ARM Tech Symposia 2016

京都マイクロコンピュータ株式会社

ARM v8アーキテクチュアのシステム性能解析と、
LLVMを用いた統合開発プラットホーム

PARTNER-Jet2

ARM V8プロセッサ 64bit対応デバッグについて

PARTNER Jet2

PARTNER-Jet2　Model 10
USBバスパワー対応で、
低価格で導入しやすい普及モデル

4Gバイト/8Gバイトトレースメモリ
搭載の高機能モデル
PARTNER-Jet2　Model 20/30

PARTNER-Jet2の利便性

- USBバスパワー対応
  - PARTNER-Jet2 Model10は、USBバスパワーで動作
  - USB3.0のみならず、多くのPCではUSB2.0でも動作可能
    - 電圧低下時はLEDでそれを表示。その場合はACアダプタをご利用下さい

- プローブホットプラグ対応
  - ターゲット動作中に、後からプローブを接続してデバッグを開始したり、取り外したりする事が可能
  - 障害が発生してから、JTAGで状態確認が可能
  - ARMプロセッサーとIntelプロセッサーで対応

ARMv8 64bit
実行状態と命令セット

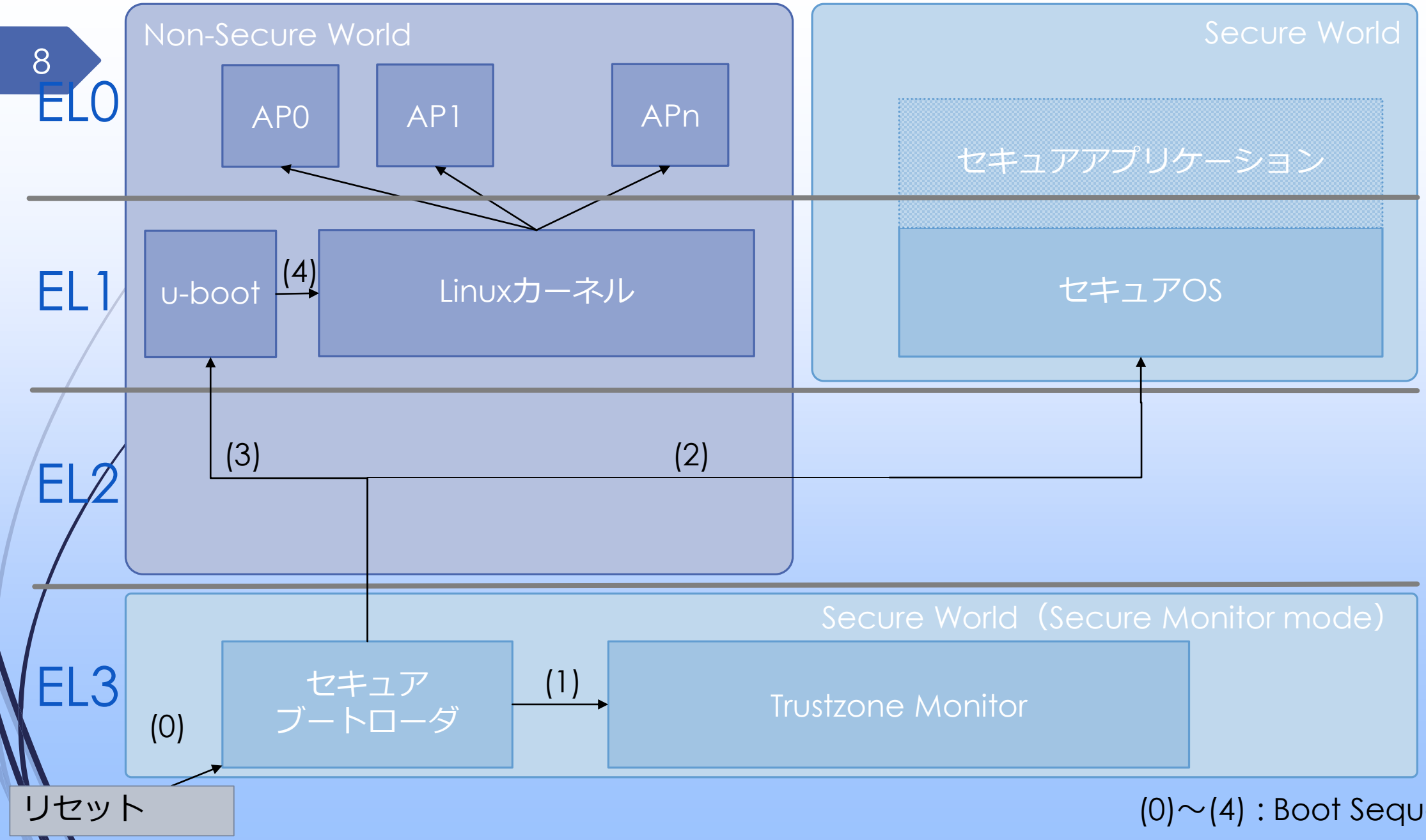
- AArch64　,64bit実行状態
  - 64bitレジスタや64bitアドレッシングなど、64bitのプログラマモデルを実行する
  - A64命令セット
    - ARMv8で新規に導入された32bit固定長の命令セット
- AArch32　,32bit実行状態
  - 32bitレジスタや32bitアドレッシングなど、32bitのプログラマモデルを実行する
  - A32命令セット
    - ARMv7までのARM命令セット相当
  - T32命令セット
    - ARMv7までのTHUMB命令セット相当
- AArch64とAArch32の切り替えは、例外(割込)と復帰のみ

ARMv8 64bit
A64レジスタセット

- 31個の64bit汎用レジスタ（X0～X30）
  - 従来のARM/THUMBの例外によるレジスタバンク無し
  - レジスタ番号による命令制限なし
- 0固定のレジスタ（XZR）
  - 殆どの命令で利用可能
- 汎用レジスタではない専用のスタックポインタ（SP）とプログラムカウンタ（PC）
  - PC/SPも64bit
- プロセッサ状態を示すステータスレジスタ（SPSR）
- 32個の浮動小数点レジスタ（V0～V31）
  - 128bit幅の浮動小数点レジスタ
  - 32個の64bit幅の浮動小数点レジスタにすることもできる

例外レベルと、セキュア状態

		非セキュア	セキュア
EL0	アプリケーション	あり	あり
EL1	カーネル	あり	あり
EL2	仮想マシンモニタ	あり	なし
EL3	セキュアモニタ	なし	あり

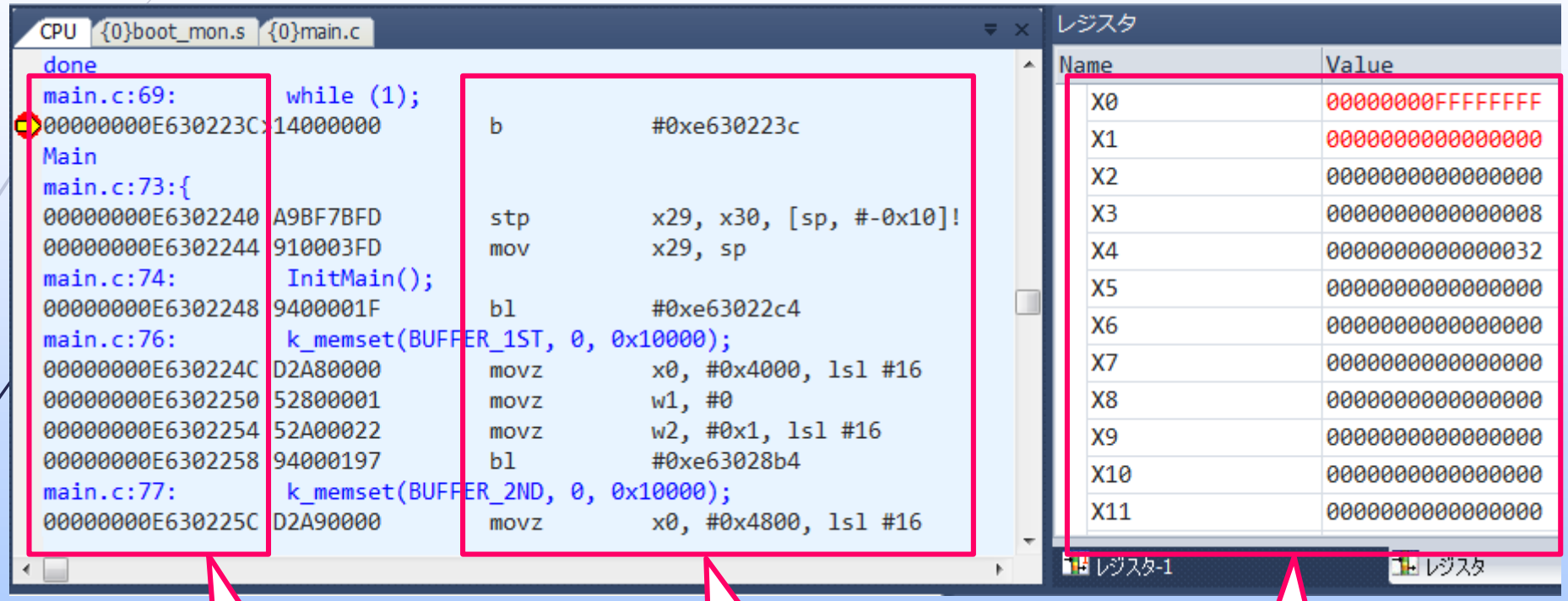


(0)～(4)：Boot Sequence

PARTNER-Jet2
ARMv8 64bit対応

- 64bitアドレス空間への対応
- 64bit化されたレジスタセットへの対応
- 実行状態に応じたデバッガI/Fの切り替え
  - AArch64では64bit表現
    - アドレス表記が64bitで、レジスタセットもX0～X30などの新しいレジスタセット
  - AARch32では32bit表現
    - アドレス表記が32bitで、レジスタセットがR0～R15などの今までのレジスタセット
- A64命令とA32命令とT32命令をサポート
  - 逆アセンブラ、アセンブラについても対応
- 実行状態や命令セットが混在していても、自動認識してデバッガI/Fや命令デコードを変更
- A64のシステムレジスタアクセス機能
  - PIS/POSコマンドで、デバッガから TTBR0_EL1 などのシステムレジスタの読み書き可能

デバッガスクリーンショット

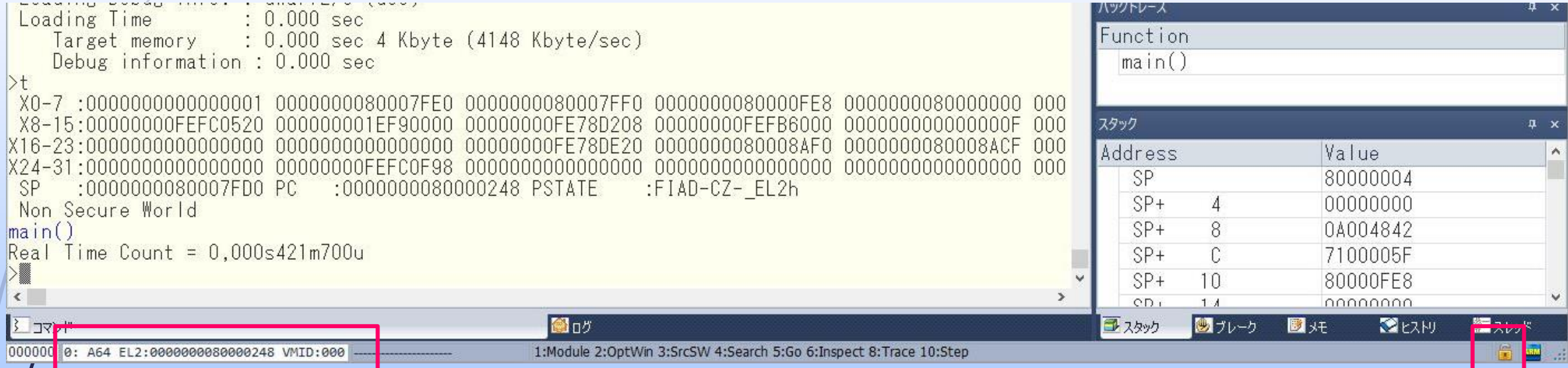


64bit化された
アドレス

新しいA64命令の
逆アセンブル

64bit化された
レジスタ

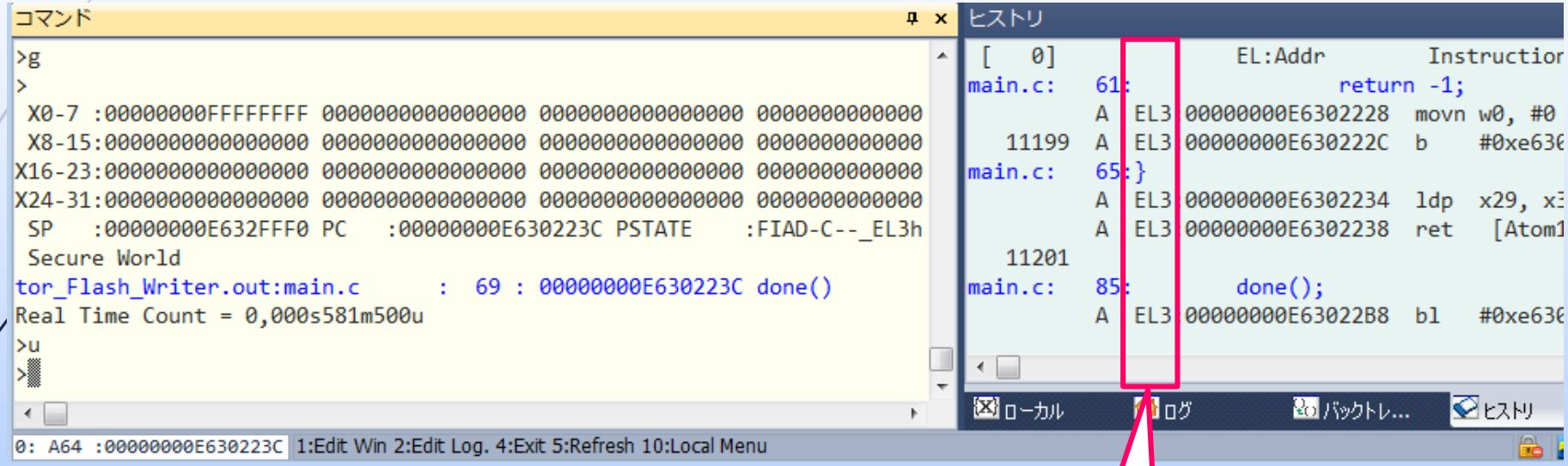
デバッガスクリーンショット



現在のCPU状況の表示（左から）
CPUコア番号: 命令レベル（A64/A32/T32）：例外レベル：プログラムカウンタ：仮想マシンID

セキュア／
非セキュア
の表示

デバッガスクリーンショット



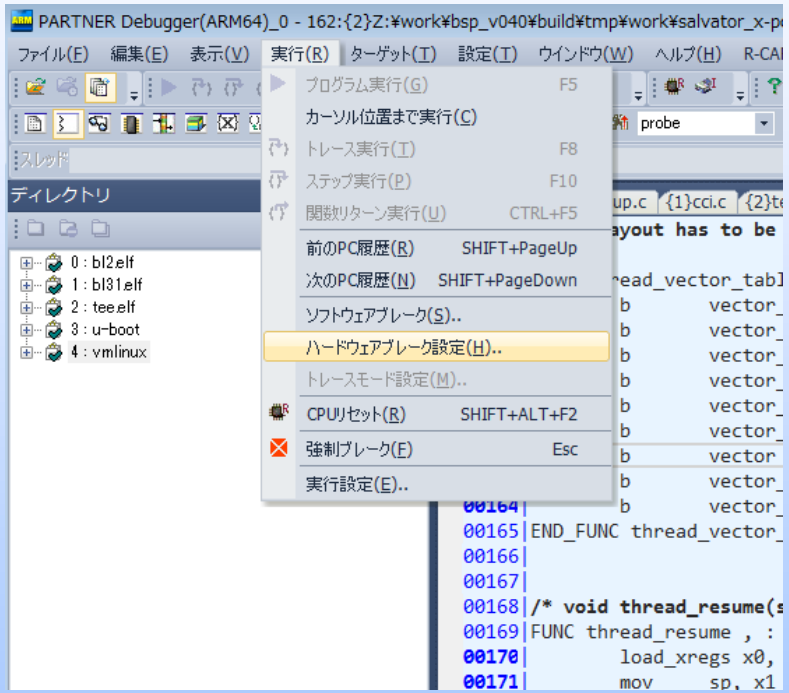
ETMトレースに、
例外レベルの表示

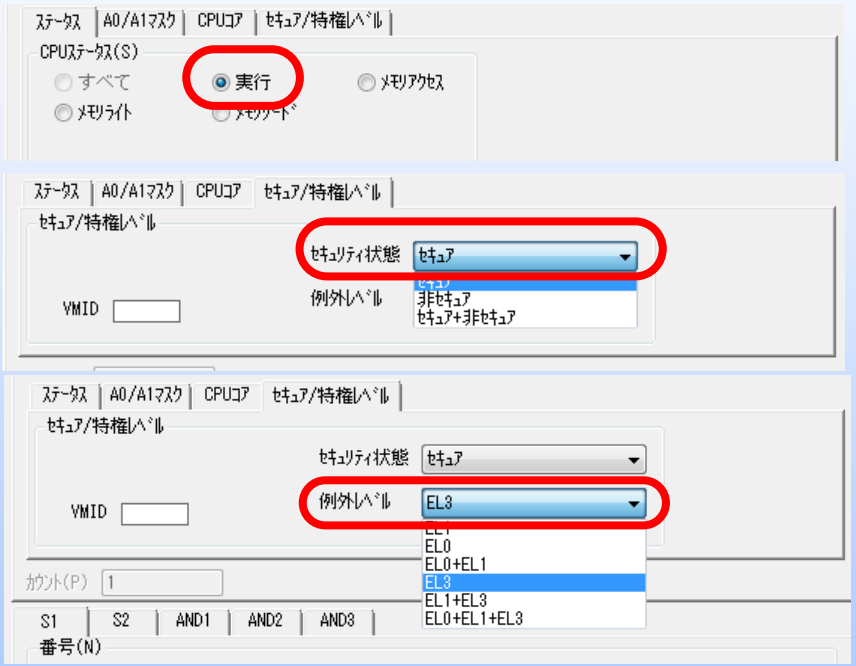
ハードウェアブレークとベクタキャッチブレーク

- ハードウェアブレークは、アドレスの他に、例外レベルや仮想マシンIDを紐付けて設定可能
  - EL3で0xF800_0000番地を実行したらブレーク
  - VMID=5の0x4000_0000番地をリードしたブレーク
  - EL1でVMID=1の0x8000_0000番地をライトしたらブレーク
- ARMv7まであったベクタキャッチブレークは簡略化
  - ハイパーバイザモニタのIRQベクタでブレーク、などのような設定はできなくなった
- Exeptionキャッチブレーク機能が追加
  - Exception Levelの変化を捕まえることが可能
  - 次にEL3を実行したらブレークする、次にEL1/セキュアを実行したらブレークする、など

Exception Levelを指定してH/Wブレークを設定する

"実行"メニュー → "ハードウェアブレーク設定"

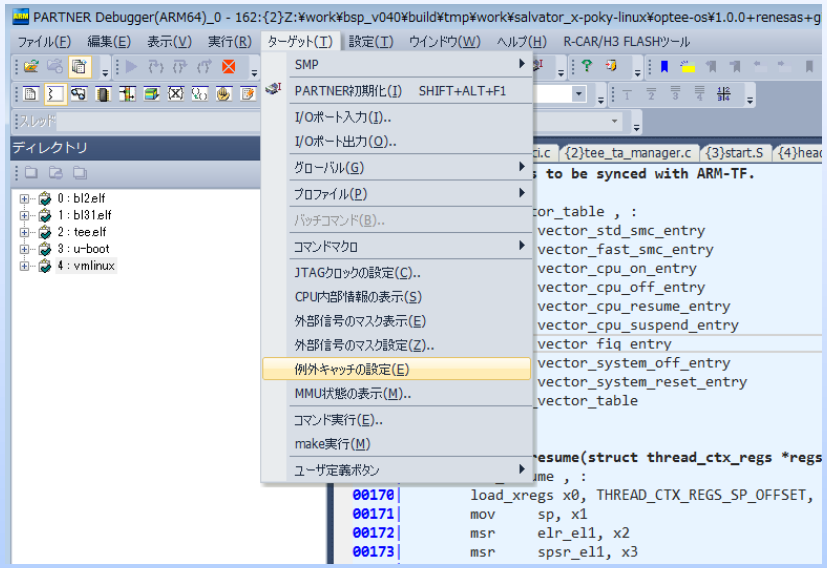


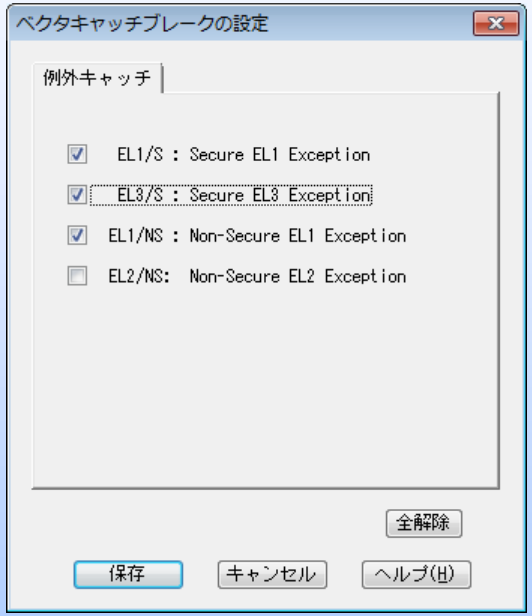


ハードウェアイベント設定ダイアログの、"ステータス"項目や"セキュア／特権レベル"項目を設定。
ステータスは実行だけでなく、メモリアクセスでの選択も可能。
それを設定することで、特定のアドレスに対して、EL3でのメモリアクセスだけをブレークさせる、などが出来る。

Exception Levelの変化を捕まえる

“ターゲット”メニュー → “例外キャッチの設定”





このように設定することで、Exception Level,Secure/Non-Secureが変化したところを捕まえられる。
一度捕まえた時に、その実行状態のキャッチを自動解除する（プロセッサ仕様のため）。
したがって、複数設定しておかないと、同一の実行レベルの変化を連続的に捕まえることはできない。
CPU実行中でも本設定を変更することができます。

PARTNER-Jet2
ARMv8 64bit対応（デバッガ機能）

- 対応済み64bitプロセッサ
  - Cortex-A53, Cortex-A57, Cortex-A72
- 新しいETMトレース（ETM v4）対応
  - 従来のETMv3.5よりも、パケット効率がよくなった
  - トレースバッファ（ETB）では対応、動作確認済み
  - LVDS化されたETM HSSTPへの対応
    - R-Car H3（Cortex-A57/A53）で対応、動作確認済み
- PARTNER-Jet/Jet2で提供してきた各種機能への対応
  - スナップショットデバッグ
  - ホットプラグ接続
  - プロファイル機能
  - その他…

PARTNER-Jet2

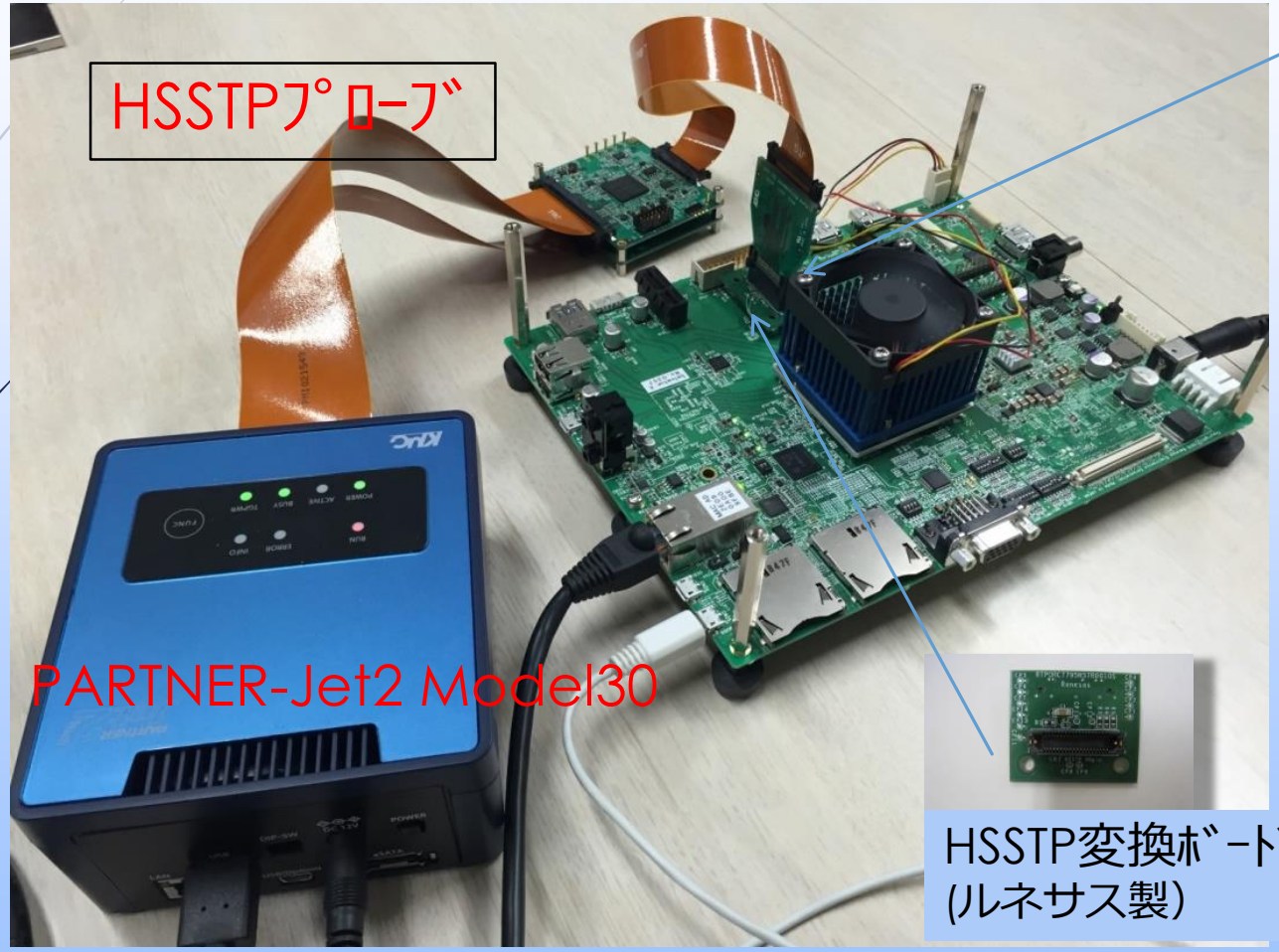
HSSTPを使った、ARM Vv8のシステム解析

HSSTPプローブ

- ARM High Speed Serial Trace Port仕様対応のプローブ
- 4 Lane HSSTP　サポート
- 2.5Gbps/5.0Gbps対応
  - ARM ETMv4.0対応
  - ARM STM対応　　400MByte/sec

  PARTNER-Jet2/Model30との組み合わせで使用

R-Car H3 Salvator HSSTP接続



CN6

HSSTP
(5Giga bps Trace)

STM 転送実績
440Mbyte/s

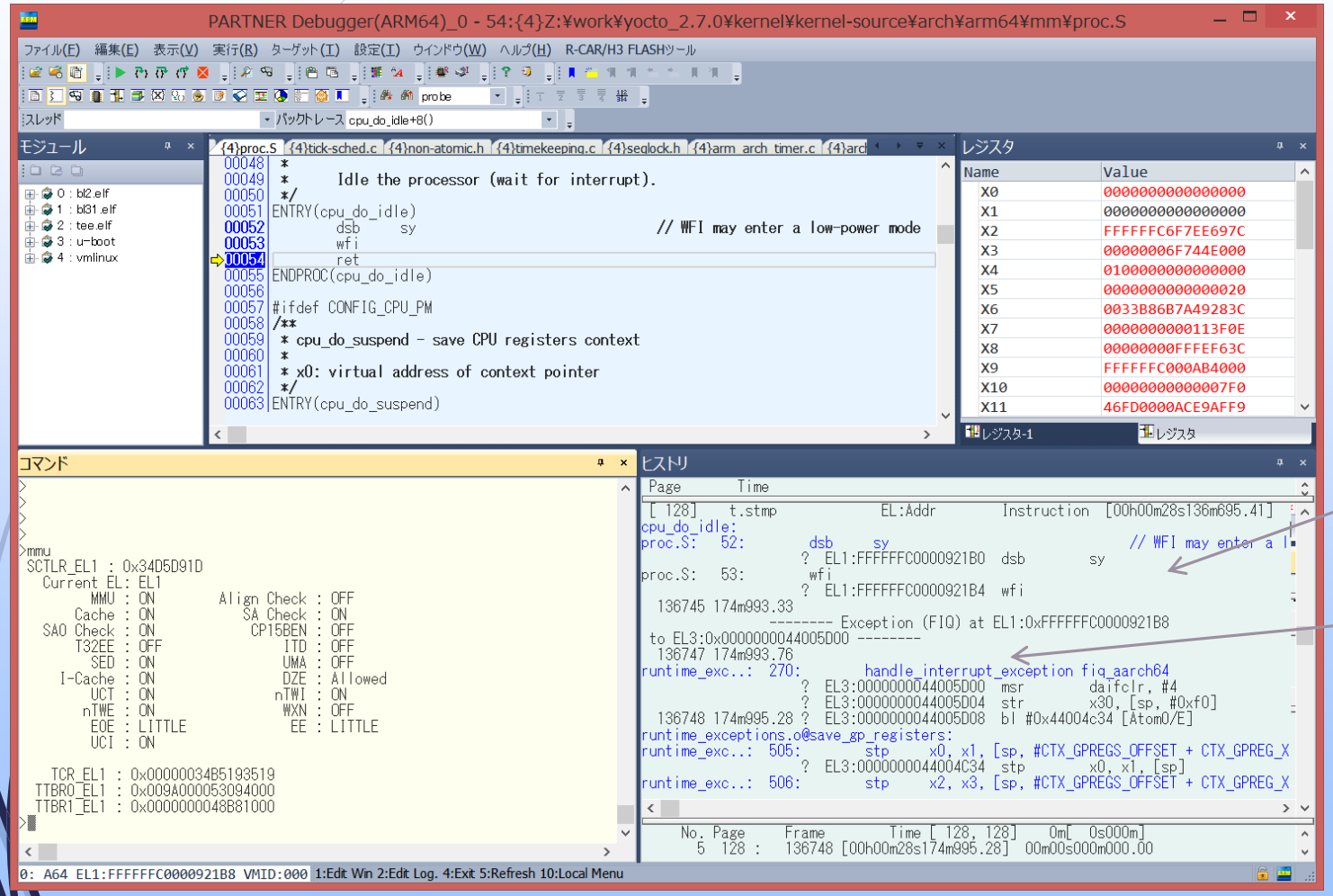
HSSTPで何を出すか？

- HSSTPは、トレースデータ出力の転送フォーマット
  - ETMやSTMのトレースを出力するのが主か？
  - ETM/PTM
    - CPUの実行状況のトレース（分岐トレース）
    - 命令実行の履歴などを再現できる
    - トレースデータの中身は決まっているので、仕様に従い解析・表示機能を作成できる
  - STM
    - 色々あるが、簡単なのは任意のメモリデータを出力することができる
    - どういうデータを出力するのかにより、解析・表示は異なる

HSSTPプローブ ETM mode

- Cortex-A57/53/R7の命令レベルリアルタイムトレース
- AArch64/32混在トレース
- 非セキュア・セキュア空間トレース
- EL0からEL3のレベル遷移トレース
- マルチコアトレース
- 10nsの分解能のタイムスタンプ
- 長時間トレース(10秒程度のロギング)
- Qprobe（オプション）による動的性能解析
  - 関数解析、コンテキスト解析等

HSSTPプローブ ETM mode



プログラムトレースによる実行レベル遷移の表示

EL1実行中

EL3へ遷移

ETMの活用

- ARM v8のソフトウェアスタックがフル実装されている時の動作解析に有効か
  - 分岐パケットではSecure/Non-Secureの区別、またEL0/1/2/3の各Exceptionレベルが記録されている
  - 実行モードの遷移するときの状況把握
  - 実行モードの比率を確認

HSSTPプローブ STMモード

- STMを以下のような使い方を検討
- OS/アプリレベルの実行状況を出力
  - カーネルへのパッチ、関数へのフックポイント挿入で、ソフトウェアの動きをSTM使って高速に出力
- 指定領域のメモリを高速に外部に保存
  - DDR → DMA → STM → HSSTPのルートで、R-Car H3の場合400MByte/secでメモリ出力が可能
  - ギガバイト単位のデータを、検査・デバッグ用に高速に外部保存する事も可能か

HSSTPプローブ STMモード

- STMを使ったソフトレベルトレース（現在開発中）
- Cortex-A57/53のLinuxソースレベルトレース
  - トレース内容
  - 関数トレース機能
  - コンテキストスイッチ
  - システムコール
  - 割り込み
  - 任意のトレースポイント
  - Printf的なログ出力
- PMU(キャッシュヒット率等)
- Qprobe（オプション）による動的性能解析
  - 関数解析、コンテキスト解析等

STMの活用

- 取り込んだ画像イメージを、テスト用に外部に出力する
  - 400MByte/secという帯域を使って画像イメージを出力し、PCに連続保存して、検証に応用する事が可能
    - システムへの負荷計算は必要
- 障害発生時のメモリ保存
  - JTAGデバッガからDMAを設定しSTM経由でメモリをPCに保存
  - 問題発生時のメモリを、詳細に静的検証
  - １ギガバイトのメモリを、数秒で転送

2017年提供 新製品のご紹介

ソフトウェア開発プラットフォーム

ソフトウェア開発プラットフォーム

SOLID

京都マイクロコンピュータ 2017年 提供予定新製品

新しくソフトウェア開発プラットフォームを提供する意義

- KMCはデバッガやGCCコンパイラなどの開発ツールを提供してきました
- しかし、RTOSやOSレス環境での、組み込みソフトウェアの開発手法、特にデバッグやテストの手法については、ここ10数年大きく変わっていない
  - 変わっていないのは、僕たちツールベンダに責任がありますね....
- 一方、iOSのアプリケーション開発環境などは、今まで以上に便利な開発環境に進化
  - Xcode開発環境は、アドレスサニタイザー機能など、多くの開発者に評価されている
- これらの進化は、ツール単位では難しく、プラットフォームOSとツールを一体化させることで実現できる可能性があり、あらたにソフトウェア開発プラットフォームを提供する、に至った

SOLID

かしこく開発・スマートにデバッグ

プログラムの問題を自動検出, RTOS とコンパイラを一体化した
革新的開発プラットフォーム

SOLID

RTOS

コンパイラ

IDE

デバッガ

SOLIDが目指すもの

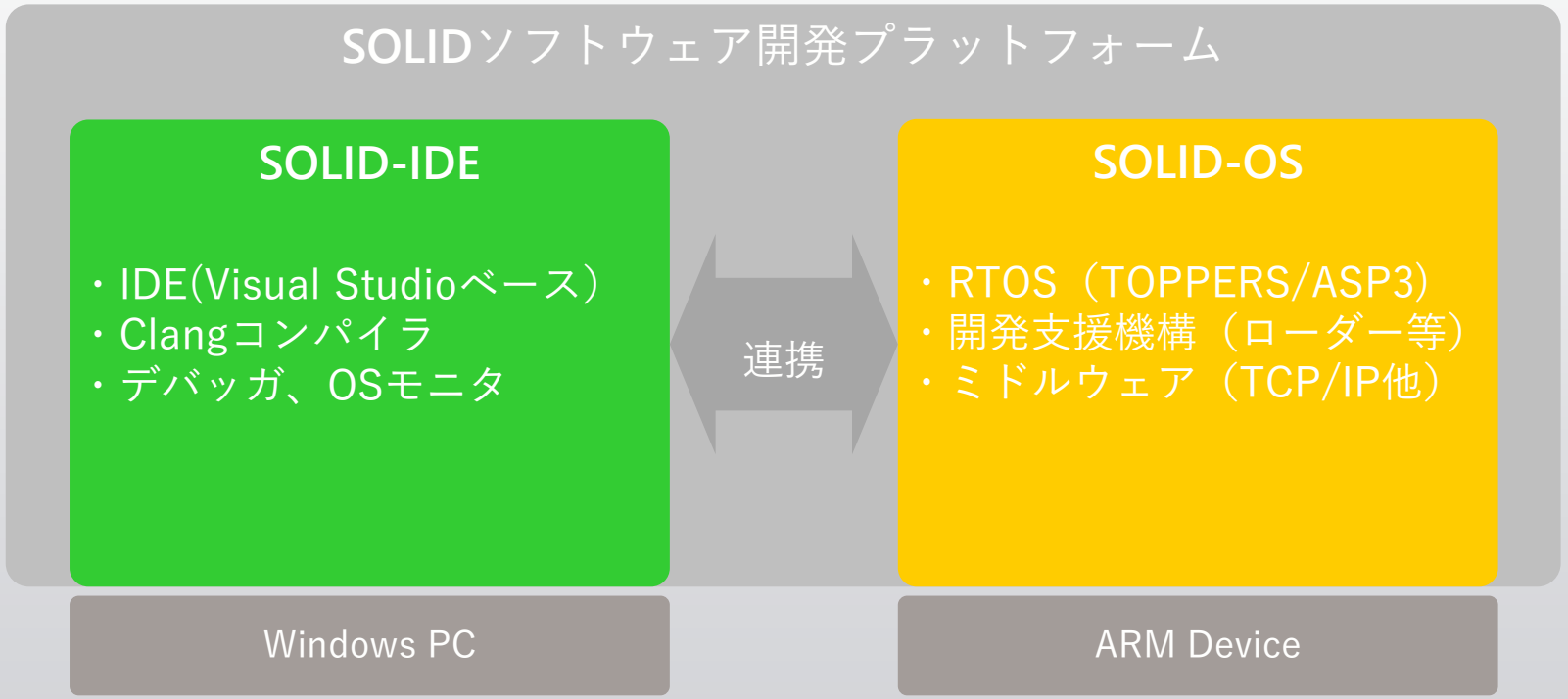
- 「ツール屋」であるKMCが、デバッガやコンパイラを提供してきた経験をもとに、開発者に”楽しく”, ”快適に”開発に取り組めるソフトウェア開発プラットフォームを提供する
- 開発者が本来のクリエイティブな業務に最大のパフォーマンスを発揮して「かしこく開発」し、また、不具合のモグラ叩きに翻弄されずに「スマートにデバッグ」する環境を提供する
- 組み込み機器の開発手順をシンプルに変える

SOLIDの構成

- SOLIDは組み込み用「リアルタイムOS」と「開発ツール」を一体化したソフトウエア開発プラットフォーム
  - リアルタイムOSには、TCP/IPなども付属する
  - 開発ツールには、IDEやコンパイラ・デバッガが含まれる

- SOLIDは、RTOS含んだSOLID-OSと、統合開発環境SOLID-IDEから構成される

SOLIDの構成


SOLIDソフトウェア開発プラットフォーム
SOLID-IDE
・IDE(Visual Studioベース)
・Clangコンパイラ
・デバッガ、OSモニタ
連携
SOLID-OS
・RTOS（TOPPERS/ASP3）
・開発支援機構（ローダー等）
・ミドルウェア（TCP/IP他）
Windows PC
ARM Device

SOLIDの構成
SOLID-OS

- SOLID-OSとして、名古屋大学を中心としたTOPPERSプロジェクトで開発されたオープンソースカーネルである**TOPPERS/ASP3**を採用します。
  - 組み込み機器として実績のあるμITRON 4.0仕様準拠
  - ASP3のティックレス仕様により実行効率・電力効率が良い割込み制御が可能
  - SOLID-OSとして
    - カーネル本体、プロセッサ依存部、BSPを提供
    - TCP/IP, Fileシステム、スクリプトエンジン等のミドルウエアも提供
    - ロイヤリティフリー

かしこく開発
SOLID-IDE

- 開発準備から性能評価までの作業をシンプルに変えるため、SOLID-IDEはユーザーインタフェースに定評のある**Visual Studioをベース**に独自開発しました。
  - コンパイルからデバッグまで、全てWindows上で作業できるので、「ビルド→転送→実行→デバッグ」、の手順がシンプル
  - Intellisense 機能（エディタでのコード補完）をはじめ、Visual Studio の多くの特徴的な機能が使える
  - Visual Studio Shellをベースに独自に開発したIDEは、ロイヤリティフリー
  - Clangコンパイラで検出したエラーを、IDE上で分かりやすく表示
  - 静的解析、動的解析ともに、問題箇所をIDE上に表示

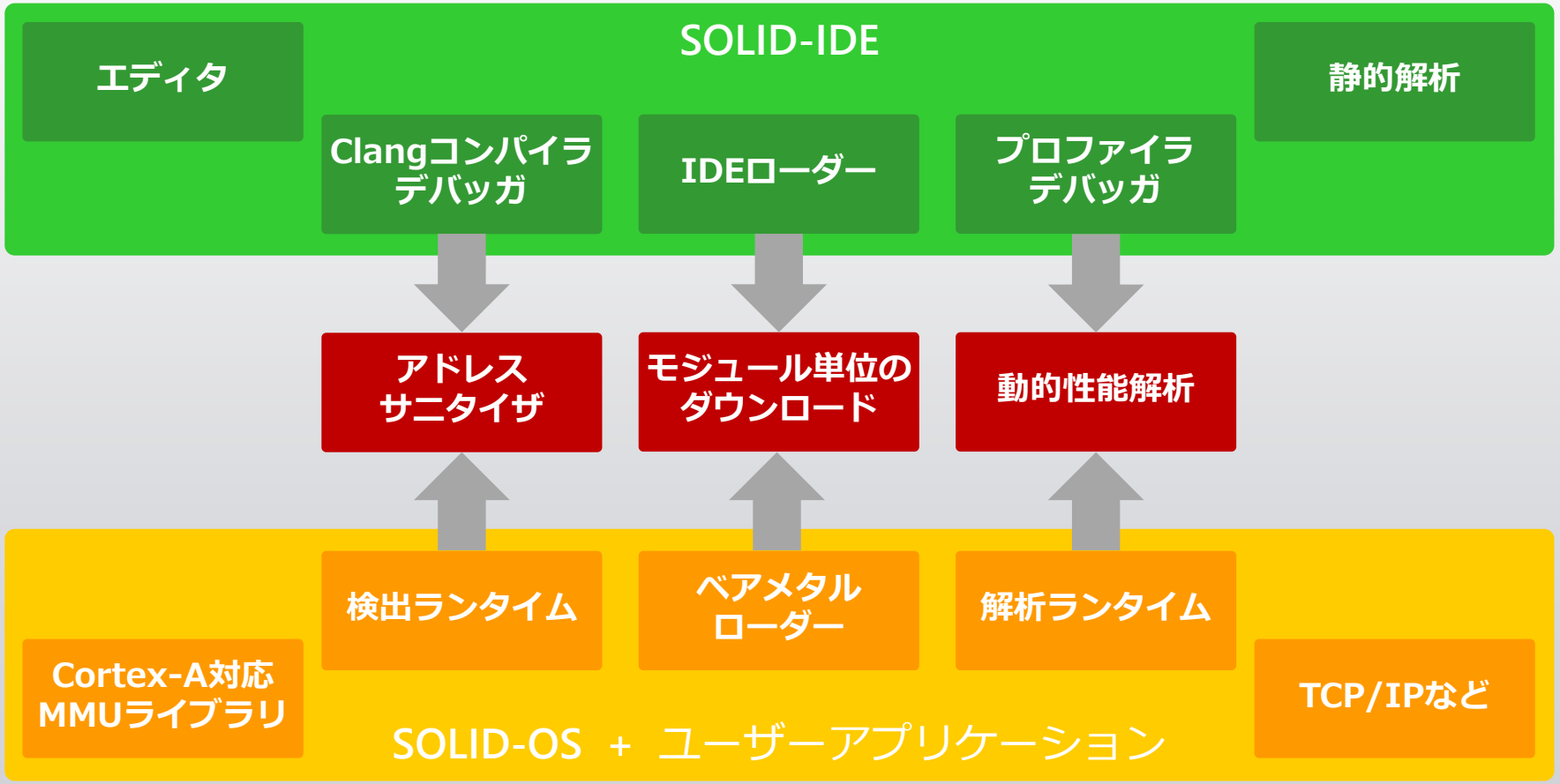
かしこく開発
Clangコンパイラ

- SOLIDプラットフォームでは、静的解析ツール、動的解析ツール機能を豊富に備えた**LLVM/Clangコンパイラ**を採用します。
  - LLVM/Clangは次世代のコンパイラとして、利用されはじめている
    - MacOS/iOS開発環境の標準コンパイラ、ARM Compiler 6、FreeBSDの標準コンパイラなど、多くの分野で標準的に利用されている
  - Clangコンパイラは、オプションや言語仕様拡張など、GCCコンパイラとの互換性が高い
  - ビルド時に静的解析ツールとしてClangを使用することにより、「未初期化変数の利用」「メモリリーク（解放もれ）パスの検出」などが検出可能
  - 実行時に「アドレスサニタイザ」などデバッガと連動した動的解析が可能

かしこく開発
Clangコンパイラ 静的解析

```
17  int main()
18  {
19      vector<int> vect;
20    char *str = (char *)malloc(100);
                    (1) Memory is allocated →
21    if (str == NULL) {
      (2) ← Taking false branch →
22      return -1;
23    }
24      vect.push_back(4);
        (3) ← Potential leak of memory pointed to by 'str'
25      vect.push_back(1);
26      vect.push_back(10);
27      vect.push_back(-3);
28      vect.push_back(1);
29      vect.push_back(-5);
30
31      print(vect);
32      sort(vect.begin(), vect.end());
33      print(vect);
34      sort(vect.begin(), vect.end(), greater<int>());
35      print(vect);
36
37      return 0;
38  }
```

スマートにデバッグ
SOLID-IDEとSOLID-OSの密結合



スマートにデバッグ
アドレスサニタイザ

組み込み初登場！アドレスサニタイザ

- アドレスサニタイザとは？
  - LLVM/Clang コンパイラのテスト支援機能で、メモリ破壊やリークなどを実行時に検出
  - iOSアプリケーション開発環境のXcodeで使えることで有名

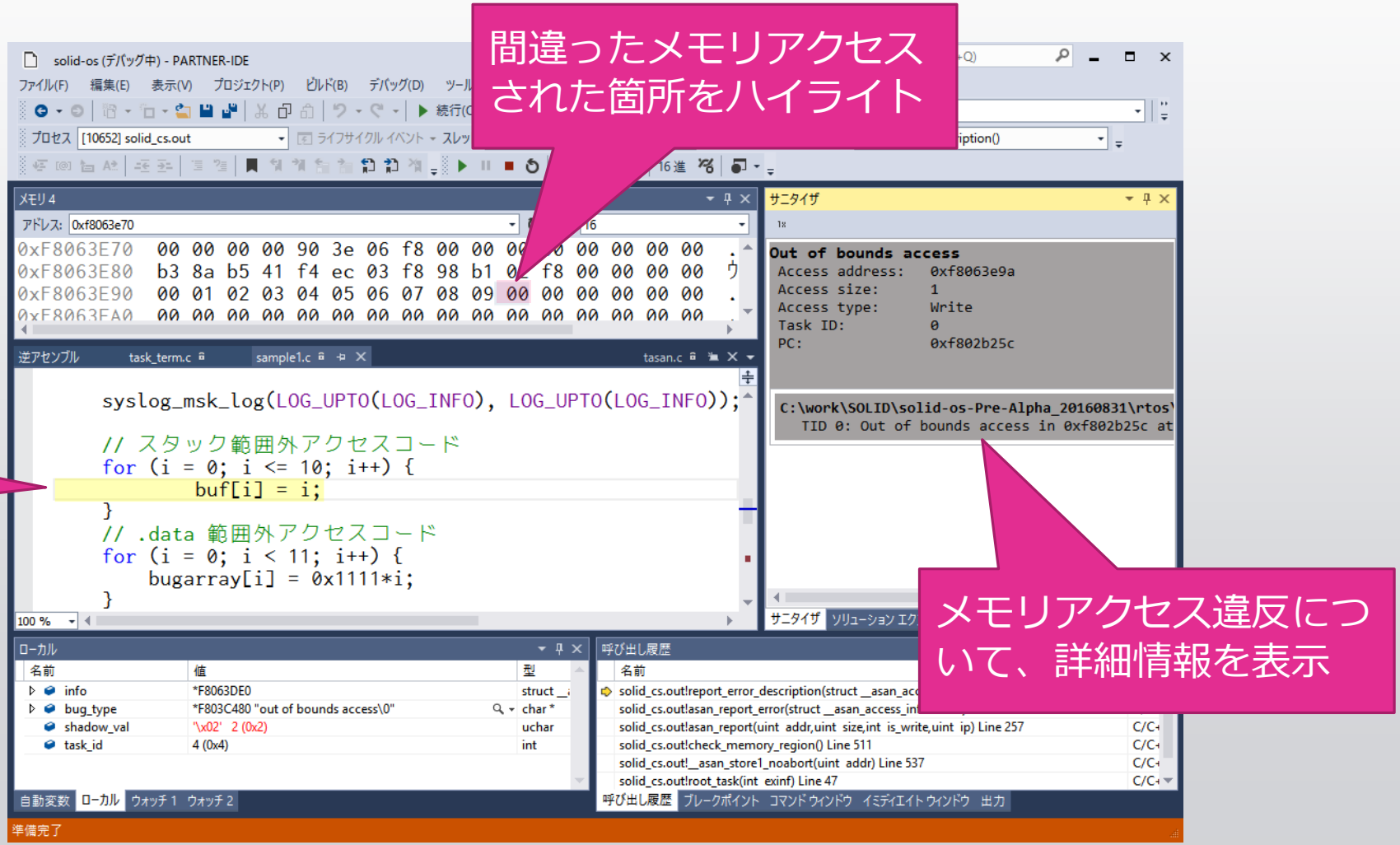
  https://developer.apple.com/xcode/jp/
- 簡単に使える
  - SOLID-IDEでビルド・実行モードを「アドレスサニタイザモード」に設定するだけ
  - 事前にバグがありそうな箇所などの検討必要なし
  - 実行するだけで、間違ったメモリアクセスを自動的に検出します

SOLID-OSとSOLID-IDEが連携するので、わかりやすく、簡単に使えます

スマートにデバッグ
アドレスサニタイザ

デモンストレーション

スマートにデバッグ
アドレスサニタイザ



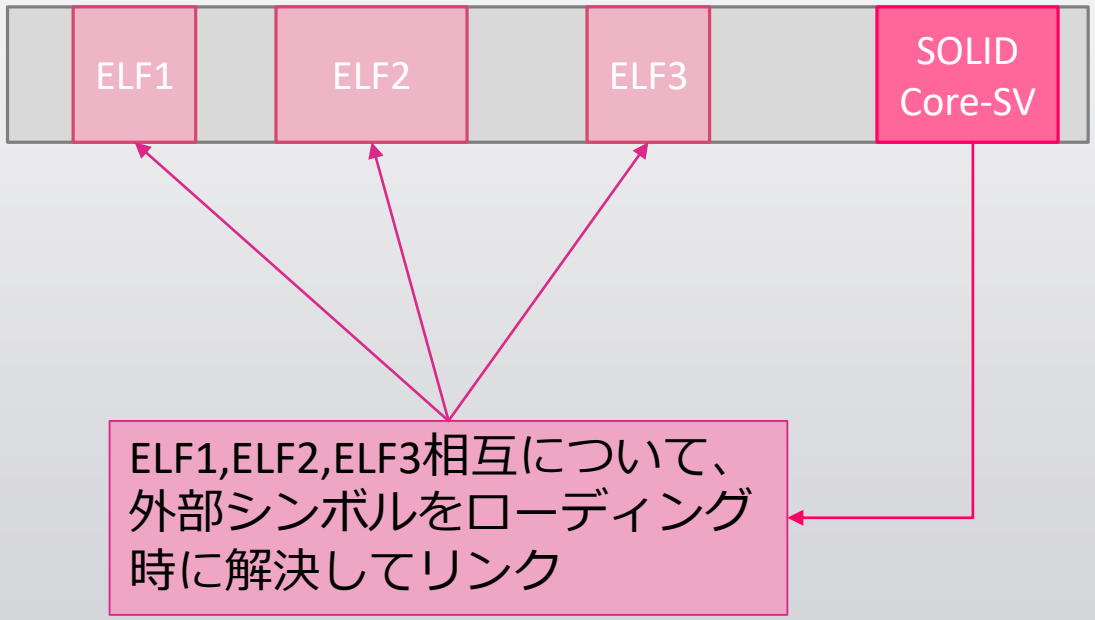
かしこく開発
シンプルベアメタルローダー

- RTOSやベアメタル環境では、複数拠点などでの分割開発時に、個別に分割したプログラム単位でのローディングや実行が煩雑になっている

- 新規にMMU対応のシンプルベアメタルローダーを開発
  - MMUは仮想アドレスを使うが、シンプルに使えるよう単一空間のみ
- 複数に分割されたプログラムでも、それぞれ個別に開発・ローディング可能
  - 分割ローディング時のアドレス解決機構を搭載
  - メモリの利用効率が低下しないよう、MMUを使ってアドレス割り付け
  - MMUのプロテクションを有効にし、デバッグ効率も向上
  - IDEと連携したローディング
  - アップデートなどの効率化

かしこく開発
シンプルベアメタルローダー

- 仮想アドレス空間にELFモジュールをロード
  - ロード元はROMなどでもよく、ファイルシステム必須ではない
  - ローディング時に、必要な領域をMMUで生成
- ELFモジュールは、アドレス固定でリンク
  - 仮想アドレス空間を使う事で、広い目のエリアをあらかじめ予約し、モジュール間の重なり防止
- 外部解決シンボルについては、ELFローダーがシンボル解決
  - ソース中に特定のキーワードをつける事で、ビルド環境が自動的に外部解決シンボルに設定



2017年リリース 対応プロセッサ

- ARM® Cortex®-A9 プロセッサに代表される、ARMv7アーキテクチュアのマイクロプロセッサを搭載したシステムを1st ターゲットとします。
  - 組み込み機器向けの汎用プロセッサおよびSoCにおける採用例が多い
  - 性能 vs 消費電力　に最適化されたコア
  - MMUによるメモリプロテクションや仮想アドレスの使用が可能
  - 標準となるハードウエア（評価ボード）用のBSPを提供
  - ハードウエアがなくても、シミュレータですぐに動作確認が可能

*今後　ARM Cortex-Mx プロセッサも対応計画中

2017年リリース

2017年度　1Q提供予定

http://www.kmckk.co.jp/SOLID/

デバイス・マイコンベンダー様と、ボードとセットにした開発キットを企画中

enjoy Development

ありがとうございました

Kyoto Microcomputer Co.,Ltd.

http://www.kmckk.co.jp

KMG Inc.

http://www.kmg-inc.jp