リアルタイムOSを導入した組込みシステムの開発技術

Development Technologies for Embedded System with Real-Time Operating System

特集

山内　信之
YAMAUCHI Nobuyuki

リアルタイムOSを導入した組込みシステムに対して，デバッグ支援環境の整備，環境設定支援ツールの開発，ソフトウェア開発手法の確立が急務となっている。デバッグには，リアルタイムOS管理データやプログラム実行履歴をユーザーにとってわかりやすい形式で表示する技術が必要である。システム環境定義については，不具合の早期排除を目的とした設定データのチェック機構が有効となる。また，ソフトウェア開発手法としてオブジェクト指向技術を適用する試みもある。いずれもリアルタイムOSの視点に立った支援が必要となる技術である。
ここでは，当社が提案する開発ツールと手法について述べる。

In the development of an embedded system with a real-time operating system (RTOS), it is necessary to prepare debugging and configuration aid tools and to establish the software development method to be used. For debugging, it is necessary to indicate RTOS management data and program trace data to the users. For system configuration, it is effective to check for errors in the set data to remove bugs at an early stage. There is also the approach of applying object-oriented technology to application system development. In all cases, support from the standpoint of the RTOS is important.
This paper provides an outline of these technologies.

1 まえがき

近年，組込みマイコン用システムに対してリアルタイムOS（RTOS）を適用するケースが増加している。この背景には，アプリケーションシステムが要求する機能が飛躍的に高くなっていることがある。一般にRTOSを使用したアプリケーションの開発には，デバッグ環境，アプリケーション構築環境が重要であると言われている。RTOSを使用したシステム設計手法に関しては，現状では確立した手法は存在しない。オブジェクト指向技術を適用する試みも報告されているが，実際の製品に組み込んだ事例は多くない。

ここでは，ユーザーからの視点に立ち，RTOS使用環境として要求される項目について述べると同時に，一つのソリューションとして当社が提案する開発ツールおよび手法を紹介する。なお，特に断らないかぎり，ここではRTOSとしてμITRON仕様OSを想定する。

2 RTOSを用いた組込みシステムの開発

RTOSを用いた組込みシステムの開発において，アプリケーション開発者からの要求が多い事項と研究・開発課題となっている事項について述べる。

2.1 デバッグ支援環境

RTOSを使用したアプリケーションでは，実行最小単位であるタスクが並行に動作するプログラム構造となる。すなわち，複数のタスクの実行権が切り換わりながら，プログラムが進行する。この実行権の切換えは，RTOSが制御するが，一般にはユーザーはRTOSの内部動作を把握する必要はなく，ブラックボックスとして扱うことができる。

ところがデバッグ作業において，不具合の原因がRTOSの使用方法にあるのか，アプリケーション自体のバグなのかの判断が困難な場合がある。多くの場合，RTOSの内部状況を確認することで判断したほうが効率的であるため，RTOS内部データ構造の提示を求めるユーザーも少なくない。しかし，一般にはRTOS内部構造を理解することは容易でなく，たとえユーザーが内部データ構造資料を参照したとしても，デバッグ自体はそれほど容易にはならない。

このような状況を考えた場合，RTOSを完全なブラックボックスとしてしまうよりも，その内部データをわかりやすくユーザーに提示することのほうが，現場において有効であると予想できる。筆者らのヒヤリング調査においても，ユーザーから強く要求されたのは，RTOSを使用したシステムに対応した効果的なデバッグ支援ツールであった。

2.2 システム環境設定

RTOSをアプリケーションシステムに初めて導入したユーザーにとって，システム環境設定は直感的にわかりにくい作業としてみられる傾向がある。この環境設定にはRTOSが提供する機能（タスク管理，時間管理など）やハードウェアの設定が含まれる。例えば当社のTLCS$_{TM}$-900 RTOS TR900は，システム環境定義ファイルと呼ばれるテキストファイルを編集することで環境設定を行う。このファイルはアセンブラ言語で記述されており，環境設定とそれに伴

う動作を詳細に把握できる利点をもつ一方，次のような課題もあった。

(1) 設定部分のわかりやすい提示方法の実現　システム環境定義ファイルは約500行あり，ユーザーが目を通す必要がある範囲は比較的広いものとなっていた。

(2) 設定時のエラー判定の実現　システム環境定義ファイルの項目には，それぞれが関連をもつものがある。例えば，ユーザーが使用可能なメモリの総容量と，メモリ割当て最小単位の合計などである。従来は，すべてユーザーがチェックしていたが，人為的ミスによりデータ間で矛盾が生ずる場合もあった。

2.3 システム設計・実装技法

ソフトウェアの肥大化に伴う，ソフトウェア生産性の問題は，多くのユーザーにとっての解決すべき課題となっている。特に，組込みマイコンシステムにおいては，サイズとパフォーマンスの問題が付きまとうため，実製品に対して効果的な方法をとることは困難であった。

このような背景の下，オブジェクト指向技術をソフトウェアの設計と実装に適用することで，ソフトウェア部品の流用率を高める動向がある。特にRTOSをすでに使用しているアプリケーションシステムにおいては，オブジェクト指向技術の適用部分を見定めることが重要となっている。

3 RTOS対応デバッガ

ここでは，当社のRTOS TR900用デバッグ機能を組み入れたデバッガ TLCS$_{TM}$-900 ファミリデバッガ ver.2.1 (以下，デバッガ2.1と呼ぶ) に関して，RTOS側の視点からその実現方法を解説する。このデバッガの特長は，RTOS自体の管理領域の内容を表示するとともに，実行履歴をグラフィカルに表示できることである。これにより，従来困難であったRTOS内部状態の確認を容易にすることができた。

OS対応デバッガに要求される機能は次のとおりである。

(1) RTOS内部の管理データの表示　例えば，アプリケーション開発者の想定外の条件が発生し，すべてのタスクが待ち状態となってシステムが停止してしまった場合を考える。このとき，「すべてのタスクが待ち状態である」ということを知らせる機能を提供することで，ユーザーは容易に対応を行うことが可能となる。

(2) RTOS機能の実行履歴表示　プログラムの想定どおりにシステム状態が変更しない場合，予期せぬ割込みが発生していたり，システムコールの使用方法が誤っている場合がある。この際に有効な機能として，RTOSのサービスが行われた履歴をアプリケーションの実行履歴と関連させて表示する方法がある。

上記(1)の機能には，デバッガがRTOSの管理データを獲得する仕組みが必要になる。デバッガ開発者とRTOS開発者が異なる場合は珍しくなく，通常は，RTOSの管理データをデバッガが把握していることは少ない。また，データがRTOS内部の複数のテーブルに格納される場合もあるため，通常その獲得のための手順は複雑になる傾向がある。

TR900では，その管理データにアクセスするためのインタフェースをデバッガ開発者に提示することで，RTOS管理データ表示機能を実現した。この概念を図1に示す。図1は，RTOS側が提示したデータ獲得部によって，RTOS管理データがデバッガに渡される構造を示している。

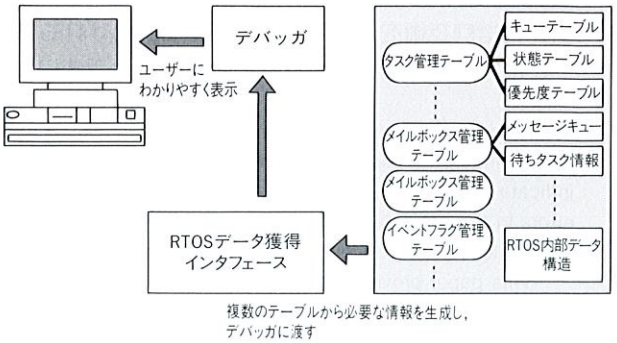

図1．RTOS管理データの受渡し　RTOS内部の管理データをRTOSデータ獲得インタフェースがまとめ，デバッガに渡す。
Flow of RTOS management data

(2)に関しては，通常デバッガには実行トレース機能があり，アプリケーションプログラムの履歴情報の獲得はもともと可能である。しかし，この情報はアプリケーションプログラムとRTOSの実行履歴が混合されたものであるため，一般には不具合の原因を特定することは容易でない。また，ユーザーが望むものはタスクやRTOS内部の動作というよりも，タスク間の同期，通信の状況であることが多い。しかし，従来のデバッガのトレース表示機能には，これらを表現する手段が十分には備わっていなかった。

デバッガver.2.1のRTOS対応機能を実現するために，TR900はシステムコール実行履歴，ディスパッチ履歴，ハンドラ実行履歴を採取するための機能モジュールを新たに備えた。このことで，実行履歴を示すデータを特定メモリ領域に書き込むことが可能になった。書き込んだデータは，その時点でICE (RTE model 25/15) のトレースメモリに転送される。転送されたデータはユーザーにとって見やすいように整形され，表示される。

RTOSの視点から見て，この機能の実装におけるポイントは，採取するデータの決定であった。すなわち，履歴情報が多ければ多いほど，デバッガで表示する項目を増やすことができるが，その一方でRTOSのオーバヘッドも増加してしまうことを考慮する必要があった。ユーザーによっては，デバッグ時に使用したRTOSをそのまま製品に組み

込むことも考えられるため，オーバヘッドはできるかぎり最小限に抑えることが望まれた。TR900では，検討の結果，全体的なオーバヘッドの増加を20％まで許せば，比較的十分なデバッグ情報が得られると判断した。これに従い，例えばシステムコール実行履歴の場合には，次のデータだけに絞り，履歴情報として獲得するように実装した。

(1) トレースデータ種別
(2) 発行タスクID
(3) システムコールID
(4) 対象オブジェクト

実行履歴獲得機能を含むTR900の内部構造を**図2**に示す。図2は，RTOS内部のシステムコール実行部分，割込みハンドラ実行部分，ディスパッチャの内部にそれぞれコードを加えてあることを示している。また，**図3**にデバッガver.2.1を使用し，RTOSを使用したアプリケーションの実行履歴を表示させた画面を示す。

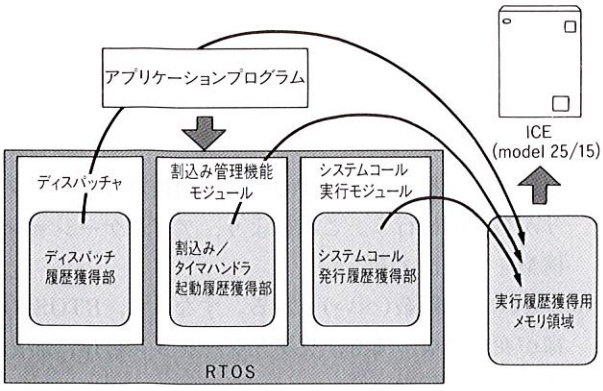

図2．履歴獲得部を実装したRTOS内部構造　RTOSの各機能モジュールに実行履歴獲得部が組み込まれている。実行履歴は実行履歴獲得用メモリ領域に書き込まれ，ICEがこれを読み込む。

RTOS structure for implementation of trace data acquisition functions

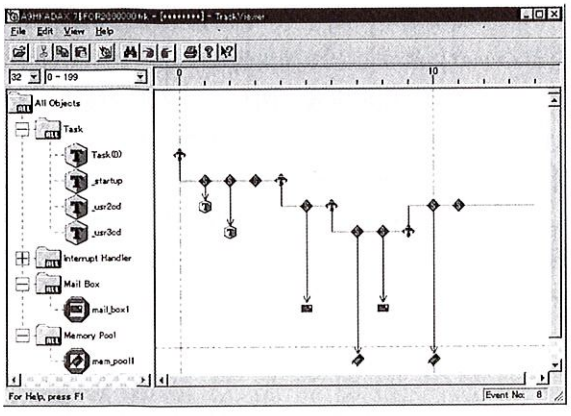

図3．デバッガver.2.1実行画面　RTOSを使用したアプリケーションの実行履歴がグラフィカルに表示されている。

Screen display of debugger Ver. 2.1 showing trace data of RTOS application

4 RTOSアプリケーション環境設定ツール

第2章で述べたように，RTOSを使用したアプリケーションの開発工程において，システム環境設定作業は大きな意味をもつ。システム環境設定に対して望まれる項目は次のとおりである。

(1) 設定すべき項目の局所化
(2) 設定項目間の関連性を利用したデータの自動生成
(3) 設定項目間の関連性を利用した設定エラー排除機能

上記の項目を考慮し，TR900の次期バージョンには環境設定ツールを付加することを予定している。このツールはWindows®(注1) 95/NTのダイアログボックス形式をとり，設定の必要な項目が一見して判断できる。また，設定できる範囲を関連する項目に従って絞り込むくふうをした。例えばアプリケーションが時間管理機能を使用しない場合，タイマの設定部分の入力を受け付けないようになる。従来の設定ファイルが約500行のファイルであったことを考えると，この環境設定ツールを使用することで，ユーザーが見るべき範囲を大幅に削減できることが期待できる。

次に，設定時点で判断できるエラーチェック機能を仕様に加えた。これは，互いに関連性のあるデータを参照し，整合がとれないものをあらかじめ排除しようとするものである。タスクに関係する設定項目を例にとると，タスク管理領域の確保数と最大タスク数，設定優先度数と最大優先度数，初期実行タスクIDとタスク数などが関連をもつ。例えば，システムに存在する最大タスク数を1としながら，タスク自体を2個以上定義してしまった場合，タスク切換え後のプログラム動作が保証できなくなってしまう。このような単純なミスを早期に防ぐことにより，ユーザーはプログラムの論理的なデバッグに注力することができるようになる。この環境設定ツールのモジュール構成を**図4**に，その実行画面を**図5**に示す。

5 オブジェクト指向技術の適用

オブジェクト指向技術を組込みマイコンシステムへ適用する試みは，以前から報告されている。しかし，RTOSをすでに使用して開発されているシステムを再設計し，実装を行う試みはあまり聞かない。大きな理由の一つとして，現状のソフトウェア構造にかかわる大幅な変更は極力少なくしたいという要求が開発現場にあることが挙げられる。すなわち，オブジェクト指向的アプローチをとって分析，設計されたソフトウェア構造が，それまでのタスク構造と同一であるとは限らないために，場合によっては大部分のモジュールを見直す必要さえ生ずる。このことは，開発コストの増大を招くだけでなく，システムの安定性を低下さ

(注1) Windowsは，Microsoft社の商標。

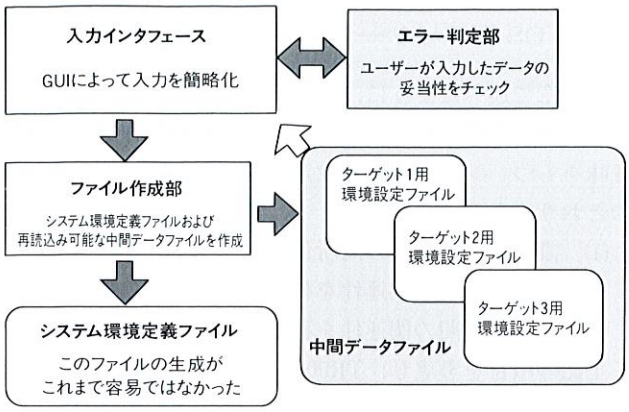

図4．環境設定ツールのモジュール構成　エラー判定部を備えることで，環境設定時のバグの混入を防ぐことができる。

Structure of system environment definition tool

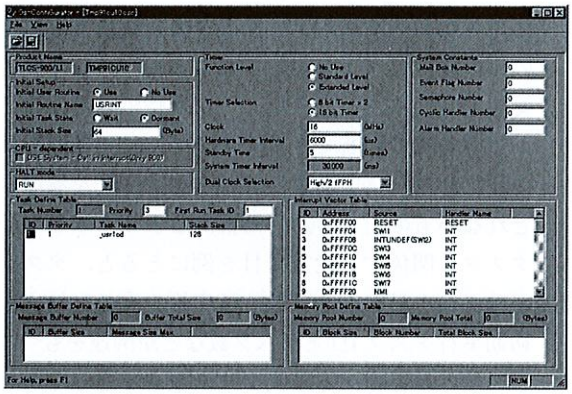

図5．環境設定ツール実行画面　システム環境設定ツールの表示画面。設定すべき項目が一画面で確認できる。

Screen display of system environment definition tool

せてしまう可能性がある。

また，実際の製品に組み込まれているソフトウェアを調査すると，モジュール間で，出来，不出来にばらつきが見られることがある。このことは，製品間におけるソフトウェア部品の流用率の低下につながってしまう。

このような事態が発生する原因の一つにRTOSの使用方法が必ずしも単純ではないことが挙げられる。例えばメッセージ通信を行う場合，通信の前後ではメモリ管理を考慮してメッセージ作成を行う必要がある。ユーザーが本来行いたいのは通信だけであるが，実際のコードは通信の前後処理のほうが長くなる傾向がある。この前後処理の作込みしだいでモジュールの独立性を損なう状況が生じていた。

また，同様なシステムを別のRTOS上に移植する場合がある。通常は，RTOSが異なればアプリケーション自体にも変更が必要となるため，開発には多くの工数を見積る必要がある。

以上のような状況から，従来のタスク構成を崩さず，RTOSの使用勝手を向上させる手段が重要となることが予想できる。この手段は，同時にソフトウェア部品の流用率を高めるようになっていなければならない。このような課題に対し，当社からはRTOS対応のクラスライブラリを提案している。このライブラリを使用した場合のシステム構成を図6に示す。その特長は次のとおりである。

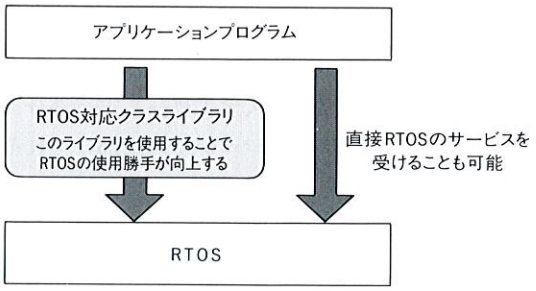

図6．RTOS対応クラスライブラリ　RTOSの機能を使いやすくするとともに，アプリケーション部品の流用率を向上させる。

Class library for RTOS

(1) RTOSの機能を使用するための前後処理はライブラリが自動的に行う。これにより，アプリケーションの構築を容易にする。

(2) RTOSが隠蔽(ぺい)される。すなわち，RTOSの仕様が変更になっても，アプリケーション自体に加える変更は発生しない。

(3) ライブラリを使用しないで，直接RTOSの機能を使用することを許す。このことで，ソフトウェア設計に柔軟性をもたせることが可能となる。

6 あとがき

デバッグ支援環境に関して，RTOSの管理情報，実行履歴をデバッガに渡すことで効果的なデバッグが可能であることと，システム環境定義に関しての必要な機能を述べ，TR900における実装例を示した。また，アプリケーション設計においてRTOS対応のクラスライブラリが有効であることを述べた。どれもRTOSを使用したシステムの効率的な開発を可能にするものである。今後，いっそうの機能向上を図っていく予定である。

山内　信之　YAMAUCHI Nobuyuki

COS事業推進部 ソフトウェア技術担当主務。
リアルタイムOSの開発に従事。情報処理学会会員。
COS Div.