インサーキット・エミュレータを利用したOSデバッグ機能の実現

1H－3

橋本 一也＊　酒井 貴＊　福田 政広＊＊　宗 雅俊＊

＊日本電気アイシーマイコンシステム(株)　九州LSI開発センター
＊＊日本電気(株)　ULSIシステム開発研究所

1　はじめに

近年、マイコン組み込みシステムにおいて、プログラミングの容易さや分散処理による効率化のために、リアルタイムOS(以下OS)を用いることが多くなってきている。これに伴い、OS上のタスクを効率よくデバッグできるタスク・デバッガが開発されてきた。

従来、このタスク・デバッガは、モニタやデバッガ用のタスクで実現されていたが、これらの方式では、ユーザによる構築が必要であったり、メモリの制約を受ける等の欠点が存在していた。

本論文では、これら欠点を解決すべく、インサーキット・エミュレータ(以下IE)を利用したタスク・デバッガを提案し、その実現にあたっての必要機能を洗い出し、それらについて考察する。

2　タスク・デバッガの機能と特徴

本論文では、当社製タスク・デバッガについて論じる。

本タスク・デバッガは、特定のタスクに着目してデバッグを行なうツールであり、ターゲット・システムは、デバッグ中でも動作し、外部割り込みを受け付けることができる点が特徴である。

また、今回対象としたタスク・デバッグ機能を以下に定義する。

- システムコールの発行
- 特定タスクのブレーク / 実行
- 特定タスクのレジスタ表示 / 設定
- OS資源(オブジェクト状態 / キュー)表示
- システムコール / 割り込みトレース

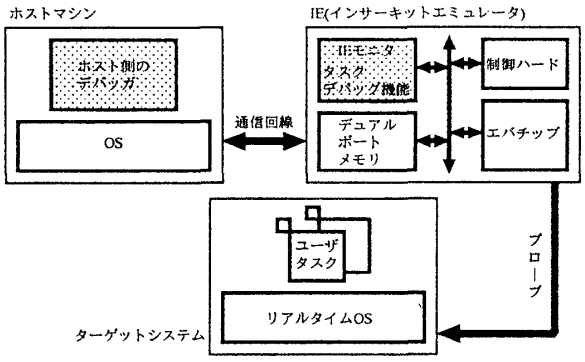
以降では、これらのタスク・デバッガの機能を実現するためのシステム構成と実現方法についてを述べる。

3　システム構成

本タスク・デバッガのシステム構成を図1に示す。

本タスク・デバッガでは、デバッグ処理中でもターゲットを動作させる必要があるため、デバッグ処理のうち高速処理を必要とする部分は、IEモニタ(IE内にありターゲットがブレークした時にエバチップにより実行されるプログラム)で行ない、それ以外の処理は、ホスト側のデバッガで処理を行なうようにし、できるだけ高速にデバッグ処理ができるように考慮した。

図1: システム構成



4　必要機能と問題点

IEにより本タスク・デバッガを実現する上での必要機能と問題点を以下にまとめる。

(1) IEモニタによるデバッグ処理方法

本タスク・デバッガでは、デバッグ処理を行なっていない間、ターゲットを常に実行させていなければならない。このため、ホスト側のデバッガがIEモニタへ処理依頼(IEモニタ・コマンドを発行)する際には特別なプロトコルが必要となる(a)。

(2) システムコールの発行

OS上のタスクに対してデバッガ側からOSのシステムコールを発行する際には、ターゲット上にデバッガ用のコンテキストが存在しないため、あるタスクのコンテキストを借りて、システムコールを発行しなければならない。この時、コンテキストを借りたタスクの優先度が低ければ、システムコールの応答が悪くなり、また、コンテキストを借りたタスクによっては、発行できないシステムコールが存在するという問題点がある(b)。

(3) 特定タスクのブレーク / 実行

タスクをサスペンド、および解除するシステムコールを発行して実現する。

(4) 特定タスクのレジスタ表示 / 設定

メモリ上のタスク・コンテキスト中に格納されているレジスタの値をIEからメモリ・アクセスすることによって実現する。

Implementation of OS debugger using In-circuit Emulater.

Kazuya Hashimoto*, Takashi Sakai*,
Masahiro Fukuda**, Masatoshi Sou*
*NEC IC Microcomputer Systems, Ltd.
**NEC Corporation.

(5) OS 資源 (オブジェクト状態 / キュー) 表示
(4) と同様にメモリ・アクセスして実現する。ただし、複数のオブジェクトの情報を表示する場合は、データの整合性を維持するためにターゲットをブレーク状態 (割り込み禁止状態) にしたまま一斉に全ての情報を読み出す必要があり、このため割り込み禁止時間が長くなるという問題点がある (c)。

(6) システムコール / 割り込みトレース
このトレースは、システム全体の時間的な動作を把握するためのものであるので、トレースデータの収集処理はできるだけ高速に行なう必要がある (d)。

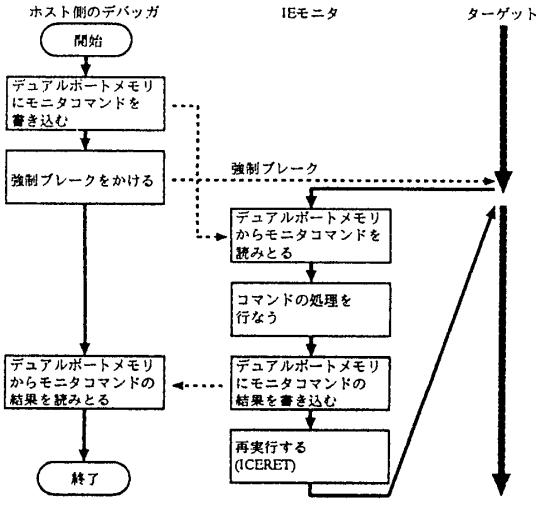
5 実現方法

前章であげた問題点に対し、(a) については、IE モニタ・コマンド発行プロトコルを導入し、(b)、(c) については、フリーズ・モード、また (d) についてはトレース用フック機能を OS 側に追加することにより解決した。以下では、その実現方法を具体的に説明する。

5.1 IE モニタ・コマンド発行プロトコル

ホスト側から IE モニタ側へのコマンド発行時には、一時的にターゲットをブレークさせ IE モニタを実行状態にする必要がある。これは、IE が持つ強制ブレーク機能によって実現し、コマンド実行後は、再びターゲットを実行させる (図 2)。

図 2: IE モニタ・コマンドの受け渡し方法



5.2 フリーズ・モード

OS 資源表示とシステムコール発行における問題を解決するために、全てのユーザ・タスクを実行状態にしない状態にするフリーズ・モード機能を OS に追加した。この状態は、OS が常に idle タスクをディスパッチし、idle タスクのみを実行状態にさせることによって創り出すことができる。

この機能を利用すれば、OS 資源表示は、フリーズ・モードにした後、表示処理 (メモリ・ダンプ) を複数回に分けて処理し、システムコール発行は、フリーズ・モードにした後、idle タスクのコンテキストでシステムコールを発行することで実現できる。

5.3 トレース用フック機能

トレースは、以下のイベントに対して行なう ([1])。

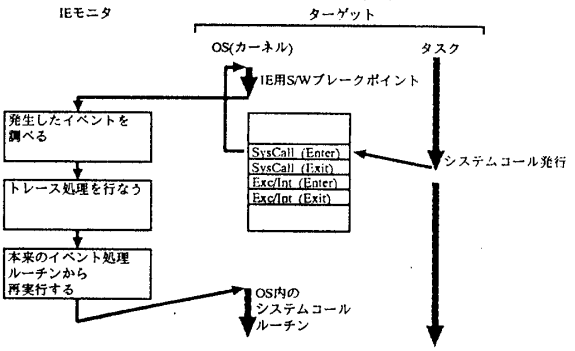
- システムコールの入口
- システムコールの出口
- 例外 / 割り込みハンドラの入口
- 例外 / 割り込みハンドラの出口

上記イベントが発生した時点でトレース・データ収集のため IE モニタに制御を移動させる機能が必要となり、OS にフック機能を追加した。

例えば、上記イベントの処理コードのアドレスを OS 内のテーブルに登録しておき、そのイベントを実行する際は、該当のテーブルの値で間接ジャンプするという方式である。この方式では、テーブルの値を、IE の S/W ブレークポイントを埋め込んであるアドレスに書き換えることにより、IE モニタへのフックが可能となる。

イベント発生時、IE モニタに制御が移動すると、IE モニタは、そのイベントに対してトレース処理を行なう。処理が終ると、IE モニタは、本来のイベント処理アドレスから再実行を行なうようにする (図 3)。

図 3: トレース方法



6 まとめ

本論文では、タスク・デバッガに対して IE を利用することを提案し、それを実現するための方法を検討した。

これを基に、我々は、V53(NEC 製 16 ビット MPU) 用リアルタイム OS 用タスク・デバッガの試作を行ない、技術的には実用レベルまで達することを確認できた。

今回の方式では、ソフトウェア (IE モニタ) により制御を行なったため、オーバヘッドがまだ生じている。今後は、このオーバヘッドをできるだけ縮小させるために、タスク・デバッグ機能を IE のハードウェアにより実現させる方法を検討していきたい。

参考文献

[1] 大管他, "実時間処理システムの動作解析 / 評価ツール", 情処第 42 回全国大会講演論文集