文書番号：NGKA_REQ-02-301

NCES

次世代車載システム向け RTOS
ハードウェア要求仕様書

Ver.3.0.1

2013/3/12

名古屋大学大学院情報科学研究科附属 組込みシステム研究センター

Copyright (C) 2011-2014 by Center for Embedded Computing Systems
Graduate School of Information Science, Nagoya Univ., JAPAN
Copyright (C) 2011-2014 by FUJISOFT INCORPORATED, JAPAN
Copyright (C) 2011-2013 by Spansion LLC, USA
Copyright (C) 2011-2013 by NEC Communication Systems, Ltd., JAPAN
Copyright (C) 2011-2014 by Panasonic Advanced Technology Development Co., Ltd., JAPAN
Copyright (C) 2011-2014 by Renesas Electronics Corporation, JAPAN
Copyright (C) 2011-2014 by Sunny Giken Inc., JAPAN
Copyright (C) 2011-2014 by TOSHIBA CORPORATION, JAPAN
Copyright (C) 2011-2014 by Witz Corporation, JAPAN

上記著作権者は，以下の (1)～(3)の条件を満たす場合に限り，本ドキュメント（本ドキュメントを改変したものを含む．以下同じ）を使用・複製・改変・再配布（以下，利用と呼ぶ）することを無償で許諾する．

(1) 本ドキュメントを利用する場合には，上記の著作権表示，この利用条件および下記の無保証規定が，そのままの形でドキュメント中に含まれていること．

(2) 本ドキュメントを改変する場合には，ドキュメントを改変した旨の記述を，改変後のドキュメント中に含めること．ただし，改変後のドキュメントが，TOPPERS プロジェクト指定の開発成果物である場合には，この限りではない．

(3) 本ドキュメントの利用により直接的または間接的に生じるいかなる損害からも，上記著作権者および TOPPERS プロジェクトを免責すること．また，本ドキュメントのユーザまたはエンドユーザからのいかなる理由に基づく請求からも，上記著作権者および TOPPERS プロジェクトを免責すること．

本ドキュメントは，AUTOSAR（AUTomotive Open System ARchitecture）仕様に基づいている．上記の許諾は，AUTOSAR の知的財産権を許諾するものではない．AUTOSAR は，AUTOSAR 仕様に基づいたソフトウェアを商用目的で利用する者に対して，AUTOSAR パートナーになることを求めている．

本ドキュメントは，無保証で提供されているものである．上記著作権者および TOPPERS プロジェクトは，本ドキュメントに関して，特定の使用目的に対する適合性も含めて，いかなる保証も行わない．また，本ドキュメントの利用により直接的または間接的に生じたいかなる損害に関しても，その責任を負わない．

＜目次＞

1. 概要

1.1　本文書の目的

本文書は「次世代車載システム向け RTOS 外部仕様書」で規定された OS が動作するために必要とするハードウェアのうち，重点としている機能(マルチコアシステム，保護機能)に関するハードウェアの要件を規定した仕様書である．

なお，基本的な OS 機能(シングルコアシステム，OSEK/VDX 相当機能)の実現に必要なハードウェアの条件に関しては本文書の対象外である．

1.2　関連文書

文書名	バージョン
次世代車載システム向け RTOS 外部仕様書	Ver.3.0.0
次世代車載システム向け RTOS 用語集	Ver.3.0.0

1.3　凡例

本文書では各要件に対して以下に示す要件番号を付加して管理を行う．

要件番号	内容
【HWxxx】	個々のハードウェア要件番号. 各々のハードウェア要素, 要件のレベル(後述)ごと，要件として規定する観点ごとに個別の番号を付与する．

2. 検討方針

2.1　検討の流れ

一般に，ハードウェア(CPU，周辺回路)の構成や機能は多種多様であり，OS の機能と性能に大きな影響を与える. 次世代車載システム向け RTOS の機能要件と性能要件を満たすことが可能なハードウェア要求仕様を規定するために，以下のような手順で検討を実施した.

ハードウェア・アーキテクチャと OS の関連性の検討

次世代車載システム向け RTOS において重視する機能(マルチコアシステム，保護機能)に関連するハードウェアの要素ごとに，OS の機能や実装(性能)との関連性を明確化した.

ハードウェア要件の規定

前段階で検討したハードウェア・アーキテクチャと OS の関連性を踏まえ，OS が要求される機能要件と性能要件を満たすためのハードウェアの機能要件を規定した.

実装，評価，フィードバック

規定したハードウェア要件に従ったハードウェアを用いて OS の実装と評価を行った．その過程において，性能向上のためのハードウェア要件を定め，これらを「性能改善のための要件」として追加した.

2.2　要件のレベル分け

ハードウェアに対する要件を，次世代車載システム向け RTOS を動作させるために必要な度合いに応じて 3 段階に分けて定義した.

- 最低限の要件(MR：Minimal Requirements)

  次世代車載システム向け RTOS の最低限の機能(機能レベルを規定しているものに関しては，そのレベル 1 の機能)を動作させるために必要となる要件．この要件を満たさないハードウェアには，次世代車載システム向け RTOS を実装することができない.

- 標準的な要件(SR：Standard Requirements)

  次世代車載システム向け RTOS のすべての機能を実用的な性能で動作させるために必要となる要件．この要件を満たさないハードウェアでは，次世代車載システム向け RTOS のすべての機能が実装できないか，実装できたとしても性能的な問題が生じる可能性がある.

- 性能改善のための要件(AR：Advanced Requirements)

  次世代車載システム向け RTOS を高い性能で動作させるために満たすことが望ましい要件．この要件を満たすハードウェア上に次世代車載システム向け RTOS を実装することで，性能向上(特に RTOS のオーバヘッドの低減)が期待できる.

2.3　OS アーキテクチャの前提

次世代車載システム向け RTOS のアーキテクチャの概要を述べる．詳細は「次世代車載システム向け RTOS 外部仕様書」を参照のこと．

OS 構成：1 リンクモデル

OS として搭載するすべての機能を単一のメモリ空間で実装し，オーバヘッド，フットプリントの軽量化を図る．

マルチコア対応 OS でのシステムサービスの実装方式：直接操作法

マルチコアシステムにおけるコアを跨いだ(システムサービス発行元のコアと，操作対象オブジェクトが割り付けられているコアが異なる)システムサービス発行の実現方式として，システムサービス発行元コアのカーネルが，操作対象オブジェクトの管理のための内部データを直接操作する方式をとる．

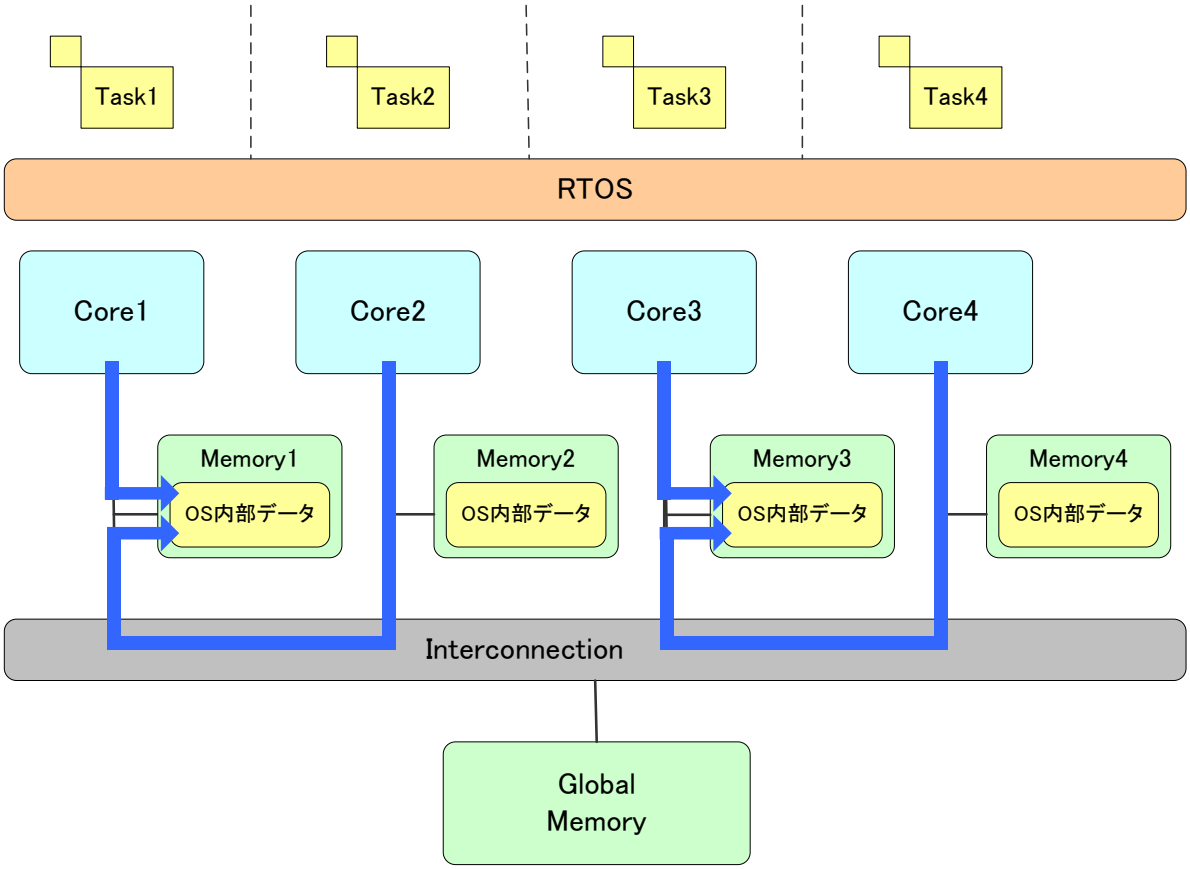

図 2-1　直接操作法によるシステムサービス処理

3. 要件

3.1　OS 実装機能と要件一覧

OS が実装する機能ごとに対応が必要なハードウェア要件の一覧を表 3-1 に示す.

なお，SC1 に相当する基本的な OS 機能の実現に必要なハードウェアの条件に関しては本文書の対象外である．従って，SC1 相当の OS 機能を実現可能なハードウェアの要件は満たしていることを前提とする．ただし，【HW020】は SC1 相当の要件であるが，SC2, MC 要件の前提となる要件なので本文書で規定する.

SC4 は，SC2 と SC3 を組み合わせたスケーラビリティクラスであるので，各一覧には記載しない．同様に，SC4-MC は SC2-MC と SC3-MC の組み合わせであるので，記載しない.

表 3-1 OS 実装機能ごとに対応が必要なハードウェア要件一覧

要件 / OS 実装機能 HW 要素	要件レベル	要件番号	SC2	SC3	SC1-MC	SC2-MC	SC3-MC
コア個数	SR	【HW001】			○	○	○
コア構成	SR	【HW002】			○	○	○
コア接続	SR	【HW003】			○	○	○
コア間排他制御機構	MR	【HW004】			○	○	○
	SR	【HW005】			○	○	○
		【HW006】			○	○	○
	AR	【HW007】			○	○	○
		【HW008】			○	○	○
コア ID	SR	【HW009】			○	○	○
	AR	【HW010】			○	○	○
メモリ	MR	【HW011】			○	○	○
	SR	【HW012】			○	○	○
	AR	【HW013】			○	○	○
		【HW014】			○	○	○
キャッシュ	SR	【HW015】			○	○	○
割込み	MR	【HW016】		○			○
		【HW017】			○	○	○
	SR	【HW018】			○	○	○
	AR	【HW019】			○	○	○
タイマ	MR	【HW020】	○	○	○	○	○
		【HW021】	○				
		【HW022】			○		
		【HW023】				○	
	SR	【HW024】	○				
		【HW025】	○				
		【HW026】				○	

要件 / OS 実装機能			SC2	SC3	SC1-MC	SC2-MC	SC3-MC
HW 要素	要件レベル	要件番号					
		【HW027】				○	
	AR	【HW028】	○			○	
MPU	MR	【HW029】		○			
		【HW030】		○			
		【HW031】		○			
		【HW032】		○			
		【HW033】		○			
		【HW034】		○			
		【HW035】		○			
		【HW036】		○			
		【HW037】					○
	SR	【HW038】		○			○
		【HW039】		○			○
	AR	【HW040】		○			○
		【HW041】		○			○

OS 実装機能：スケーラビリティクラス(SC2,3,4)ならびにマルチコア対応(MC)を表す.

○：OS 実装機能ごとに，その OS が動作するために，ハードウェアが準拠しなければならない要件を表す.

3.2　コアの個数

マルチコアシステムを構成するコアの個数を規定する．本節の規定はすべてマルチコアシステムに対する要件である．

3.2.1　標準的な要件【SR】

マルチコアシステムには，最小で 2 個，最大で 4 個のコアを搭載すること【HW001】.

【要件決定の理由】

OS がサポート可能な最大のコア数はデータ構造とアルゴリズムの設計に影響する．

マルチコア対応 OS の制御対象は，現在は主にシングルコアシステムで搭載されているリアルタイム制御である．現在のシステムからの性能向上，ならびに動作周波数や消費電力の抑制を目的としたマルチコアシステムにおいては，2 個から 4 個程度のコアを搭載したハードウェアが最も有力であると捉えている．

3.3　コアの構成

マルチコアシステムを構成するコアの組み合わせを規定する．本節の規定はすべてマルチコアシステムに対する要件である．

3.3.1　標準的な要件【SR】

マルチコアシステムを構成するコアは，同一の命令セットやエンディアンをもつ，いわゆるホモジニアス構成であること【HW002】.

【要件決定の理由】

各コアで同一の OS やアプリケーションのバイナリを共有するために必要な要件である．この要件を満たさない場合は，コア毎に異なる OS やアプリケーションのバイナリが必要となることや，コアを跨いだシステムサービスの実現が困難となるために定めた．

3.4　コアの接続形態

マルチコアシステムを構成するコア間の接続形態について規定する．本節の規定はすべてマルチコアシステムに対する要件である．

3.4.1　標準的な要件【SR】

密結合マルチコアシステムであること．すなわち，システム中のいずれのコアからもアクセス可能な共有メモリが存在すること【HW003】.

【要件決定の理由】

一般に，共有メモリを持たない疎結合マルチコアシステムにおけるコア間通信はネットワークを介することになる．

密結合マルチコアシステムと疎結合マルチコアシステムを比較すると，密結合マルチコアシステムはコア間で共有する情報に対して各コアが直接アクセスすることが可能であり，一般的にアクセスレイテンシが小さい．しかしながら，それに対し，複数のコアが同一の情報に対して同時にアクセスしないようコア間排他制御が必要となる．

一方，疎結合マルチコアシステムはその情報を保持しているコアに対してアクセスを依頼する必要があるため，一般的にアクセスレイテンシが大きい．しかしながら，情報へのアクセスに対してコア間の排他制御は必要ない．

このように，両者は一長一短ではあるが，共有メモリへのアクセスレイテンシが小さければ，密結合マルチコアシステムの方が，応答時間やリアルタイム性の点で有利であることを考慮して，密結合マルチコアシステムを前提とする．

3.5　コア間排他制御機構

マルチコア対応 OS の実現に必要な排他制御機構の要件を規定する．本節の規定はすべてマルチコアシステムに対する要件である．

コア間排他制御の必要性

直接操作法によって構成される OS では，複数のコアが OS 内部の同一のデータを同時にアクセスする状況が起こりうるため，コア内の排他制御(割込み禁止)に加えてコア間排他制御機構が必要となる．図 3-1 はマルチコア対応 OS におけるコア間排他制御の実装の一例である．

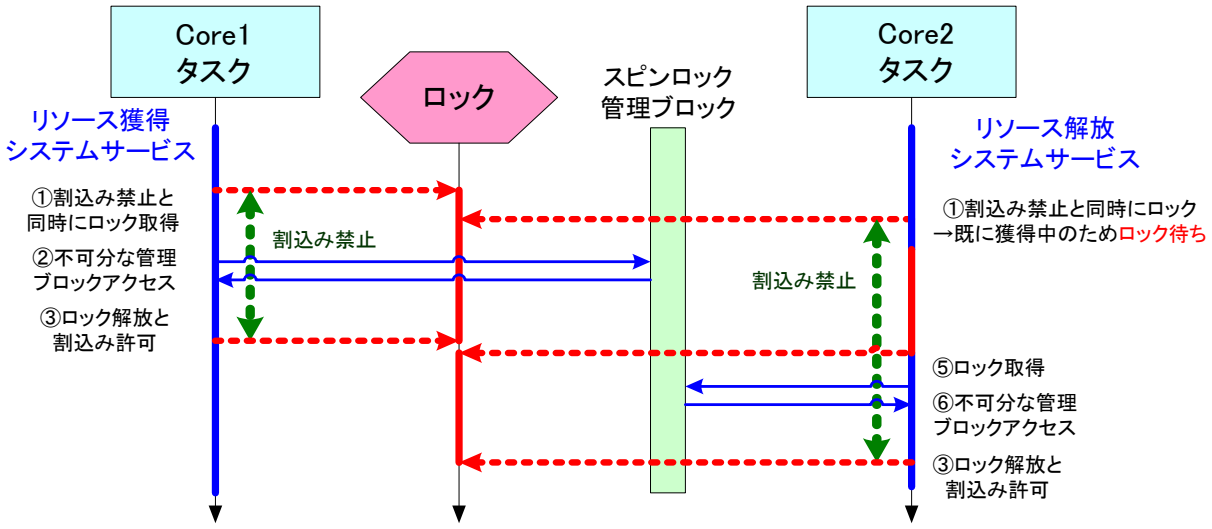

図 3-1　コア間排他制御の例

コア間排他制御の実現方法

コア間排他制御の実現方法として，大きく以下の 2 種類が考えられる．

・ アトミック命令による実現

アトミック命令によって，共有メモリ上のある領域をロックの実体として使用する．ロックの実体の個数制限は，アトミック命令によるアクセスが許されている共有メモリの容量によるため，実質的にほぼ無制限にロック実体を確保可能だが，ロック取得のロジックをソフトウェアによって実現する必要があり，レイテンシの増加につながる．

・ 排他制御専用ハードウェアによる実現

ロックを周辺回路(ハードウェア)として実現するもの．ハードウェアで実現するため，ロックの個数に制限が課される場合が多い．周辺回路へのアクセスを共有メモリの接続されているバスと別構成とすることで，排他制御のレイテンシを低減させることができる．または，応答時間の特性の良いロックアルゴリズムをハードウェア実装することでも，排他制御のレイテンシを低減させることができる．

3.5.1　最低限の要件【MR】

システム全体で 1 個のロックが実現可能であること．つまり，命令セットに何らかのアトミック命令が搭載されている，もしくは排他制御専用ハードウェアを最低 1 個搭載していること【HW004】.

【要件決定の理由】
マルチコアシステム全体の OS 内部データを一括して排他制御すること(ジャイアントロック方式)が最低限の実装であり，排他制御の単位(ロック単位)を 1 個設定できることが最低限の要件となる．

3.5.2　標準的な要件【SR】

<u>コア数 3 個以上のシステムにおける，OS の最悪実行時間の上限の保証のための要件</u>

コア数が 3 個以上のマルチコアシステムでは，アトミック命令として，Compare and Swap(以降，CAS と呼ぶ)もしくは Load-Link/Store-Conditional(以降，LL/SC と呼ぶ)を持つこと【HW005】.
（この要件は「最低限の要件」に対する追加である．なお，専用ハードウェアを用いる方法の場合の要件は「性能改善のための要件」としている．）

【要件決定の理由】
最も単純なコア間排他制御アルゴリズムは「Test and Set スピンロック」である．単純でかつ低オーバヘッドという利点があるが，コア数が 3 個以上の場合にロック競合時のロック獲得順序がランダムとなってしまい，OS 処理の最悪実行時間の上限が保証できなくなる．
最悪実行時間の上限の保証のために，ロック獲得順序を到着順によって管理する「キューイングスピンロック」の導入が効果を示す．キューイングスピンロックの実現には CAS や LL/SC といった，より高度なアトミック命令が必要となる．

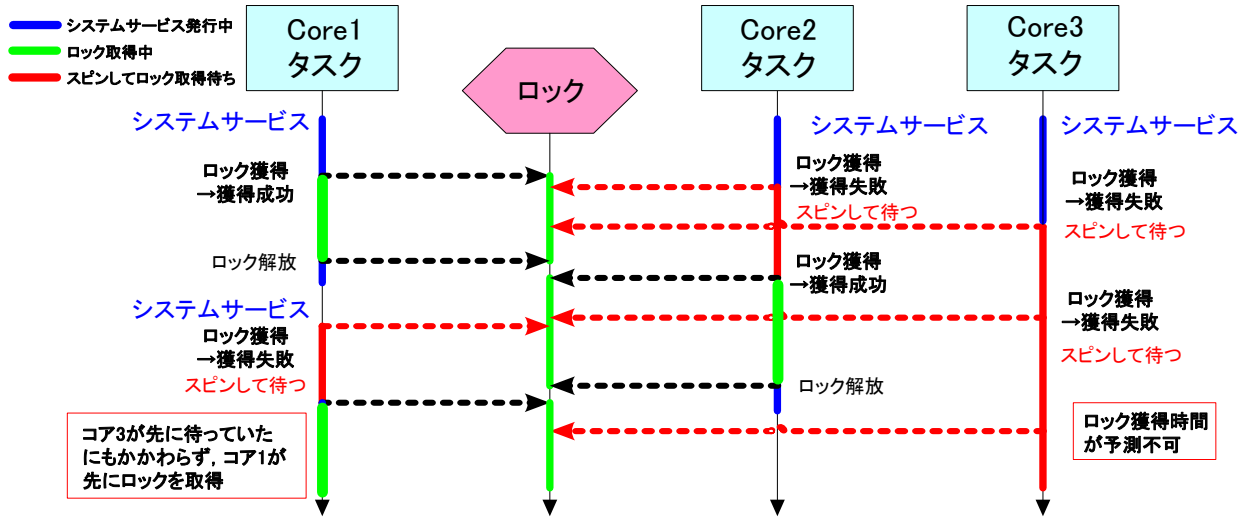

図 3-2　Test and Set スピンロックの問題点

OS 処理におけるコア間の並列性の向上，競合の低減のための要件

コアごとに 2 個のロックと，システム全体で 2 個のロックが実現可能であること．具体的には，命令セットに何らかのアトミック命令が搭載されているか，もしくは排他制御専用ハードウェアを最低でも(コア数×2)＋2 個搭載していること【HW006】．なお，ソフトウェアでロックを実現する場合は，排他制御専用ハードウェアを最低 1 個搭載していればよい．(この要件は「最低限の要件」を包含している．)

【要件決定の理由】

ジャイアントロック方式の場合，システム中の複数のコアが同時にシステムサービス処理を実行することが不能になるため，コア間における OS 処理の並列性がなくなる．同時に複数のコアで並列して OS 処理を実行可能とするためには，ロック単位を分割する必要がある．コア間の並列性を確保しつつ，デッドロックが発生しないロック単位の設定を検討し，(コア数×2)＋2 個という要件を導出した．コアごとに 2 個である理由は，タスク，カウンタに対するロックが必要であるからで，さらに 2 個必要である理由は，スピンロック，IOC に対するロックである．

なお，CPU がアトミック命令を備えている場合には，その命令がアクセスを許されるメモリの容量の分だけロック単位を設定可能ということになり，個数の面での問題はない．

3.5.3　性能改善のための要件【AR】

OS 処理におけるコア間の並列性の向上，競合の低減のための要件

コアごとに 1 個のロックに加え，カウンタオブジェクトごとに 1 個，コア間排他制御オブジェクト(リソースオブジェクト，ミューテックスオブジェクト)ごとに 1 個のロックが実現可能であること．具体的には，命令セットに何らかのアトミック命令が搭載されているか，もしくは排他制御専用ハードウェアを(コア数＋カウンタオブジェクト数＋コア間排他制御オブジェクト数)個搭載していること【HW007】．なお，ソフトウェアでロックを実現する場合は，排他制御専用ハードウェアを最低 1 個搭載していればよい．

(この要件は「最低限の要件」を包含している．この要件を満たせば、【HW006】を満たす必要はない．)

【要件決定の理由】

標準的な要件からさらに個数を増やし，コア間排他制御機能のオブジェクト，カウンタオブジェクト個別のロック単位に分割することで，OS 処理の並列性は向上する．

コア数 3 個以上のシステムにおける，OS の最悪実行時間の上限の保証のための要件(ロックのオーバヘッド削減)

キューイングスピンロックアルゴリズムを実現可能な排他制御ハードウェアを搭載していること【HW008】．

(この要件は「最低限の要件」に対する追加である．なお，アトミック命令を用いた方法による場合の要件は「標準的な要件」としている．)

要件を満たすハードウェア仕様の詳細は 5 章を参照のこと．

3.6　コア ID

マルチコアシステムを構成するコアをソフトウェアから識別するためのハードウェアに関する要件を規定する．本節の規定はすべてマルチコアシステムに対する要件である．

3.6.1　標準的な要件【SR】

マルチコアシステムを構成する各々のコア固有の定数値(コア ID)をソフトウェアから何らかの方法で取得できること【HW009】.

【要件決定の理由】

OS の実行コードをすべてのコアで共有する場合には, OS の実行コードがどのコアで動作しているかを各々のコアで判別する必要があるため．ソフトウェアからコア ID が判別できない場合には，コアごとに個別の OS 実行コードを用意する必要があり，必要となる ROM サイズが増加する．また，コアごとに個別の OS 実行コードとした場合，共有変数の実現が困難である．

3.6.2　性能改善のための要件【AR】

コア ID の値として，コアごとに異なる RAM 上の特定のアドレスを指し示すこと．具体的には，読出し専用レジスタにメモリ領域の先頭アドレスを保持するか，コア ID とは別にコアごとに任意のアドレス値をソフトウェアから書込み，読出し可能なレジスタを持つこと【HW010】.

（この要件は標準的な要件を包含している．）

【要件決定の理由】

コア ID の値の体系として最も一般的なのは, 連続した整数値(1~コア数もしくは 0~コア数-1)である．しかし，OS ではコア ID 番号をそのまま使用することは少なく，コア ID 番号からコア固有データへアクセスするためにアドレス計算を行う．コア固有データへのアクセス頻度は高く，アクセスする度にアドレス計算を行うのはオーバヘッドの増大に繋がる．

このようなアドレス演算のオーバヘッドを低減するために，コアごとにコア固有データの先頭アドレスを保持するのが有効である．アドレス値を保持する方法として，読出し専用レジスタとして搭載する方法のほか, 読出し専用レジスタにはコア ID 番号を保持し, コア固有データのアドレス専用の読出し，書込み可能レジスタを OS 予約とする方法がある．この，コア固有データアドレス専用のレジスタはコンパイラの呼出し規約に影響を与えないよう，汎用レジスタとは別の専用レジスタとして装備されていることが望ましい．

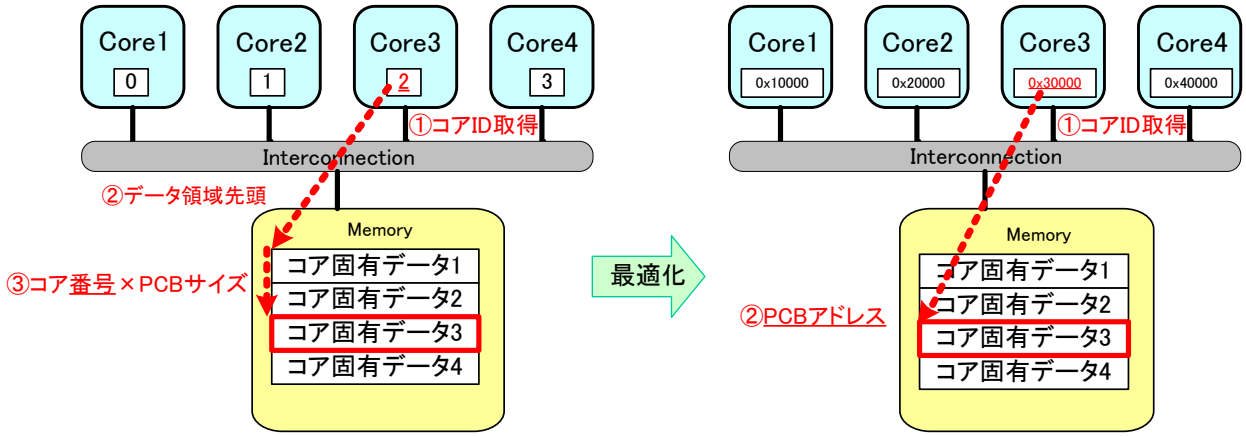

図 3-3　コア ID の値による最適化

3.7　メモリアーキテクチャのモデル

マルチコアシステムを構成するハードウェアのメモリについて規定する．本節の規定はすべてマルチコアシステムに対する要件である．

本文書ならびに「次世代車載システム向け RTOS 外部仕様書」においては，マルチコア環境におけるメモリアーキテクチャの説明のために，以下のようにアーキテクチャをモデル化する．

共有メモリとバスアーキテクチャ

- プライベートメモリ(共有不可メモリ)

  特定のコアからのみアクセス可能な共有不可メモリ．

- ローカル/リモートメモリ(共有メモリ)

  すべてのコアからアクセス可能な共有メモリ．ただし，アクセスレイテンシは特定のコアからのアクセス(ローカルメモリへのアクセス)において最も低く，それ以外のコアからのアクセス(リモートメモリへのアクセス)はグローバルなバスを介するため，相対的に高レイテンシである．

- グローバルメモリ(共有メモリ)

  すべてのコアからアクセス可能な共有メモリ．グローバルなバスアクセスを伴い，プライベートメモリやローカルメモリへのアクセスよりも相対的に高レイテンシである．ただし，すべてのコアからのアクセスにおいて同等のレイテンシである．

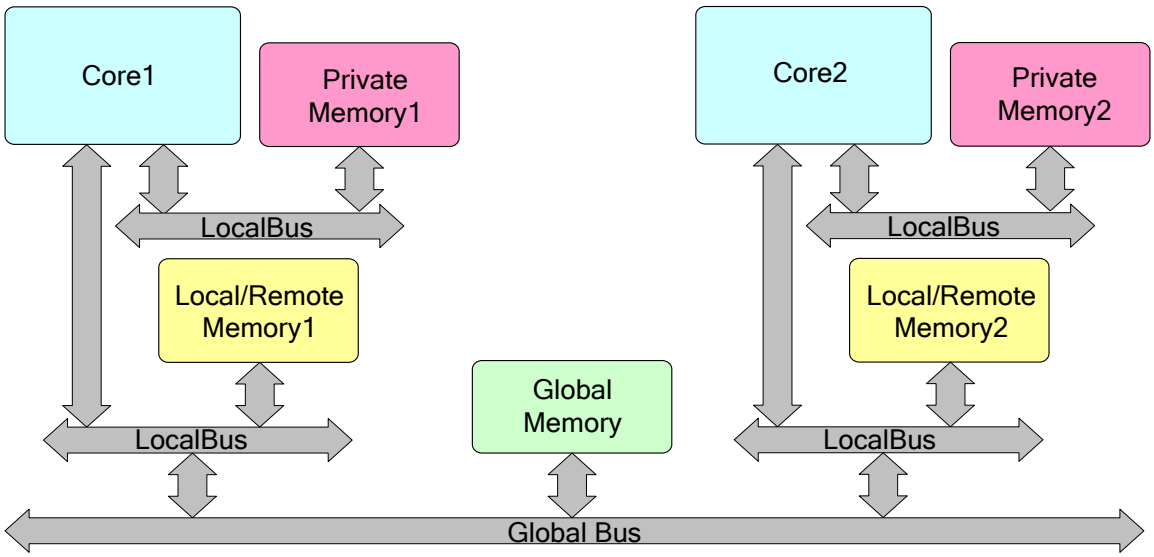

図 3-4　メモリアーキテクチャのモデル

メモリアドレスアーキテクチャ

・ Uniform アドレス

共有メモリをすべてのコアから同じアドレスでアクセスする．(図 3-5 中の Memory1-1 と Memory2-1)

・ Overlay アドレス

各コアのプライベートメモリ，ローカルメモリを各コアの同一のアドレスにマッピングしてアクセスする．(図 3-5 中の Memory2-1 と Memory2-2)すなわち，このアドレス空間に各コアがアクセスすると，それぞれ異なるメモリ(プライベートメモリ，ローカルメモリ)にアクセスすることになる．

・ Multiform アドレス

すべてのコアのローカルメモリをすべてのコアからリモートメモリとして Uniform アドレスでアクセスできるのと同時に，各コアのローカルメモリを Overlay されたアドレスでもアクセス可能．図 3-5 中の Core1 の 2 つの Memory3 のアドレスにマッピングされたメモリの実体は同じものである．

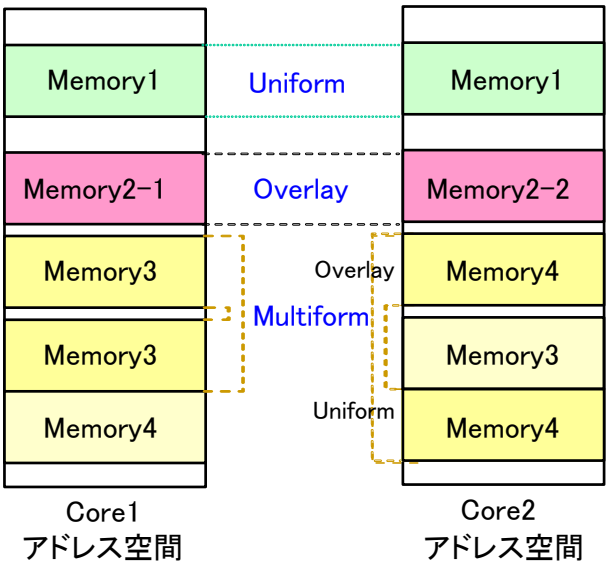

図 3-5　メモリアドレスアーキテクチャ

3.7.1　最低限の要件【MR】

マルチコアシステム中のすべてのコアから，同じアドレスでアクセス(読出し，書込み，実行)可能な RAM が搭載されていること【HW011】.

【要件決定の理由】

直接操作法によって構成される OS では，すべてのコアから OS 内部データを読み書きする可能性があるため，すべてのコアからアクセス可能な RAM が搭載されていることが必須となる．また，すべてのコアからアクセスする個々のデータの所在を一意に特定するために，各コアから同じアドレスでアクセスできる，つまり Uniform アドレスである必要がある．

3.7.2　標準的な要件【SR】

各コアにローカルメモリが搭載されていること【HW012】.

(この要件は最低限の要件に対する追加である．)

【要件決定の理由】

あるデータに対して，特定のコアが高頻度でアクセスする状況では，高頻度にアクセスするコアに付随するローカルメモリにそのデータを配置することによって，GlobalBus(グローバルメモリが接続されているバス)の負荷を軽減することが可能となる．

3.7.3　性能改善のための要件【AR】

<u>コア固有データアクセス最適化のための要件</u>

標準的な要件で規定されるローカルメモリが，Multiform アドレス(Uniform アドレス(リモートメモリ)と Overlay アドレス(ローカル)の両方でアクセス可能)であることが望ましい【HW013】.

(この要件は最低限の要件，標準的な用件に対する追加である．)

【要件決定の理由】

ローカルメモリを示すアドレス空間が各コアにおいて同一となっていれば，OS コードにおけるデータアクセス最適化が可能となる．例えば各コアにおけるコア固有のデータを同じアドレスに配置することで，コア固有データのアドレスを求める操作を省略するような制御が考えられる．

ただし，このような最適化の実現にはリンカの機能に依存するところが大きく，(ハードウェア，ソフトウェア開発環境を含めて)実現可能なターゲットが限られることから，オプショナルの位置づけとした．

バス競合回避のための要件

各コアにプライベートメモリが搭載されていること【HW014】.

【要件決定の理由】

特定のコアのみがアクセスするデータに関しては，共有メモリに配置する必要はない．そのようなデータをプライベートメモリに配置することでグローバルなバスアクセスを回避し，バスアクセス競合を低減させることが可能である．

3.8　キャッシュメモリ

マルチコアシステムにキャッシュメモリが搭載される場合の，キャッシュコントローラに対する要件について規定する．本節の規定はすべてマルチコアシステムに対する要件である．

3.8.1　標準的な要件【SR】

コアごとに独立したデータキャッシュを搭載する場合には，データキャッシュ間のコヒーレンシがハードウェアによって保証されること【HW015】.

【要件決定の理由】

OS の管理する内部データには，共有メモリ上にあり複数のコアからアクセスされるものが多い．それらのデータアクセスに関わるキャッシュメモリのコア間のコヒーレンシを OS によって確保するのは不可能であるため.

3.9　割込み

割込みコントローラとコア間の割込み，外部割込みの接続について規定する．実装する OS 機能ごとに，対応が必要な要件の一覧を表 3-2 に示す.

表 3-2 OS 実装機能ごとに対応が必要な割込みハードウェア要件

要件 ＼ OS 実装機能		SC3	SC1-MC	SC2-MC	SC3-MC
最低限	【HW016】	○			○
	【HW017】		○	○	○
標準	【HW018】		○	○	○
性能改善	【HW019】		○	○	○

○：OS 実装機能(SC/MC)ごとに，その OS が動作するために，ハードウェアが準拠しなければならない要件を表す.

3.9.1　最低限の要件【MR】

<u>OS アプリケーション機能のための要件</u>

割込み要因ごとに割込み許可，禁止を設定できること【HW016】.

【要件決定の理由】

OS 仕様で規定される割込みを個別の要因ごとに禁止，許可する機能(DisableInterruptSource，EnableInterruptSource)を実現するため，割込み要因ごとに割込みの受付を禁止，許可が出来る必要がある.

<u>マルチコアシステムにおける要件</u>

マルチコアシステムを構成する各コアから，他の任意のコアに対して割込み(コア間割込み)を発生することが可能であること【HW017】.

【要件決定の理由】

マルチコア対応 OS の最も基本的な性質として，マルチコアシステムを構成する各コアが協調動作することが必要である．OS 内部のあらゆる事象をコア間で相互に通知するために，コア間割込みハードウェアの搭載が必要となる.

3.9.2　標準的な要件【SR】

マルチコアシステムにおける要件

外部割込みは静的に決められた特定のコアに接続されていること【HW018】.
(この要件は最低限の要件に対する追加である．)

【要件決定の理由】

外部割込み発生時に，割込み処理コアを処理負荷などから動的に決定するアーキテクチャの場合，割込み処理コアを決定する処理によって OS 処理のリアルタイム性を損なう可能性がある．

この要件は，ランタイムにおいて割込みごとに割込み処理コアが静的に決定しているという意味であり，割込みに対して割込み処理コアを動的(OS 起動時)に設定可能なハードウェア仕様を禁ずるものではない．図 3-6 で挙げた割込み接続の形態例のうち(a)は(b)の形態に対して割込み処理を行うコアを設定可能にしたものといえる．両者とも割込み通知までの上限時間の保証が容易なので要件を満たす．(c)に関しては，各コアの処理負荷の判断によっては通知までの上限が定まらない上に，割込み処理を行うコアも不定となるので，本 OS の要求に合致しない．

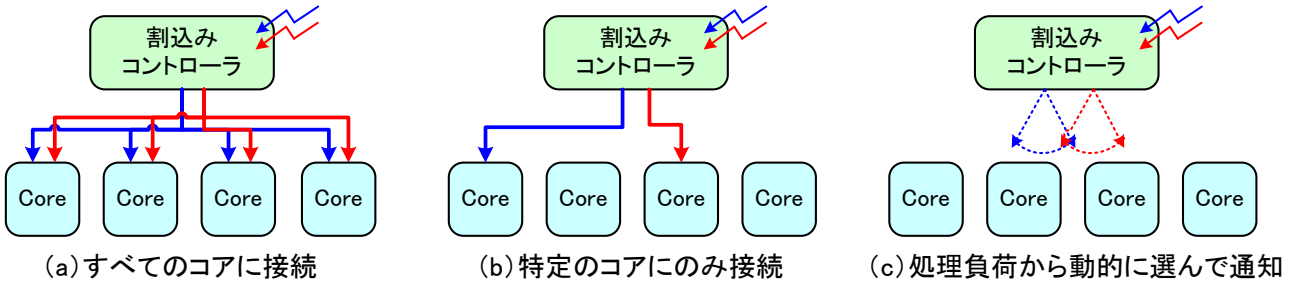

図 3-6　マルチコアシステムにおける割込みの接続形態

3.9.3　性能改善のための要件【AR】

マルチコアシステムにおける要件

割込み禁止状態において，個別に割込み許可としている割込み要因のうち，いずれかの割込み要因で要求が発生しているということが，ソフトウェアから低オーバヘッドで判別可能であること【HW019】.

（この要件は最低限の要件，標準的な要件に対する追加である．）

【要件決定の理由】

低オーバヘッドで割込み要求を判別する機構は，コア間排他制御における「中断可能なロックアルゴリズム」の実現に必要なもので，保留されている割込み要求があった場合にのみ割込みを許可するという操作のためのものである．割込み要求判別操作が高オーバヘッドであった場合には，保留されている割込み要求の有無に関らず割込みを許可するという手続きとなる．しかし，OS 搭載機能とハードウェア構成(例：タイミング保護をマスカブル割込みで実装)によっては，割込み許可操作も高オーバヘッドとなる可能性がある．

要件をみたすハードウェアの例を図 3-7 に示す．同図中「個別要因の割込み要求」で示されるように n 個の割込み要求を個別に保持する機構であった場合には，全割込み要因に対して割込み要求の有無を調べるという操作をソフトウェアで実現する必要がある．一方，図 3-7 のように，これらを OR 接続した回路を用意することで，このオーバヘッドが削減できる．そのほか，1 命令で読み込み可能なレジスタの各ビットに割込み要求を割り振るというアプローチも考えられる．

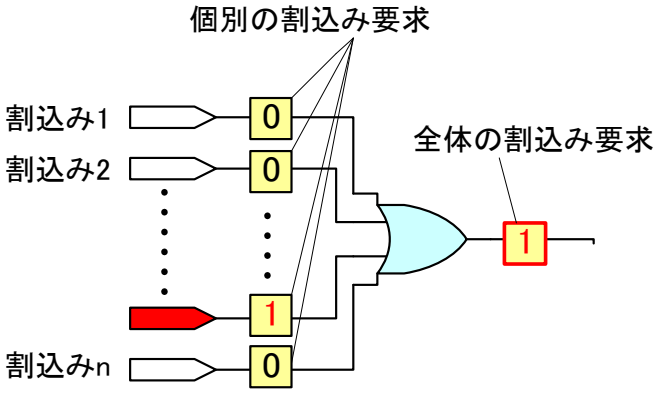

図 3-7　割込み要求の有無の判別

3.10 タイマ

時間駆動の OS 機能の実現に必要となるタイマの要件を規定する．実装する OS 機能ごとに，対応が必要な要件の一覧を表 3-3 に示す．

表 3-3 OS 実装機能ごとに対応が必要なタイマハードウェア要件

要件 ＼ OS 実装機能		SC2	SC1-MC	SC2-MC
最低限	【HW020】※注 1	○	○	○
	【HW021】	○		
	【HW022】		○	
	【HW023】			○
標準	【HW024】	○		
	【HW025】	○		
	【HW026】			○
	【HW027】			○
性能改善	【HW028】	○		○

○：OS 実装機能(SC/MC)ごとに，その OS が動作するために，ハードウェアが準拠しなければならない要件を表す．

注1 ：【HW020】は SC1 相当の要件であるが，SC2, MC 要件の前提となる要件なので本文書で規定する．

3.10.1 最低限の要件【MR】

<u>シングルコアシステム・タイミング保護なしにおける要件</u>

OS が提供する，周期起動に必要なタイマのカウント間隔で繰り返し割込みを発生するタイマが 1 個存在すること【HW020】．

【要件決定の理由】

タスクの起動やイベントのセットといった動作をカウンタ駆動で行うため，OS のカウンタをインクリメントする基準となる割込みを発生するタイマが必要である．このタイマ割込みハンドラによってカウンタオブジェクトを駆動することで，そのカウンタオブジェクトに接続されたアラーム，スケジュールテーブルオブジェクトの満了アクションを実行可能となる．

なお，この要件は SC1 機能相当の要件であるが，〔HW021〕，〔HW022〕の前提となる要件なので本文書で規定している．

シングルコアシステム・タイミング保護ありにおける要件

前述の「シングルコアシステム・タイミング保護なしにおける要件」で規定されるタイマに加えて，以下の性質を持つタイマが 1 個存在すること【HW021】.

- 設定した任意のカウント(単位マイクロ秒)後に割込みを発生する
- 途中で停止することが可能で，停止した時の残りのカウント値が読出し可能

【要件決定の理由】

タイミング保護機能のうち，実行時間監視，リソース占有時間監視，OS 管理割込み禁止時間監視に必要なタイマである．タスクや ISR のプリエンプションによる監視対象のタスク，ISR の切替えや，リソース獲得，解放や OS 管理割込み禁止，許可による監視対象の切替えにより，バジェット値の切替えが必要になるため，途中停止とそのときのカウント値の読出しを行う必要がある．単位をマイクロ秒としたので，近年の車載ソフトウェアにおける需要による．

マルチコアシステム・タイミング保護なしにおける要件

以下の性質をもつタイマが存在すること【HW022】.

- OS が提供する，周期起動に必要なタイマのカウント間隔で，すべてのコアに対して繰り返し割込みを発生させるタイマ

【要件決定の理由】

シングルコアシステムと同様の要件である．なお，1 つのタイマからすべてのコアにタイマ割込みを通知するか，コア個別にタイマを用意してそれぞれが 1 個のコアに対して割込みを通知するかは問わない(図 3-8).

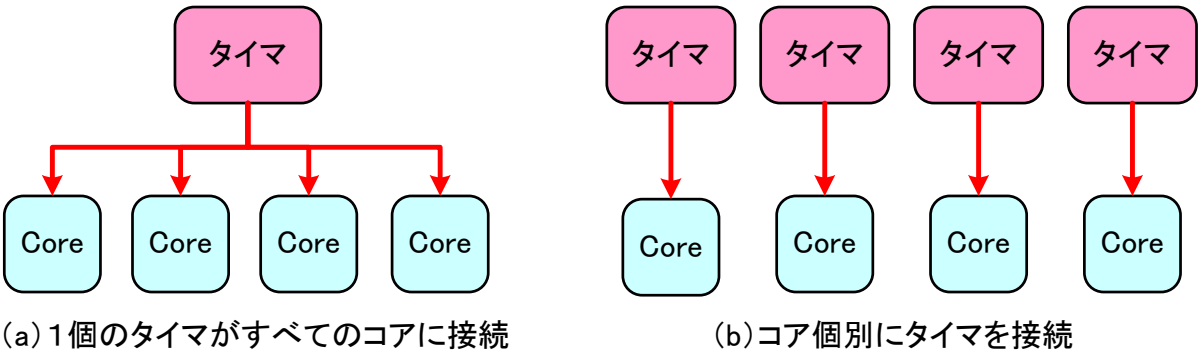

図 3-8　マルチコアシステムにおけるタイマの接続形態

マルチコアシステム・タイミング保護ありにおける要件

前述の「マルチコアシステム・タイミング保護なしにおける要件」で規定されるタイマに加えて，〔HW021〕を満たすタイマが各コア個別に搭載されていること【HW023】

【要件決定の理由】

各コアで並列にタスク，ISR が実行されるため，それぞれの実行時間，リソース占有時間，割込み禁止時間を同時並行に監視する必要があるため.

3.10.2 標準的な要件【SR】

シングルコアシステム・タイミング保護(経過時間監視)における要件

最低限の要件で規定されるタイマに加え，以下の性質を持つタイマが 1 個存在すること【HW024】.

- 設定した任意のカウント(単位マイクロ秒)後に割込みを発生する
- 途中で停止することが可能
- NMI(ノンマスカブル割込み)にも接続可能

【要件決定の理由】

タイミング保護機能のうち，全割込み禁止時間監視に必要なタイマである．全割込み禁止解除のため，途中停止は必要であるが，実行時間等のように監視対象の切替えは必要ないためカウント値の読出しは必要ない.

また，NMI(ノンマスカブル割込み)に接続した場合，OS 機能の全割込み禁止，許可をコアの割込み禁止，許可で実現可能となる.

シングルコアシステム・タイミング保護(到着時間監視)における要件

最低限の要件で規定されるタイマに加え，以下の性質を持つタイマ(間隔監視)が 1 個存在すること【HW025】.

- マイクロ秒単位のカウンタであり，現在値が読出し可能

【要件決定の理由】

タイミング保護機能のうち，到着時間監視に必要なタイマである．タスクならびに ISR の到着(起動)から次の到着までの間隔を監視するために，直近のタスク，ISR の到着が発生した時刻と現在時刻を取得可能である必要がある．割込みの発生は必要ない.

マルチコアシステム・タイミング保護(経過時間監視)における要件

〔HW024〕を満たすタイマが各コア個別に搭載されていること【HW026】

【要件決定の理由】

各コアで並列にタスク，ISR が実行されるため，それぞれの実行時間，割込み禁止時間を同時並行に監視する必要があるため.

マルチコアシステム・タイミング保護(到着時間監視)における要件

〔HW025〕を満たすタイマのカウント値が，すべてのコアにおいて共通して読み出せること【HW027】.

【要件決定の理由】

タスク到着の契機となる操作(OS 機能におけるタスク起動, イベントセット)はコアを跨って行われる，つまり，監視対象タスクが割付けられたコアとタスク到着契機となる操作を行ったタスク，ISR の割付コアが異なる場合があり，両者は統一した時間管理が必要となる．よって，すべてのコアから同一のタイマの値が読み出せる必要がある.

3.10.3 性能改善のための要件【AR】

標準的な要件の「シングルコアシステム・タイミング保護(経過時間監視)」ならびに「マルチコアシステム・タイミング保護(経過時間監視)」における要件で規定のタイマが，実行時間監視とリソース占有時間監視を低オーバヘッドにて切替え可能な「実行時間監視タイマ」であること【HW028】.
(この要件は「標準的な要件」に対する追加である)

「実行時間監視タイマ」の仕様の詳細は 4 章を参照のこと.

【要件決定の理由】

タイミング保護機能のうち，実行時間監視，リソース占有時間監視，OS 管理割込み禁止時間監視に必要なタイマである．実行時間監視の開始(タスク，ISR の起動)と終了，プリエンプトによる中断と復帰，リソース占有時間監視，OS 管理割込み禁止時間監視の開始と終了の動作におけるソフトウェアオーバヘッドを低減するよう，これらの動作をハードウェア化したものである.

3.11 メモリ保護ユニット(MPU)

メモリ保護機能を実現するために必要なメモリ保護ユニット(Memory Protection Unit：以降，MPU と呼ぶ)のハードウェア要件について規定する．実装する OS 機能ごとに，対応が必要な要件の一覧を表 3-4 に示す．

表 3-4 OS 実装機能ごとに対応が必要な MPU ハードウェア要件

要件 ＼ OS 実装機能			SC3	SC3-MC
MPU	最低限	【HW029】	○	○
		【HW030】	○	○
		【HW031】	○	○
		【HW032】	○	○
		【HW033】	○	○
		【HW034】	○	○
		【HW035】	○	○
		【HW036】	○	○
		【HW037】		○
	標準的	【HW038】	○	○
		【HW039】	○	○
	性能改善	【HW040】	○	○
		【HW041】	○	○

○：OS 実装機能(SC/MC)ごとに，その OS が動作するハードウェアが準拠していることが必要な要件を表す．

MPU の概要

MPU はプロセッサによるメモリアクセスを監視し，メモリ保護を実現するためのハードウェアである．メモリ管理ユニット(Memory Management Unit：以降，MMU と呼ぶ)とは異なり，仮想記憶を実現するための論理アドレスの変換等は行わない，メモリ保護の機能に特化したハードウェアである．

図 3-9 に MPU の概念図を示す．コンフィギュレーションにて，アクセスの許可を設定する連続したメモリ領域を保護領域と呼ぶ．保護領域に関する設定値を保持するために，MPU には保護領域レジスタが搭載されている．MPU はプロセッサからのメモリアクセスを，保護領域レジスタの設定値を元に判定し，禁止されているメモリアクセスだと判定した場合には，そのアクセスを抑止するとともに，プロセッサにメモリ保護違反例外を通知する．

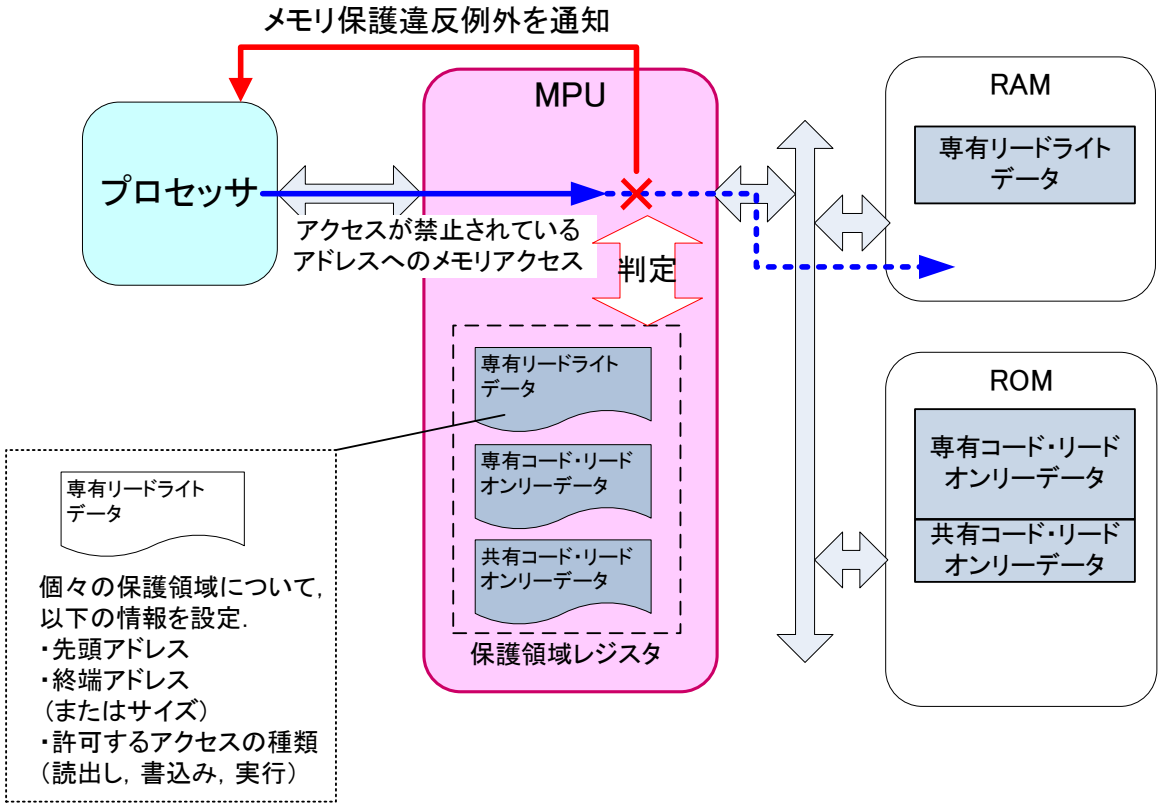

図 3-9　MPU によるメモリ保護の原理

保護領域レジスタの設定

保護領域レジスタは，以下の情報を設定可能なレジスタの組で構成される．

・ 保護領域のアドレス範囲

先頭アドレスと終端アドレス，もしくは先頭アドレスとサイズ等の方式が考えられる．また，これらの設定値にはアラインメント制約などのハードウェア依存の制約が課される場合がある．

・ 保護領域のアクセス許可

許可されるアクセスの種類(読出し，書込み，命令実行)を指定する．

なお，以降の要件定義，ならびに解説においては，あるメモリアドレスが保護領域レジスタに保持する値が示す保護領域のいずれかに含まれるとき，そのメモリアドレスが保護領域レジスタに「マッチする」と表現する．

保護領域のオーバラップ

複数の保護領域レジスタが保持する保護領域設定の指すアドレス範囲がオーバラップしている場合，プロセッサがアクセスしたメモリアドレスが，複数の保護領域レジスタにマッチする場合がある．この場合のハードウェア仕様には，次のいずれかの方式が考えられる．

・ 保護領域がオーバラップするアドレス範囲設定が可能
  - 保護領域レジスタ間の優先順位を決め，最も高い優先度を持つ保護領域レジスタで設定されている情報の組を優先して適用する方式.
  - アクセスの種類ごとに，アクセス許可する設定を優先して適用する方式.
    (例：禁止と許可の両方が設定されていた場合には，許可を優先)

・ 保護領域がオーバラップするアドレス範囲設定が不可能

後述の保護領域レジスタの組数に関する要件において，保護領域がオーバラップするアドレス範囲設定の可，不可によって要件が変わる場合には，両方の場合に対して個別に規定する．

コード MPU とデータ MPU

コードに対するアクセスとデータに対するアクセスを，別の MPU により監視するというアプローチも考えられる．コードアクセス監視のための MPU(以降，コード MPU と呼ぶ)とデータアクセス監視のための MPU(以降，データ MPU と呼ぶ)が，それぞれ別の機構となっている場合には，保護領域レジスタの組数に関する要件を，両者に対して個別に規定する．

3.11.1 最低限の要件【MR】

メモリ保護ハードウェアに関する要件

メモリ保護を目的に搭載するハードウェアは，リアルタイム性に優れたメモリ保護機能ユニット(MPU)であること【HW029】.

【要件決定の理由】

一般的な汎用 OS では，MMU を用いて仮想記憶とメモリ保護の両方を実現している．メモリ保護機能を MMU によって実現することも可能であるが，MMU のもつ仮想記憶機能(論理・物理アドレス変換)は高レイテンシで，かつ応答時間の上限の見積もりが困難であり，リアルタイム性の確保も困難となる．一方で，本仕様が対象とする車載システムでは，プログラムやデータの動的なロードは行われず，仮想記憶機能の必要性は低い．これらの理由により，本仕様では，車載システム向け RTOS の保護機能を実現するためのハードウェアとして MMU ではなく，MPU を要求する．

OS アプリケーション個別の保護領域設定のための要件

保護領域レジスタは，ソフトウェアによって動的に設定内容を変更できること【HW030】.

【要件決定の理由】

保護領域の設定は，OS アプリケーションごとに個別に定義されることから，OS アプリケーションごとにアクセスが許可されるメモリ領域とそのアクセスの種類が異なるため，保護領域レジスタの設定値はソフトウェアから書換え可能である必要がある.

アクセスが禁止される領域に対するメモリアクセス監視のための要件

プロセッサの動作モードが非特権の状態において，MPU は，以下のいずれかに該当するメモリアクセスをメモリ保護違反として検出できること【HW031】.

- 対象メモリアドレスがいずれの保護領域レジスタにもマッチしないアクセス
- 対象メモリアドレスがいずれかの保護領域レジスタにマッチしている場合で，その保護領域のアクセス許可設定で禁止されている種類のアクセス

なお，前者の要件を満たさない場合でも，保護領域がオーバラップするアドレス範囲設定が可能であれば，この要件を満たすことができる．その場合には，以降で規定される保護領域レジスタの組数に関する要件に，アドレス空間全体のメモリアクセスを禁止する設定のための保護領域レジスタを加味する必要がある.

【要件決定の理由】

保護領域を設定していない領域は，コード，データが未配置の領域である．MPU が前者の要件を満たさない(保護領域レジスタにマッチしないアクセスを扱えない)場合には，未配置の領域に明示的にアクセス禁止の設定をする必要が生ずる．しかし，未配置領域は断片化されている可能性があり，未配置領域の設定に必要な保護領域レジスタの組数が増大してしまう．このため，未配置領域については保護領域レジスタに設定しなくてもアクセスが禁止されるようにするか，アドレス空間全体のメモリアクセスを禁止するという設定をオーバラップできるようにする必要がある.

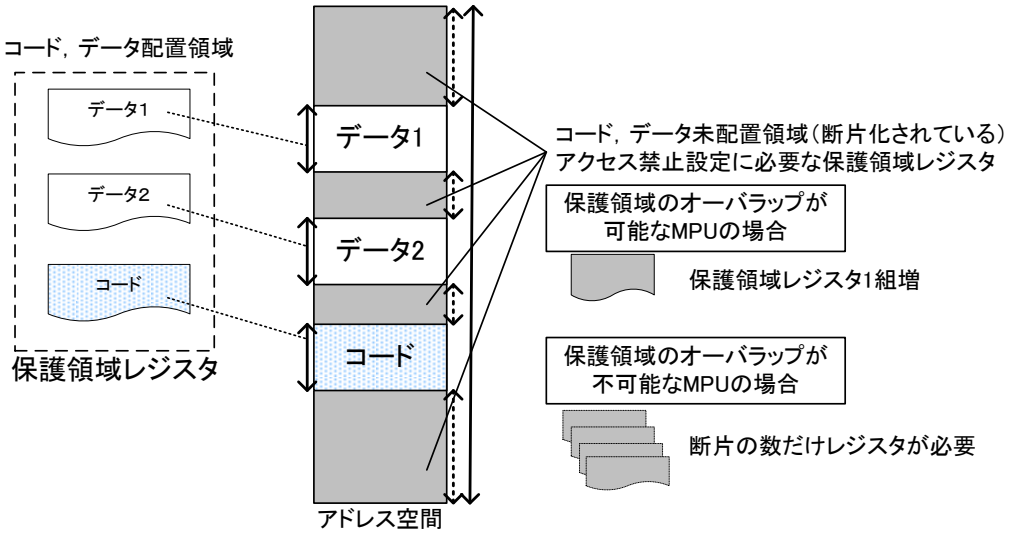

図 3-10 未配置領域に対するアクセス禁止設定
(保護領域レジスタにマッチしないアクセスが許されない MPU)

メモリ保護違反の通知のための要件

MPU がメモリ保護違反を検出した際に発生する割込みは， OS 管理下の割込みより高い優先度の割込みか，NMI(ノンマスカブル割込み)であること【HW032】.

【要件決定の理由】

OS 管理下のいずれの処理単位においてもメモリ保護違反を通知するためには, OS 管理下の最高優先度の割込みを処理中であっても，メモリ保護違反が通知されることを保証しなければならないため.

特権モードにおけるアクセス許可のための要件

プロセッサが特権モードで実行している間は, 以下のいずれかの振舞いが可能であること【HW033】.

- MPU の機能を停止
- すべてのメモリ領域に対するすべての種類のアクセスを許可

なお，後者の振舞いの設定に別途保護領域レジスタを必要とする場合，後述の保護領域レジスタの組数に関する要件に，特権モード中の振舞いを実現するための保護領域レジスタを加味する必要がある.

【要件決定の理由】

信頼 OS アプリケーションと OS 処理は同じアクセス権を持つものと外部仕様で規定されており, 両者は特権モードで動作する．OS が動作するモードにおいては，任意のメモリ領域に対する任意の種類のアクセスが許可となる設定が可能である必要があるため.

アクセス禁止のための要件

保護領域レジスタにて設定する保護領域に対して，書込みアクセスを禁止することができること【HW034】.

【要件決定の理由】

メモリ保護仕様レベル 1 は，書込みアクセスからの保護のみが含まれており，書込みアクセスの保護のみが必須となる.

アクセス許可のための要件

すべてのメモリアドレスに対して，読出しアクセスと実行アクセスを許可するように設定できること【HW035】.

なお，読出しアクセス，実行アクセスの許可の設定に，保護領域レジスタを必要とする場合，後述の保護領域レジスタの組数に関する要件に，読出しアクセス，実行アクセスの設定に必要な保護領域レジスタの組数を加味する必要がある.

【要件決定の理由】

メモリ保護仕様レベル 1 は，書込みアクセスの監視のみが含まれており，読出しアクセスと実行アクセスが許可されることは前提となっているため.

保護領域(書込みアクセス監視のみ)導入のための要件

MPU に設定可能な保護領域の情報の組が，以下に規定する数以上であること【HW036】.

- コード MPU とデータ MPU の区別が無い場合は 3 組以上
- コード MPU とデータ MPU が別の場合には，データ MPU に対して 3 組以上

【要件決定の理由】

3 組の保護領域レジスタは，以下の 3 つのメモリ領域への書込みアクセスを許可するために使用することを想定している.

- タスクまたはカテゴリ 2 ISR のスタック領域
- OS アプリケーションの専有リードライトデータ領域および共有リード専有ライトデータ領域
- 共有リードライトデータ領域

マルチコアシステム・メモリ保護ありにおける要件

マルチコアシステムにおいては，以下の性質を持った MPU を各コアに搭載すること【HW037】.

- 〔HW029〕～〔HW036〕で規定される要件をすべて満たす.
- メモリ保護違反が発生した場合, MPU はメモリ保護違反を発生させた処理単位が動作していたコアのみに対してメモリ保護違反を通知する.

【要件決定の理由】

各々のコアでは，必要となる保護領域設定が異なる処理単位が同時並列で実行されるため，それぞれ独立した動作をする MPU が必要となる．また，特定のコアで発生したメモリ保護違反が必ずしも他のコアに関係あるとは限らないため，保護違反を通知する対象のコアは保護違反が発生したコアのみとする．保護違反発生コア以外の他のコアに対して保護違反を通知する必要がある場合には，コア間割込みハードウェアと，それを利用した OS 機能(コア間割り込み機能)によって対応する.

3.11.2 標準的な要件【SR】

アクセス禁止のための要件

保護領域レジスタにて設定する保護領域に対して，読出しアクセス，書込みアクセスを個別に禁止することができること【HW038】．なお，実行アクセスは読出しアクセスに含むことを想定しているが，別の種類のアクセスとして区別しても良い.

（この要件は最低限の要件を包含している．）

【要件決定の理由】

メモリ保護機能レベル 2 およびレベル 3 は，読出し(実行を含む)，書込みアクセス対する監視が含まれており，個別にそれぞれのアクセスを禁止する設定ができなければならない.

保護領域(読出し，書込みアクセス監視)導入のための要件

MPU に設定可能な保護領域の情報の組が，以下に規定する数以上であること【HW039】.

- コード MPU とデータ MPU の区別がない場合は 8 組以上
- コード MPU とデータ MPU が別の場合，コード MPU 用に 2 組以上，データ MPU 用に 8 組以上
- ショートデータ最適化を用いる場合，データアクセス監視用に 1 組追加
- 保護領域がオーバラップするアドレス範囲設定が不可能な場合，データアクセス監視用に 1 組追加

（この要件は最低限の要件を包含している．）

【要件決定の理由】

8 組の保護領域レジスタは，以下の 8 領域へのアクセス保護を実現するために使用することを想定している.

- 実行中のタスクまたはカテゴリ 2 ISR のスタック領域
- 実行中の OS アプリケーションの専有コード領域および専有リードオンリーデータ領域
- 実行中の OS アプリケーションの専有リードライトデータ領域
- 実行中の OS アプリケーションの共有リード専有ライトデータ領域
- すべての OS アプリケーションの共有リード専有ライトデータ領域
- 共有コード領域および共有リードオンリーデータ領域
- 共有リードライトデータ領域
- 周辺デバイスがマップされたメモリ領域

ショートデータ最適化が適用可能なターゲットの場合は，上記の 8 領域に加えて，以下のメモリ領域を含めた 9 領域のアクセスを許可するために使用することを想定している.

- 実行中の OS アプリケーションの専有リードライトショートデータ領域

また，「すべての OS アプリケーションの共有リード専有ライトデータ領域」の実現に際して，保護領域が「実行中の OS アプリケーションの共有リード専有ライトデータ領域」とオーバラップするアドレス範囲設定が可能な場合にはデータ MPU の保護領域レジスタのうち 2 組を使用するのに対し，オーバラップ不可能の場合には 3 組を使用する.

3.11.3 性能改善のための要件【AR】

保護領域(読出し，書込みアクセス監視)導入のための要件

ショートデータ最適化を用いる場合には，MPU に設定可能な保護領域の情報の組が，以下に規定する数以上であること【HW040】.

- コード MPU とデータ MPU の区別がない場合は 12 組以上
- コード MPU とデータ MPU が別の場合，コード MPU 用に 2 組以上，データ MPU 用に 12 組以上
- 保護領域がオーバラップするアドレス範囲設定が不可能な場合，データアクセス監視用に 1 組追加

（この要件は最低限の要件，標準的な要件を包含している．）

なお，ショートデータ最適化を用いない場合は，標準的な要件と同じとなる.

【要件決定の理由】

標準的な要件でサポートされる 9 領域に加えて，ショートデータ最適化が適用される以下の 3 領域を加えた 12 領域へのアクセス保護を実現するために使用することを想定している.

- 実行中の OS アプリケーションの共有リード専有ライトショートデータ領域
- すべての OS アプリケーションの共有リード専有ライトショートデータ領域
- 共有リードライトショートデータ領域

実行中の OS アプリケーションの専有リードライトショートデータ領域のみ標準的な要件としたのは，ある OS アプリケーションのみで使用するグローバル変数が多い場合に，これをショートデータ領域に配置できないと，大きな性能劣化が予想されるためである.

ソフトウェアからのメモリアクセス可否判定のオーバヘッド低減のための要件

MPU は，領域サーチ機能として以下の機能を持つ【HW041】.

- ソフトウェアによって指定されたメモリアドレスが，いずれかの保護領域レジスタにマッチするか否かを判定し，その結果をソフトウェアに返す.

- ソフトウェアに返すサーチ結果には，マッチした保護領域レジスタの識別番号(マッチする保護領域レジスタがない場合は，そのことを示す値)と，指定されたメモリアドレスに対して禁止されているアクセスの種類を含む.

- 保護領域がオーバラップするアドレス範囲設定が可能で，指定されたメモリアドレスが複数の保護領域レジスタにマッチした場合，サーチ結果として以下のいずれかの方法で保護領域レジスタの識別番号を返す.
  - 保護領域レジスタ間に優先順位を決める方式の場合には，指定されたメモリアドレスにマッチする保護領域レジスタの中で，最も高い優先度を持つ保護領域レジスタの識別番号
  - アクセスの種類ごとに，許可する設定を優先して適用する方式の場合には，指定されたメモリアドレスにマッチする保護領域レジスタのリスト(ビットマップで表現すればよい)

- 特権モード中に MPU を停止した場合でも，MPU の保護領域レジスタに設定された値によって，領域サーチが可能であること.

（この要件は最低限の要件，標準的な要件に対する追加である．）

【要件決定の理由】

領域サーチ機能は，あるアドレスに対するメモリアクセス可否をハードウェア的に低レイテンシで判別するための機能で， CheckTaskMemoryAccess システムサービスにおいて自タスク(システムサービスを発行したタスク)を指定した場合と，CheckISRMemoryAccess システムサービスにおいて自 ISR(システムサービスを発行した ISR)を指定した場合のオーバヘッドを削減するためのものである.

領域サーチ機能が搭載されない場合には，メモリアクセス可否の判定のために，保護領域レジスタの各組の設定値を逐次比較する方法，実際にメモリアクセスを実行して保護違反例外が生ずるか否かを判定すると言う方法が考えられるが，どちらも高オーバヘッドとなる恐れがある.

4. 実行時間監視タイマに対する要件の詳細

本章では要件【HW028】を満たすハードウェアの詳細について述べる．なお，本章で述べるタイマは，オーバヘッドが小さいダウンカウンタを想定して記述しているが，アップカウンタで実現しても差し支えない．

4.1　目的

実行時間監視タイマは，タイミング保護機能のうち，タスク実行時間またはカテゴリ 2 ISR 実行時間と，リソース占有時間，OS 管理割込み禁止時間の監視を 1 個のタイマで実現する上で，少ないアクセス回数で制御できるハードウェアである．

タスク実行時間またはカテゴリ 2 ISR 実行時間と，リソース占有時間，OS 管理割込み禁止時間の監視は，標準的な要件のタイミング保護(経過時間監視)のタイマにより実現可能であるが，監視の開始，終了の動作(図 4-1)において，一般的にプロセッサより動作周波数の遅いタイマへ複数回のアクセスを行うため，ソフトウェアのオーバヘッドが大きいという問題がある．実行時間監視タイマは，演算処理とその結果に連動したタイマの制御をハードウェア化し，この問題を解決する．

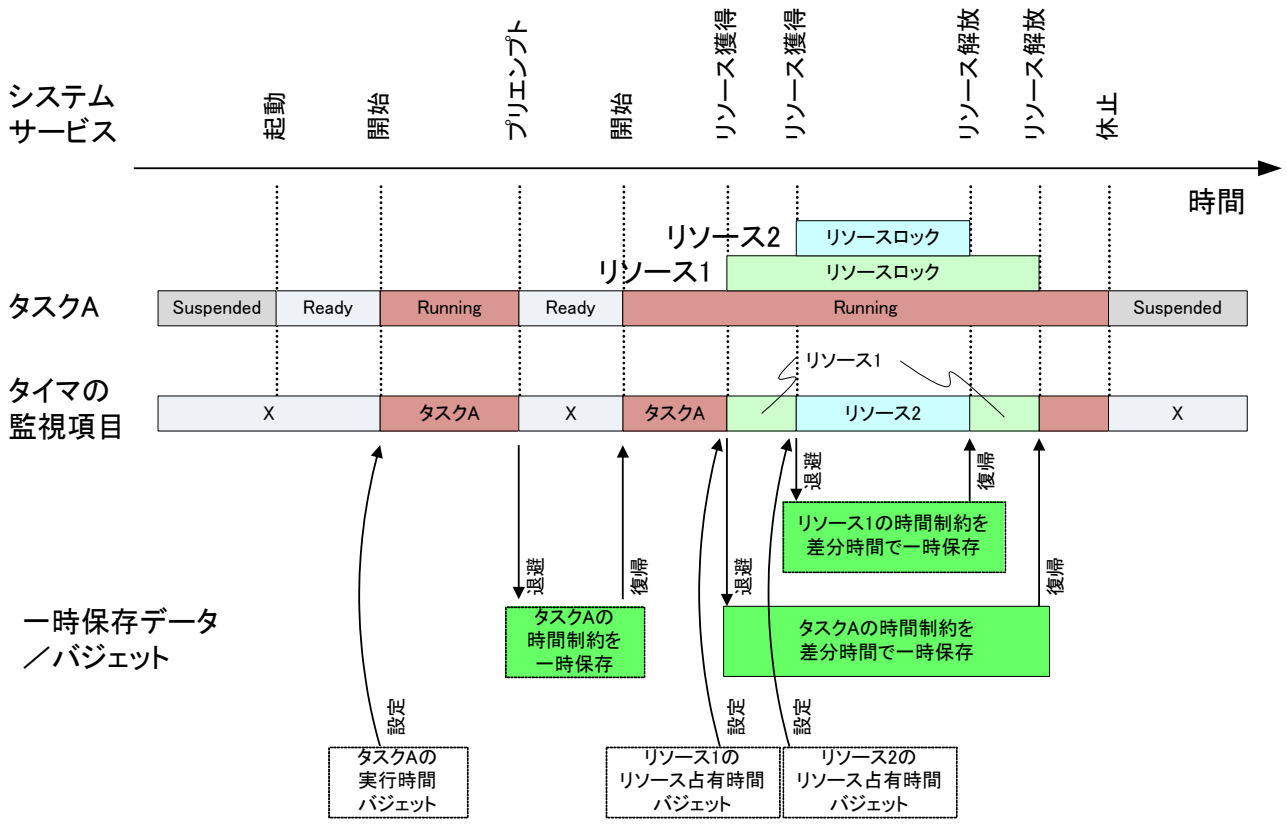

図 4-1　サービスコールとタイマの監視項目

4.2　概要

実行時間監視タイマは，標準的な要件のシングルコアシステム・タイミング保護(経過時間監視)における要件で規定のタイマの機能に加え，以下の機能を持つ.

<u>タイマの残り時間の設定機能</u>

・ (設定 1)残り時間直接入力

入力データをタイマの残り時間に設定し，タイマ状態を遷移させる.

・ (設定 2)残り時間比較入力

入力データが現状のタイマの残り時間より小さい場合，入力データをタイマの残り時間に設定し，タイマ状態を遷移させる．現状のタイマの残り時間から入力データを減算した結果を差分時間として保持する.

・ (設定 3)差分時間入力

入力データをタイマの残り時間に加算し，タイマ状態を遷移させる.

<u>タイマの残り時間の参照機能</u>

・ (参照 1)残り時間出力＜タイマ停止＞

タイマを停止状態に遷移して，タイマの残り時間を参照する.

・ (参照 2)残り時間出力＜タイマ状態維持＞

タイマ状態を維持して，タイマの残り時間を参照する.

<u>差分時間(設定 2 の結果)の参照機能</u>

・ (参照 3)差分時間出力

タイマ状態を維持して，(設定 2)で算出した差分時間を参照する.

4.3　機能要件

タイマ構成の概念図を図 4-2 に示す．ソフトウェアの設定により制御する経路を赤線で示す．

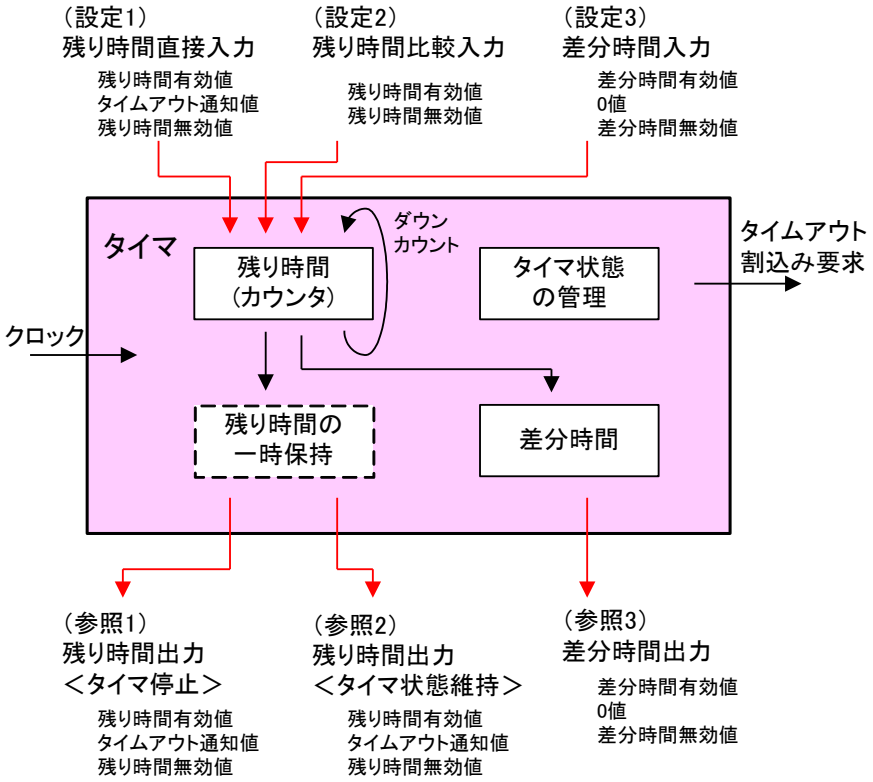

図 4-2　タイマ構成の概念図

4.3.1　タイマ状態

タイマは表 4-1 に示す 3 つの状態を持つ．状態遷移の概念図を図 4-3 に示す．ソフトウェアの設定により制御する経路を赤線で示す．

表 4-1 タイマ状態

状態	状態の概要
停止状態	・リセット発生時，初期状態として，この状態となる ・残り時間を保持する ・ソフトウェアの設定により，この状態から動作状態，タイムアウト状態へ遷移する
動作状態	・残り時間≠0 の場合，残り時間を一定周期でダウンカウントし， 残り時間＝0 の場合，タイムアウト状態へ遷移する ・ソフトウェアの設定により，この状態から停止状態へ遷移する
タイムアウト状態	・残り時間を保持する ・タイムアウト割込み要求をセットして出力する ・ソフトウェアの設定により，タイムアウト割込み要求をクリアし， この状態から停止状態へ遷移する

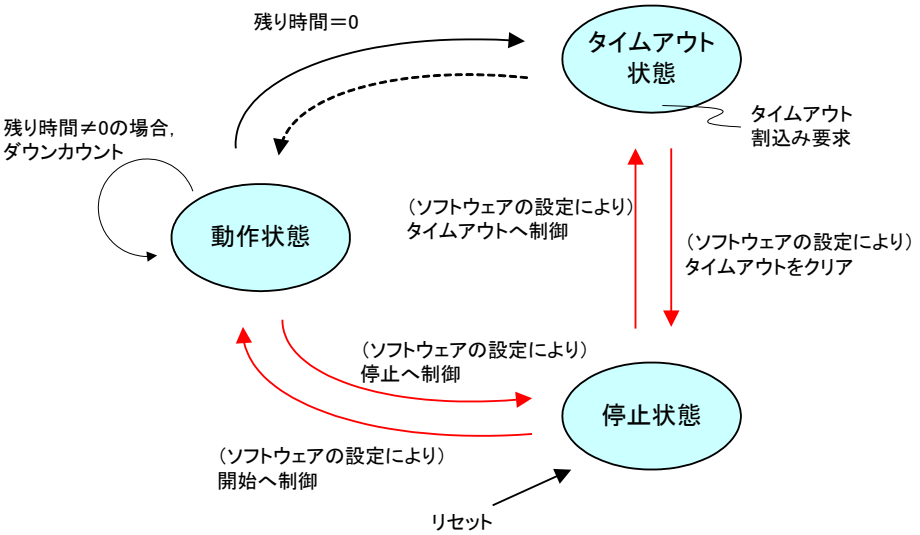

図 4-3　タイマの状態遷移の概念図

4.3.2　タイマの設定/参照データ

タイマへ設定するデータと参照するデータは表 4-2，表 4-3 に示す分類に分けられる．

表 4-2 残り時間の分類(設定 1，設定 2，参照 1，参照 2)

分類	説明
残り時間有効値	監視する残り時間を示す値
タイムアウト通知値	タイムアウト状態であることを示す値(残り時間=0 に相当)
残り時間無効値	監視なしを示す値

表 4-3 差分時間の分類(設定 3，参照 3)

分類	説明
差分時間有効値	(設定 2)残り時間比較入力によるタイマの残り時間の減少値を示す値
0 値	(設定 2)残り時間比較入力によるタイマの残り時間の更新なしを示す値
差分時間無効値	(設定 2)残り時間比較入力の前に監視なしであったことを示す値

4.3.3　タイマの残り時間の設定

(設定 1)残り時間直接入力

入力データを，残り時間有効値，タイムアウト通知値，残り時間無効値に分類判定する．判定した入力データ，タイマ状態に応じて，表 4-4 のとおりに残り時間の設定と，タイマ状態の制御をする．

表 4-4 (設定 1)残り時間直接入力

タイマ状態 / 入力データ	動作状態	タイムアウト状態	停止状態
残り時間有効値	未定義	未定義	(設定 1-A)
タイムアウト通知値	未定義	未定義	(設定 1-B)
残り時間無効値	未定義	未定義	(設定 1-C)

・ (設定 1-A)の制御

残り時間に入力データを設定する．タイマ状態を動作状態へ遷移する．

・ (設定 1-B)の制御

タイマ状態をタイムアウト状態へ遷移する．

※一旦動作状態へ遷移して，直ちにタイムアウトを発生させてもよい．

・ (設定 1-C)の制御

タイマ状態は現状の状態(停止状態)を維持する．

(設定 2) 残り時間比較入力

入力データを，残り時間有効値，残り時間無効値に分類判定する．判定した入力データ，現状の残り時間，タイマ状態に応じて，表 4-5 のとおりに残り時間の設定と，タイマ状態の制御と，差分時間の設定をする．

表 4-5 (設定 2)残り時間比較入力

タイマ状態 / 入力データ	動作状態	タイムアウト状態	停止状態
残り時間有効値	“残り時間”>入力データ →(設定 2-A)	(設定 2-C)	(設定 2-D)
	“残り時間”≦入力データ →(設定 2-B)		
残り時間無効値	(設定 2-C)	(設定 2-C)	(設定 2-E)

・ (設定 2-A)の制御

残り時間に入力データを設定する．タイマ状態は現状の状態(動作状態)を維持する．差分時間に残り時間から入力データを減算した値(差分時間有効値)を設定する．

・ (設定 2-B)の制御

残り時間は現状を維持する．タイマ状態は現状の状態(動作状態)を維持する．差分時間に 0 値を設定する．

・ (設定 2-C)の制御

残り時間は現状を維持する．タイマ状態は現状の状態を維持する．差分時間に 0 値を設定する．

・ (設定 2-D)の制御

残り時間に入力データを設定する．タイマ状態を動作状態へ遷移する．差分時間に差分時間無効値を設定する

・ (設定 2-E)の制御

タイマ状態は現状の状態(停止状態)を維持する．差分時間に差分時間無効値を設定する．

(設定 3)差分時間入力

入力データを，差分時間有効値，0 値，差分時間無効値に分類判定する．判定した入力データ，タイマ状態に応じて，表 4-6 のとおりに残り時間の設定と，タイマ状態の制御をする．

表 4-6 (設定 3)差分時間入力

タイマ状態 / 入力データ	動作状態	タイムアウト状態	停止状態
差分時間有効値	(設定 3-A)	(設定 3-A)	未定義
0 値	(設定 3-B)	(設定 3-B)	未定義
差分時間無効値	(設定 3-C)	(設定 3-C)	(設定 3-C)

・ (設定 3-A)の制御

残り時間に入力データを加算する．タイマ状態を動作状態へ遷移する．

・ (設定 3-B)の制御

残り時間は現状を維持する．タイマ状態は現状の状態を維持する．

・ (設定 3-C)の制御

タイマ状態を停止状態へ遷移する

4.3.4　タイマの残り時間の参照

(参照 1)残り時間出力＜タイマ停止＞

タイマ状態に応じて，表 4-7 のとおりに残り時間，タイムアウト通知値，残り時間無効値のいずれかを読出し，タイマ状態の制御をする．

表 4-7 (参照 1)残り時間出力＜タイマ停止＞

タイマ状態 / 入力データ	動作状態	タイムアウト状態	停止状態
(なし)	(参照 1-A)	(参照 1-B)	(参照 1-C)

・ (参照 1-A)の制御

残り時間を読み出す．タイマ状態を停止状態へ遷移する．

・ (参照 1-B)の制御

タイムアウト通知値を読み出す．タイムアウト割込み要求をクリアし，タイマ状態を停止状態へ遷移する．

・ (参照 1-C)の制御

残り時間無効値を読み出す．タイマ状態は現状の状態(停止状態)を維持する．

(参照 2)残り時間出力＜タイマ状態維持＞

タイマ状態に応じて，表 4-8 のとおりに残り時間，タイムアウト通知値，残り時間無効値のいずれかを読み出す．タイマ状態は現状を維持する．残り時間のビット幅が読出しバスのビット幅より大きい場合，複数回にまたがるアクセスの途中にデータが更新されないように，残り時間を一時保持して読み出す．

表 4-8 (参照 2)残り時間出力＜タイマ状態維持＞

タイマ状態 / 入力データ	動作状態	タイムアウト状態	停止状態
(なし)	(参照 2-A)	(参照 2-B)	(参照 2-C)

・ (参照 2-A)の制御

残り時間を読み出す．タイマ状態は現状の状態(動作状態)を維持する．

・ (参照 2-B)の制御

タイムアウト通知値を読み出す．タイマ状態は現状の状態(タイムアウト状態)を維持する．

・ (参照 2-C)の制御

残り時間無効値を読み出す．タイマ状態は現状の状態(停止状態)を維持する．

4.3.5　差分時間の参照

(参照 3)差分時間出力

(設定 2)残り時間比較入力にてタイマ内部で設定した差分時間を読み出す．

4.4　ソフトウェアの実現例

実行時間監視タイマを用いたソフトウェアの実現例について解説する．

<u>タスクのタイミング保護に対する実行時間監視タイマの設定/参照</u>

実行時間監視タイマを，タスク実行時間，リソース占有時間，OS 管理割込み禁止時間の監視に適用するにあたり，以下の条件で実行時間監視タイマの設定，参照を行う．

・ 開始

タスク起動後の最初の実行開始時，(設定 1)残り時間直接入力で実行時間バジェットをタイマの残り時間に設定し，タスク実行時間監視を開始する．

プリエンプトされた後のタスク実行再開時，(設定 1)残り時間直接入力でプリエンプト時に退避した残り時間をタイマの残り時間を設定し，タスク実行時間監視を再開する．

・ プリエンプト

プリエンプトによるタスク切替え時，(参照 1)残り時間出力＜タイマ停止＞でタイマの残り時間を読出して退避し，タイミング保護の監視を途中停止する．

・ リソースの獲得

リソースの獲得時，(設定 2)残り時間比較入力でリソース占有時間バジェットをタイマの残り時間に設定し，リソース占有時間監視を開始する．(設定 2)残り時間比較入力で算出したタイマの残り時間の減少値を，(参照 3)差分時間出力で読出して退避する．

・ リソースの解放

リソースの解放時，(設定 3)差分時間入力でリソースの獲得時に退避したタイマの残り時間の減少値をタイマの残り時間に加算し，リソース占有時間の監視を終了して，リソースの獲得前の監視を復帰する．

・ OS 管理割込みの禁止

OS 管理割込みの禁止時, (設定 2)残り時間比較入力で OS 管理割込み禁止時間バジェットをタイマの残り時間に設定し，OS 管理割込み禁止時間監視を開始する．(設定 2)残り時間比較入力で算出したタイマの残り時間の減少値を，(参照 3)差分時間出力で読出して退避する．

・ OS 管理割込みの許可

OS 管理割込みの許可時, (設定 3)差分時間入力で OS 管理割込みの禁止時に退避したタイマの残り時間の減少値をタイマの残り時間に加算し, OS 管理割込み禁止時間の監視を終了して, OS 管理割込みの禁止前の監視を復帰する．

・ 休止，待ち

休止状態，待ち状態への状態遷移時，標準的な要件の機能によりタイマを途中で停止し，タスク実行時間監視を終了する．

・ デバッグでタイマの参照

デバッグ時，(参照 2)残り時間出力＜タイマ状態維持＞でタイマを停止せずに(参照 1)残り時間出力＜タイマ停止＞と同様のタイマの残り時間を読み出す．

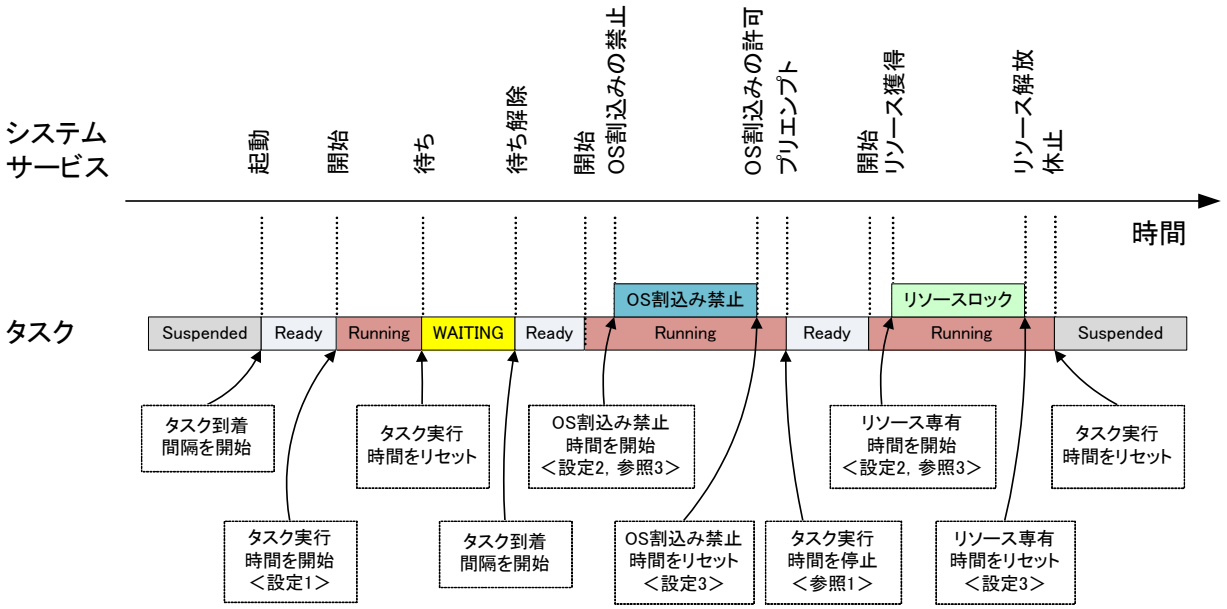

図 4-4　実行時間監視タイマを用いたタスクタイミング保護の対応

<u>カテゴリ 2 ISR のタイミング保護に対する実行時間監視タイマの設定/参照</u>

実行時間監視タイマを，カテゴリ 2 ISR 実行時間，リソース占有時間，OS 管理割込み禁止時間の監視に適用するにあたり，前記のタスクと同様に，実行時間監視タイマの設定，参照を行う．

5. キューイングスピンロックハードウェアに対する要件の詳細

本章では要件【HW008】を満たすハードウェアの詳細について述べる．

5.1　目的

キューイングスピンロックハードウェアは，コア数が 3 個以上のマルチコアシステムにおいて，OS の最悪実行時間の上限を定めるために必要となるキューイングスピンロックを実現する排他制御ハードウェアである．

キューイングスピンロックは，CAS 命令や LL/SC 命令を用いることにより，ソフトウェアで実現することが可能であるが，Test and Set スピンロックと比較してアルゴリズムが複雑であるため，実行オーバヘッドが大きいという問題がある．キューイングスピンロックハードウェアは，アルゴリズムのハードウェア化により，この問題を解決する．

5.2　概要

キューイングスピンロックハードウェアは，優先度発行ユニットと優先度順スピンロックユニットから構成されており，これらを用いてロックは次のように実現される．まずロックを取得したいコアは優先度発行ユニットから優先度を取得する．この優先度はロックを取得する順序の決定に用いる．次に取得した優先度を自コアの優先度として優先度順スピンロックユニットに設定する．優先度順スピンロックユニットは，ロックが解放されていれば，設定されたコアごとの優先度のうち，最も優先度の高いコアに対してロックを与える．

5.3　構成と概念

5.3.1　構成

キューイングスピンロックハードウェアは，次の 2 種類のハードウェア(ユニット)から構成されている．それぞれのユニットはバス接続等によりすべてのコアと接続する必要がある．

- 優先度発行ユニット
- 優先度順スピンロックユニット

優先度発行ユニットはシステムで 1 個必要である．優先度順スピンロックユニットは，ロック単位と同じ個数必要である．

5.3.2　ロック取得優先度

キューイングスピンロックハードウェアで扱うロック取得順決定のための優先度について規定する．優先度は，コアがロックを取得する順序を決定するために用いられる．

優先度は，次の値に分類される．

- 有効値
  - 通常値(最小値～最大値)
  - 最高値
- 無効値

それぞれの値の具体的な数値については規定しないが，例えば次のように数値を割り当てることができる．

例)

- 通常値：0x0000～0xffff
- 最高値：0x10000
- 無効値：0x20000

優先度の各値の優先関係は次の通りである．なお，通常値は基本的には値が小さい方が優先度は高いが，循環することを考慮して扱う．

- 最高値 > 通常値(最大値～最小値) > 無効値

5.4　優先度発行ユニットの機能要件

優先度発行ユニットは, コアに対して優先度を発行するユニットである. 概念モデルを図 5-1 に示す.

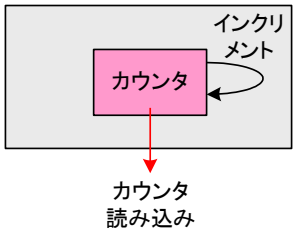

図 5-1　優先度発行ユニットの概念モデル

5.4.1　カウンタ

優先度発行ユニットは，優先度(通常値)を保持するカウンタを 1 個持つ．なお，最高値，無効値は保持しない．また，リセット時の内容は規定しない(どのような値でもよい).

コアから，カウンタを読み出すと，カウンタの内容がインクリメントされる．複数のコアが，同時にカウンタからの読み込み要求を出した場合でも，それぞれのコアは異なる通常値を読み込む．カウンタが最大値を保持している場合に，カウンタのインクリメントを行うと，最小値となる.

5.5　優先度順スピンロックユニットの機能要件

優先度順スピンロックユニットは，ロックを与えるコアを決定するユニットである．概念モデルを図 5-2 に示す．

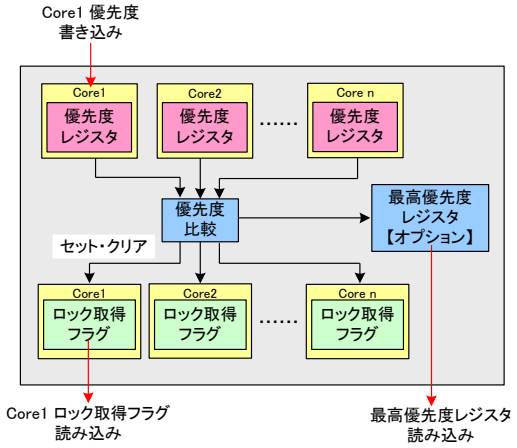

図 5-2　優先度順スピンロックユニットの概念モデル

5.5.1　優先度レジスタとロック取得フラグ

コアごとの優先度レジスタとロック取得フラグの組をシステム上のコアの個数分持つ．

優先度レジスタは，コアから，通常値，最高値，無効値のいずれかを書き込める．優先度レジスタの内容は，リセット時には無効値となる．

ロック取得フラグは，1 ビットのフラグであり，コアから読み込める．ロック取得フラグの内容は，リセット時には，クリアされる．

5.5.2　最高優先度レジスタ【オプション】

優先度レジスタの内容で最も優先度の高い値が設定される．コアから内容を読み込める．

5.5.3　状態遷移

優先度順スピンロックユニットの状態遷移を図 5-3 に示す．状態としては，アンロックとロックの 2 状態を持つ．リセット時はアンロック状態となる．

アンロック状態の場合は，有効値を保持している優先度レジスタの内容を比較し，最も高い優先度の優先度レジスタに対応するロック取得フラグをセットして，ロック状態に遷移する．最も高い優先度を複数の優先度レジスタが保持する場合，それらのうち，どの優先度レジスタに対応するロック取得フラグがセットされるかは規定しない．全優先度レジスタの内容が無効値の場合は，アンロック状態に留まる．

ロック状態の場合は，セットされているロック取得フラグに対応する優先度レジスタに無効値が書き込まれると，ロック取得フラグをクリアして，アンロック状態に遷移する．コアからは，優先度レジスタに無効値が書き込まれた後，ロック取得フラグがクリアされる前の過度的な状態(セットされた状態)が見えてはならない．

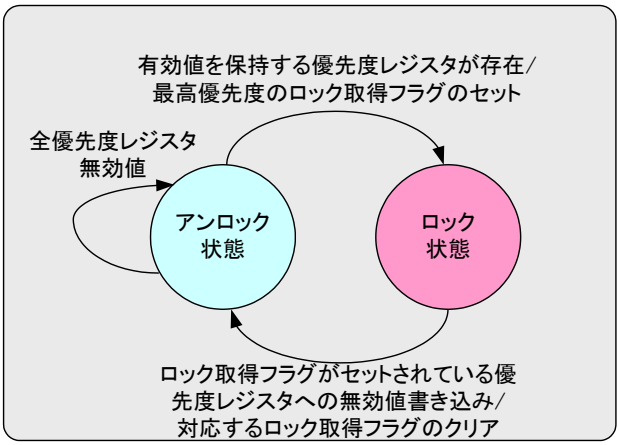

図 5-3　優先度順スピンロックユニットの状態遷移

5.5.4　優先度比較における循環の考慮

通常値は基本的には値が小さい方が優先度は高いが，循環することを考慮して扱う．このとき，同時に優先度レジスタに設定される通常値の範囲が，数の円の半周以上にまたがることはないと仮定して良い(図 5-4)．半周以上にまたがった場合には，最も優先度の高い優先度レジスタに対応するロック取得フラグをセットすることを保証しなくてもよい．

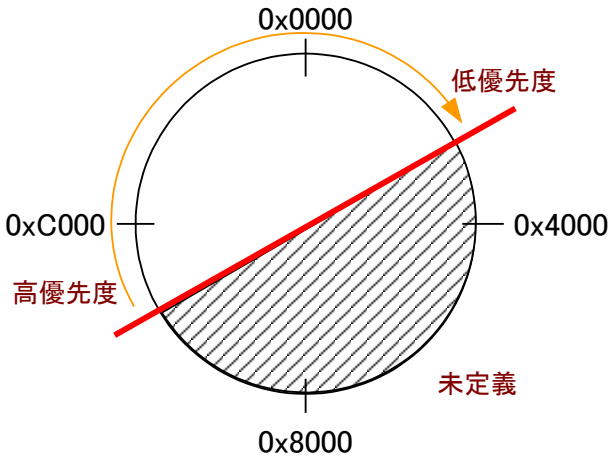

図 5-4　16 ビットの数の円

5.5.5　コアとの接続【オプション】

どのコアからも，自コアの優先度レジスタとロック取得フラグは，同じアドレスとする(Overlay アドレス)．

各コアから，自コアのロック取得フラグへのアクセスにより，他コアの実行を阻害しないようにする．

5.6　ソフトウェアの実現例

キューイングスピンロックハードウェアにアクセスするためのソフトウェアの API と，その API を使用したロック取得ルーチンの例について解説する．

5.6.1　API

データ型

LOCK　優先度順スピンロックユニットの ID を保持する型
PRI　　ロック取得優先度を保持する型
bool　　真偽値を保持する型

スピンロックの開始

【API】

void enter_spinlock(LOCK lock, PRI priority)

【パラメータ】

LOCK lock　　：優先度順スピンロックユニットの ID
PRI　priority　：優先度レジスタへの設定値

【機能】

lock で指定された優先度順スピンロックユニットの自コアに対応した優先度レジスタへ priority を書き込む．

スピンロックの取得待ち

【API】

bool try_spinlock(LOCK lock)

【パラメータ】

LOCK lock　　：優先度順スピンロックユニットの ID

【リターンパラメータ】

bool success　　：ロックを取得できたか

【機能】

lock で指定された優先度順スピンロックユニットの自コアに対応したロック取得フラグを読み込み，セットなら true(ロックが取得成功)を，クリアなら false(ロック取得失敗)を返す．

スピンロックの中断

【API】

void suspend_spinlock(LOCK lock)

【パラメータ】

LOCK lock　　：優先度順スピンロックユニットの ID

【機能】

lock で指定された優先度順スピンロックユニットの自コアに対応した優先度レジスタへ無効値を書込み，ロックの取得の試行を中止する．

スピンロックの返却

【API】

void release_spinlock(LOCK lock)

【パラメータ】

LOCK lock　　：優先度順スピンロックユニットの ID

【機能】

lock で指定された優先度順スピンロックユニットの自コアに対応した優先度レジスタへ無効値を書込み，ロックを返却する．

最高優先度の取得

【API】

PRI get_hpriority(LOCK lock)

【パラメータ】

LOCK lock　　：優先度順スピンロックユニットの ID

【リターンパラメータ】

PRI priority　　：最高優先度

【機能】

lock で指定された優先度順スピンロックユニットの最高優先度レジスタの値を読み込む．

優先度の取得

【API】

PRI get_priority(void)

【リターンパラメータ】

PRI priority　　：優先度

【機能】

優先度発行ユニットのカウンタを読み込む．

5.6.2　1 段ロックの取得ルーチン

割込み発生の有無をソフトウェアから低オーバヘッドで判定不可能なプロセッサ用．ネストロックは考慮していない．

グローバル変数

- PRI priority[TNUM_CORE]
  - 初期値は無効値(INVALID)
- LOCK spinning[TNUM_CORE]
  - 初期値は NULL

関数

- enable_interrupt()
  - 割込みの許可
- disable_interrupt()
  - 割込みの禁止

ローカル変数

- coreid
  - 自コアのコア ID(0 オリジン)
- lock
  - ロック対象のロックに対応する優先度順スピンロックユニットのハンドル

ロック取得ルーチン

```
disable_interrupt();
assert(spinning == NULL);
retry:
if (priority[prcid] == INVALID) {
    priority[prcid] = get_priority();
}
spinning[prcid] = lock;
enter_spinlock(spinning, priority[prcid]);
while (!try_spinlock(spinning[prcid])) {
    enable_interrupt();
    disable_interrupt();
    if (spinning[prcid] == NULL) {
        goto retry;
    }
}
```

ロック解放ルーチン

```
release_spinlock(spinning[prcid]);
spinning[prcid] = NULL;
priority[prcid] = INVALID;
enable_interrupt();
```

割込み入口処理(割込み禁止区間内で実行)

```
if (spinning[prcid] != NULL) {
    suspend_spinlock(spinning[prcid]);
    spinning[prcid] = NULL;
}
```

5.6.3　2 段ロックの取得ルーチン

割込み発生の有無が低オーバヘッドで判定不可能なプロセッサ用．2 段のネストロックの場合のルーチン．

グローバル変数

- PRI priority[TNUM_CORE]
  - 初期値は無効値(INVALID)
- LOCK spinning[TNUM_CORE]
  - 初期値は NULL
- LOCK spinning2[TNUM_CORE]
  - 初期値は NULL

関数

- enable_interrupt()
  - 割込みの許可
- disable_interrupt()
  - 割込みの禁止

ローカル変数

- coreid
  - 自コアのコア ID(0 オリジン)
- lock
  - ロック対象のロックに対応する優先度順スピンロックユニットのハンドル

ロック取得ルーチン(1 段目)

```
    disable_interrupt();
    assert(spinning1[prcid] == NULL);
    assert(spinning2[prcid] == NULL);
retry:
    if (priority[prcid] == INVALID) {
        priority[prcid] = get_priority();
    }
    spinning1[prcid] = lock1;
    enter_spinlock(spinning1[prcid], priority[prcid]);
    while (!try_spinlock(spinning1[prcid])) {
        enable_interrupt();
        disable_interrupt();
        if (spinning1[prcid] == NULL) {
            goto retry;
        }
    }
```

ロック取得ルーチン(2 段目)

```
    spinning2[prcid] = lock2;
    enter_spinlock(spinning2[prcid],
                                        priority[prcid]);
    while (!try_spinlock(spinning2[prcid])) {
        enable_interrupt();
        disable_interrupt();
        if (spinning1[prcid] == NULL) {
            assert(spinning2[prcid] == NULL);
            goto retry;
        }
    }
```

ロック解放ルーチン

```
release_spinlock(spinning2[prcid]);
spinning2[prcid] = NULL;
release_spinlock(spinning1[prcid]);
spinning1[prcid] = NULL;
```

割込み入口処理(割込み禁止区間内で実行)

```
if (spinning2[coreid] != NULL) {
    suspend_spinlock(spinning2[coreid]);
    spinning2[coreid] = NULL;
}
if (spinning1[coreid] != NULL) {
    suspend_spinlock(spinning1[coreid]);
    spinning1[coreid] = NULL;
}
```

5.7　優先度(最高値)と最高優先度レジスタの使用方法

5.7.1　優先度(最高値)

優先度(最高値)を用いることにより，ローカル優先ロックを実現可能である．

特定のコアは，ロック取得時に常に最高値を設定することにより，最大でも 1 つのコアにしか待たされない．

使用例としては，RTOS のロック単位がコアごとに設定されており，各オブジェクトはコアに割り付けてられている場合，コアに閉じたロック獲得(ローカルなロック獲得)を優先することで，自コアに閉じた処理の最悪実行時間を抑えることができる．

5.7.2　最高優先度レジスタ

ネストロックでの優先度継承を実現可能．

前段までのロックでロック取得待ちとなっている(自分が待たせている)高優先度の処理単位の優先度を用いてロックを取得する．2 段目以上のロックを取得する場合，優先度発行ユニットから取得した優先度と，前段の優先度スピンロックユニットの最高優先度レジスタの内容を比較して，高い優先度をロック取得に使用する．図 5-5 に優先度継承の例を示す．図中の赤い数字は，優先度として用いるタイムスタンプであり，値は 1 から始まり，数が小さいほどロック取得の優先度が高いものとする．

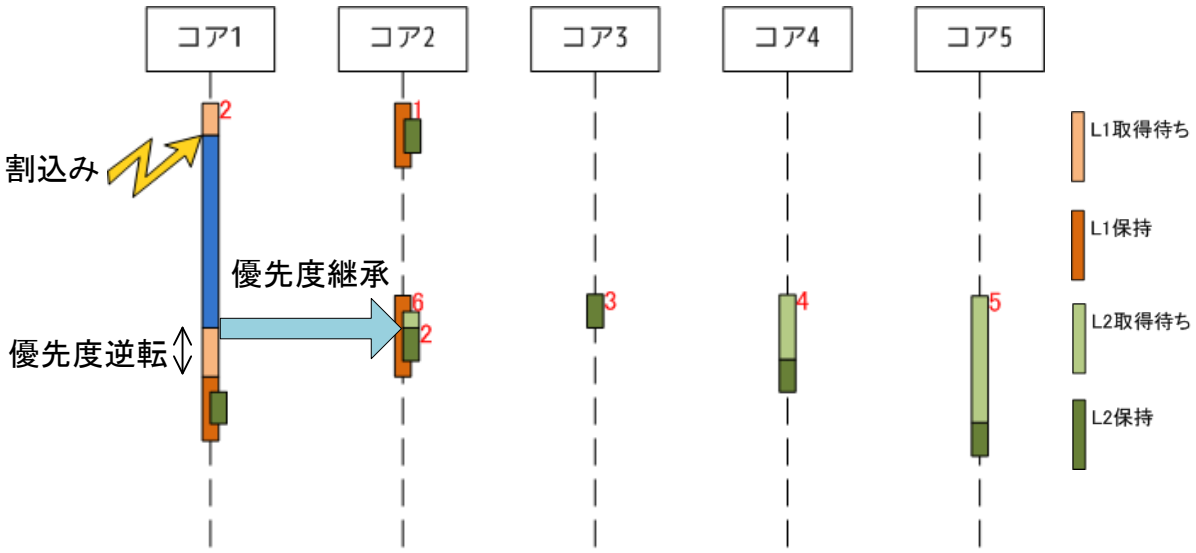

図 5-5　ネストロックにおける優先度継承

コア 2 が優先度 6 で L2 の取得を要求するとき，すでにコア 3 が L2 を保持しているため，コア 2 は L2 を取得できない．コア 2 は再度 L2 の取得を試みる前に，コア 2 より高い優先度のコアが L1 の取得

待ちになっていないか，最高優先度レジスタを用いてチェックを行う．この優先度のチェックは，L2 のロック取得に失敗し，再度取得を試みる前に毎回行う．コア 1 の割込み処理終了後にチェックを行うと，コア 2 より優先度の高いコア 1 が取得待ちをしているため，コア 2 はコア 1 の優先度 2 を継承し，L2 を取得する優先度として設定する．コア 3 がロック L2 を解放すると，コア 2 はコア 4 とコア 5 より優先度が高いため L2 を取得する．コア 2 が L1 を解放するとコア 1 が L1 を取得する．そして，コア 1 が L2 を取得を試みる際には，既にコア 4 が L2 を取得しているため，コア 4 が L2 を解放するまでコア 1 が L2 を取得できないのは避けられない．この例では，コア 1 が待たされる低優先度のコアはコア 2，コア 3，コア 4 の 3 つになる．

このように，優先度継承を導入することによって高優先度のコアが待たされる低優先度のコアを 3 つに抑えることができる．したがって，優先度継承は 4 コア以上のシステムでのみ有効である．

変更履歴

Version	Date	Detail	Editor
1.0.0	2010/3/5	NCES 内部リリース	NCES
1.1.0	2010/12/1	TOPPERS 会員向け早期リリース	NCES
1.1.1	2011/12/28	ATK2 コンソーシアム向けリリース	NCES
1.1.2	2012/3/30	・ファイル名から文書番号を削除 ・コピーライトを記載 ・誤字，脱字，説明不足等を修正	NCES
2.0.0	2013/1/22	一般向けリリース	NCES
3.0.0	2013/6/28	AUTOSAR のマルチコア仕様に対応	NCES
3.0.1	2014/3/12	コピーライト修正	NCES