リアルタイムオペレーティングシステムーI-TRON

坂村 健
（東京大学理学部・情報科学科）

1. I-TRON設計の思想

TRON[1][2][3][4][5][6]とは、32ビット幅ノイマン型のアーキテクチャの将来のVLSI化を考慮し、80年代後期に焦点を合わせて設計された、レンジの広いマイクロプロセッサ用リアルタイムマルチタスクアーキテクチャである。

TRONの対象とするレンジは広いため、応用をもとに、産業用に使われる組み込みシステム用I-TRON、主としてパーソナルコンピュータ用を考慮したB-TRON、家庭での利用を考慮したH-TRON等々、いくつかのシリーズよりなる。

I-TRON（Industrial-TRON）[7]は、特に産業用の組み込みシステムへの応用を考えており、次のような特徴を持つ。

① I-TRONは、核の機能として、タスク関連操作、同期・通信機能、割り込み処理機能、例外処理機能、メモリ管理機能を持つ。そして、その他に基本入出力処理、基本ファイル処理機能、基本デバッグ機能を持つマルチタスクOSである。

② I-TRONは、効率を最大にすることを考慮して設計されている。複数のプロセッサ上に、I-TRONをのせる場合、バーチャルマシン的アプローチは取らない。バーチャルマシンアプローチは、効率を最も重視する組み込み用リアルタイムOSには適さない。アダプティブという思想に基づき、I-TRONをのせるチップ上で最もよい実行効率が出るよう、各プロセッサごとにI-TRONを作る。

現在、I-TRON実現を考慮しているチップは、モトローラ68000系と、インテルiAPX86系である。本論文では、対象マシンにとらわれず、I-TRONに共通する設計思想と機能について概説した後、モトローラ68000系チップ上に実現したI-TRON／68K[8]、インテルiAPX86系チップ上に実現したI-TRON／86[9]のアーキテクチャについて述べ、最後に実験的インプリメンテーションの結果について概説する。

2. I-TRONの構成

将来のVLSI化を考慮したため、I-TRONの構成を大きく二つに分けた。一方はI-TRONチップ核であり、他方はI-TRONチップ核周辺である。

将来I-TRONを内蔵したマイクロプロセッサ（これをTRONチップという）を開発する場合、チップに内蔵されるOS部分つまり基本インストラクションとなるべき部分が、I-TRONチップ核である。

チップ化とROM化とでは考慮すべき点が若干異なる。そのため、ROM化の際には当然含まれてしかるべき機能でもチップ化には向かないという部分（チップ化するには重いが機能として持ちたい部分や、比較的高速性を必要としない部分等）をI-TRONチップ核周辺としてカバーすることとした。

I-TRONチップ核は、タスク関連操作、同期・通信機能、割り込み処理機能、例外処理機能、メモリ管理機能から構成される。システムコールは表1に示す通りである。

I-TRONチップ核周辺は、基本入出力管理機能、基本ファイル管理機能、基本デバッグ機能、その他から構成される。

基本入出力機能としては、コンソール、モデムを対象とする直列インタフェース、セントロニクス方式の8ビット並列入出力、GPIBなど、現在標準的に使われているものすべてを用意する予定である。

基本ファイル機能の特徴としては、高速メモリと二次記憶を同一次元で見ることができるファイル構成を取り、データの物理的所在をシステムジェネレーション時にユーザの要求に合わせ自由に設定できることがあげられる。

基本デバッグ機能としては、基本的なモニタのためのコマンドインタプリタ、ブレークポイントの設定といった基本機能から、バリュートレース機能までを持つ。

その他の機能として、時刻表現の間での変換、日付計算といったタイマ関連補助機能がある。

3. 適応化

TRONは適応化を重点に置いて設計されたアーキテクチャである。I-TRONは一つの標準的モデルではあるが、システムジェネレーション時の自由度は大きい。不必要なモジュールの取りはずし、前述したファイルの物理的割り付け、タスク切り換え（コンテキストスイッチ）時の退避レジスタの指定などが可能である。

ユーザの必要とする最小限の大きさで、最高の実行効率が実現できるよう設計されている。

適応化については、個々のプロセッサ上の実現の方法と深く関る問題であるので、後述するI-TRON／68K、I-TRON／86の部分で詳述する。

4. I-TRON／68K

4-1 I-TRON／68Kの特徴

I-TRON／68Kの設計の要旨を68Kのアーキテクチャの特徴とともに述べる。

- 68Kには32ビットの汎用レジスタが15個あり、レジスタ数が多い。
  - → レジスタを最大限に活用するため、I-TRON／68Kではスケジューリングの際に、タスクのTCB(Task Control Block)に退避されるレジスタを指定することができるようにした。詳細は後述する。
- 68Kではユーザモードとスーパバイザモードが分離している。
  - → タスク部分のプログラムはユーザモードで動き、I-TRON自体や割り込みハンドラなどの非タスク部分のプログラムをスーパバイザモードで動かす。このようなモード切り換えによって、メモリ管理のハードウェアがない場合にも、ある程度のデータ保護ができるようにした。その結果、ユーザにとっても、タスク部分のプログラムと非タスク部分のプログラムの区別がはっきりする。
- 68Kには例外割り込みの種類が多い。
  - → 例外割り込みはバスエラー、0除算など、プログラム実行にともなって発生

システムコール名		機能
1. タスク関連操作		
Cre_Tsk	Create_Task	タスクを生成する
Sta_Tsk	Start_Task	タスクを起動する
Del_Tsk	Delete_Task	タスクを削除する
Def_Ext	Define_Exit_Routine	終了時処理ルーチンを定義する
Exi_Tsk	Exit_Task	自タスクを正常終了する
ExiD_Tsk	Exit_and_Delete_Task	自タスクを正常終了後、削除する
Abo_Tsk	Abort_Task	自タスクを異常終了させる
Ter_Tsk	Terminate_Task	他タスクを強制終了させる
Che_Pri	Change_Task_Priority	タスク優先度を変更する
Rot_RQ	Rotate_Ready_Queue	タスクレディキューを操作する
TCB_Adr	Get_TCB_Address	タスクTCBアドレスを得る
Sus_Tsk	Suspend_Task	タスクを強制待ち状態へ移行する
Res_Tsk	Resume_Task	強制待ち状態のタスクを再開する
Sle_Tsk	Sleep_Task	タスクを待ち状態へ移行する
Wak_Tsk	Wakeup_Task	待ち状態のタスクを起床させる
CWak_Tsk	Cyclic_Wakeup_Task	タスクを周期起床する
Can_CWak	Cancel_Cyclic_Wakeup_Task	タスクに出された周期起床要求を無効にする
2. 同期・通信機能		
Cre_Flg	Create_EventFlag	イベントフラグを生成する
Del_Flg	Delete_EventFlag	イベントフラグを削除する
Set_Flg	Set_EventFlag	イベントフラグをセットする
Wai_Flg	Wait_for_EventFlag_to_be_Set	イベントフラグの条件満足を待つ
Flg_Adr	Get_EventFlag_AccessAddress	イベントフラグアクセスアドレスを得る
Cre_Sem	Create_Semaphore	セマフォを生成する
Del_Sem	Delete_Semaphore	セマフォを削除する
Wai_Sem	Wait_on_Semaphore	セマフォのP命令
Sig_Sem	Signal_Semaphore	セマフォのV命令
Sem_Adr	Get_Semaphore_AccessAddress	セマフォアクセスアドレスを得る
Cre_Mbox	Create_Mailbox	メイルボックスを生成する
Del_Mbox	Delete_Mailbox	メイルボックスを削除する
Sen_Mes	Send_Message	メッセージを送信する
Rec_Mes	Receive_Message	メッセージを受信する
Mbox_Adr	Get_Mailbox_AccessAddress	メイルボックスアクセスアドレスを得る
3. 割り込み処理機能		
Def_IntH	Define_Interrupt_Handler	割り込みハンドラを定義する
Ret_IntH	Return_from_Interrupt_Handler	割り込みハンドラから復帰する
Dis_Int	Disable_Interrupt	割り込みを禁止する
Ena_Int	Enable_Interrupt	割り込みを許可する
Fet_Data*	Fetch_Data	割り込みハンドラ内で使うDSをセットする
Dev_No*	Get_Device_NO	処理中の割り込みを発生したデバイス番号を知る
4. 例外処理機能		
Def_ExH	Define_Exeption_Handler	例外処理ハンドラを定義する
Ret_ExH	Return_from_Exception_Handler	例外処理ハンドラから復帰する
5. メモリ管理機能		
Cre_Pool	Create_Memory_Pool	メモリプールを生成する
Del_Pool	Delete_Memory_Pool	メモリプールを削除する
Get_Blk	Get_Memory_Block	メモリブロックを獲得する
Rel_Blk	Release_Memory_Block	メモリブロックを解放する
Pool_Adr	Get_MemoryPool_AccessAddress	メモリプールアクセスアドレスを得る
6. その他		
Set_Time	Set_Time	システムクロックを設定する
Get_Time	Get_Time	システムクロックの値を読み出す

注) *印の付いたシステムコールは、I-TRON／86固有のシステムコールである。

〔表1〕I-TRONチップ核システムコール一覧表

する割り込みである。これに対する処理ルーチンはタスクごとに指定することができるようにした。すなわち、例外割り込みが発生すると、I-TRONは現在実行中のタスクを調べ、そのTCBから例外処理ルーチンのアドレスを読み出し、そこへジャンプするようにした。例外割り込みの処理はそれほど高速性を要求されないので、使いやすさの方を重視した。

- 68Kには7つの割り込み優先度があり、ステータスレジスタと一体化して管理できる。また、周辺装置に割り込みベクトルを送る機能がないときは、自動ベクトル方式をとる。
  - → 割り込みハンドラ中でさらに優先度の高い割り込みがかかることがあるため、割り込みハンドラのネストを許した。また、自動ベクトルを用いる場合はポーリングが不可欠なので、割り込みハンドラがそれに対処できるようにした。詳細については後述する。

4-2 I-TRON／68Kにおける適応化
（コンテキストスイッチ時のレジスタ退避）

従来のリアルタイムOSでは、ディスパッチャが動くごとに内部レジスタすべてがTCBの中に退避される。これは既存のマイクロプロセッサの内部レジスタが少ないことによる必然的な措置であった。しかし、68Kは32ビットの汎用レジスタを15個持ち、人間がアセンブラでプログラミングした場合に、1つのタスクの中ですべてのレジスタを使うことは少ない。そこで、I-TRON／68Kでは、スケジューリングの際にタスクのTCBに退避されるレジスタをシステムジェネレーション時に指定できるようにした。これによって、不必要なレジスタの退避時間やメモリのオーバーヘッドを最小限にすることが可能となった。

また、タスク間の共通変数として、退避されないレジスタを利用することができるので、タスク間での通信を高速化できる。8MHz の68Kでは、レジスタ間の転送が4サイクル（0.5 マイクロ秒）に対して、絶対アドレッシングによるレジスターメモリ間の転送は20サイクル（2.5 マイクロ秒）になるので、レジスタを利用する効果は大きい。

4-3 割り込み処理

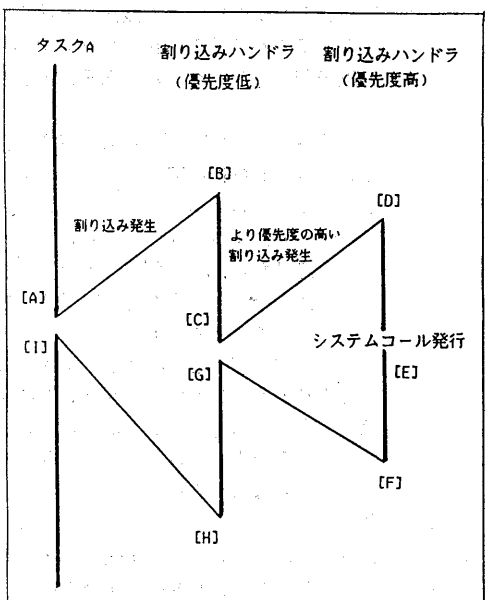
割り込み要因、割り込みハンドラ、実際に割り込み処理を進めるタスクの3つが1対1に対応していることが使いやすさの点から望ましい。このためには、割り込みを発生した周辺装置が固有の割り込みベクトルを送る必要があり、68K用に設計された周辺チップにはこの機能がある。

しかし、それ以外の周辺装置、特に MC6800 用の周辺チップを使うと、割り込みベクトルを送る機能がないので、自動ベクトル方式を取らざるを得ない。

自動ベクトル方式では1つの優先度に1つのベクトルしか割り当てられていないので、複数の割り込み要因を1つの割り込みハンドラで扱い、1つの割り込みハンドラから複数のタスクが起動できなければならない。

このため、割り込みハンドラにはかなり柔軟な機能が要求され、その中で使うシステムコールには制限を設けないことにした。つまり、待ち状態に入るものを除いたすべてのシステムコールが割り込みハンドラの中で書けるようにした。

ただし、割り込みハンドラの中でディスパッチャが動くと、割り込みハンドラがネストしたような場合に問題が起きる。タスクAの実行中にある優先度の割り込みがかかりその処理中に、より高い優先度の割り込みがかかった場合のプログラムの流れを図（割り込み処理のネストとスケジューリング）に示した。



［割り込み処理のネストとスケジューリング］

もし割り込みハンドラの中で発行されたシステムコール（[E]点）でスケジューラが動作すると、割り込みハンドラの残りの部分（[E]－[F]間と[G]－[H]間）がタスクAの一部として実行されてしまう。したがって、すべての割り込み処理が終了する[H]点までスケジューラを動作させてはならない。そこで、スーパバイザモードからユーザモードに遷移する直前にスケジューリングをするようにした。ユー

ザスタックとスーパーバイザスタックが別になっているという68Kの特徴を生かし、これを実現した。

5. I-TRON／86

5-1 I-TRON／86の特徴

I-TRON／86の設計の要旨を86のアーキテクチャの特徴とともに述べる。

- 86ではシステムコール呼び出しに使う内部割り込み命令（INT 命令）に256通りのベクトル番号を与えることができる。
  - → 割り込みハンドラで発行されるシステムコールや、高速性を要求されるシステムコールについては、INT命令のベクトル番号を別にした。これによって、I-TRON／86へのエントリーポイントを変え、高速な応答ができるようになった。それ以外のシステムコールについては、すべて同じベクトル番号の INT命令で起動され、AX レジスタに置かれた機能コードによってシステムコールを区別する。
- 86では割り込みコントローラ（8259）を接続することによって、割り込みを起こした周辺デバイスから直接割り込みベクトルを送ることができる。
  - → 割り込みコントローラ(8259)の使用をI-TRON／86動作の前提とし、これをサポートするシステムコール Disable_Interrupt, Enable_Interrupt, Get_Device_NO を設けた。また、割り込みコントローラを使った場合、割り込み終了時にコントローラに対して EOI (End Of Interrupt) コマンドを送る必要があるが、これは割り込みハンドラから戻るときに自動的に送られるようにした。その結果、ユーザが直接割り込みコントローラを操作する必要はなくなった。
- 86はセグメント値(16ビット) * 16 + オフセット値(16ビット)の形でメモリを参照し、プログラムやデータをセグメント単位で管理するアーキテクチャとなっている。
  - → すべてのアドレス空間を有効に使うため、タスクや割り込みハンドラには、それぞれ異なったセグメントを割り当てられるようにした。詳しくは後述する。

5-2 I-TRON／86 のセグメント管理

86上で大きなプログラムを動かす場合、セグメントの扱いが問題となる。I-TRON／86では、コンテキストスイッチの時に CS,DS レジスタの値も自動的に切り換えるため、タスク毎に異なったコードセグメントやデータセグメントを使うことを可能とした。個々のタスクの使用する CS,DS の値は Create_Task のシステムコールで指定するようにした。また、終了時処理ルーチン、例外処理ルーチンについても、もとのタスクとは別のコードセグメントにすることができるようにした。

I-TRON／86では割り込みに対する応答を速くするため、割り込みハンドラがI-TRONの介入なしに直接起動するように設計した。しかし、割り込みハンドラの中でそのままメモリ操作を行なうと、割り込まれたタスクのデータセグメントが対象となってしまう。これを避けるために Fetch_Data システムコールを設け、割り込みハンドラ内でこのシステムコールを発行することにより、独立したデータセグメントを使えるようにし、高級言語で割り込みハンドラを書くことも可能とした。

メッセージの先頭アドレスやメモリブロックの先頭アドレスもセグメント値を含めて指定することができる。

高速化のため、タスク、セマフォなどの指定には、ID番号ではなく管理ブロックのアドレス（以下アクセスアドレスという）を直接用いているが、アクセスアドレスはセグメント値16ビットだけで表わし効率化をはかった。

6. アセンブラインタフェース

システムコールに対する引数や戻り値（リターン値）は、原則としてレジスタに置くようにした。プロセッサごとにレジスタの使い方に関する原則を設け、そのもとになる考え方は統一した。I-TRON／68K 及び I-TRON／86におけるレジスタの使い方に関しての原則は以下の通りである。

[I-TRON／68K]

- レジスタ数とシステムコールの引数から考えて、5つのレジスタをシステムコールとのインタフェースに使うことにする。使用するレジスタは、D0,D1,D2,A0,A1 とする。
- システムコールの種類を示す機能コードは、システムコールに対するオプションと共に D0 に置く。
- 戻り値のうち、エラーコードは D0 に返る。この値が0のとき、エラーがないことを示し、同時に効率向上のため、Zフラグ（ステータスレジスタ内）をセットする。
- 引数の数が多いもの、データの長いものは、メモリ上にパケットを作ってその先頭アドレスを A0 に置く。これに該当するシステムコールは、

Create_Task などのシステム生成に関係するものと、 Set_Time などの時間関係のものである。

- 高速化のため、タスク、セマフォなどの指定には、ID番号ではなく管理ブロックのアドレス（以下アクセスアドレスという）を直接用いる。アクセスアドレスは引数、戻り値とも A1 を使う。
- ID番号、タスク優先度、終了コード、ビットパターンなどの1ワードに入るデータは D1 を使う。ただし、このような引数が2つあるときは、D2 も使う。
- メッセージアドレス、割り込みハンドラの先頭アドレスなど、アクセスアドレス以外でアドレスを表わす引数があれば，A0 に置く。
- 戻り値として使わないレジスタはすべて保存する。

[I-TRON／86]

- 4つのレジスタ AX,DX,SI,DI をシステムコールとのインタフェースに使うことにする。
- 機能コードは、システムコールに対するオプションと共に AX に置く。
- 戻り値のうち、エラーコードは AX に返る。この値が0のとき、エラーがないことを示し、Zフラグがセットされる。
- 引数の数が多いもの、データの長いものは、メモリ上にパケットを作ってその先頭アドレスのオフセット値を SI に置く。
- 高速化のため、タスク、セマフォなどの指定には、ID番号ではなくアクセスアドレスを用い、セグメントのみの16ビットで表現する。アクセスアドレスは引数、戻り値とも DI を使う。
- ID番号、タスク優先度、終了コード、ビットパターンなどの１ワードに入るデータは DX を使う。ただし、このような引数が２つあるときは、DI も使う。
- メッセージアドレス、割り込みハンドラの先頭アドレスなど、アクセスアドレス以外でアドレスを表わす引数があれば， セグメント値を DX に、オフセット値を SI に置く。
- 戻り値として使わないレジスタはすべて保存される。

7. 高級言語インタフェース

I-TRONは複数のプロセッサ上に実現され、本論文のI-TRON／68Kもそのひとつである。その際、ユーザプログラムのプロセッサ間の相違を吸収する方法は、高級言語による記述以外にはない。I-TRON標準の高級言語は、現在Cである。

システムコールはCの関数として呼び出す。機械語レベルでのシステムコールとCの環境の間にはインタフェースルーチンがはいり、引数の調整などを行なう。

言語Cからシステムコールを呼び出す場合の引数の受け渡し方は次のような原則に基づいている。

- Cの関数としての戻り値はエラーコードとする。
- 引数の順序は、オプション、エラーコード以外の戻り値をセットするアドレス、その他の引数、である。
- アクセスアドレスについては、他の引数よりも前に置く。
- 引数が多い場合にはメモリパケットを使っていたが、これについては同じ形式のままCの構造体として扱う。

I-TRON／86の場合、現在利用できる 86用Cコンパイラでは、セグメントを固定しているために 64Kバイトまでのアドレス空間しか扱えないもの（SMALL MODEL）が多い一方、I-TRON／86の性能を十分に引き出すためには、 1Mバイトのアドレス空間を扱うCコンパイラ（LARGE MODEL）を用いるのが望ましい。SMALL MODEL のCコンパイラと LARGE MODEL のCコンパイラでは、スタックの使い方が異なっているが、これは I-TRON／86と言語Cとの間のインターフェースルーチンで吸収することにした。

言語Cから、Create Taskシステムコールを呼び出す場合のプログラム例を以下に示す。

```
/* 宣言部 */
Struct CreatePacket {
  offset   TaskStartOffset;
  segment  TaskStartSegment;
  short    UserStackSize;
  short    TaskPriority;
  short    Option;
  segment  DS;
};
segment *TaskAccessAddressP;
int  TaskID;
struct  CreatePacket *Packet;
int  ErrorCode;

/* システムコール呼び出し */
ErrorCode = Cre_Tsk(
              TaskAccessAddressP,
              TaskID,
              Packet);
```

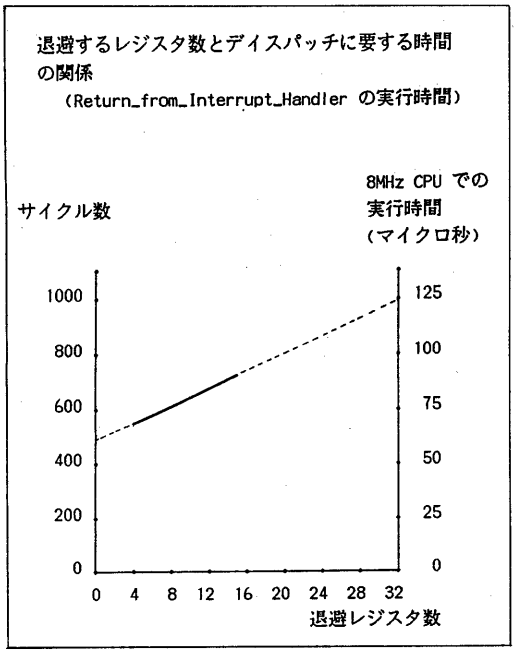
8. 実験的インプリメンテーション

現在、I-TRON／68Kのモトローラ VMEシングルボードコンピュータ上への実験的インプリメンテーションをすすめている。

オブジェクトコードの大きさは、タスク関連操作部で約3Kバイト、チップ核全体で約8Kバイトである。オブジェクトコードの大きさについては、I-TRON／86についても同程度になることが報告されている[10][11]。TCBの大きさはコンテキストスイッチで退避するレジスタ数によって変わってくるが、15個全部を退避した場合116バイト、8個のみ退避した場合には88バイトになる。

タスクスイッチングの時間としては、割り込みハンドラから戻りディスパッチするシステムコール(Return_from_Interrupt_Handler)について、その実行サイクル数の計算、実行時間の測定を行ない、全部のレジスタを退避した場合に724サイクル(8MHzの68000で、91μs)という結果を得た。これは、現在の他のリアルタイムオペレーティングシステムと比較しても最高水準にある。また、コンテキストスイッチで8個しか退避しないときは、612サイクル(8MHzの68000で、77μs)となり、全部のレジスタを退避した場合と比較して20%程度の速度向上となった。

退避するレジスタ数とディスパッチに要する時間の関係を図に示す。



9. おわりに

TRONとは一つの設計思想であり、異なるプロセッサ上にいかにして実現していくかについての基本的考え方を示すものである。その具体例として、I-TRON／68KとI-TRON／86における実現について述べた。

今後、他のアーキテクチャを持ったハードウェア上にのせる場合にも、ここで述べた考え方が基本となる。

参考文献

[1] 坂村 健「TRONプロジェクトの設計思想」bit 1984.9

[2] 坂村 健「TRONプロジェクト」マイクロコンピュータ応用国際コンファレンス(IMAC)'84

[3] 坂村 健「マイクロプロセッサ用標準リアルタイムオペレーティングシステム原案:TRON 1. アーキテクチャ」情報処理学会第28回(昭和59年前期)全国大会

[4] 梅田 望夫、坂村 健「マイクロプロセッサ用標準リアルタイムオペレーティングシステム原案:TRON 2. カーネル(1)」情報処理学会第28回(昭和59年前期)全国大会

[5] 松為 彰、坂村 健「マイクロプロセッサ用標準リアルタイムオペレーティングシステム原案:TRON 3. カーネル(2)」情報処理学会第28回(昭和59年前期)全国大会

[6] 清水 徹、坂村 健「マイクロプロセッサ用標準リアルタイムオペレーティングシステム原案:TRON 4. 開発環境」情報処理学会第28回(昭和59年前期)全国大会

[7] 坂村 健「I-TRONアーキテクチャ」情報処理学会第29回(昭和59年後期)全国大会

[8] 梅田 望夫、坂村 健「I-TRON／68K」情報処理学会第29回(昭和59年後期)全国大会

[9] 松為 彰、坂村 健「I-TRON／86」情報処理学会第29回(昭和59年後期)全国大会

[10] 長谷川 薫、高橋 久、門田 浩「V20/30用リアルタイムOSのインプリメンテーション[I]」情報処理学会第29回(昭和59年後期)全国大会

[11] 長谷川 薫、高橋 久、門田 浩「V20/30用リアルタイムOSのインプリメンテーション[II]」情報処理学会第29回(昭和59年後期)全国大会