Vol.2013-OS-124 No.4
2013/2/28

組込みＲＴＯＳにおけるタスク起動機構冗長化とタスク起動順位保持

平井勝彦† 伊原彰紀† 門田暁人† 松本健一†

本稿では，組込みＲＴＯＳ（Real-Time Operating System）におけるコンテキスト切替えオーバーヘッドの低減を目的として，「タスク起動機構の冗長化」，および，「タスク起動優先順位の保持」の方式を提案する．組込みシステムの多くは，プリミティブ・レベルのインタフェース方式により周辺デバイス群をバスに直接接続するプロセッサ構成となっている．ＯＳとタスク間で頻繁に生ずるインタラクションなどオーバーヘッドの除去は，ソフトウェア方式設計時の重要課題の一つである．提案方式では，タスク起動要求の到着間隔の実効時間短縮を図るタスク統合化環境およびタスク起動機構の冗長化アルゴリズムにより，コンテキスト切替えオーバーヘッドの低減を実現すると共に，冗長化に伴うスケジューリング優先性の錯綜を防ぐ．

Discussion of A Method for Redundancy to Invoke Tasks and A Method To Keep A Priority Order for Dispatching A Task in the Embedded RTOS

KATSUHIKO HIRAI† AKINORI IHARA†
AKITO MONDEN† KENICHI MATSUMOTO†

This manuscript discusses an idea for reducing the overhead of the context switch in the embedded real-time operating software, and proposes a method for redundancy to invoke tasks and a method to keep a priority order for dispatching a task as the task has specified the order. The typical embedded system configures the processor through the bus directly connected to devices with the primitive level interface architecture. In the software architecture design, it is one of important themes to eliminate any overheads caused by the frequent interaction between RTOS and tasks. This manuscript proposes three methods as follows: First is to implement the reduction of the overhead to switch the context with the algorithm for redundancy to invoke tasks. Second is to implement the environment to integrate several tasks into the single task for shortening effective intervals of the task activation. Third is to prevent the complexity of task scheduling caused by the algorithm for redundancy.

1. はじめに

近年，情報システム技術分野における組込みシステムの役割は，社会基盤として，不可欠なシステム領域の一部を担っている．しかし，組込みシステムのソフトウェア開発において設計手法や開発管理の更なる合理性が求められるので[1]，開発プロジェクトには３つの課題が残る．

(1) Design Scalability・・・設計拡張性
(2) Productive Development・・・開発生産性
(3) Software Maintainability・・・保守容易性

設計拡張性には，ハードウェア環境の拡張制約の下で，高いソフトウェアスケーラビリティ環境要件が求められる．タスク増設による機能拡張が頻繁に行われる組込みソフトウェアにおいては，RTOS タスク管理によるコンテキスト切替えが増加することから，コンテキスト切替えオーバーヘッドはスケーラビリティ劣化の要因となる．設計拡張性の提案においては，RTOS オーバーヘッド低減を図るタスク起動機構に着目した．

開発生産性には，プロダクトライン動向，ハードウェア方式仕様，並びにユーザビリティなどに関する要求分析の妥当性が求められる．しかし，要求事項への変更頻発は，ソフトウェア方式設計において不適切なタスク分割を誘発し，ソフトウェアテスト工数の増加などを伴う副作用となるので，不適切なタスク分割は開発生産性劣化の要因である．開発生産性の提案においては，要求分析の弊害を緩和するタスク起動機構に着目した．

保守容易性には，対象のモジュールやユニットの特定，変更，並びに保守完了の確証などの要件が求められる．しかし，中小規模の組込みソフトウェアの多くが RTOS を採用せず，割込み処理と内部処理を一体化しているので，保守性が劣化し易い．一体化したソフトウェア構造の分解保守の手順要件には，内部処理ソフトウェア部分を，一旦，タスクに格納し，タスク分割技法を用いてモジュールの独立化を図り，保守性を段階的に高めることが求められる．そのための課題は，割込みベクタの優先順位と等価となるタスク起動に関する優先順位を保持するソフトウェア構造の装備が条件である．保守容易性の提案においては，タスク起動の優先順位を保持できるタスク起動機構に着目した．

本稿の構成は第 2 章に本研究の背景，第 3 章に設計拡張の阻害要素と対策，第 4 章に開発生産性の阻害要素と対策を示す. 第 5 章に保守容易性の阻害要素と対策を提案する．第 6 章は本研究に関する今後の目標と課題を明らかにする．

† 奈良先端大学院大学情報科学研究科
Nara Institute of Science and Technology

ⓒ2013 Information Processing Society of Japan

2. 本研究の背景

組込みソフトウェアのソフトウェア方式設計において，タスクの優先順位を検討する際，タスク実行周期に基づく制約デッドラインの算出，タスク実行のスケジュール可能性の判定，そしてタスク優先順位の決定を行う．しかし，問題となるのは，RTOS 処理オーバーヘッドに関する発生確率に基づく RTOS 実行時間のバイアス値などである．

組込みシステムにおいて，RTOS 処理のオーバーヘッドは基本的な管理処理である以下の項目で占められる．

(1) Task Management・・・タスク切替え，タスク実行順序監視，およびタスク起動などの管理機構
(2) Time Management・・・時間監視と時刻管理機構
(3) Memory Management・・・メモリ領域の貸与管理機構
(4) Interruption Management ・・・割込み管理，および遅延ディスパッチ管理機構
(5) Inter-Task Communication Management・・・タスク間通信の仲介管理機構
(6) Inter-Processor Management・・・マルチ CPU 間並列化管理，および同期化管理機構
(7) Configuration Management・・・ソフトウェア最新版更新管理機構
(8) System Mode Management・・・システム状態監視管理機構

比較的高い発生確率を占める RTOS 処理オーバーヘッドはタスク管理であり，タスク動作の並行性の変化に比例して負荷変動を示すオーバーヘッドである．

タスク管理の計算量変化の原因は，管理対象のタスク数の増加に伴いスケジュール待ち行列にて待機するタスクの行列長が伸びるので，タスク探査時間が増加することにある．一般的に，実行待ちタスクのスケジュール待ち行列において，FIFO(First-In-First-Out)方式を行列順序の基準にしてタスク優先順位を重みとする行列を形成するので，優先順位の低いタスクを切替えるときのタスク探査時間が増加する傾向がある．

他方，時間待ちタスクのスケジュール待ち行列は待ち時間数を，そしてメモリ領域貸与待ちタスクのスケジュール待ち行列は FIFO(First-In-First-Out)方式を行列順序の基準とするので，時間管理やメモリ管理も各スケジュール待ち行列にあるタスク数に依存した計算量が発生する．しかし，スケジューリング密度が低いので，これらのオーバーヘッドは著しいレベルではない．また，割込み管理，タスク間通信管理, およびマルチ CPU 管理におけるスケジューリングは，全てタスク管理に集約して実行されるので，タスク管理のオーバーヘッドに含まれる．しかし，発生頻度は低い．

尚，ソフトウェア最新版更新管理は，更新対象のタスクを動作停止するため，オーバーヘッドにならない．更新版ソフトウェアの転送処理，そしてシステム状態監視管理などの方がオーバーヘッドに含まれるが，システムが低い負荷を示すタイミングにおいて実行されるサービスであるので，無視できる，

以上より，RTOS 処理によるオーバーヘッドはタスク管理に注目すべきであることが分かる．しかも，タスク管理の発生頻度は，タスク・アルゴリズム自体の挙動に依存するので，タスク動作の実態を解明し，RTOS のタスク管理並びにタスク・アルゴリズムの合理的な仕組みに関する研究の重要性に着目した．参考文献[2]において，マルチ CPU での並列タスク・スケジューリング手法による，スケジュール可能性からの効率向上の提案が示された．

しかし，如何に RTOS において合理的とするタスク・スケジューリング方式を確立できても，残念ながら，CPU 計算量を生み出すことは難しいと考える．

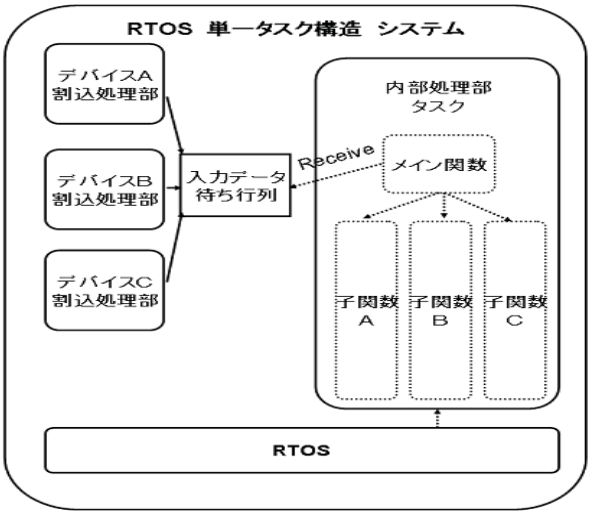
一方，RTOS 処理オーバーベッドを低減できれば，その計算量はタスク・アルゴリズムに割り当てるので，利用者の観点から CPU 実効計算量は増加と認識できる．

そこで，RTOS のオーバーヘッド発生を避けるために，タスク管理処理が実行しない基本条件を検討した．参考文献[3]において，Linux でのスレッド管理の実行を回避し，コンテキスト切替えオーバーヘッドを低減する方式が提案されたが，スレッド環境という制約下でのオーバーヘッド低減では広く一般の RTOS 環境への適用は困難であると考察した．そこで，式(1)の特性をもつタスク・アルゴリズムを抽出することにより，一般解が得られないかと考えた．

（入力到着時間間隔）＜（タスク内の計算時間） (1)

並行処理性能は期待できないけれども，極端な例として，図 1 に示すような単一タスク構造のシステムを構築すれば，絶え間なく入力が到着するので，計算量が重い限り，タスク管理（即ち，コンテキスト切替え）は発生しないという仮定が成立するのである．

図 1　コンテキスト切替えのない単一タスク構造



ⓒ2013 Information Processing Society of Japan

3. 組込みシステムにおける設計拡張性

組込みシステムにおいてソフトウェアの拡張性はハードウェアとソフトウェアの双方から厳しい制約が課せられ，設計目標と実装結果との乖離が重い問題となる．例えば，ハードウェア面から，メモリサイズ，非 DMA データ転送速度，並びに CPU バウンド処理などの制限がソフトウェア設計の制約条件となり，ソフトウェアサイズや CPU 計算量の節約に関する要求事項が与えられる．

組込みシステムは高いリアルタイム処理性能を有するマルチタスク構造のソフトウェアを内蔵する．図 2 は組込みソフトウェアの実行タイミングと制約要素を示している．

図 2　マルチタスク実行タイミング例と制約要素

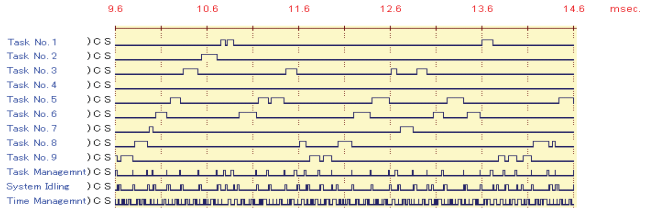


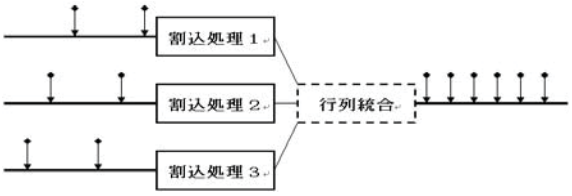
表 1 では，システムにおけるコンテキスト切替えがオーバーヘッドとなり，その発生増加に伴いタスクの実行占有率が減少傾向にあることを表す．RTOS によるオーバーヘッドはソフトウェアスケーラビリティのリニアリティ劣化の制約要素となっている．ソフトウェア面から，タスク分割に関する階層構造，機能配置，並列関係，並びに優先順位などの制限が制約条件となって要求事項が与えられる．

表 1　RTOS タスク管理によるオーバーヘッド例

CPU: H8-3687/ 20MHz (計測単位: 1mS)		タスク管理のコンテキスト切替え オーバーヘッド変化による影響				
切替え頻度		4 回	5 回	6 回	7 回	8 回
Task 関連 CPU 消費	User Task	92%	88%	82%	75%	70%
	Task Management	7%	11%	18%	24%	29%
	Overhead per Switch	11uS	14uS	14uS	15uS	16uS

これら制約の緩和への必要条件は RTOS オーバーヘッドの削減である．実現への課題は RTOS システムコール発行によってタスクが待ち状態に遷移する頻度を低減させる環境を整えることにある．即ち，タスク実行時間内に次の処理要求が到着してしまう環境を組立てることである．具体策は①タスクの計算量粒度拡大，②処理要求の到着間隔高密化の 2 案を検討した．しかし，前者は処理の並列性が劣化するので，図 3 に示す通り，後者の処理要求の到着間隔高密化方式を採用した．

図 3　処理要求の到着間隔高密化モデル



入力デバイス，または他タスクからの処理要求など複数の発生源から入力されるメッセージの行列統合には①単一化順次受信方式，②冗長化包括受信方式の 2 案を検討した．しかし，前者の単一化順次受信方式では処理要求の発生源間に定められた優先順位を識別できないので，優先順位を継承できる後者の冗長化包括受信方式を採用した．

処理要求の冗長化包括受信に①待ち行列チェイン方式，②待ち行列配列方式の 2 案を検討したが，検索効率性の良さに着目し，後者を採用した．

冗長化包括受信方式 (RCA: Redundant and Comprehensive incoming Architecture) の実現に求められるタスク起動機構には，図 4 の通り，①包括ディレクトリ，②包括された待ち行列毎に受信処理用ユーザ関数を指定するデータ構造を加え，RTOS には③包括受信アルゴリズム及び送信アルゴリズムを装備する．

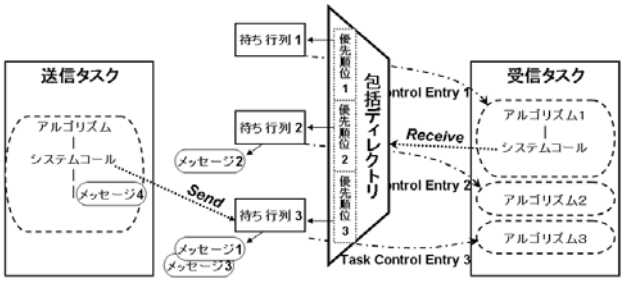
包括ディレクトリとは，包括下にある各待ち行列および待ち状態にあるタスクを格納して，メッセージ受信要求の際に，RTOS システムコール［包括受信］にて指定される．

メッセージ受信用の待ち行列毎に異なるアルゴリズムを実行できるように，包括下にある待ち行列毎に独立した受信処理用ユーザ関数を指定する．この指定により，個々の待ち行列に異なる特性をもつメッセージを取り扱うアルゴリズムの独立性を担保できるように，各アルゴリズムを束ねて 1 個のタスクを形成する．アルゴリズムは個々にタスク制御エントリを備えてタスク構造を単純化できるので，タスクの論理的強度の上昇を抑制する効果がある．

包括受信アルゴリズムは，RTOS 内部に配置され，タスクが実行する RTOS システムコール［包括受信］により呼び出される．このアルゴリズムは RTOS システムコール［包括受信］に指定された包括ディレクトリ内に到着メッセージをもつ待ち行列があれば，そのタスクは実行状態を継続する．もしも包括ディレクトリ内の全ての待ち行列に到着メッセージがなければ，そのタスクを待ち状態に遷移して，当該包括ディレクトリに収容されて待機させる．

送信アルゴリズムは，RTOS 内部に配置され，タスクが実行する RTOS システムコール［送信］により呼び出される．このアルゴリズムは RTOS システムコール［送信］に指定された待ち行列に送信メッセージを行列し，その待ち行列を格納する包括ディレクトリに待ち状態のタスクが収容されていれば，そのタスクを実行可能状態に遷移する．

図 4　冗長化包括受信方式(RCA)タスク起動機構モデル



4. 組込みシステムにおける開発生産性

この章では，オペレーティングシステムの観点から，ソフトウェア開発生産性向上に関する課題と提案を示す．

参考文献 [1]によれば，表 2 の通り，組込みシステム開発の実態はプロダクトライン上の差分開発/派生開発/改修開発/保守開発など再開発プロジェクトが過半数を占める．機能の向上・追加・変更など改変作業を主体とするソフトウェア開発であり，ソフトウェア再利用率の高い設計活動である．開発要員投入は開発上流工程に手厚く，不具合発生の影響を抑制している．要求分析の妥当性評価レビューは高い実施率を示し，製品企画に基づくプロダクトラインの分析，ハードウェア方式に関する拡張可能性の解析，利用者ニーズのユーザビリティの分析など重点活動を行える実態にある．

しかし，工程別レビュー完全実施率減少と不具合発生原因率増加の相関は，ソフトウェア設計工程以降におけるハードウェア設計とソフトウェア設計のコンカレント開発による副作用の存在を示している．即ち，ソフトウェア設計における要求事項の変更頻発は要求事項の追跡可能性を困難にするので，レビュー成果並びに変更管理の徹底などを損なう．したがって，ソフトウェア方式設計の妥当性評価レビューが阻害されると，不適切なタスク分割や統合を抑制監視できなくなり，ソフトウェア不具合によるソフトウェアテストの作業要因が増加することに懸念が深まる．

表 2　組込みソフトウェア開発の実態

プロジェクト種別		開発事由		実装内容区分	
差分開発 派生開発 改修開発 保守開発 など	53%	機能の改変	54%	新規開発部	37%
		バグ対応	16%	既存改変部	34%
		Hardware/ OS 変更	14%	既存流用部	20%
		非機能の向上	12%	OSS 導入部	6%
新規開発	41%				

工程	要員 平均投入率	レビュー 完全実施率	不具合 発生原因率
企画・要求分析	10%	40%	12%
システム設計	10%	37%	16%
ソフトウェア設計	20%	29%	34%
実装・デバッグ	26%	21%	27%
SW テスト	16%	27%	5%
SYS テスト	10%	32%	3%
運用・実機テスト	8%	30%	4%

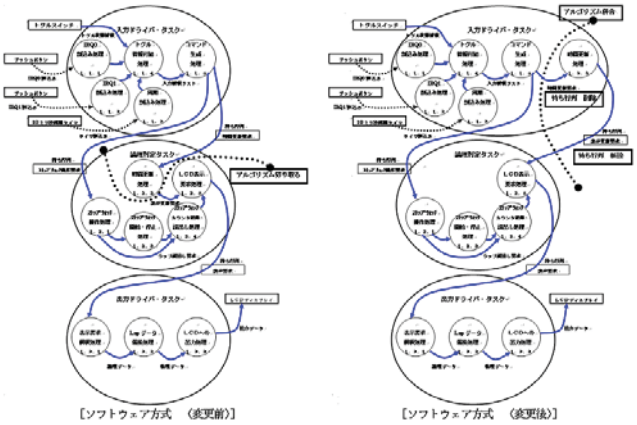
レビュー並びに変更管理の徹底を図って，開発生産性を向上するには，①要求事項の追跡可能性の確保，並びに②不適切なタスク分割や統合の防止などが求められる．

要求事項の追跡可能性の確保はソフトウェア設計工程以降でもソフトウェア方式設計の妥当性について常に確認できることが課題となる．この課題は，ソフトウェア設計内容とソフトウェア動作環境の一致を検証することにより達成できる．

タスク分割されたソフトウェアに関する要求事項の追跡可能性を目的とするソフトウェア動作環境は，タスク間通信インタフェースデータを対象にデータフロー分析手続きとする．しかし，意図の異なるデータの混合データフローでは分析が困難となるので，データ特性毎に独立したデータフローを形成できることが求められる．ソフトウェア設計に冗長化包括受信方式(RCA)をタスク間通信に導入することにより，目的とするソフトウェア動作環境を構築できる．ソフトウェア方式設計の妥当性を評価するには，タスク間通信に定義されている待ち行列，タスク，包括ディレクトリ，並びにタスク制御エントリ先のアルゴリズムを識別することにより，データフロー分析を行う．

タスク内部構造の複雑化による不適切なタスク分割や統合の発生防止策には，冗長化包括受信方式(RCA)におけるデータフロー別に確保できるアルゴリズムの独立性を活用した可視化を行う．即ち，タスク制御エントリを複数設定できることにより，データフロー別にアルゴリズムを独立でき，残された部分のソフトウェア方式の原型を損なうことなく，可視化環境下でタスク内部のアルゴリズム単位に機能の追加・削除・変更を行う．また，タスク間において，待ち行列と一対にアルゴリズムを切取り，あるいは切り取ったアルゴリズムを待ち行列と一対でタスクに貼付けるという一対処理の制約によりレビューが容易である．この可視化を含めた防止策により変更管理の実効性を高められる．概要例を図 5 に示す．

図 5　タスク分割の変更手続きとデータフロー図



一方，設計レビューやテスト工程で検証漏れが発生し易いソフトウェア例外処理設計において，冗長化包括受信方式(RCA)を導入することを提案する．

ソフトウェア動作環境において，システム全体の例外処理を一元的には定義できないので，ソフトウェア例外処理の詳細化に関する妥当性の評価はタスク単位の個別検証に依存するという問題がある．この課題には，タスク間通信に冗長化包括受信方式(RCA)を活用することにより，通例処理系と例外処理系の双方について 2 元的にデータフローを併記できるので，タスク間連携例外処理およびタスク固有例外処理に関しても統括的な設計および検証を実現する．

ⓒ2013 Information Processing Society of Japan

5. 組込みシステムにおける保守容易性

この章では，オペレーティングシステムの観点から，RTOS を装備しない組込みシステムを対象に, RTOS 導入への移行阻止の技術的原因を示し，保守容易性実現への課題と提案を示す.

参考文献[1]によれば，表 3 の通り，組込みシステムにおいて開発ソフトウェアの実行環境の 1/4 は OS を使用していないという調査結果が出ている.

表 3　組込み開発ソフトウェアの実行環境

実行環境	割合	実行環境	割合
Windows 系	55.4%	T-Engine 仕様	10.7%
Unix/Linux 系	55.4%	DOS 系	9.6%
ITRON 仕様	49.0%	自社独自	7.0%
OS は使用しない	26.0%	その他	4.8%
IDE に組込まれた OS	21.8%	不明	0.7%

RTOS を装備しない理由について，ハードウェアの制約であるメモリサイズに起因する．一方，ソフトウェア面からは①小規模なソフトウェア，②RTOS のオーバーヘッド，③Reengineering コスト負担などが想定される. しかし, 2004 年調査ではソースコード規模 1 万行以下が 37.4%を占めていた新規ソフトウェアは，2010 年調査では 17.6%に減少を示し，開発ソフトウェアは規模拡大に伴う RTOS 導入など保守性強化に向けて Reengineering コスト低減策がプロダクトライン開発の課題である.

また，RTOS 導入への移行阻止の技術的原因は，タスク内部アルゴリズムの観点から, その動作環境が CPU ハードウェア周辺回路インタフェースとの等価性を提供しないことにある．RTOS 装備下のタスク環境にソフトウェア内部処理アルゴリズムを移行するには周辺デバイスの割込み優先順位にしたがったマルチタスク構造に再構築した上での移行が求められる．しかし，冗長化包括受信方式(RCA)では包括ディレクトリにおいて待ち行列に優先順位を付与できるので，周辺デバイス毎に待ち行列を設定することにより，CPU ハードウェア周辺回路インタフェースとの等価性を提供するタスク環境を構築できる.

ここに，Reengineering コスト低減策の課題解決として，冗長化包括受信方式(RCA)の活用による段階的移行(MPS: Migration Path Spirals) Reengineering を提案する．段階的移行は, 最終目標であるソフトウェア構築まで, ①特性分離, ②環境移行，③移行前後実体照合，④移行前実体撤去検証の 4 段階を繰り返してアプローチする Reengineering 活動である．概要を図 6 に示す.

移行のシナリオは，移行対象のソフトウェアを 1 個のタスクに格納し，移行以前と等価な動作環境を構築することにより，動作確認を行い，そのタスクを複数にタスク分割して．RTOS 導入の要求事項を段階的に実装していく.

RTOS を装備しない組込みソフトウェアに関する RTOS 導入の移行手順例を基に，段階的移行(MPS) Reengineering プロセスを示す．スパイラル第一環では割込処理部を分割し，内部処理部は第二環以降にタスク分割を繰り返し行い，マルチタスク構造を構築する方針とする.

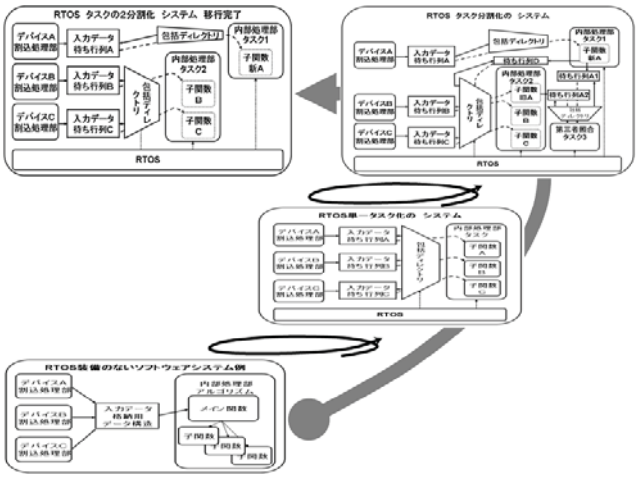
まず，第一段階の特性分離において，第一環では，対象ソフトウェアの制御構造を解析し，内部処理部と割込処理部の論理領域を識別する．論理制御構造にしたがって分割すべきモジュール群を特定すると，タスク内部に格納する内部処理部と割込みベクタテーブルに配置する割込処理部を決定できる．第二環以降では，内部処理部の論理制御構造にしたがってタスク分割すべきモジュール群を特定する.

第二段階の環境移行において，第一環では，デバイスに従属する各割込処理部と内部処理部間のデータフローに待ち行列群を設定する．待ち行列群を格納する包括ディレクトリを定義することにより，タスクが格納した内部処理アルゴリズムに受信データを分配するデータフローを決定して，内部処理部のメイン関数にある受信データ分配機能は撤去して RTOS のタスク起動機構からタスク制御コールを直接受ける, 単一タスク方式の組込みシステムを構築する．第二環以降では，リアルタイム性能の向上のために計算量の短縮化を図るタスク分割を行い，タスク間通信に基づく待ち行列とデータフローを決定する.

第三段階の移行前後実体照合において，第一環では，デバイス割込み優先順位と包括ディレクトリの待ち行列優先順位との整合性を確認し，単一タスク方式実装の実行性能を評価し，移行前のシステムと性能比較を検証する．第二環以降では, タスク内部アルゴリズムのタスク分割を行う．旧内部アルゴリズムと分割先タスクの新アルゴリズムの処理結果を確認するための新設した第三者照合タスクも連携できる動作環境を構築して，旧処理と新処理の動作照合をリアルタイムに検査する.

第四段階の移行前実体撤去検証において，第三段階で分割した機能に係る旧処理部を撤去した状態でシステム動作を確認する.

図 6　段階的移行(MPS) Reengineering のスパイラル例



6. 今後の目標と課題

本稿では，アイデアの提示に終始し，実証実験の成果に基づく検証を示すことができなかったことを深く反省材料とする．

本研究の今後は，組込みシステム開発において，ソフトウェア方式設計の容易化，ソフトウェア例外処理系設計の一貫性，そしてソフトウェア保守の容易化を目指すオペレーティングシステムを追求していくと共に，更に，小数の開発要員体制においても可視化に基づく要求事項の追跡可能性を高めていける開発プラットフォームを構築することを目標としたい．

当面の課題は，本稿で示したアイデアについて，まずは小規模 CPU 環境において動作できるリアルタイム・モニターを構築して，実証実験を行うものとする．その検証テーマには，①冗長化包括受信方式タスク起動機構(RCA: Redundant and Comprehensive incoming Architecture)の効率性評価，②段階的移行(MPS: Migration Path Spirals) Reengineering によるタスク起動優先順位保持の妥当性評価，③パッシブ・タスク方式によるコンテキスト切替え無きタスク間通信(PTA: Passive Tasking Architecture)の実現性評価，④マルチ CPU 方式における冗長化包括受信方式タスク起動機構(RCA)の妥当性評価と問題点の抽出とする．

参考文献

1) IPA 技術本部ソフトウェア・エンジニアリング・センター: 2011 年度「ソフトウェア産業の実態把握に関する調査」報告書, 独立行政法人情報処理推進機構(2012)
http://sec.ipa.go.jp/reports/20120427.html
2) Mike Holenderski, Reinder J. Bril and Johan J. Lukkien: Parallel-Task Scheduling on Multiple Resources, the 24th Euromicro Conference on Real-Time Systems, 2012
3) Francis M. David, Jeffrey C. Carlyle and Roy H. Campbell: Context Switch Overheads for Linux on ARM Platforms, Proceedings of the 2007 workshop on Experimental Computer Science, 2007, San Diego CA

ⓒ2013 Information Processing Society of Japan