修 士 論 文

タスクの静的優先度に従った バイナリの構築法に関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

笹山 高志

2009 年 3 月

修 士 論 文

タスクの静的優先度に従った バイナリの構築法に関する研究

指導教官　田中清史 准教授

審査委員主査　田中清史 准教授
審査委員　日比野靖 教授
審査委員　篠田陽一 教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

0610040 笹山 高志

提出年月: 2009 年 2 月

概 要

近年の組込みシステムでは，厳しいコストの制約や，ハードウェアの物理的な大きさの制限により，搭載できるメモリ容量が限られている．そのため，コードサイズが小さく，使用メモリ容量を節約できるシステムが望まれている．組込み用途向けのプロセッサにはコードサイズを削減することを目的に異なる命令長を備えているものがある．

また，近年の組込みシステムの開発において，組込み用 OS が使用される機会が多くなっている．これは OS の導入によりリアルタイム性の向上や，アプリケーション開発の負担が軽減される等の利点があるためである．

本研究では，組込み用途向けプロセッサ上で動作する組込み用 OS を実装し，その OS の上で動作するタスクの静的優先度に従って，リアルタイム性，コードサイズに最適化された実行バイナリを，開発者の負担を減らしつつ構築することを目的とする．そして実行バイナリのリアルタイム性をシミュレーションによって評価する．

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　研究の背景

近年の組込みシステムでは，厳しいコストの制約や，ハードウェアの物理的な大きさの制限により，搭載できるメモリ容量が限られている．そのため，コードサイズが小さく，使用メモリ容量を節約できるシステムが望まれている．

組込み用途向けのプロセッサにはコードサイズを削減することを目的に命令長を可変としているものがある．その代表的なプロセッサにARM[1]がある．ARMプロセッサは近年，組込みシステムなどで広く利用されている32ビットRISCプロセッサであり，通常の32ビットARMモードとコード圧縮を優先した16ビットThumbモードの２つのモードを備える．ThumbモードはARMモードに比べて，実行命令数増加により実行速度が遅いというデメリットがあるが，総コードサイズがARMモードに比べて圧縮できるというメリットがある．この2つのモードは1つのバイナリの中で混在することが可能であり，実行速度の求められる箇所はARMモード，比較的実行速度の求められない箇所はThumbモードを設定することにより，リアルタイム性，コードサイズに最適化されたバイナリを得ることが可能となる．

また，近年の組込みシステムの開発において，組込み用OSが使用される機会が多くなっている．これはOSの導入によりリアルタイム性の向上や，アプリケーションの開発負担が軽減される等の利点があるためである．しかし，組込み用OSでは汎用OSに比べて，よりリアルタイム性が求められる傾向があり，デッドラインミス，平均応答時間などの情報が重要になる．そこで本研究では，組込み用OSのリアルタイム性を計測するシステムを実装する．

1.2　研究の目的

本研究では，ARMプロセッサ上で動作する，μITRON4.0仕様[2]アプリケーションをターゲットとした，バイナリの最適化システムと，バイナリのリアルタイム性を評価可能なシミュレータを実装する．バイナリの最適化システムは，OS上で動作するタスクの静的優先度をもとに，自動でタスクと，タスクで使用されるOSのサービスコールをARMモード，Thumbモードと切替え，プログラマが手動で行うバイナリの最適化処理を自動化し，開発の負担を軽減しつつ最適化されたバイナリを得ることを目的としている．バイナリの最適化システムのアルゴリズムは，ある一定の閾値をプログラマが設定し，その閾

値と静的優先度を比較して，優先度が高い場合は ARM モード，低い場合は Thumb モードと切替える．シミュレータは，タスクの平均応答速度，デッドラインミス情報などを計測可能とし，バイナリのリアルタイム性を考慮しながら開発を支援することを目的としている．なおシミュレータは，ARM プロセッサ (ARM7TDMI) 搭載機器をターゲットとしたものである．

1.3 本論文の構成

本論文の構成を以下に示す．
第 2 章では，ARM プロセッサと µITRON4.0 仕様 OS について述べる．
第 3 章では，提案手法のコンフィギュレーションアルゴリズムについて述べる．
第 4 章では，評価環境について述べた後，提案手法の評価結果を述べる．
第 5 章では，まとめと今後の課題について述べる．

第2章　ARMプロセッサとμITRON4.0仕様OS

本章では，組込みシステムで広く利用されているARMプロセッサと，同じく組込みシステムで広く利用されているμITRON4.0仕様OSについて説明を行う．

2.1　ARMプロセッサ

ARMプロセッサとは，主に組込みシステムで広く利用されている32ビットRISCアーキテクチャのプロセッサである．本節では，このARMプロセッサの特徴について説明を行う．

2.1.1　ARMプロセッサの特徴

ARMプロセッサはバッテリー搭載機器などの多い組込み用途向けに最適化されており，超低消費電力で動作することが可能である．この理由は，ARMプロセッサが比較的単純なアーキテクチャで構成されており，小さなダイサイズでチップ化が可能だからである．他にも，ARMプロセッサは，特徴として以下の機能を備える．

- **インライン・バレル・シフタ**
  インライン・バレル・シフタは，命令で使用するレジスタオペランドに対して，前もって各種シフト操作を行うことができるハードウェアである．これにより，シフト操作を各種命令と統合する事が可能となり，シフト命令を別命令として実装しているプロセッサに比べて，コード密度が高くなる．

- **複数レジスタ転送命令**
  複数レジスタ転送命令は，1命令で複数のレジスタに対してロード，もしくはストアを実行することができる命令のことである．この命令は複数クロックサイクルかかることがあるが，コード密度が高くなる利点がある．

- **条件実行**
  条件実行は，特定の条件（オーバーフロー発生，キャリー発生など）を満たした時

にだけ，命令を実行する機能である．ARM 命令セットでは，全ての命令において条件実行が可能であり，この機能によって分岐命令の削減が可能である．

- **Thumb16 ビット圧縮命令セット**
  ARM はプロセッサコアを拡張して，Thumb と呼ばれる 16 ビットの命令セットを実装している．同じプログラムにおいて，この Thumb 命令セットは，ARM 命令セットに比べて実効命令数が増加する傾向があるが，総コードサイズを削減することが可能である．Thumb 命令セットの詳細は次節で説明を行う．

- **IRQ，FIQ ２つの外部割込み**
  ARM プロセッサには，外部からの割込み要求に IRQ(通常割込み) と FIQ(高速割込み) の 2 種類が用意されている．FIQ と IRQ の異なる点を以下に挙げる．
  - 例外の優先順位が FIQ の方が高く設定されている．
  - モードが切り替わった際に使用可能になるバンクレジスタの数が IRQ は 2 つだが，FIQ は 7 つとなっており，レジスタ退避のペナルティが軽減される．
  - FIQ はベクタアドレスが最上位に設置されており，直接例外処理を記述することが可能である．

2.1.2 Thumb 命令セット

一部の ARM プロセッサに搭載されている Thumb 命令セットは，メモリサイズの制約が厳しい組込みシステムにおいて，命令コードサイズの圧縮を目的とした命令セットである．Thumb の特徴として 16 ビットの命令長であることがあげられる，これにより命令長が 32 ビットの ARM 命令セットに比べて約 30%の命令コードサイズの削減が可能となる．しかし，Thumb 命令セットは，ほぼ全てのデータ処理命令が 2 オペランド方式を採用しており，3 オペランド方式の ARM 命令セットに比べてプログラムの実行速度が遅くなる傾向がある．その他にも表 2.1 のようにコア命令数，扱えるレジスタの数などにも Thumb 命令セットは制限がある．

表 2.1: ARM，Thumb 機能比較

機能	ARM	Thumb
命令コードサイズ	32 ビット	16 ビット
コア命令	58	30
レジスタ	16 の汎用レジスタ	8 の汎用レジスタ
インライン・バレル・シフタ	使用可	使用不可
条件実行	全命令で使用可	一部命令で使用可

Thumb モードを使用する場合は，コンパイル時にオプションを指定する事により，ルーチン単位で使用する事が可能となる．

2.1.3 ARM-Thumb インターワーキング

ARM モードから Thumb モードへの状態遷移は ARM プロセッサのバージョンが ARMv4T 以前の場合においては，ベニア関数により行う．図 2.1 はベニア関数により状態の遷移を表したものである．

図 2.1: ベニア関数によるモード遷移

ベニア関数には ARM から Thumb に遷移する場合と，Thumb から ARM に遷移する場合の 2 種類が存在し，機能としてはモードのチェンジと，対象ルーチンへのジャンプを行うことである．ベニア関数は，現在のモードから，他のモードのルーチンを呼び出す回数分だけ必要になる．なおベニア関数はオブジェクトのリンク時に生成される．ベニア関数の例を図 2.2 に示す．

ベニア関数 (ARM→Thumb)

アドレス	命令	モード	備考
1000:	ldr r12 [pc , #0]	ARM	PC + 8 のアドレス(1008)からロード
1004:	bx r12	ARM	r12に格納されたアドレスにジャンプ
			r12[0] == 1 のためThumb モードに状態遷移
1008:	02000951	DATA	Thumbコードの書かれたルーチンのアドレス

ベニア関数 (Thumb→ARM)

アドレス	命令	モード	備考
2000:	bx pc	Thumb	PC + 4 のアドレス(2004)へジャンプ
			PC[0] == 0 のためARMモードに状態遷移
2002:	nop (mov r8 r8)	Thumb	PC + 4 の実現のために必要なNOP命令
2004:	b 02000004	ARM	ARMコードの書かれたルーチンへジャンプ

ARMv4Tの場合

図 2.2: ベニア関数の例

なお ARMv4T 以降の ARM プロセッサでは，BLX 命令が導入されており，この命令のみでベニア関数と同等の動作を実現することができる．

2.2 μITRON4.0 仕様 OS

本節では，リアルタイムオペレーティングシステム (RTOS) の 1 つである，μITRON4.0 仕様 OS の特徴について説明をする．

2.2.1 μITRON4.0 仕様 OS の機能

μITRON4.0 仕様 OS では，機能として，タスク管理機能，タスク付属同期機能，同期・通信機能，メモリプール管理機能，時間管理機能，システム状態管理機能，割込み管理機能がある．それらの機能について説明を行う．

- **タスク管理機能**
  タスク管理機能は大きく分けて，タスクの管理，タスクの状態，スケジューリングにわかれる．以下はその概要である．詳細については，[2] を参考されたし．

  - **タスクの管理**
    μITRON4.0 仕様では，並行処理するプログラムの単位をタスクと呼ぶ．タスクを管理する主な情報として，タスク ID(タスクの名前)，タスクの状態，タスクの優先度，タスクのスタートアドレス，スタックに関する情報の格納領域，タスクの使用するレジスタのセーブ領域などがある．これらの管理情報はタスクコントロールブロック (TCB) に格納される．
  - **タスクの状態**
    μITRON4.0 仕様では，タスクのとりうる状態として，
    * 実行状態 (RUNNING)
    * 実行可能状態 (READY)
    * 待ち状態 (WAITING)
    * 強制待ち状態 (SUSPENDED)
    * 二重待ち状態 (WAITING SUSPENDED)
    * 休止状態 (DORMANT)
    * 未登録状態 (NON-EXISTENT)

    の 7 つの状態が定められており，その状態遷移は図 2.3 となる．
  - **スケジューリング**
    μITRON4.0 仕様でのタスクのスケジューリングは，タスク間の優先度を基準にして行われる．例えば，複数のタスクが実行可能状態にある時，実行されるタスクは最も優先度の高いものとなる．また優先度が同じタスク同士の場合は，First Come First Served 方式 (最初に来たもの順) で実行される．

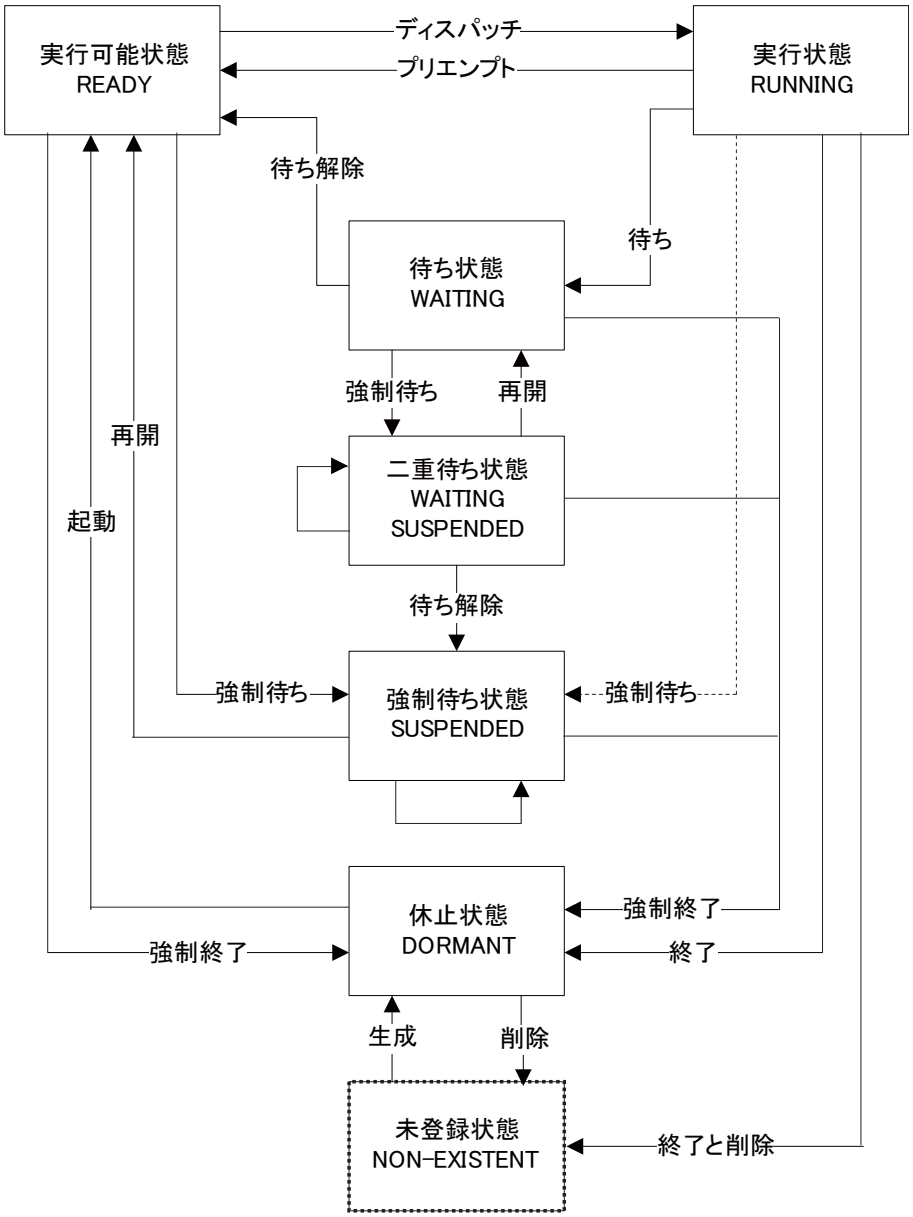

図 2.3: タスクの状態遷移

- **タスク付属同期機能**
  タスク付属同期は，タスクの状態を直接的に操作することで同期を行うための機能である．具体的にはタスクを起床待ち状態にしたり，待ち状態から起床させたりする機能，タスクの起床要求をキャンセルする機能などがある．

- **同期・通信機能**
  μITRON4.0 仕様では，タスク間の同期・通信機能として，主に以下の機能が挙げられる．

- **セマフォ**
  主に資源の排他的利用で使用される．
- **イベントフラグ**
  主にタスク間同期に使用される．
- **データキュー**
  タスク間での通信やデータ交換で使用される．(優先度なし)
- **メールボックス**
  タスク間での通信やデータ交換に使用される．(優先度あり)

これらの機能を使い分けることにより，アプリケーションに最適なシステムを組み上げることができる．

- **メモリプール管理機能**
  メモリプールはアプリケーションが動的にメモリを使用する場合において使用される．

- **時間管理機能**
  時間管理は，時間に依存した処理を行うための機能である．具体的には，システム時刻を操作する機能としてシステム時刻を設定/参照，タイムティック (OS の時間管理単位) を供給してシステム時刻を更新する機能がある．また，一定周期で起動される周期ハンドラ機能として周期ハンドラの生成/削除，動作の開始/停止，状態を参照する機能がある．

- **システム状態管理機能**
  システム状態管理は，システムの状態を変更，参照するための機能である．具体的には，タスクの優先度を回転 (ラウンドロビンスケジューリング等で使用)，実行状態のタスク ID を参照する，CPU をロック状態へ移行/解除，タスクディスパッチの禁止/解除，コンテキストやシステム状態を参照する機能がある．

- **割込み管理機能**
  μITRON4.0 仕様では，割込みによって起動される処理として，割込みハンドラと割込みサービスルーチンがあり，いずれか一方あるいは両方の機能が提供される．

2.2.2 静的 API とコンフィギュレータ

システムコンフィギュレーションファイル中に記述し，カーネルやソフトウェア部品の構成を決定したり，オブジェクトの初期状態を定義するためのインターフェースを静的 API と呼ぶ．静的 API は，μITRON4.0 仕様になって定められた機構であり，静的 API の導入は，これまで実装ごとに定められていた静的オブジェクト情報の記述方法を一般化し，ソースコードの流用を容易にすることを目的としている．

静的 API を利用するには，システムコンフィギュレーションツール (コンフィギュレータ) を用意する必要がある．コンフィギュレータの詳細に関しては，次章で説明を行う．

第3章　μITRON4.0仕様アプリケーション用バイナリコンフィギュレータの実装

本研究では，ARM プロセッサを対象としたμITRON4.0 仕様アプリケーション用のバイナリコンフィギュレータの実装を行った．本章では，μITRON4.0 仕様アプリケーション従来のコンフィギュレータと，実装したバイナリコンフィギュレータのアルゴリズムについて説明を行う．

3.1　μITRON4.0仕様アプリケーションのコンフィギュレーション

コンフィギュレータとは，システムコンフィギュレーションファイルに記述された静的API 情報を解析し，μITRON4.0 仕様アプリケーションの実行に必要な，ヘッダーファイル，カーネル情報ファイルを出力することを目的としたプログラムである．ヘッダーファイルには，システムを構成するオブジェクトのID 番号の定義がされており，タスクのソースファイルにインクルードされる．カーネル情報ファイルは，オブジェクトの管理ブロックなどの情報が記述されており，カーネルとリンクする．μITRON4.0 仕様では，コンフィギュレーション方法も標準化されており，その処理のおおまかな流れは図3.1 になる．まず system.cfg というシステムコンフィギュレーションファイルをC 言語プリプロセッサに通す．次にプリプロセッサに通したシステムコンフィギュレーションファイルをコンフィギュレータにかける．そしてコンフィギュレータがヘッダーファイルとカーネル情報ファイルを出力する．

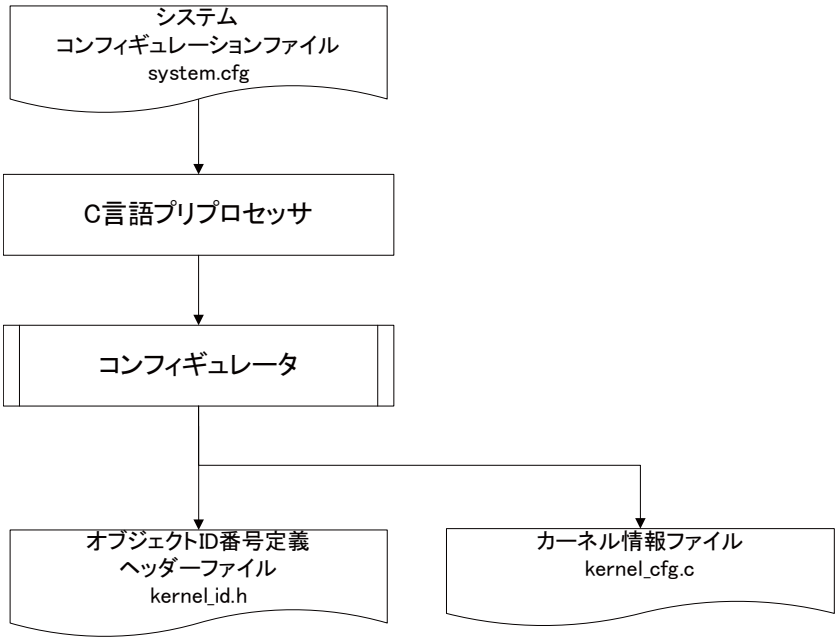

図 3.1: 従来のコンフィギュレーション手順

3.2 提案手法のコンフィギュレーション

本研究で提案するμITRON4.0 仕様アプリケーション用バイナリコンフィギュレータは図 3.2 の処理手順となる．処理の流れは C 言語プリプロセッサに通した，システムコンフィギュレーションファイルと，タスク用関数ファイル (各タスクの実行する関数を全てひとまとめにしたファイル) の 2 つを解析し，ITRON ライブラリ用の Makefile のひな形と，ITRON タスク用 Makefile のひな形を基に，2 つの Makefile を出力する．

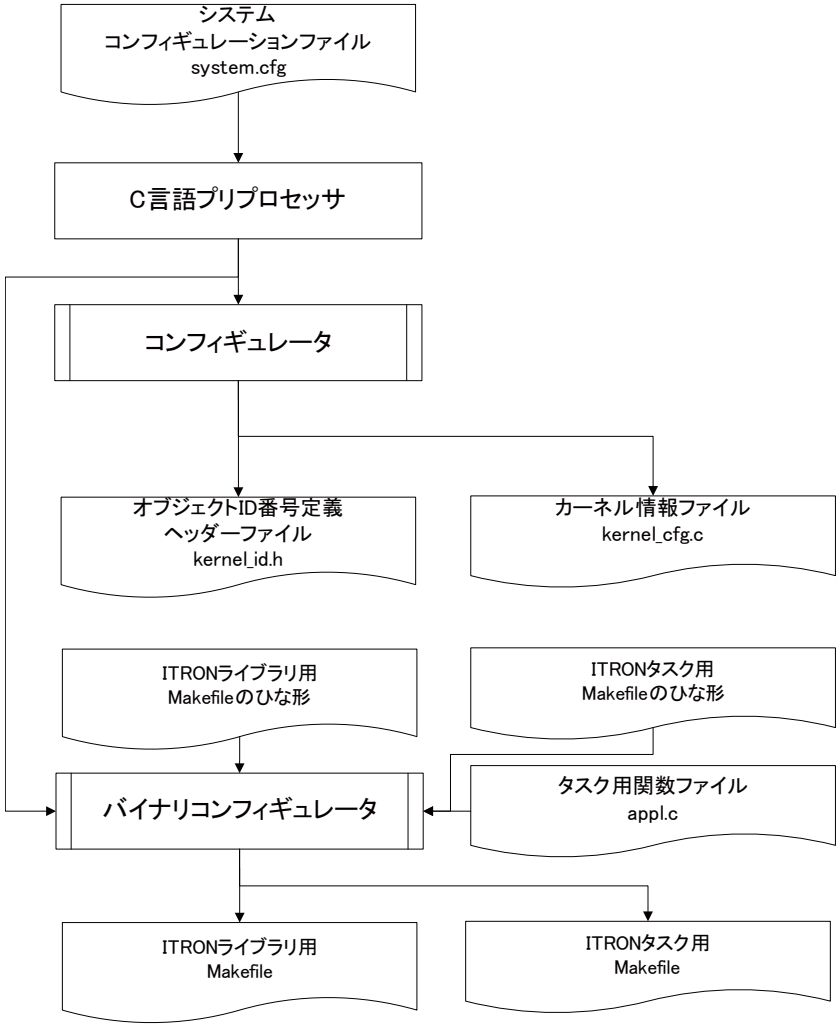

図 3.2: 提案手法のコンフィギュレーション手順

3.3 提案手法のアルゴリズム

本節では実装したバイナリコンフィギュレータのアルゴリズムについて説明を行う．バイナリコンフィギュレータによるコンフィギュレーションは 2 段階に分かれており，1 段階目は ITRON ライブラリに対して，2 段階目は ITRON タスクに対して行う．

3.3.1 ITRON ライブラリのコンフィギュレーション

本研究で使用する µITRON4.0 仕様 OS は 79 個のサービスコールとカーネルで構成されている．79 個のサービスコールは全て C 言語で記述されており，Makefile に記述されている各サービスコールのコンパイルオプションを適切に設定することにより，ARM モード,

Thumb モードとモードを変更することが可能である．ここではサービスコールとカーネルをまとめたITRON ライブラリに対してのコンフィギュレーションについて説明を行う．

ライブラリのコンフィギュレーションの例は図 3.3 となる．処理の流れを説明すると，Makefile 生成ツールが静的 API 情報に記述されたタスクの静的優先度を基に Makefile に記述されたサービスコール用のコンパイルオプションを変更し，Makefile の生成を行う，そして，最後に生成された Makefile で Make を実行しライブラリを生成するという流れになる．その後，ライブラリはITRON バイナリの生成時にリンクして使用される．

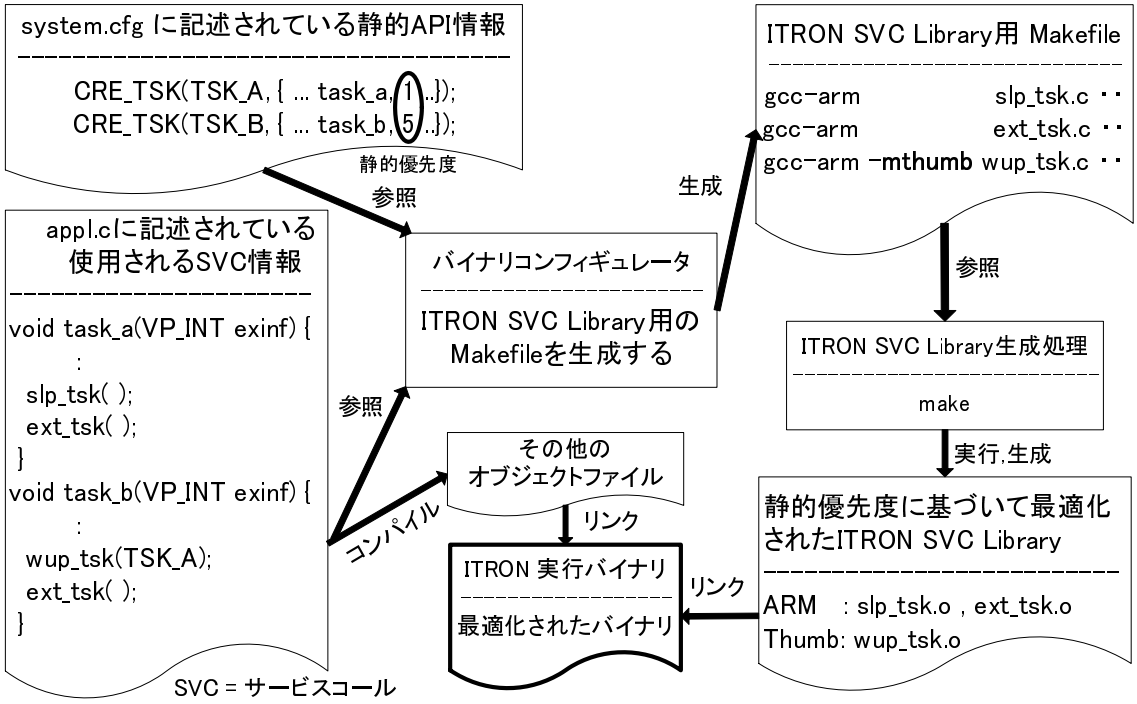

図 3.3: サービスコールライブラリに対するコンフィギュレーションの概要

Makefile の変更を行うバイナリコンフィギュレータのアルゴリズムは

1. コンフィギュレーション前処理
2. サービスコールオプション決定処理
3. 静的優先度に従ったモードフラグ決定処理
4. フラグに基づいた Makefile 生成処理

の順になる．

図 3.4 はコンフィギュレーションの前処理をフローチャートで表したものである．コンフィギュレーションは，μITRON のコンフィギュレーションファイルである system.cfg をプリプロセッサに通し，オープンする処理から始まる．このファイルには静的 API の情報が記述されている．

次にオープンしたコンフィギュレーションファイルから１行読み出し，そこに記述されている文字列が静的 API である場合は，以下の処理を行う．静的 API が CRE_TSK(タスクの定義) の場合は，タスクの静的優先度と実行する関数の情報が記述されているため，それを取得する．静的 API が CRE_CYC(周期ハンドラの定義) , DEF_INH(割込みハンドラの定義) , ATT_INI(初期化ルーチンの定義) のいずれかであった場合は，実行する関数情報が記述されているので，それを取得する．他にも静的 API は存在するが，それらに対して処理は行わない．この処理はコンフィギュレーションファイルの末尾に至るまで行う．

次に，ITRON サービスコールのモードを決定する処理について説明を行う．図 3.5 はサービスコールのモードを決定する処理をフローチャートで表したものである．この処理では最初に，静的優先度の閾値を設定する．この閾値に従って，ITRON タスク，ITRON サービスコールは ARM モードか，Thumb モードのどちらかのモードでコンパイルするかが決定される．

次に全タスクの実行する関数情報が記述されているファイル appl.c をプリプロセッサを通してからオープンする．この appl.c を１行ずつ読み込んでいき，先のコンフィギュレーション前処理で得た関数名が出現した場合，関数内で使用されているサービスコールを検索する．サービスコールが出現した場合，図 3.6 のフローチャートにある静的優先度に従ったモードフラグ決定処理に移る．この静的優先度に従ったモードフラグ決定処理では，まず関数が非タスクの設定であるかどうかをチェックする．非タスクとは，CRE_TSK 静的 API 以外で定義されたタスクの実行する関数のことを指す．非タスクには割込みハンドラ，周期ハンドラ，初期化ルーチンがあり，システムにおいて速度が求められる重要な処理が多い．よって非タスクの関数から呼び出されるサービスコールは全て ARM モードでコンパイルするフラグを設定する．次に静的優先度の閾値とその関数を呼び出すタスクの静的優先度を比較する．閾値より優先度が高い場合は ARM モードでコンパイルするフラグを設定する．

次に，先の処理で設定したフラグを基にサービスコールのオプションを決定する処理について説明を行う．図 3.7 はフラグに基づいた Makefile 生成処理の処理手順を表したものである．この処理では最初に，ITRON サービスコールライブラリの Makefile をオープンし，そこに記述されている，79 個のサービスコールの生成処理に対してコンパイルオプションの変更を行う．具体的には，ARM モードでコンパイルするフラグが設定されているサービスコールは，コンパイルオプションに何も変更を加えず，それ以外のサービスコールはオプションに -mthumb (Thumb モードでコンパイルするためのオプション) を追加する．この処理を 79 個全てのサービスコールに対して行う．

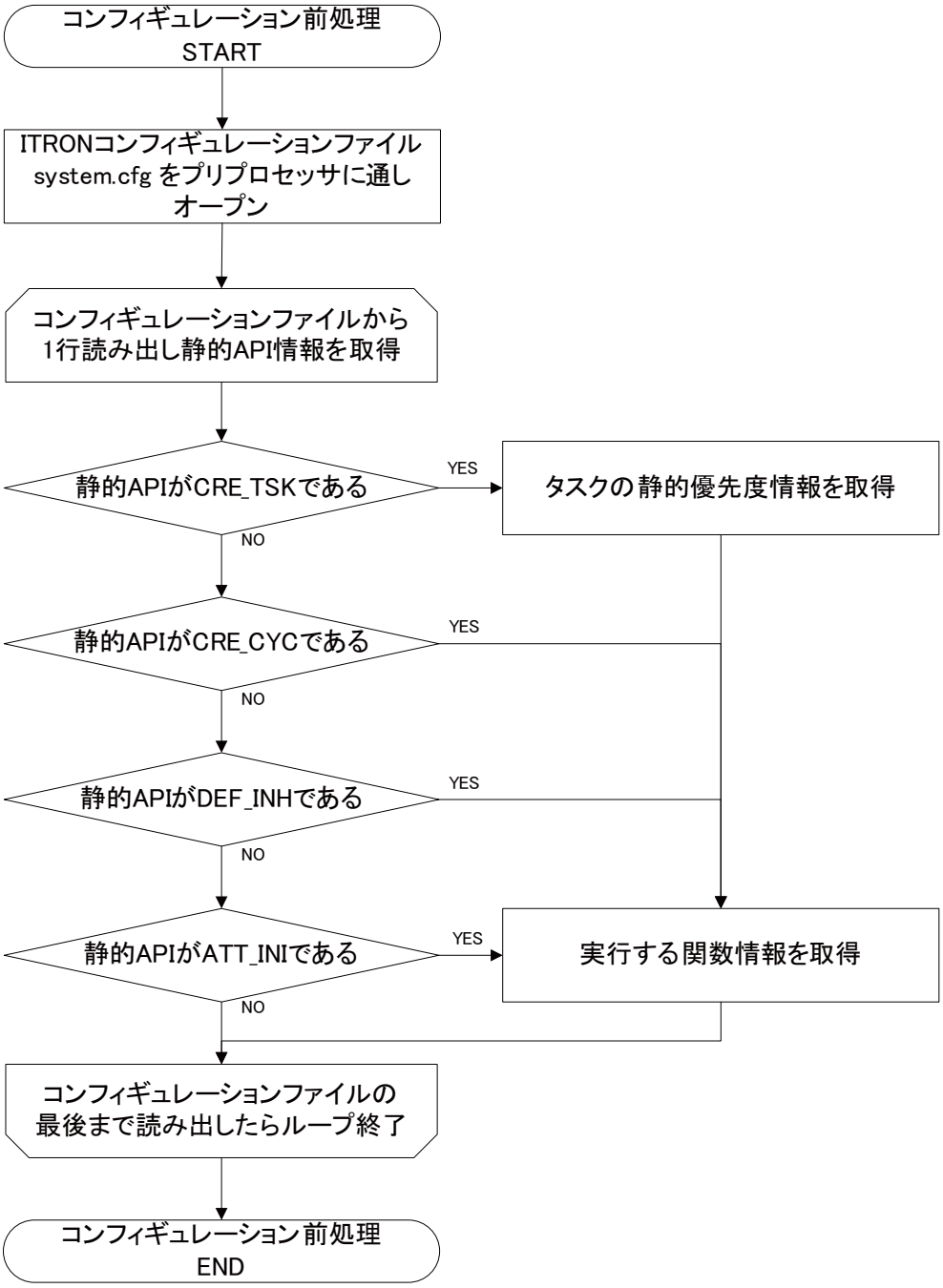

図 3.4: コンフィギュレーション前処理

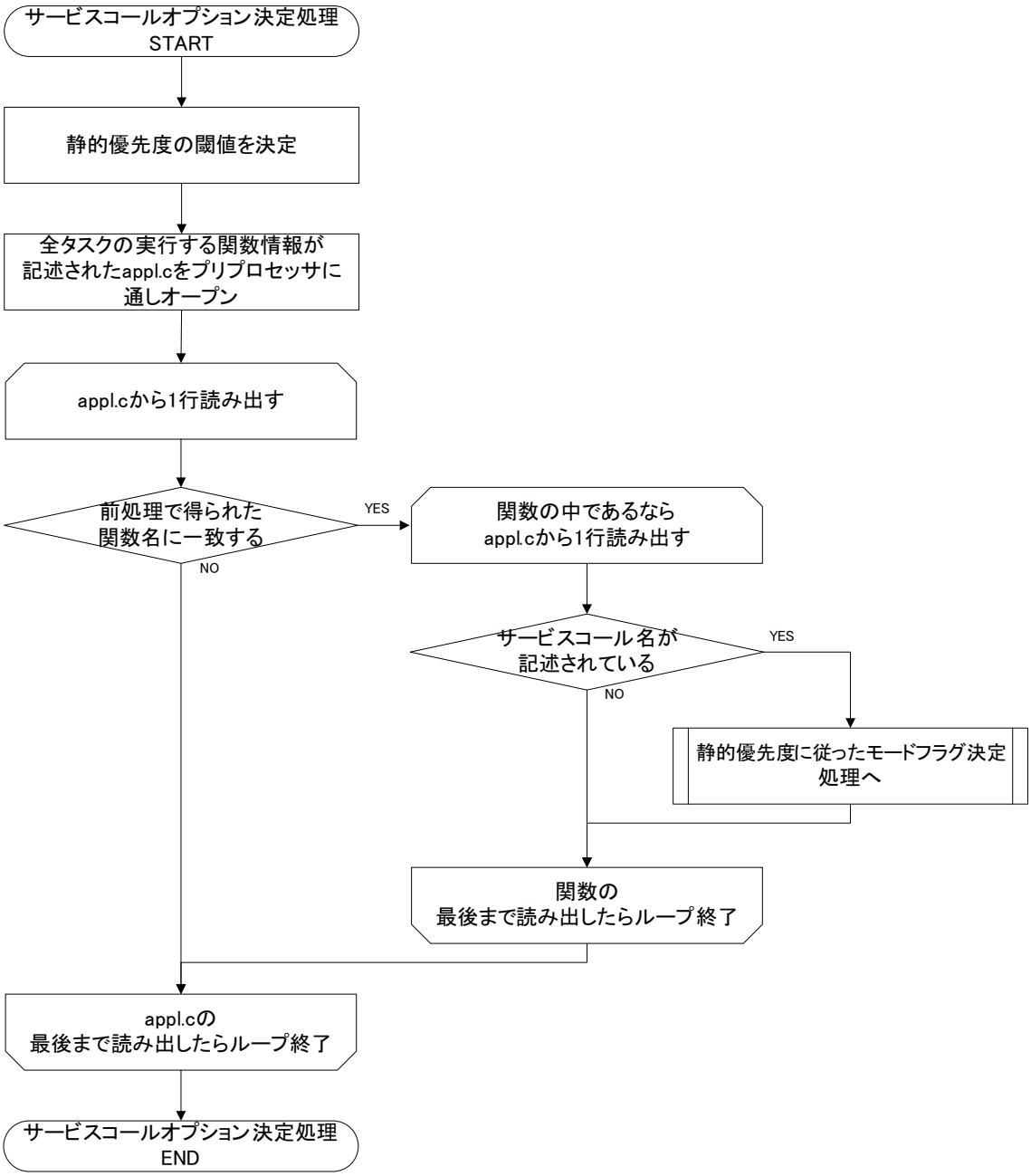

図 3.5: サービスコールオプション決定処理

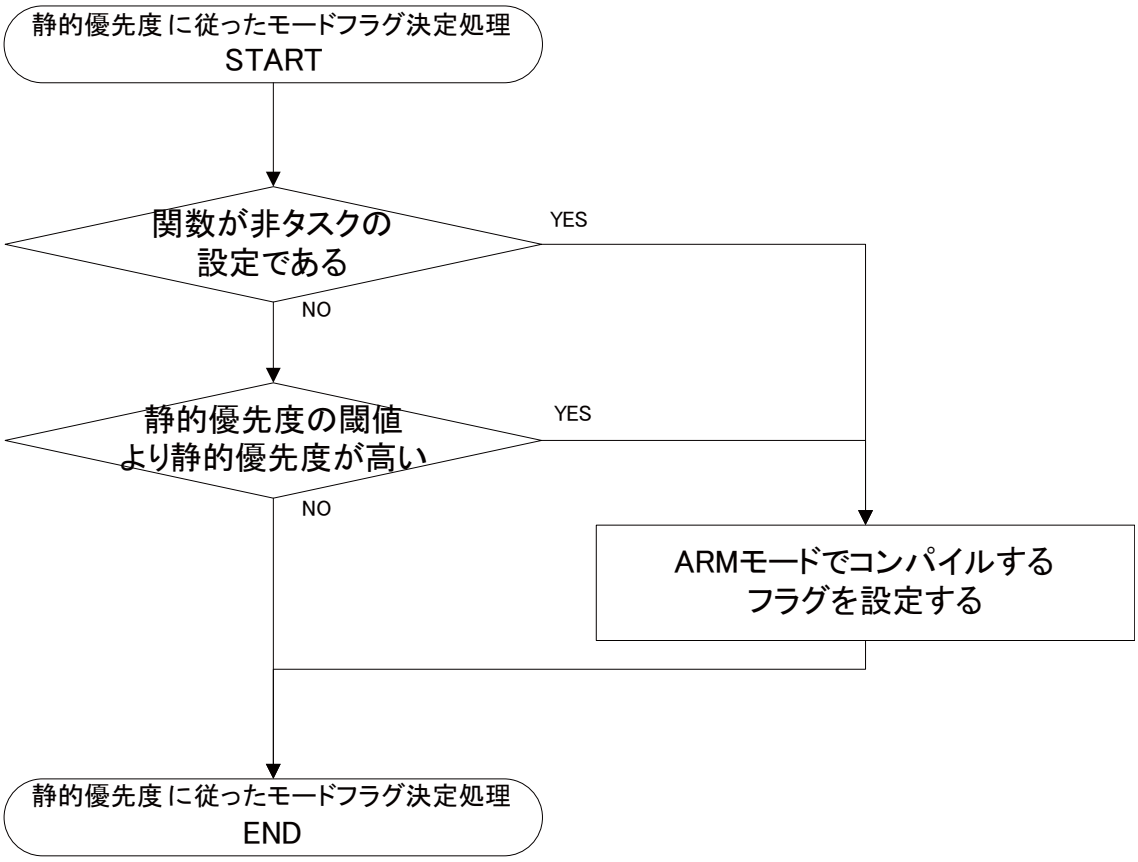

図 3.6: 静的優先度に従ったモードフラグ決定処理

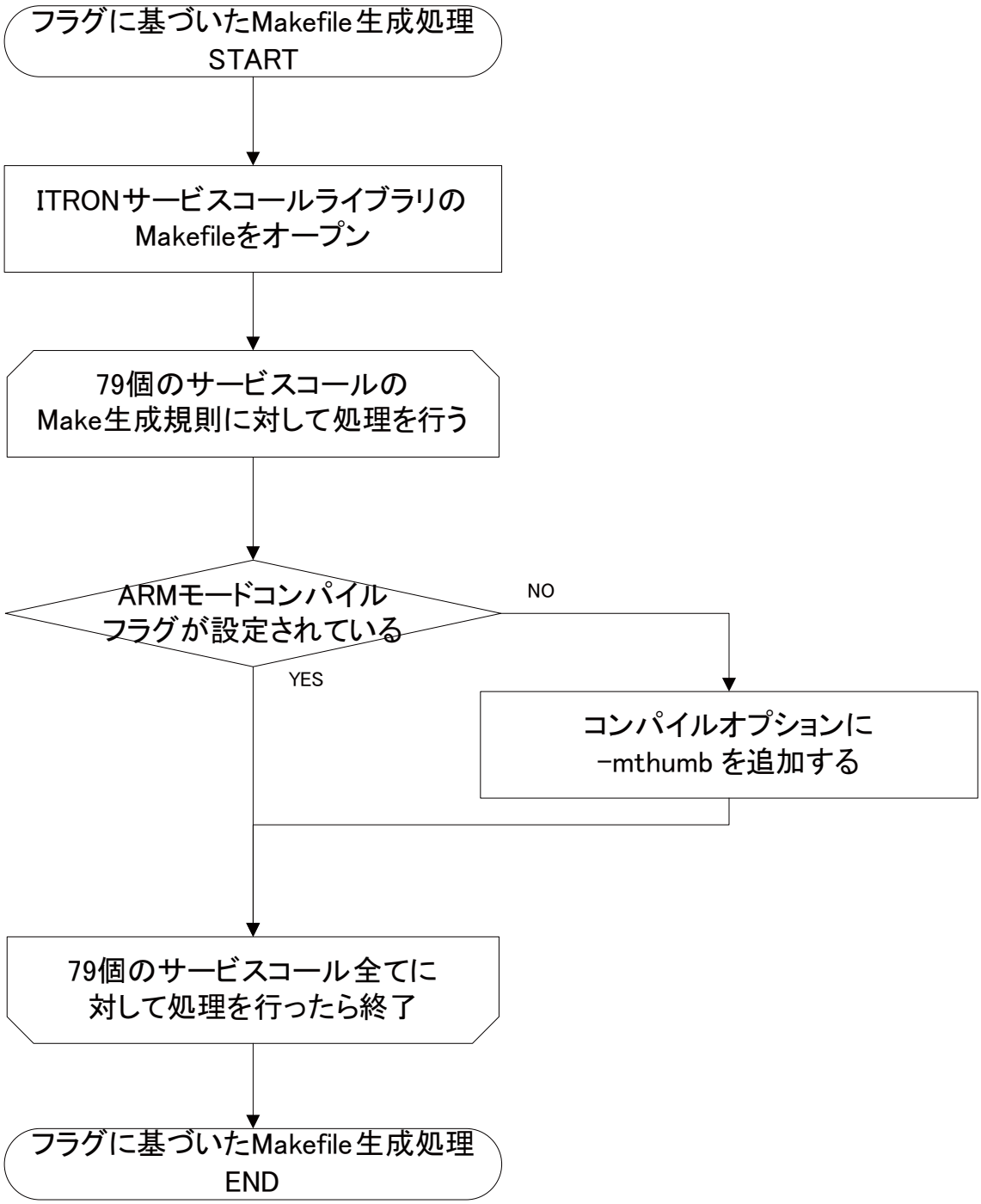

図 3.7: フラグに基づいた Makefile 生成処理

3.3.2 ITRON タスクのコンフィギュレーション

次に，ITRON タスクのコンフィギュレーションついて説明を行う．タスクに対するコンフィギュレーションの例は図 3.8 となる．処理の流れとしては，まず ITRON ライブラリのコンフィギュレーションと同様に，システムコンフィギュレーションファイルに記述された CRE_TSK 静的 API から静的優先度，実行する関数の情報を取得する．次にバイナリコンフィギュレータが ITRON タスク用の Makefile のひな型を参照し，設定された静的優先度の閾値とタスクの静的優先度を比較して，タスクを ARM モードでコンパイルするか，Thumb モードでコンパイルするかを決定する．その後，バイナリコンフィギュレータが ITRON タスク用の Makefile を生成する．生成された Makefile を用いて，Make を実行しオブジェクトファイルを生成する．そしてそれらのオブジェクトファイルをリンクして ITRON バイナリを生成する．

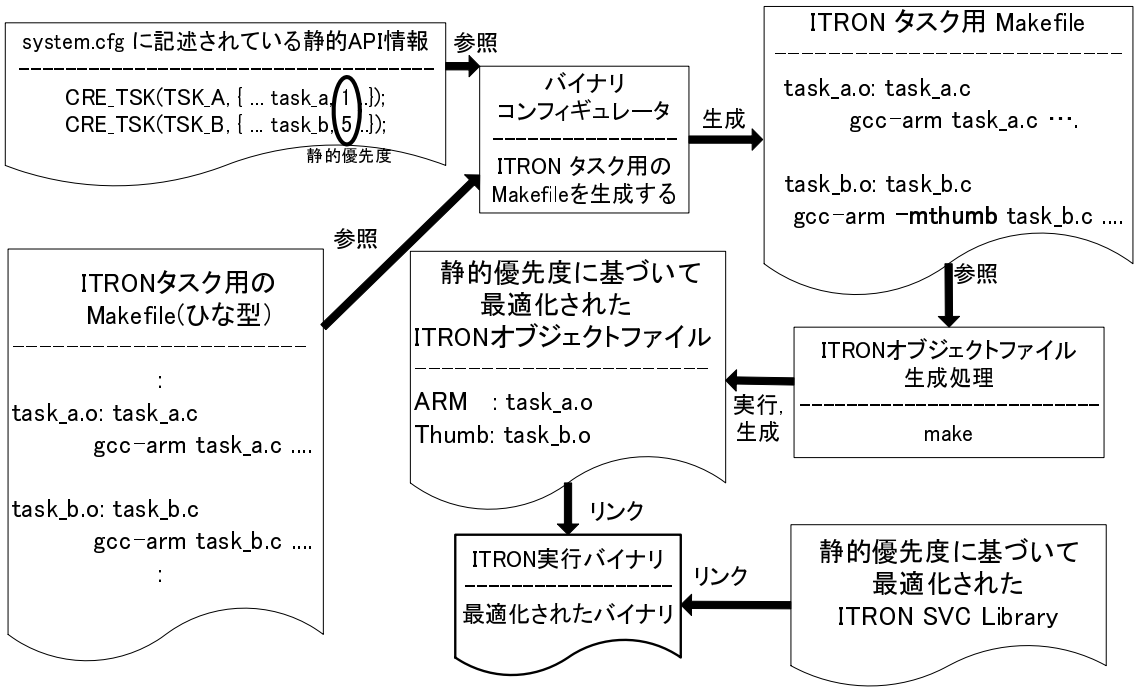

図 3.8: タスクに対するコンフィギュレーションの概要

第4章　評価

本章では，シミュレーション環境，評価関数について述べた後，提案手法をシミュレーションで評価した結果と，その考察について述べる．

4.1　評価環境

本節では，評価に用いる ARM シミュレータの仕様と，評価用タスクセットについて説明を行う．

4.1.1　ARM シミュレータ仕様

本研究では ARM7TDMI 搭載機器 (ARMv4T) に準拠し，μITRON4.0 仕様アプリケーションのバイナリを実行することを想定したシミュレータを作成した．シミュレータは ARM コード，Thumb コードの両方が実行可能であり，命令，クロックサイクル，実行タスクの単位でシミュレーションを行うことができる．ARM7TDMI は組込み用途向けのプロセッサのため，キャッシュメモリ，MMU などは搭載していない．命令は基本的に 1 命令 1 クロックサイクルで実行を行うが，以下の命令に関してはペナルティが発生する．

- **乗算命令**
  +2 クロックサイクル
- **複数ロード/ストア命令**
  +転送するレジスタ数 − 1 クロックサイクル

割込みに関しては，14 本の外部割込みチャネルを搭載し，設定ファイルによりクロックサイクル単位で制御することが可能である．14 本の内 4 つはタイマー機能としており，4 つの 16 ビットカウンタを内部に持つ．カウンタ値がオーバーフローすると割込みが発生する．

バイナリのシミュレーションを実行後，シミュレータは以下の情報を出力する．

- 各タスクの平均応答時間
- 全タスクの平均応答時間

- タスクの実行完了数
- タスクのデッドラインミス数
- デッドラインミスしたタスクの平均優先度
- CPU 使用率
- ARM，Thumb の CPU 使用比率
- 実行命令数
- 実行クロックサイクル数

4.1.2 実行バイナリの生成

評価に用いる実行バイナリの生成について説明を行う．図 4.1 は実行バイナリの生成から ARM シミュレータでバイナリをシミュレーションするまでの流れを表したものである．実行バイナリは，ITRON サービスコールライブラリと，ITRON タスク，その他関数 (割込みハンドラ，文字列処理関数，文字列出力関数など) をコンパイル，リンクしたもので構成される．なお本研究で使用する C 言語クロスコンパイラは GCC Ver3.3.2 である．実行バイナリ生成後は，ARM シミュレータにてシミュレーションを行い，結果を得る流れとなる．

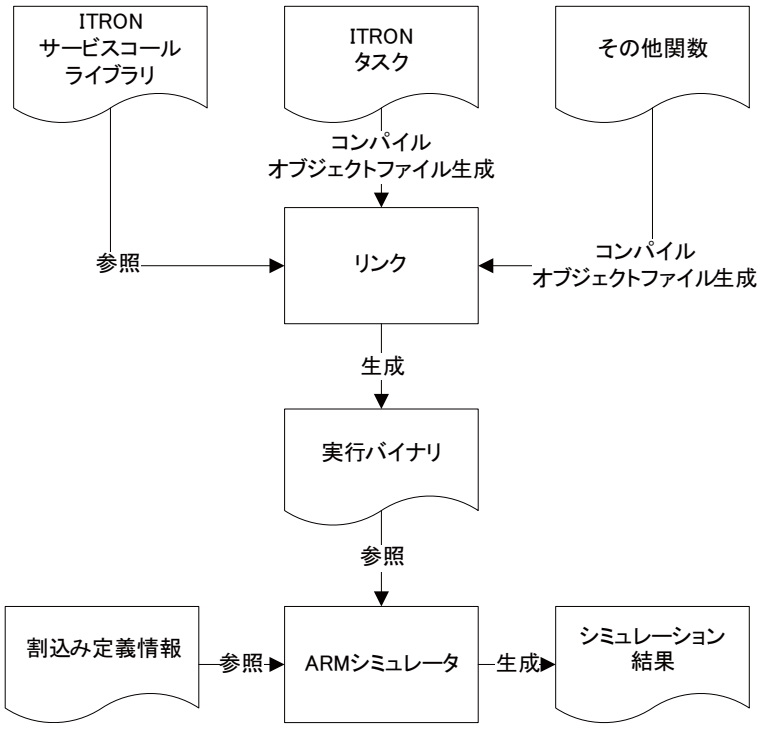

図 4.1: 実行バイナリ生成からシミュレーションまで

4.1.3 評価用タスクセット

評価に用いるタスクセットの概要について説明を行う．タスクセットを構成するタスクの詳細を表 4.1 に示す．優先度は値が小さいほど，優先度が高いことを意味する．タスクには周期タスクと，非周期タスクの 2 種類が存在する．本研究ではタスクを 12 個使用する．

- **非周期タスク**
  タスク ID が 2 のタスクは初期化ルーチンが起動要求を発行する．タスク ID が 3〜9 のタスクは，タスク ID が 2 のタスクが起動要求を発行する．

- **周期タスク**
  タスク ID が 10〜13 のタスクは，定期的な割込みが発生する度に，割込みハンドラから起動要求を発行する．

上記の他，タスクセットには非タスクとなる初期化ルーチン，周期ハンドラも含まれている．初期化ルーチンはスタートアップ時に 1 回，周期ハンドラは 200tick ごとに実行される．なお 1tick は 16780 クロック (約 1ms) と設定されている．相対デッドライン時間の設定は，後述する ALL_ARM モードで実行バイナリを作成し，それをシミュレータに通し，得られたタスクごとの平均応答時間を参考に設定した．タスク ID が 2 のタスクの相対デッドライン時間を大きな値に設定しているのは，このタスクが無限ループを使用しているため，タスクの平均応答時間が計測できないからであり，そのため，デッドラインミスが発生しないように大きな値を設定した．また全タスクセットで使用したサービスコールの数は 31 個である．

表 4.1: 評価用タスクの情報

タスク ID	優先度	周期/非周期	周期 (クロック)	相対デッドライン時間
2	10	非周期	-	16780000
3	1	非周期	-	62000
4	2	非周期	-	50000
5	3	非周期	-	70000
6	4	非周期	-	4000
7	5	非周期	-	1000
8	6	非周期	-	60000
9	7	非周期	-	7000
10	0	周期	500000	2500
11	0	周期	500000	3500
12	0	周期	1000000	4500
13	0	周期	1000000	5000

4.1.4 タスクの時間情報 [3]

本研究におけるタスクの時間情報を以下に定義する (図 4.2).
なお括弧の中に記述されているのは対応するサービスコール名である.

- **起動要求時刻(a)**
  タスクの実行要求 (act_tsk，iact_tsk) が発行され実行可能状態に遷移した時刻.
- **開始時刻(s)**
  タスクが最初に実行を開始した時刻.
- **終了時刻(f)**
  タスクの実行が終了した時刻 (ext_tsk).
- **実行時間(C)**
  プロセッサがタスクの実行時間に費やした時間.
- **応答時間(R)**
  タスクの起動要求が発生してからタスクの実行が完了するまでの時間．$f - a$ によって求まる.
- **相対デッドライン時間(D)**
  タスクの実行要求が発行されてから絶対デッドライン時刻までの時間．各タスクごとに固有の値を持つ.
- **絶対デッドライン時刻(d)**
  タスクの実行が完了していなければならない締め切り時刻．$a + D$ によって求まる.
- **余裕時間(L)**
  タスクの実行が完了した時の絶対デッドライン時刻までの残り時間．$d - f$ によって求まる.
- **周期(T)**
  規則的に繰り返し起動要求が発生するタスクの起動間隔 (図 4.2 には未記載).

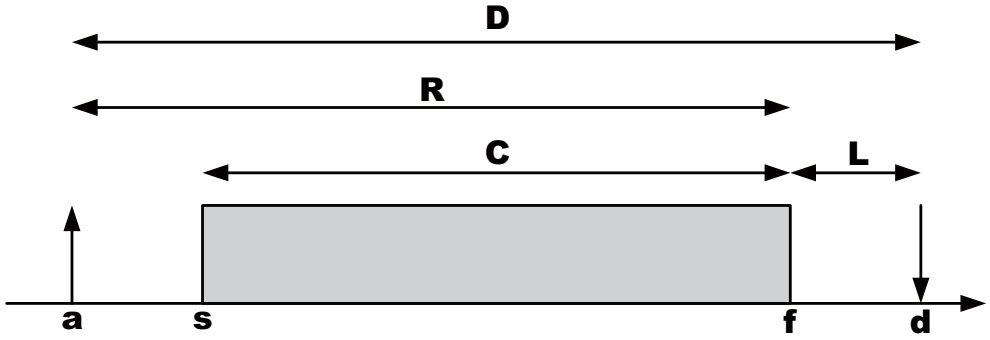

図 4.2: タスクの時間情報

4.1.5 評価関数の定義

提案手法のリアルタイム性を評価する際，デッドラインミス数と平均応答時間を指標とする．これらの指標を評価関数として以下に定義する．

- **デッドラインミス数**
  n 個のタスクが実行完了したとき，デッドラインミス数 N_{miss} を以下に定義する．

$$N_{miss} = \sum_{i=1}^{n} miss(f_i)$$

  ただし $miss(f_i)$ は以下の定義とする．

$$miss(f_i) = \begin{cases} 0 & (f_i \leq d_i \text{ のとき}) \\ 1 & (\text{それ以外のとき}) \end{cases}$$

- **平均応答時間**
  n 個のタスクが実行完了したとき，平均応答時間 R_{avg} を以下に定義する．

$$R_{avg} = \frac{1}{n} \sum_{i=1}^{n} (f_i - a_i)$$

4.2 サービスコールの命令数，コードサイズ

サービスコールを ARM モード，Thumb モードでそれぞれコンパイルし，命令数を比較したものが図 4.3，図 4.4 となる．横軸がサービスコール名，縦軸がサービスコールの命令数となる．コードサイズを比較したものが，図 4.5，図 4.6 となる．横軸がサービスコール名、縦軸がサービスコールのコードサイズとなる．サービスコールは計 79 個あり，命令数の増加率は最大で 71.43%，最小が 13.85%となり，平均は 30.90%となった．コードサイズの縮小率は最大で 43.08%，最小が 14.29%となり，平均は 34.55%となった．

4.3 実行バイナリの評価

本節では，実行バイナリを構成する際に指定したモードと，モード別の実行バイナリの各種評価について説明を行う．

4.3.1 最適化モードの定義

実行バイナリの評価を行うにあたり，最適化モードを以下に定義する．

- **ALL_ARM**
  全タスク，全サービスコール，その他関数を ARM モードに設定．(閾値を 11 に設定)
- **最適化レベル３ (OPT3)**
  閾値を 3 に設定．
- **最適化レベル０ (OPT0)**
  閾値を 0 に設定．
- **ALL_Thumb**
  全タスク，全サービスコール，その他関数を Thumb モードに設定．
  (割込みハンドラ，周期ハンドラは除く)
  OPT0 の最適化モードでは全てのサービスコールは Thumb モードとならない．これは非タスクで使用されているサービスコールは．ARM モードでコンパイルされる仕様になっているためである．(詳細は 3.3.1 節を参照されたし) 非タスクで使用されているサービスコールを全て手動で Thumb モードに設定したのがこの最適化モードである．

ALL_Thumb で割込みハンドラ，周期ハンドラを Thumb モードにしていないのは，それぞれ，ARM コードで記述されたアセンブリソース，ソース内でインラインアセンブラを使用し ARM コードを記述していることが理由である．

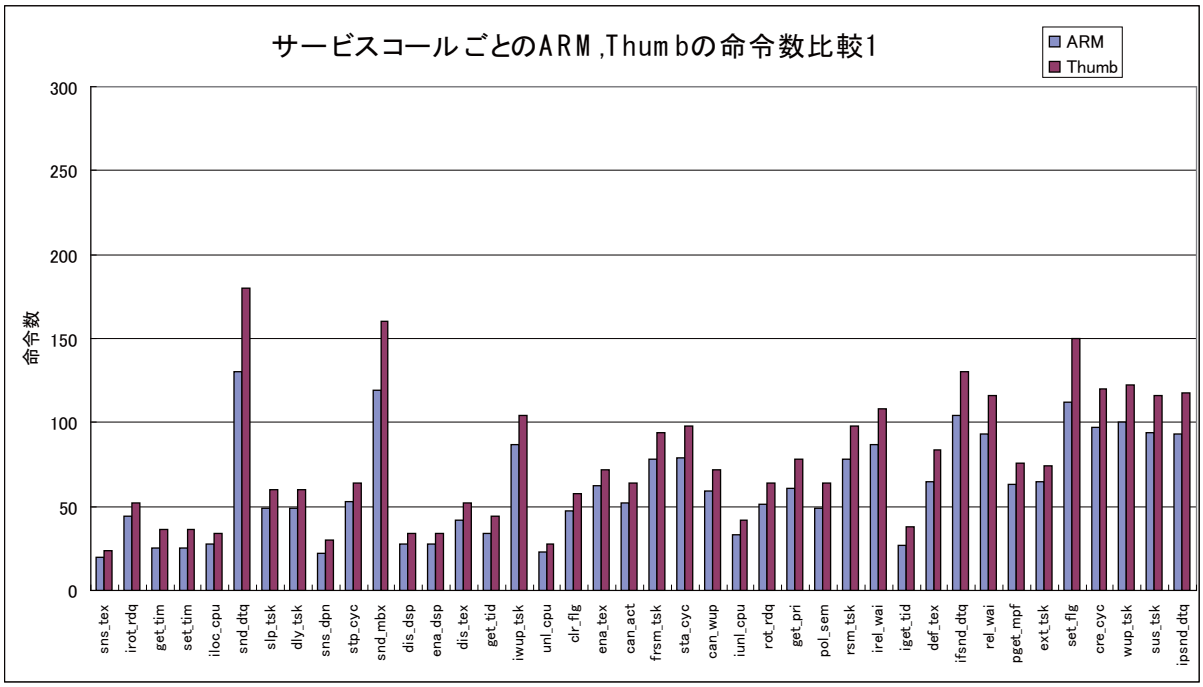

図 4.3: サービスコールの命令数比較 1

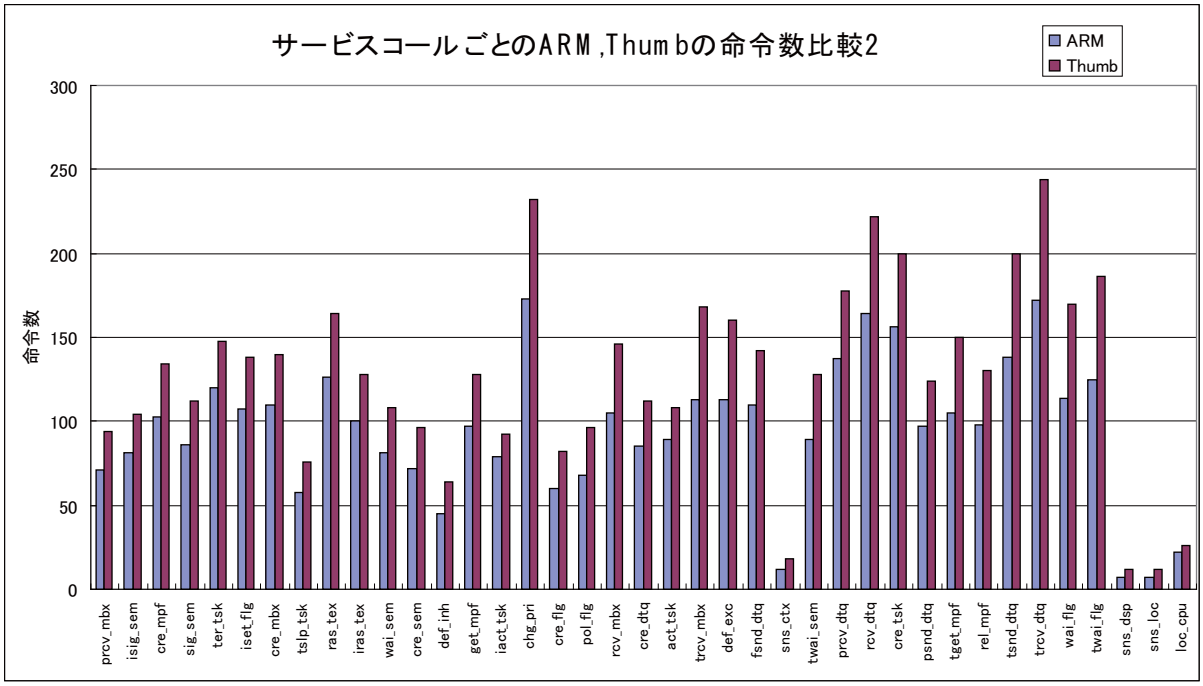

図 4.4: サービスコールの命令数比較 2

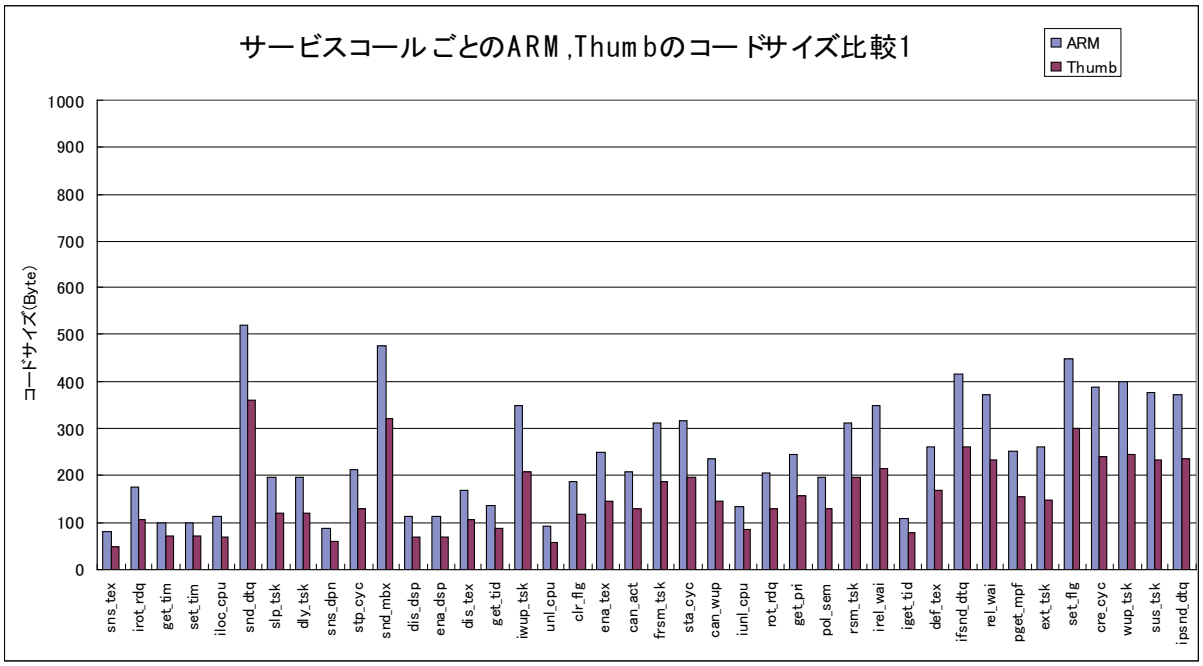

図 4.5: サービスコールのサイズ比較1

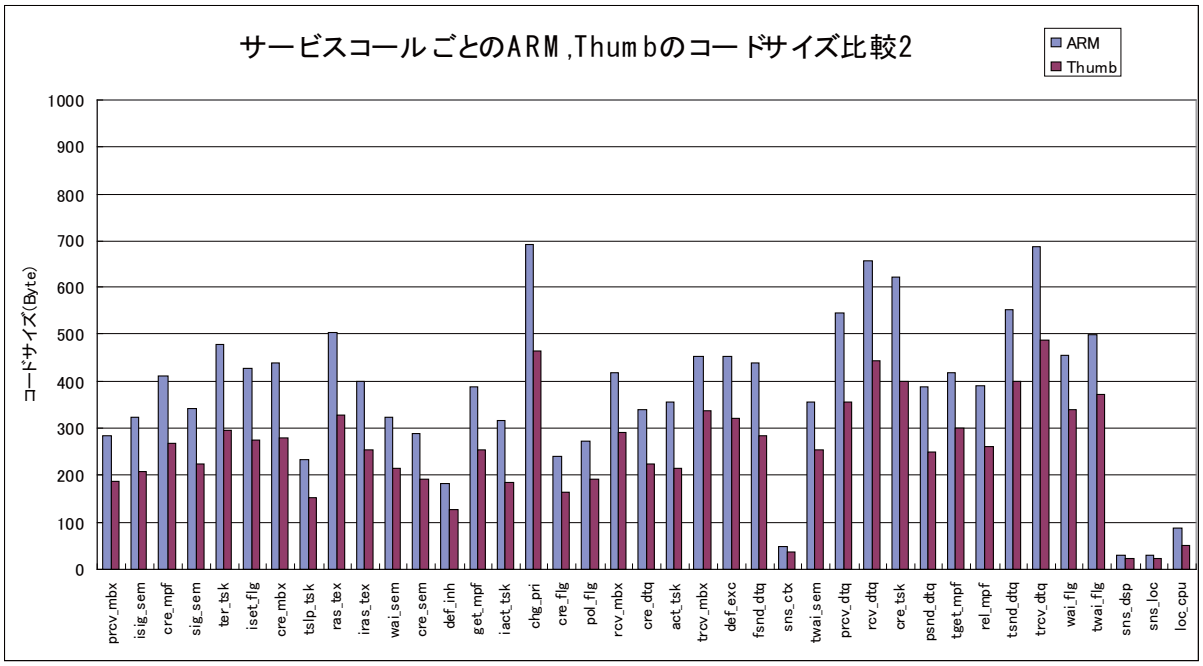

図 4.6: サービスコールのサイズ比較2

各最適化モードでサービスコール，タスクのモード状態を表したのが表 4.2，表 4.3 になる．どちらの表も，表の下に行くほど，Thumb モードの要素が多くなり，ARM の要素が減っている．

表 4.2: 最適化モードごとのサービスコール状態

モード名	ARM サービスコール	Thumb サービスコール
ALL_ARM	31	0
OPT3	26	5
OPT0	11	20
ALL_Thumb	0	31

表 4.3: 最適化モードごとのタスク状態

モード名	ARM タスク	Thumb タスク
ALL_ARM	12	0
OPT3	6	6
OPT0	0	12
ALL_Thumb	0	12

4.3.2 最適化モードごとの実行バイナリサイズ評価

表 4.4 は最適化モードごとの実行バイナリサイズと，その縮小率を ALL_ARM モードを基準に示したものである．モードの構成で Thumb の比率が高くなるほど実行バイナリサイズが縮小している．

表 4.4: 最適化モードごとの実行バイナリサイズ

モード名	バイナリサイズ (byte)	縮小率 (%)
ALL_ARM	31266	0.00
OPT3	30654	1.96
OPT0	28490	8.88
ALL_Thumb	27450	12.20

4.4　シミュレーション結果

本節では実行バイナリを ARM シミュレータで実行し，得られた結果について説明を行う．

4.4.1　シミュレーション条件

シミュレーション実行時の条件について説明を行う．シミュレーションの対象機器は ARM7TDMI 搭載機器で 16.78MHz で動作するものとする，シミュレーションの実行時間は 16780000 クロックとする，これは対象機器において約 1 秒となる．シミュレータで実行されるバイナリを構成するタスクセットの詳細については 4.1.3 節を参照されたし．

4.4.2　タスクの平均応答時間

表 4.5 は，最適化モードごとの全タスク平均応答時間を ALL_ARM を基準に示したものである．増加率は ALL_ARM を基準にどれだけ平均応答時間が増加しているかを示したものである．

表 4.5: 最適化モードごとの全タスク平均応答時間

モード名	平均応答時間	増加率 (%)
ALL_ARM	18745	0.00
OPT3	20141	7.45
OPT0	28906	54.21
ALL_Thumb	28905	54.20

4.4.3 タスクの実行完了数

表 4.6 は，最適化モードごとのタスク実行完了数を ALL_ARM を基準に示したものである．減少率は ALL_ARM を基準にどれだけタスクの実行完了数が減少したかを示したものである．

表 4.6: 最適化モードごとのタスク実行完了数

モード名	タスクの実行完了数	減少率 (%)
ALL_ARM	845	0.00
OPT3	776	8.17
OPT0	579	31.48
ALL_Thumb	579	31.48

4.4.4 デッドラインミス

表 4.7 は，最適化モードごとのデッドラインミス数を示したものである．ミス比率は，タスクの実行完了数の内，デッドラインミスがどれだけ発生しているかを示したものである．

表 4.7: 最適化モードごとのデッドラインミス数

モード名	デッドラインミス数	ミス比率 (%)
ALL_ARM	0	0.00
OPT3	17	2.19
OPT0	220	38.00
ALL_Thumb	220	38.00

4.4.5 シミュレーション結果考察

本節では，先に示した各種計測結果に関して考察を行う．

- **タスクの平均応答時間**
  表 4.5 を考察する．ALL_ARM を基準に見ると，他の最適化モードは全て平均応答時間が増加した．これは Thumb の比率が高くなったためである．

  OPT3 は ALL_ARM に比べると 7.45%の増加となり，他の最適化モードに比べると増加率は低い，OPT0 は 54.21%，ALL_Thumb は 54.20%と近い値となった．これは，表 4.3 にあるように，2 つの最適化モードが全て，タスクのモードが Thumb となっているためだと考えられる．

  また OPT0，ALL_Thumb は，ほぼ同じ平均応答時間となったが，この結果が示すことは，OPT0 を使用するより，より実行バイナリサイズの小さい ALL_Thumb を使用したほうがメモリ使用量の削減が見込め，平均応答時間も増加しないということである．

  また OPT0 と ALL_Thumb を比較すると，若干だが ALL_Thumb の方が平均応答時間が短いが，これはベニア関数の減少が影響していると考えられる．Thumb の比率が高くなると，Thumb から ARM になるベニア関数が少なくなる，結果，ベニア関数のオーバヘッドが小さくなり，このような結果になったと考えられる．

- **タスクの実行完了数**
  表 4.6 を考察する．ALL_ARM を基準に見ると，他の最適化モードは全てタスクの実行完了数が減少した．これはタスクの平均応答時間の増加と同様の現象がおきており，Thumb の比率増大による実行命令数増加の影響である．

  また減少率については，OPT3 は ALL_ARM に比べて，減少率は 8.17%と，他の最適化モードに比べると減少率は低い，OPT0 は 31.48%，ALL_Thumb は 31.48%となり 2 つとも，同じ減少率となった．

- **デッドラインミス数**
  表 4.7 を考察する．相対デッドライン時間の基準を ALL_ARM にしたため，ALL_ARM はデッドラインミス数が 0 である．他の最適化モードは全てデッドラインミスが発生した．これも平均応答時間の増加と同じ現象がおきているためである．

  またミス比率は OPT3 は 2.19%と比較的低い値となった．OPT0 は 38.00%，ALL_Thumb は 38.00%となり 2 つとも，同じミス比率となった．

第5章　おわりに

5.1　まとめ

本研究では，ARM プロセッサ上で動作する µITRON4.0 仕様 OS をターゲットとした，実行バイナリの最適化システムを実装した．そして，その最適化システムを使用して，自動でタスクの静的優先度に従って ARM モードと Thumb モードの混在した実行バイナリを構築した．評価の結果，提案手法により実行バイナリのサイズを削減できることが確認できた．

また，実行バイナリのリアルタイム性を評価するために，ARM シミュレータを実装した．シミュレータによりタスクの平均応答時間，デッドラインミス数などを計測することができた．

5.2　今後の課題

今後の課題として以下の項目が挙げられる．

- **シミュレータ**
  本研究で実装したシミュレータは，主に小規模な組込みシステムで利用される ARMv4T に準拠したものであり，比較的，規模の大きい組込みシステムで使用される ARMv5T 以降のアーキテクチャには対応していない．ARMv5T 以降ではベニア関数ではなく BLX 命令で ARM，Thumb のモード切替えが可能である．この命令により，更に実行バイナリサイズを削減，応答時間を短縮することが可能と考えられる．

- **バイナリコンフィギュレータ**
  現在のバイナリコンフィギュレータの仕様では，どのサービスコールがどの関数で使用されているかは，ソースコードを見る以外に知る方法は無い．そのため，どの関数から，サービスコールが呼ばれているかの情報を示す機能があると最適化の良い判断材料になると考えられる．

謝辞

本研究を行うにあたり，熱心にかつ懇切丁寧に御指導を賜りました，田中清史 准教授に心から深く感謝するとともに，ここに御礼申し上げます．適切な御助言をいただきました日比野 靖 教授，篠田 陽一 教授，井口 寧 准教授に深く感謝致します．

その他，貴重な御意見，後討論をいただきました，田中・日比野研究室の皆様をはじめとする多くの方々の御助言にに対して厚く御礼申し上げます．

最後に，日頃から暖かく支援してくださいました両親に感謝致します．

参考文献

[1] ARM Limited.“ ARM Architecture Reference Manual ”,Pearson Education, 1996-2000.

[2] トロン協会.“ μITRON4.0 仕様 Ver.4.03.00 ”.

[3] G. C. Buttazzo.“ Hard Real-Time Computing Systems：Predictable Scheduling Algorithums And Applications ”Springer, 2004.