C-027

ベアメタルから RTOS 対応ソフトウェアへの移行設計手法
The design methods for migrating software from bare metal to RTOS

奥村　彩花†　　澤田　孝雄†　　玉田　竜一†
Ayaka Okumura　Takao Sawada　Ryuichi Tamada

1. はじめに

近年の組込機器の IoT 化に伴い，従来はベアメタル（OS 未搭載）でソフトウェアを実装していた組込機器にも，ネットワーク機能等の高機能化が求められている．特に，非同期で動作するネットワーク機能はタイミング制御設計が複雑となるため，RTOS を導入することで設計容易化だけではなく保守性や拡張性の向上を図ることができる．一方で，ベアメタルで開発していた開発者にとっては，RTOS 導入に伴うソフトウェア設計の変更や新たな設計手法の習得が大きな課題となる．

本稿では，ベアメタルから RTOS への移行に伴う課題を整理し，RTOS 適用の判断基準と効果的な移行設計手法を報告する．

2. ベアメタルと RTOS 対応ソフトウェア

ベアメタルは，高速な制御が可能で，シンプルな構成や低リソース消費といった利点を持つ．一方で，ソフトウェアの規模が大きくなる場合，RTOS を利用することでマルチタスク設計が容易になり，複雑な非同期処理の設計が効率化される．これにより，ソースコードの保守性や拡張性が向上する．ただし，ベアメタルから RTOS 対応ソフトウェアへ移行する際には，リソース使用量の増加や RTOS のオーバーヘッドによる処理時間の増加といった点を考慮した上で適用を検討する必要がある．

ベアメタルと RTOS 対応ソフトウェアの比較を表 1 に示す．

表 1　ベアメタルと RTOS 対応ソフトウェア比較

特徴	ベアメタル	RTOS
プログラム構成	単一ループ処理 または割込みの組合せ	マルチタスク
リソース使用量	少ない	多い
開発規模	小規模開発向き	中規模開発向き
移植性	低い	高い
高速制御	非常に高速	高速
用途	シンプルなシステム	複雑なシステム

3. RTOS の適用判断

RTOS の適用を検討する際の判断基準の例を下記に示す．

① リソース（ROM/RAM），CPU 使用率に余裕がある
② 制御周期が 1ms 以上
③ プログラムサイズ（バイナリ）が 64KB 以上
④ 割込み処理が多く，処理遅延や抜けが発生
⑤ 機能が 5 種類以上あり，多機能化している
⑥ ネットワーク通信やセキュリティ機能の追加が必要

これらを踏まえ RTOS の適用可否を判断する．

† 富士電機株式会社　Fuji Electric Co., Ltd.

RTOS を適用する場合の選定においては，システムの要件（機能，性能，品質，コスト）を満たすことを適用条件と合わせて確認する必要がある．

4. ベアメタルから RTOS へ移行する際の課題

当社組込機器製品において，IoT 対応などの機能拡張時の設計容易化による開発工数削減を狙い，既存のベアメタル資産を活用し RTOS 対応ソフトウェアへの移行設計を行った際に生じた課題を下記に示す．

① マルチタスク設計への移行
ベアメタルの単一ループ，複数の割込み処理構成から，複数のタスクが独立して実装されているマルチタスク構成へ移行するために，タスクの分割，適切なスケジューリングを行う必要がある．

② タスク間のインタフェース設計
タスク間で通信を行う場合は，使用する OS 機能に応じてインタフェース設計，既存処理の設計変更を行う必要がある．

③ 共有リソース／共通関数の排他処理設計
複数のタスクからアクセスする共有リソースや共通関数のデータ・処理の不整合が発生しないように排他処理設計を行う必要がある．

5. RTOS 対応ソフトウェアへの移行設計手法

本章では 4 章で挙げた課題に対する移行設計手法を示す．

5.1 マルチタスク設計方法への移行

ベアメタルのプログラム設計から RTOS に対応したマルチタスク設計へ移行する際は，タスクの構造分析，タスク候補の抽出，タスク化，タスク処理の設計，タスク優先度とタスクスタックサイズの設定を行う必要がある．以下，順を追って説明する．

5.1.1 タスクの構造分析とタスク候補の抽出

まず，既存資産のプログラム構造分析を行う．

分析の際は，メイン関数や割込み処理の最上位を起点として，関数内の処理が概ね関数の呼び出しのみで構成される 3 階層程度までの処理（関数）を目安として各処理（関数）の所属機能，実行タイミングを整理し，必要に応じて処理間のシーケンスやタイミングチャート，処理の呼び出し関係も確認する．

分析の結果から，時間的制約により処理の遅延や分割，順番の変更が許されない処理を順次処理が必要な処理，それ以外を並行動作可能な処理に分類する．

並行動作可能な処理をタスク化する候補として抽出する．

5.1.2 既存処理のタスク化

タスク候補の処理（関数）を機能毎にまとめて 1 つのタスクとする．ただし，機能毎にまとめた処理の中でも動作タイミングが異なり，統合せずに分けた方が良い場合には別タスクとする．これにより，実行タイミングの管理が容易になる．

また，順次処理が必要な場合でも，処理を分離し一部をタスク化できる場合がある．例えば，シリアル通信の受信割込み処理において，RAM へのデータコピーとその後のデータ処理で構成されている場合，データ処理部分をタスクとする．この設計により，割込み処理の負荷を軽減することが可能となる．設計変更の例を図 1 に示す．

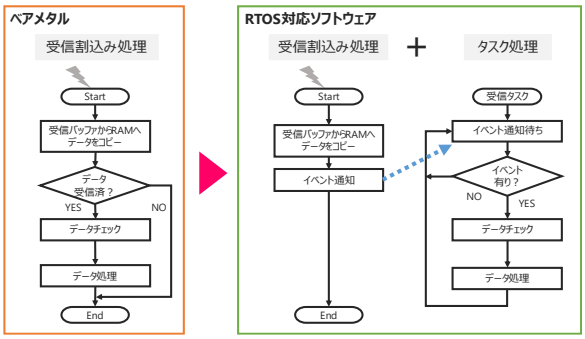

図 1　シリアル通信の受信割込み処理のタスク分離例

5.1.3 タスク処理の設計

タスク化する処理をベアメタル環境での動作に基づき分類する．一定間隔ごとに実行される処理は定周期動作，処理要求のフラグが立っていたら動作する処理をイベントドリブン動作として分類して，設計を行う．

以下に，各動作の基本処理構造について示す．

（1）定周期タスク

①自タスク内での遅延動作

定周期動作を自タスク内で遅延動作させる場合は，任意の処理後に指定時間遅延（ウェイト）し，再び任意の処理に戻り処理を繰り返し行う，といった無限ループの処理構造とする．

②自タスク以外からの定周期起動

定周期動作を自タスク以外から起動して動作させる場合は，OS タイマ機能を使用し，起動時間（周期）と動作させたい処理を設定しておき，設定時間経過後にデータ処理を呼び出すという構造とする．

または，後述のイベントドリブンタスクの構造とし，定周期でイベント通知を受けて動作する設計とする．

（2）イベントドリブンタスク

イベント通知待ち状態で，通知を受けた後に処理を行い，再びイベント通知待ち状態となる，といった無限ループの処理構造とする．

イベント通知待ちの処理は，同期・非同期，データ有無の組み合わせにより，利用する OS 機能を選択する．利用する OS 機能例を表 2 に示す．

表 2　イベント通知待ちで利用する OS 機能例

同期・非同期	データ有無	利用 OS 機能
同期	データ無し	・ イベントフラグ ・ セマフォ
	データ有り	・ メッセージ通信
非同期	データ無し	・ イベントフラグ
	データ有り	・ データキュー

5.1.4 タスク優先度，タスクスタックサイズの設定

RTOS のスケジューリングは一般的に優先度ベースが主流であり，各タスクに優先度を設定する．例えば，時間的制約の厳しいタスクから順に高い優先度を設定する．

また，RTOS では，各タスクに独立したスタック領域が割り当てられるため，タスクスタックサイズの設定が必要となる．スタックサイズの見積り方法を以下に示す．

タスクの最上位関数から最下位までの関数ツリーを作成し，各関数のスタックサイズの合計を求める．次に，コンテキストレジスタのサイズを加算し，今後の処理追加によるスタック使用量の増加を考慮し，30%以上のマージンを持たせてスタックサイズを見積もる．

尚，関数の呼び出しに関数ポインタを使用している場合は，関数ツリーに含まれず，スタックサイズの計算から漏れてしまう可能性があるため，注意する．

5.2 タスク間インタフェース設計

タスク間やタスクー割込み間で通知や通信等を行う場合は，使用する OS 機能に応じてインタフェース設計を行う．既存の割込み処理などはインタフェースの設計変更を行う．

タスク間のインタフェースについては 5.1.3 項にて説明した，タスクで使用する OS 機能に合わせて設計する．

タスクー割込み処理間のインタフェースはタスク側で使用する OS 機能に合わせて設計する．

5.3 共有リソース／共通関数のアクセス制御設計

複数のタスクまたは割込み処理からアクセスされるソフトウェア（共有変数）およびハードウェア（I/O レジスタ等）の共有リソースに対する排他制御設計を行う．

（1）ソフトウェア共有リソース

データの共有変数へのアクセスは，排他処理を含む関数を作成し，書き込み処理を局所化する．

フラグ変数は OS のイベントフラグ機能へ置き換える．

（2）ハードウェア共有リソース

I/O レジスタへの単純なアクセスでは，セマフォや割込み禁止を用いて排他制御を行う．I/O レジスタ経由で通信デバイスとアクセスする場合，セマフォやミューテックスで通信単位の排他制御，データキューやイベントフラグで通信要求を管理する．

尚，セマフォまたはミューテックスによる排他処理設計時はデッドロックが発生しないように，排他処理の順序に注意する．

共通関数については，リエントラント性やスレッドセーフ対応を再確認した上で排他処理の要否を判定し，必要に応じて排他処理を追加する．

6. おわりに

本稿では，既存のソフトウェア資産の活用を前提としたベアメタルから RTOS 対応ソフトウェアへの移行設計手法を提示した．RTOS 導入により，ソフトウェア設計の効率化や保守性の向上が期待できるが，導入時には RTOS の特性を十分に理解した上で，適切な設計を行う必要がある．

参考文献

[1] IPA, “組込みソフトウェア向け設計ガイド ESDR[事例編]”, 1 版 1 刷 (2012).
[2] TRON Forum, “ITRON 入門（中級編）中級者向けの RTOS を使ったリアルタイムシステム開発手法入門”, (2016).