ソフトウェアテストシンポジウム2004
デバッグ丸かじり

# 組み込み開発における デバッグテクニック

横河ディジタルコンピュータ株式会社
開発環境統括本部
武井　千春

## 組み込みシステムのデバッグ手法を解説すると共にこれからの課題を説明

- モニターデバッグ
  - その機能と特徴
- ICEデバッグの特徴
  - フルICE
  - JTAG ICE
- 組み込みシステムのデバッグ環境の問題点
  - ホスト上の有用な多くのツールとのつながり
  - 上流ツールとの連携
  - RTOS環境でのデバッグ
  - 評価機能、自動テスト

## モニタデバッグ

- シリアル接続： 簡単、低速
- イーサネット接続： 高速、ドライバ／スタックを用意しなければならない。
- 最近は、USBでどうか ： 標準実装、高速

ターゲットシステムとPCを接続。ターゲットプログラムの一部に小さなモニタプログラムを置く。

## モニタデバッガの制限

- メモリ保護
  - 不正アクセスからメモリを保護できない
  - 暴走して壊れれば再ダウンロード
- トレース機能
  - どんな過程を経て、実行されたかが分からない
    - 「不正な処理をしたので・・・レジスタとスタックは ・・・」と表示されても何が原因かほとんど追えない
- イベント検出機能
  - メモリアクセスなどのデータ値によるブレークなど
- 評価機能
  - カバレジ、時間性能測定など

## モニタプログラム の機能

- ホストデバッガとのコマンドの送受。
- 実行開始
  - 指定PCからユーザプログラムを実行
- ブレーク
  - 強制ブレーク：NMI相当の割り込みからモニタへ
  - ブレークポイント設定：ブレーク用命令への書き換え
  - 一般にアクセスブレークは不可
- レジスタのアクセス
  - ブレーク時にセーブした、全レジスタの参照／変更
- メモリのアクセス
  - ターゲットシステムのメモリの参照／変更

## ICEデバッグ

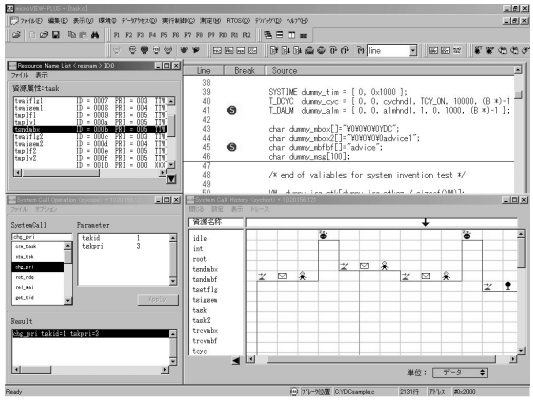
- 典型的実機デバッグの構成
  ターゲットマイコンの代わりに
  エミュレータを接続

ターゲットのマイコンの動きを全てエミュレーション

- 解析能力は最も高い
- モニタデバッグの持つ機能の全てに加え
  - トレース機能
  - イベント検出機能
  - 性能評価機能　など

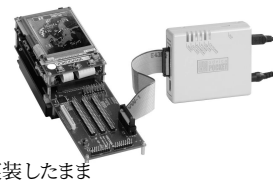
## ホスト側のデバッガと組み合わせて

- C/C++のソースレベルデバッグ
  - ソース行単位の実行
  - ソース上でのブレーク設定
  - 変数アクセスの結果によるトリガ／ブレーク
- RTOS上での タスクデバッグ
  - 資源表示
  - タスクトレース表示
  - システムコールの発行
- 性能機能評価など
  - 時間測定
  - プロファイル測定
  - カバレジ測定



## JTAGベースデバッグ

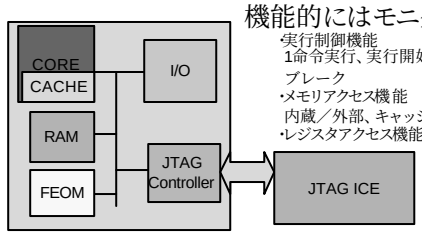
- テスト用のJTAG端子をデバッグ用のICE端子として利用



- マイコンはターゲットに実装したまま
  - 外乱に強い。ノイズ, 遅延など。
  - 接続が容易。20ピン。
  - 高速動作マイコンに強い。マイコンは自分の動作スピードで動作。JTAGラインは別なスピードで動作

## ターゲットマイコンの中にデバッグ支援機能を実装

- そもそもはマイコンのテスト機能端子
  機能的にはモニタ相当
  ・実行制御機能
  1命令実行、実行開始、強制ブレーク
  ・メモリアクセス機能
  内蔵／外部、キャッシュ
  ・レジスタアクセス機能




スキャンチェーンがつながってれば、
複数のマイコンを制御できる

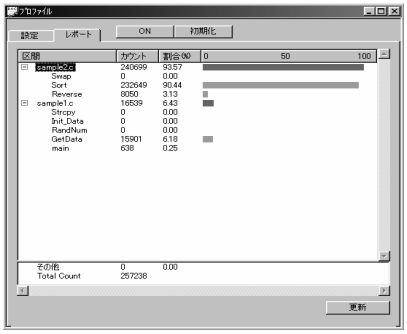
## JTAG ICEデバッグの特徴

- 最近の高速・高性能マイコンにはほとんど実装
  - 一部機能を追加するなどして各社別の呼び方
    - UDI,SDI,DSUなどなど
- 安定動作
- 低価格なICEが利用できる
  - JTAGラインの変換のみで基本はOK
  - 大勢の技術者が利用できる
- オンチップトレース機能への拡張性もあり
  - EJTAG,ETM,HUDI,NEXUSなど
  - シリコン内部にトレース収集、出力機能を実装

## 単純なデバッグでは済まない組み込みシステム

- 性能評価
  - 容量と性能は有ればあるだけ使うアプリが登場
    - 10年前 HD100MB/33MHz 今 HD100GB/3GHz
      - 容量1000倍、スピード100倍
  - 性能余力がどれだけあるか？
    - OSのオーバーヘッドは？
    - タスクを追加できる余力は？
  - ボトルネックはどのモジュールか？
    - 80%の時間は20%の モジュールが消費？いや数%。
    - 改善すべき20%を見つける手段
    - 実機上でのprofile相当の機能
- それらが、実際の制御、通信、などを行なっている状態で評価されなければならない
  - シミュレータではほとんど不可能

## プロファイル測定表示例



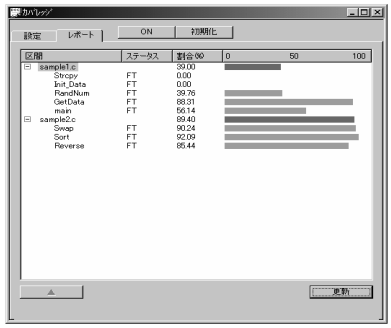

## 性能評価の方法と問題点

- 関数へのプロービング
  - 入り口、出口の時間から評価。
  - 割り込みの除去が困難
- 実行アドレスのサンプリング
  - 非常に短周期に実行アドレスをサンプリングして関数、モジュールごとの
    実行割合を測定
  - サンプリング間隔以内の変化が追えない
  - 実行アドレスをサンプリングする仕組みが困難に
  - キャッシュ上の実行が追えない
- RTOSとの協力が不可欠
  - タスクスケジューリング、割り込みハンドラ のスケジューリング時点で次官情報は収集。
    これをデバッガに渡す仕組みを構築。
  - 少なくともタスク単位の性能評価を実現

## 品質評価

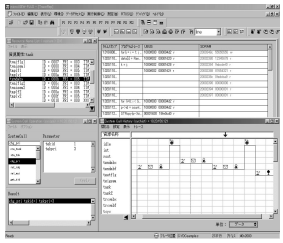
- 出荷後の回収騒ぎ
  - テストケースの漏れ
  - モジュール評価、関数評価の漏れ
  - 一度でもやってみれば発生する問題が・・・
- 世の中はいつも時間と十分性のせめぎあい
  - いかに十分性を客観的評価値として評価できるか？
- ハードウェアカバレジテスト
  - C0、S0は比較的容易。C1が困難
- コンパイラとの連携
  - 関数、着目行にプロービングコードを埋め込み
  - 特定アドレスに実行時にアクセスする事を外部で検出

## カバレジ表示例




## RTOSサポート

- 視覚的なタスク遷移トレース
  - プライオリティインバージョンなども判断しやすい
- タスク状態の表示
  - システムがハングアップ？
  - 止めて見るとアイドルルーチンにしかいない
  - タスクの状態が必須
- フラグ、メッセージ キューなどのRTOS資源の表示
- 進まないタスクにフラグを立ててやりたい
  - システムコールの代替発行



## 今後への課題

- 上流ツールとの連携
  - UMLなどの設計ツールと自動コード生成ツールが生成するモジュールを設計ツールの環境で実機デバッグ
- 自動テスト
- マルチコアデバッグ
  - 携帯電話に32ビットRISCマイコンが2個載っている現状。近い将来 2個→4個、それ以上。
  - これらの協調デバッグが必要。
  - 同一シリコン上の複数プロセッサのデバッグ
  - 同一ボード上の複数のプロセッサのデバッグ
    - JTAGスキャンチェーンがつながっている場合
      - 単一ICE
    - JTAGスキャンチェーンがつながっておらず 独立したプロセッサ の場合
      - 複数ICEの協調