービスに実装されています。MMU活用などについて、コアサービスの仕様検討から実装までの開発を一貫して担当した小林章が、その役割や特徴的な機能を解説します。

京都マイクロコンピュータ株式会社
SOLID コアサービス開発エンジニア | 小林 章

高機能を簡単に使える コアサービス



GUIツールでメモリマップ設定

実装済みSOLID MMUライブラリ

GUIで設定しビルドするのみ!!

コアサービスMMU処理
SOLID_MEM_* API群

- SOLIDコアサービスで、複雑なARM MMU処理も、プログラムを書くことなく簡単に利用可能!!
- コアサービスには、他にELFローダーなど基本機能を実装済み

Q コアサービスには、そもそもどのような機能があるのですか？

主にターゲットのリソース(メモリ/MMU, 割り込み, タイマ等)の管理と、ユーザーへの補助(ローダー,デバッグ)機能があります。
ターゲットシステムの一部に常駐していますが、システムセットアップや動的解析機能を使ったデバッグ時など、必要に応じてIDEとも連携して動作するソフトウエアです。

Q ターゲットのリソース管理とはどのようなものでしょうか？

はい、少し古いですが「BIOSのようなもの」と言えばイメージがわきやすいでしょうか。シンプルなAPI群を使ってリソースを操作するものです(左図)。
また、ブート時のMMUの仮想アドレスへの移行といった、ユーザーが操作するのが面倒な処理も受け持っています。

Q SOLIDを開発するときに最も留意した点は何ですか？

「リアルタイム」を追求した「コンパクト」な構成であることはもちろん、ユーザーが「スマート」に開発できるよう「シンプル」な仕様を心掛けています。その上でPCとターゲットのどちらでどのように処理を行うかも重要です。
ちなみに、コアサービスは、KMCの造語です。縁の下の力持ちという役割からして、結構しっくりくる名称ではないかと思っています。

Q SOLIDではMMUのどのようなメリットに着目したのでしょうか？

まず各機能(モジュール)を仮想アドレス上に分散して配置できる点です。これにより、領域外アクセスであるバッファオーバーランの検出や(右図①)、特定のメモリ領域に対してアクセス制限をするセキュリティ管理にも有効です。また複数拠点で開発する場合に、モジュールを各開発拠点単位に分割・分散配置して、開発対象モジュールだけに限定してデバッグ操作を行うといった管理が簡単に実施できるようになります(右図②)。

Q 従来のMMUの利用方法と異なる点は何ですか？

Linuxであれば、仮想アドレスで空間を多重化し、なおかつデマンドページングによって動的にメモリ割り当てを行うといった機能がありますが、RTOSの場合はリアルタイム性(再現性)を重視するので、空間の多重化は使用しません。SOLIDのアプローチとしては、単一の大きな仮想アドレス空間にプログラムを配置するのにMMUを利用しています。SoCの構成に合わせた仮想アドレスの設定は全てIDE側の簡単な操作で利用できるようにしましたので、ユーザーはメモリの設定プログラムを自分で作成する必要はありません。
また、物理空間のメモリ配置をIDE上で表示することで、何か問題が起きた時でもプログラムがどうなっているかをユーザーに分かりやすく伝える工夫もしています。

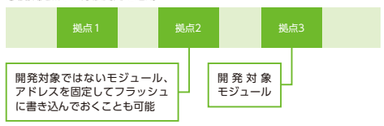
Q ユーザーにとっては嬉しい機能ですね

はい。MMUの設定は複雑ですし、それを間違えてしまうとデバッグが出来ない状態に陥ってしまうこともあるので、従来のユーザーは割と避けていた部分です。そこをツールベンダーが専門家として取り組んだ事は、ユーザーにも多いに喜んでいただける点だと思います。

MMUの有効活用なら SOLIDにお任せください

①領域外アクセス検出などのデバッグ機能強化に活用

②複数拠点での分散開発に活用



インテリジェントローダーが スマートなデバッグ環境を生み出す

ELFロード時にSOLID-IDEとARMターゲットシステム上のSOLIDコアサービスが連携。自動的にIDEデバッグを開始

Q SOLIDのローダーの特徴は何ですか？

プログラムの規模が大きくなってくると、修正の度にプログラム全体をロードする時間が開発者の負担になってきます。それなら新たに開発対象モジュール単体だけをロードする仕組みを作ろう、という事で出来上がったのがSOLIDのローダーであり、SOLIDの最も大きな特徴のひとつです。

Q モジュール単体でビルド(リンク)すると、モジュール間参照などの問題は生じませんか？

いいえ、ローダーにリンク機能を持たせていますので、モジュール間に参照が残っていても問題ありません。かつSOLIDではモジュールのビルドでリンクの問題が生じないように、かつ「コンパクト」にロードできるような工夫をしています。リンクもDLLのような形式ではないので、モジュール間の双方でシンボル参照をしていても問題ありません。またモジュールが追加されたことを、ローダーがIDEに通知するので、IDE側でシンボルを自動的にロードしたり、逆にロードするモジュールをIDE側で差し替えたりすることも可能です。

Q ローダーとIDEは深く連携しているのですね

はい、それがSOLIDの一番の特徴といえます。ローダーはRTOSカーネル情報、MMUによるマッピング情報、IDEのビルダが持っているシンボル情報やデバッグ情報の全てと連携して動作しています(左図)。SOLIDが開発環境の一体化を実現したことで、ローダーをはじめとするスマートな開発スタイルが実現できたのは間違いありません。

京都マイクロコンピュータ株式会社
http://www.kmckk.co.jp/
お問い合わせメールアドレス：jp-info@kmckk.co.jp
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：〒105-0004 東京都港区新橋2-14-4 Rビル5F　Tel.03-5157-4530

2017.5

enjoy Development

KMC

SOLID

開発準備から性能解析までをシンプルに変える

京都マイクロコンピュータ株式会社　http://www.kmckk.co.jp/SOLID/

開発環境と実行環境の一体化で実現する
SOLIDの様々な機能。

これらが、あなたの開発を加速します!!

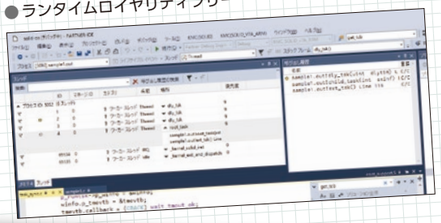
RTOS TOPPERS/ASP3

TOPPERS/ASP3を採用

- 軽量コンパクトな最新のオープンソースITRON
- ティックレスカーネル
  - 省電力性にすぐれる
  - μsecオーダーの時間管理を効率的に実現
- 資源動的生成版カーネルと、よりコンパクトな静的生成版カーネルを用意
- ランタイムロイヤリティフリー

SOLID-IDEと密接に連携

- スタックフェンス、スタック解析機能
- システムコールトレーサ
- タスク状況をIDEに表示
- RTOSの資源表示
- 実行コンテキスト(タスク、割り込み)遷移表示



SOLID-IDEのデバッガに、タスクリストや状態を表示。各タスクの状態だけでなく、関数呼び出し状況やローカル変数もタスクを切り替えて表示可能

統合環境で、リッチにRTOS分析

Visual Studio® IDE エディタ, デバッガ

実績豊富なVisual Studioを採用

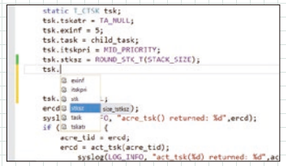
- SOLID用に専用実装をしているので、Visual Studioの別途導入や購入は必要なし

Visual Studioエディタが持つ高度な機能を、Clangコンパイラと連携して実現

- 入力リストを候補表示するインテリセンス
- ソース編集時にコンパイル時の問題を警告(入力時に自動的にバックグラウンドでコンパイラエンジンが動作)

Visual Studioデバッガによる快適なデバッグ

- ブレーク箇所周辺で参照している変数を、ローカル・グローバル問わずに集めて表示する自動変数表示
- SOLID-OSと連携し、タスク別の呼び出し参照表示
- ARM CPU例外発生時に、例外発生箇所をポイント
- タスクを認識したブレーク機能



IDEでのソース編集時に入力リストを候補表示。メンバの型も表示

IDEでのソース編集時に入力と同時にコンパイル時の問題を警告

動画で詳しく見る

サクサクコーディング

ツールチェイン GCCコンパイラ, Clangコンパイラ

選べる二種類のコンパイラを同梱

- 実績豊富なGCCコンパイラ
- 先進的機能のLLVM/Clangコンパイラ

リンカは実績あるGNU binutilsを同梱

C99,C++11仕様をサポート

生成されるソフトウェアはGPLフリー

標準ライブラリについては、ソースコードが付属

KMCデバッガと連携できる、開発時に便利なVLINKライブラリも同梱。VLINK利用もSOLID-IDEから簡単設定

コンパイラの持つ豊富な機能(プロファイルや関数フックなど)も、SOLID-IDEと連携して簡単に利用可能

リンカスクリプト設定もIDEのGUIで設定(メモリマップデザイナ画面)可能

最新強力なOSSコンパイラを自在に!

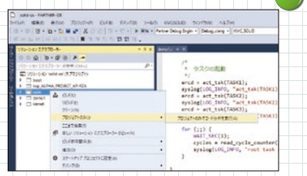
ソースコード静的解析

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

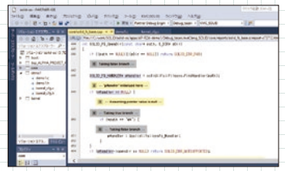
- 関数呼び出しや分岐条件を加味し、未初期化ポインタアクセスやゼロ除算など、様々な問題を指摘
- IDEのビルド環境と連携しているので、#ifdefなどの条件もビルド設定をそのまま反映

IDEのプロジェクトメニューを選択するだけで、簡単に実施可能

問題箇所と、問題に繋がる箇所をソース上でハイライト表示



IDEでメニュー選択するだけで静的解析が実行可能



解析結果もIDE上のソースにハイライト表示

スッキリソースコード解析

ARM Cortex-A メモリ管理機能

ARM® Cortex®-Aシリーズの MMUを“シンプルに”積極的に活用

- 仮想アドレスは使うが、単一空間なので常に全てのメモリ空間が見えている(わかりやすい)
- 生成されたプログラムについては、自動的にセクションに応じたプロテクションを設定
  - .text=実行可/読可, .rodata=読可, .data&.bss=読書可のような設定です

利用する仮想アドレスを設定しやすくするためのIDEメモリマップレイアウト機能

- リンカスクリプトと連携
- MMU初期化コードと連携

機能豊富だけど、設定が複雑なARM Cortex-AのMMUを、簡単に安全に使えます

IDEに搭載されるメモリマップレイアウト機能

ラクラクMMU活用

裏面も読んでね

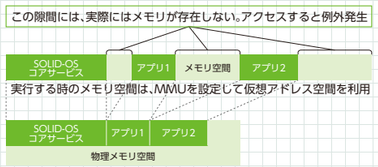
ELFローダー

分割開発を可能にする本格的なELFローダーを標準提供

- 双方向アドレス解決
- 組込向けに特化した軽量動作
  - 固定アドレスで実行
  - アンロード機能無し
- MMUを使って効率的にメモリ使用
- ローディングにMMUを使ったプロテクション設定
- 自動デバッグ開始起動
  - SOLID-IDEと連携することで、ローディング時に自動的にデバッグ開始

軽量動作の実現

- SOLIDツールチェインに、ARM側ローダーと連携した専用機能を追加
  - 開発環境と実行環境の一体化で、実行時の負荷を削減
- ROM上のプログラムを効率的にロードする専用ファイルシステム



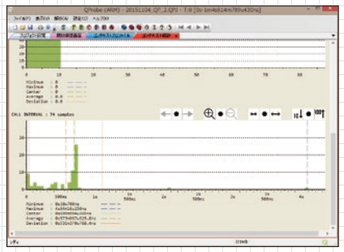
効率的なメモリ利用

ローディング時に、アプリ間のアドレス解決

プログラム更新・デバッグをスマートに!!

裏面も読んでね

性能解析プロファイラ



QProbeによるタスク実行時間分散画面

ARM Cortex CPUに搭載されるPMU(Performance Monitoring Unit)を利用した計測

- CPU負荷状況
- キャッシュミス率
- メモリアクセス回数など

RTOSと連携した性能解析

- タスク別の計測や集計

プロファイルツールQProbeとの連携

- QProbeでタスク解析するための設定を導入済み
  - 関数単位やタスク単位で、実行履歴・比率・分散状況などが把握可能

簡単性能解析!!

アドレスサニタイザ

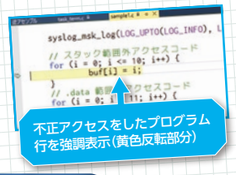
C/C++言語プログラム実行時のメモリアクセスの不正実行を自動検出

- Clangコンパイラの持つ機能と、SOLID-OS上の検出ランタイム(KMC製)を組み合わせて実現

ローカル変数、グローバル変数について、オーバーランや未定義領域アクセスの問題検出が可能

memcpy()など連続するメモリアクセスであっても、専用ライブラリ(KMC製)の利用で、効率良く問題検出が可能

不正アクセスがあったメモリ領域を表示(ピンク網掛け部分)



不正アクセスをしたプログラム行を強調表示(黄色反転部分)

使い方は簡単。アドレスサニタイザモードでビルドして実行するだけです。自分で考えてブレークポイントを設定するなど必要ありません。SOLID-IDEとSOLID-OSが連携して、間違ったメモリアクセスを自動的にあぶり出してくれます。

バグ付きのソースコード

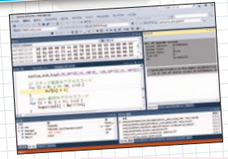
```
static unsigned short bugarray[10];

void root_task(intptr_t exinf)
{
        int             i, x = 0;
        ER              ercd;
        char            buf[10];

    // スタック範囲外アクセスコード
    for (i = 0; i <= 10; i++) {
        buf[i] = i;
    }
    // .bss 範囲外アクセスコード
    for (i = 0; i < 11; i++) {
        bugarray[i] = 0x1111*i;
    }

    /*
     * タスクの起動
     */
    ercd = act_tsk(TASK1);
```

10個の配列サイズを越えた書き込みが、i=10の時に発生する



動画で詳しく見る

メモリバグ自動検出!!