修士論文

題目

ソフトマクロを用いた
リアルタイムシステムの高性能、
低電力化に関する研究

指導教員

近藤 利夫 教授

2015年

三重大学大学院 工学研究科
博士前期課程 情報工学専攻
計算機アーキテクチャ研究室

杉山　智之 (413M515)

内容梗概

近年，組込みシステムでは高性能化に伴う消費電力の増加が問題となっている．組込み分野では，性能や電力だけでなく耐故障性や小面積性といった様々な要求が存在し，それらの要求を満す際に生じる消費電力の増加についても考慮する必要がある．本論文では，組込みシステムの要求の一つであるリアルタイム性を保証することにより生じる消費電力増加の問題に着目する．一般に，リアルタイム性を持つシステムでは，厳密な時間管理が求められ，最悪の条件が重なった場合でも正常に処理が完了することを保証しなければならない．しかし，多くの場合，実際の実行時間は最悪実行時間になる事がなく，要求される性能よりも過剰な性能でプログラムを実行しており，これが電力効率の低下を招いている．この問題を解決するためには，我々はハードウェアを用いて厳密な時間管理を行うアプローチが有効であると考えられる．しかし，このような新たな組込みシステムを開発するためには，容易に改造できるベースプロセッサが必要不可欠となる．また，ハードウェアの設計開発には膨大な時間がかかるため，効率的に開発のできる環境が必要となる．そこで本論文では，リアルタイム組込みシステムを開発するためのベースプロセッサを含む開発環境を構築し，その有効性を示すために Linux 及びリアルタイム OS である uC/OS-II を用いて性能評価をする．

Abstract

Recently, to achieve high performance, low energy consumption and high reliability is required in embedded systems. But the increase of energy consumption to guarantee reliability is a problem of embedded systems. To reduce overhead to guarantee real-time, we propose hardware support method for embedded processors. However, to design such a custom processor with novel approaches takes long time. Therefore, simple baseline processor which is easy to customize but has enough features to execute real-time OS is required. This paper develops SakuraProcessor as a baseline processor for various embedded systems. This paper also evaluates SakuraProcessor by using Linux and uC/OS-II.

目 次

図 目 次

表 目 次

1　はじめに

近年，汎用機器だけでなく組込み機器においても高性能なプロセッサの搭載が一般的となり，それに伴う消費電力の増加が問題となっている．また，組込みシステムには，耐故障性や小面積性といった様々な要求が存在し，それらの要求を満たす際に生じる消費電力についても考慮する必要がある．本研究では，組込みシステムの要求の一つであるリアルタイム性を保証するために生じる消費電力の問題について取り上げる．

リアルタイム性を持つシステムでは，厳密な時間管理が求められ，最悪の条件が重なった場合でも正常に処理が完了することを保証しなければならない．しかし，多くの場合，実際の実行時間は最悪実行時間になる事がなく，要求される性能よりも過剰な性能でプログラムを実行しており，電力効率の低下を招いている．

この問題を解決するためには，著者らはハードウェアによるリアルタイム OS のサポートが有効であると考えている．リアルタイム OS のハードウェア化に関する研究 [6],[7] は幅広く行われているが，その中でも著者らは細粒度なデッドライン管理による低消費電力化に着目している．

一般に，リアルタイム OS でも汎用 OS と同じようにタイムスライスに区切り，タスク毎の優先度やデッドライン制約に基づいてタスクのスケジューリングを行う．そのとき，タスクがデッドライン制約を満たせるかどうかは，キャッシュミスやバスの衝突，外部割り込みなど，最悪の条件が重なった場合でもデッドラインを満たせるようにスケジューリングする．しかし，一般的には最悪条件が重なるケースは少なく，実際には与えられたタイムスライスよりも早く処理を完了できる．そのため，将来の状態が既知であれば，性能は低いが電力効率の高い演算リソースを利用できるにも関わらず，過剰性能のプロセッサを用いて最悪条件を想定してタスクを処理している．著者らはタイムスライスを更に細かくサブタイムスライス単位に分割し，ハードウェアを用いることでサブタイムスライス単位でデッドライン制約を満たせるかどうかを判断し，できる限り電力効率の高いプロセッサでタスクを処理する手法を提案している [1].

しかし，ハードウェアの開発には膨大な時間がかかり，数億命令のプログラムを実行するためには，数時間から数十時間かかる．数億命令先で誤動作を起した場合，ハードウェアの改良を行い，再度実行を行う必要がある．このようにシミュレーションにかかる時間の増加が開発効率の低下を招いている．また，このようなシステムを開発するためには，シ

ステムを組込むためのプロセッサを一から設計する必要がある．

これを解決するためには，容易に改良できるベースプロセッサコア及び，効率よく開発できる環境が求められる．設計データが公開されており，かつ独自のハードウェアを追加することができるプロセッサとして，UltraSPARC T2 や OpenRisc などが挙げられる．しかし，構造が複雑であり，容易に独自のハードウェアを追加することが困難であるなど，組込みシステムのベースプロセッサとして利用しやすいものはない．そこで本論文では，リアルタイム OS の動作する組込みシステム向けのベースプロセッサ，SakuraProcessor の開発及び評価環境の構築を目指す．

2　リアルタイムシステム

リアルタイム性とは，制限時間（デッドライン）内での処理完了を保証することである．リアルタイム性を重要視するシステムをリアルタイムシステムといい，時間制約の厳密性によりハードリアルタイムシステム，ファームリアルタイムシステム，ソフトリアルタイムシステムの3つに分類される．リアルタイムシステムの分類は，その応用分野や研究者により異なっているため，本論文では文献 [2] の分類に基づいて述べる．

2.1　ハードリアルタイムシステム

ハードリアルタイムシステムとは，制限時間に厳格なシステムであり，主に安全制御システムに用いられる．ハードリアルタイムシステムでは，デッドラインを越えた場合，システムに致命的な影響を与え，制御不能な状態に陥る．自動車の安全システムを例にとると，衝突の可能性を検知した場合，一定時間内に危険物を回避する，あるいはブレーキを掛ける等の処理を完了しなければ，重大な事故につながる恐れがある．

2.2　ファームリアルタイムシステム

ファームリアルタイムシステムは，ネットワークアプリケーションやマルチメディアアプリケーションに用いられるもので，デッドラインを越えた際にシステムに致命的な影響は与えないが，そのタスクの価値は無くなるものである．例えば，車載のナビゲーションシステムにおいて，自動車が交差点を通過後に左折等の指示を出しても無意味である．また，動画像のデコード処理では，デッドライン（フレームレート）を越えた場合，そのフレームは表示されることがないため，処理内容，すなわち当該フレームのデコード処理は無価値なものとなる．しかし，ナビゲーションシステムの指示の遅れや動画像のフレーム落ちでシステムが制御不能な状態に陥るわけではないため，ハードリアルタイムとは区別される．

2.3　ソフトリアルタイムシステム

ソフトリアルタイムシステムは，通話やキーボードの入力データのハンドリングといったユーザインタラクションシステムに用いられるもの

で，処理完了時間が制限時間を越えた際に，システムに致命的な影響を与える事はないが，自身の価値が低下するシステムの事である．例えば，Apple 社の Siri や Google 社の音声検索システムなどで，ユーザが発声後，一定期間内に応答がなければシステムの価値は低下する．しかし，音声認識がデッドラインを越えたとしても，直ちにその価値が無くなる訳ではない．

本研究では，ハードリアルタイムシステムを対象とする．ハードリアルタイムシステムは正確な時間管理が必要であり，実行タスクの処理時間の正確さがシステムの信頼性に大きく影響する．そのため，最悪の条件が重なった場合でも正常に処理が完了することを保証しなければならない．これを実現するために，リアルタイムシステムではリアルタイム性を必要とする全てのタスクに対して，最悪の場合の処理完了時間として WCET（worst case execution time）が設定されており，リアルタイムシステム上で実行される全てのタスクが WCET で実行してもデッドラインを満すようにリアルタイム OS の内部でスジューリングを行っている．

前述の通り，この制約を満たすためには，極めて悲観的な想定の下にハードウェアを用意する必要がある．しかし，多くの場合，実際の実行時間は最悪実行時間になる事がなく，要求される性能よりも過剰な性能でプログラムを実行しており，電力効率の低下を招く．これを解決するために，ソフトマクロを用いて電力効率の高いリアルタイムシステムを目指す．

3　ソフトマクロプロセッサ

新たな組込みシステムを開発するためには，以下の要求を満す評価環境が必要不可欠である．

- 組み込み用途として適当な回路規模
- シンプルな構造
- メンテナンス性，拡張性
- 評価環境として十分な機能性

これらの要求を満すために，ソフトマクロを用いる事で解決する．ソフトマクロとは，ハードウェア記述言語（Hardware Design Language:HDL）などで記述された論理合成可能な設計資産（Intellectual Property:IP）である．一般に特定のプロセスやデバイスに依存しないため，様々な組込機器に搭載し易いという特徴がある．近年，組込機器でも高機能化が進んでおり，従来のような 8 ビットや 16 ビットではなく，32 ビット，64 ビットの高性能プロセッサが利用されはじめている．一般にプロセッサのソフトマクロは暗号化されていたり，高額なライセンス料が必要となることが多いが，設計者や研究者が無償で利用できるソフトマクロもいくつか提供されている．無償で利用できるソフトマクロの代表的なものを以下に挙げる．

3.1　既存のソフトマクロプロセッサ

3.1.1　OpenRISC

OpenRISC は組込みシステム向けに開発された完全にオープンソースの 32 ビット及び 64 ビットプロセッサである．MMU を搭載し，Linux をサポートしているが，ORBIS32/63(OpenRISC Basic Instruction Set) という独自の命令セットを採用しており，GCC をはじめとする無償の GNU ソフトウェア開発環境を利用できないという欠点がある[1]．また，Linux やアプリケーションソフトウェアも独自にサポートする必要があるため，最新の OS やアプリケーションの利用が困難という問題もある．

[1]GNU ツールチェーンを独自に移植したものはあるが，GNU で公式にサポートされていないため，最新のバージョンは利用できない．

3.1.2 FabScalar

FabScalar[8] はノースカロライナ州立大学で提案されたヘテロジニアスマルチコアプロセッサを自動生成するツールセットである．パラメータファイルを変更することで，任意のスーパスカラプロセッサコアを生成することが可能である．FabScalar は様々な ISA をサポートしており，GNU ツールチェーン等の開発環境や既存のソフトウェアをそのまま利用できるという利点がある．しかしながら，FabScalar は高性能なヘテロジニアスマルチコア用のベースプロセッサとして開発されたものであるため，多くの組込み用途ではハードウェア資源が過剰であるという問題がある．

3.1.3 UltraSPARC T2

UltraSPARC T2 は Sun Microsystems 社によって開発されたプロセッサであり，数多くのサーバに搭載される商用のハイパフォーマンス向けプロセッサである．UltraSPARC T2 は SMT(Simultaneous Multithreading) をサポートしており，各コアあたり 8 スレッドを実行できる．また，FPU を搭載しており，Solaris や Linux を実行できる．しかしながら，UltraSPARC T2 も高性能計算機用に開発されたものであり，多くの組込み用途では電力制約やハードウェアサイズ制約を満たせないという問題がある．

3.1.4 Nios II

Nios II は，Altera 社によって開発され，RTOS をサポートした組込み向けプロセッサであり，HDL コードは一般に公開されていない．性能及びサイズに応じて高速，エコノミー，標準の 3 点のコンフィギュレーションが可能である．高速版では，6 段パイプライン，MMU や外部ベクター割込みコントローラを搭載し，高機能な例外をサポートしている．ダイナミック分岐予測を持ち，高い性能を実現する．しかしながら，HDL コードが一般に公開されていないため，プロセッサ内部に新たな機構を内部に組込むことが出来ないという問題がある．

3.1.5 RISC-V

RISC-V[3] は，Asanovic らによって提案されている Open ISA で，RISC-V に基づいたプロセッサのデザインが無償で提供されている．RISC-V は，

32 ビットと 64 ビットの両方をサポートしている．また，例外処理や浮動小数点演算もサポートしており，商用のプロセッサと遜色ない仕様を有している．しかしながら，OpenRISC と同様，独自の ISA であるため，最新の GNU ツールチェーンが使えない，既存のソフトウェア資産が利用し難いという問題がある．更に，RISC-V のハードウェア設計データは Chisel 言語という独自言語で書かれているため，機能追加や設計データの変更が困難という問題がある．

3.2 既存のソフトマクロの問題点

表 3.1: プロセッサの特徴の比較

	OpenRISC	FabScalar	Ultra-SPARC T2	Nios II	RISC-V
HDL code	Verilog	System Verilog	Verilog	Nonpublic code	Chisel
Hardware cost	Small	Large	Very Large	Small	Small
Complexity	Simple	Complex	Very Complex	Simple	Simple
Support OS	Yes	No	Yes	Yes	Yes
Modifiablity	Good	Fair	Poor	Impossible	Poor

表 3.1 に候補となるソフトマクロの特徴を示す．OpenRISC はシンプルな構造であるが，Verilog で記述されているためメンテナンス性が低い．また，独自の命令セットを使用しているため，OS を移植することが困難である．FabScalar は HDL コードが公開されており，SystemVerilog で記述されているためメンテナンス性が高い．しかしながら，アーキテクチャが複雑でハードウェア規模が大きい．また，現状 OS はサポートされていない．UltraSPARC T2 は HDL が公開されたプロセッサであるが，ハイパフォーマンス向けのプロセッサであるため，ハードウェア規模が大きい．また，構造が複雑であり，容易に独自のハードウェアを追加することが困難である．Nios II は，ハードウェアコストが小さく，HDL コード

が一般に公開されていないため，新たな機構を内部に組込むことが出来ない．RISC-V はシンプルな構造であるが，独自言語で記述されているため，拡張性が低い．このように，候補となるソフトマクロの中に要求を全て満すものがない．そこで組込みシステムのベースとなるプロセッサ，SakuraProcessor を開発する．

4 SakuraProcessorの提案

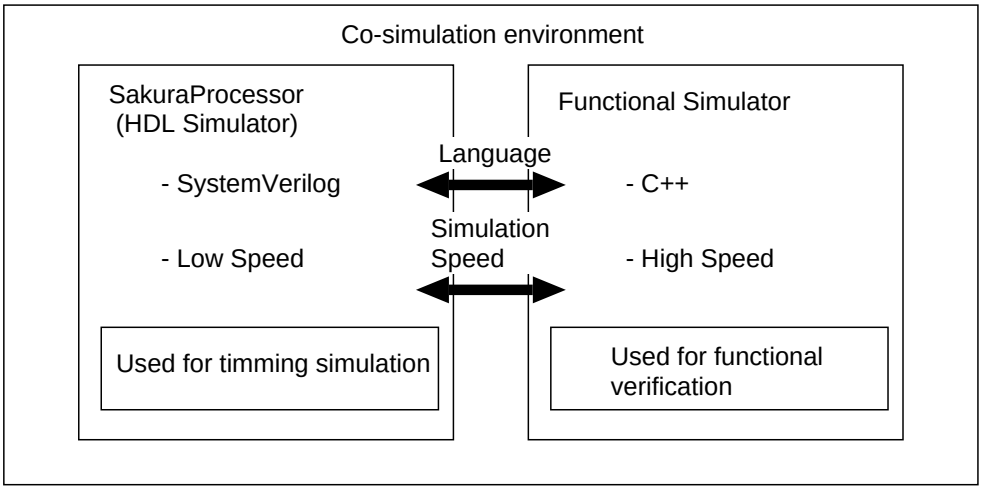

図 4.1: Structure of co-simulation framework.

本研究で提案する開発環境は図 4.1 に示す様に SystemVerilog で記述された SakuraProcessor（HDL シミュレータ）と C ＋＋で記述された機能シミュレータで構成され，DPI-C（Direct Programming Interface-C）を用いて相互に通信を行う．DPI-C は SystemVerilog によって提供される SystemVerilog 及び C 言語間のインターフェースを可能にするためのものである．HDL シミュレータはサイクルごとに DPI-C を用いて機能シミュレータと通信し，動作検証を行う．本節では SakuraProcessor の特徴や構成について述べる．

4.1 SakuraProcessorの特徴

SakuraProcessor は以下の 5 つの特徴を持つ．

4.2 パイプライン構造
シンプルなアーキテクチャで十分なパフォーマンスを達成するために 5 段のシングルパイプラインプロセッサである．

4.3 拡張性
高い拡張性，メンテナンス性を満すために，SystemVerilog を用いて設計する．

4.4 階層設計
パイプラインステージや演算器を階層モジュール化する．

4.5 命令セット
組み込みシステムで広く使用されている MIPS32R2 を採用する．

4.6 開発容易性
開発及び検証ツールとして，C++で記述された高速シミュレータを搭載する．

以降，各々の特徴について詳細に説明する．

4.2 パイプライン構造

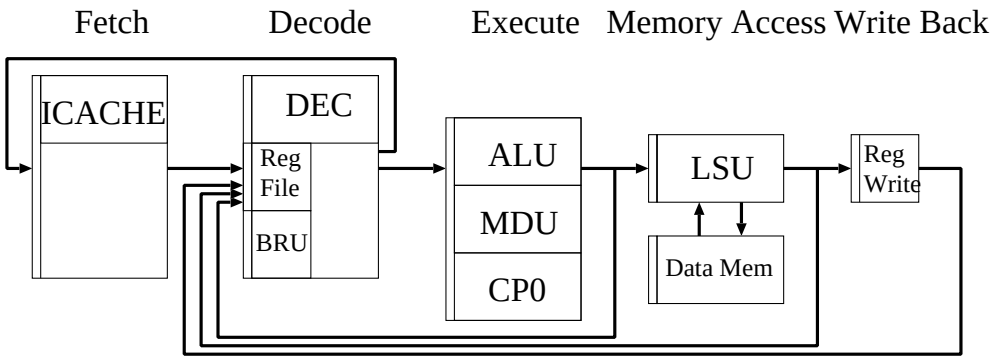

図 4.2: Block design of SakuraProcessor.

図 4.2 に示すように SakuraProcessor はフェッチ，デコード，実行，メモリアクセス，ライトバックの 5 段パイプラインである．近年，組込み用途のプロセッサでもスーパスカラ構成を採用しているものもあるが，リアルタイムシステムを必要とするシステムでハイエンドプロセッサを必要とするケースは少なく，またシンプルな構造の方が実行時間の見積りがし易いという利点があるため，本研究ではシングルパイプライン構造を採用した．SakuraProceossor はマルチコア構成もサポートしているため，より高い性能が必要な場合は複数コアを利用する，あるいは FabScalar 等のハイエンドコアと併用することで解決できる．

4.3 拡張性

SakuraProcessor は高い拡張性，メンテナンス性を満すために，SystemVerilog を用いて設計している．従来の Verilog HDL では，モジュール間の配線を個別に接続する必要があったため，ブロック図レベルでシンプルな構成をとっていても，実際の HDL 設計ではモジュール間の配線部分が複雑になり，機能拡張やデバッグ等が困難であった．SakuraProcessor では，モジュール間の信号線を機能毎にまとめ，SystemVerilog の構造体 (struct) を用いることでモジュール間の接続の認識性を向上させた．また，機能毎に配線を分類し，定義することで，機能拡張やデバッグが容易に行えるようにした．また，トップレベルの接続も Interface を用いることで機能毎に集約しており，RTL コードの可読性を高めている．

4.4 階層設計

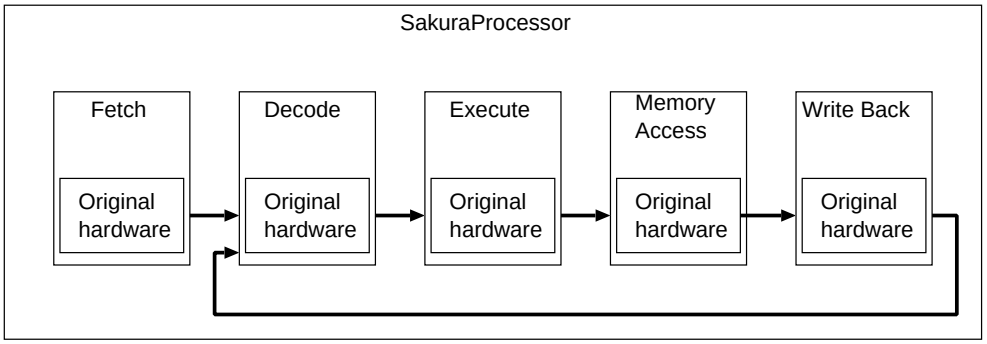

図 4.3: Top module of SakuraProcessor.

プロセッサコアの改良を容易にするために階層構造とモジュール化を積極的に利用した．図 4.3 に示す様に，トップモジュールは命令デコードやメモリアクセスといったステージモジュールのみで構成される．モジュール間の接続は SystemVerilog で提供される構造体を用いて実装されている．そのためトップモジュールはシンプルな構造になっている．ALU や TLB といった実際の回路はステージモジュールのサブモジュールとして配置される．

4.5 命令セット

SakuraProcessorでは，組み込みシステムで広く使用されているMIPS32R2を採用している．そのため，既存のOSやソフトウェア資産をそのまま利用できるという利点がある．組込み分野では，ARM社のThumb命令セットのようなコンパクトな命令セットが広く用いられており，MIPS社でもMIPS16が提案されている．現在のSakuraProcessorではMIPS16はサポートしていないが，拡張は容易であり，今後対応予定である．

4.6 開発容易性

一般に，新規にハードウェアを追加したり，既存のハードウェアを改造した場合，デバッグや動作検証が非常に困難である．そこで，HDL記述に加え，C++で記述された機能シミュレータを実装した[4, 5]．この機能シミュレータは，HDLシミュレータと協調動作させることができる．機能シミュレータとHDLシミュレータで構成されるコシミュレーション環境については5章で詳細に説明する．

5　コシミュレーション環境

機能シミュレータはHDLシミュレータと協調動作させることができ，以下のような利点がある．

5.1 ラピッド・プロトタイピング設計

5.2 HDLの論理検証

5.3 シミュレーション時間の短縮

以降，各々の特徴について詳細に説明する．

5.1　ラピッド・プロトタイピング設計

一般に，HDLの記述性は，ソフトウェア記述言語よりも低く，コードの行数が長くなったり，複雑になったりする傾向がある．そのため，新しいハードウェアを追加する場合に，HDLで記述し，性能評価を行うのは非常に効率が悪い．SakuraProcessorでは，機能シミュレータが用意されているため，まず新しいハードウェアの動作モデルをC++等で実装し，機能シミュレータ上で実行することで，早期に性能予測や機能検証を行うことができる．

5.2　HDLの論理検証

図5.4に示すように，本研究で提案するコシミュレーション環境はSakuraProcessor及び機能シミュレータで構成される．SakuraProcessorはサイクル単位で振舞うHDLシミュレータまたはLSIの設計データとして用いられる．しかし，SakuraProcessorは実際の回路として振舞うためシミュレーション時間が膨大になり，開発にかかる時間の増加を招く．また，新規のハードウェア設計では，ソフトウェアと比較して，動作検証やデバッグが困難という問題がある．これを解決するために，論理検証を行うことのできるC言語で記述された機能シミュレータを用いる．

この機能シミュレータは，DPI-CをSystemVerilogシミュレータと通信することで，サイクルごとに動作の一致比較を行っている．そのため，HDL記述と機能シミュレータで動作が異なった場合には，すぐに問題の発見，デバッグを行える環境になっている．しかし，機能シミュ

レータはサイクル単位でシミュレートを行うことができないため，これらのシミュレータは振舞いが異なる．SakuraProcessor はステージごとに１命令実行するが，機能シミュレータは１度に１命令実行する．そのため，SakuraProcessor が１命令実行した時に，機能シミュレータは１命令実行し，動作の一致比較を行っている．

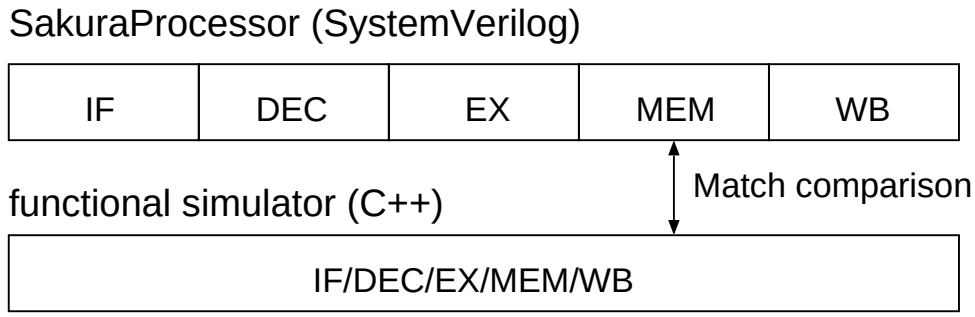

図 5.4: Verification environment.

5.3 シミュレーション時間の短縮

一般に HDL シミュレータは非常に動作が遅いため，大規模なプログラムやシステム全体のシミュレーションには膨大な時間がかかる．そこで，SakuraProcessor では，図 5.5 のように機能シミュレータを用いた高速スキップを実現している．プログラムの実行開始時点では，機能シミュレータのみを用いて高速に実行する．そして，HDL の検証を行いたい部分，あるいは性能評価を行いたい部分まで達した時点で，HDL シミュレータを起動し，機能シミュレータの内部情報をコピーする．その後，機能シミュレータと HDL シミュレータを同時に動作させることで，サイクル単位で一致比較を行いつつ，HDL シミュレーションを行うことができる．これにより，大規模なアプリケーションの一部を高速にシミュレーションすることができる．

高速スキップの具体的な方法はロード及びストア命令を用いることで実現する．その方法を図 5.6，図 5.7 を用いて説明する．図 5.6 に，機能シミュレータと相互に通信を行うための方法を示す．プログラムの実行開始時点まで，機能シミュレータのみを用いて高速に実行した後，機能シミュレータと相互に通信を行うためにメモリ空間を 7fd00104 番地から割り当

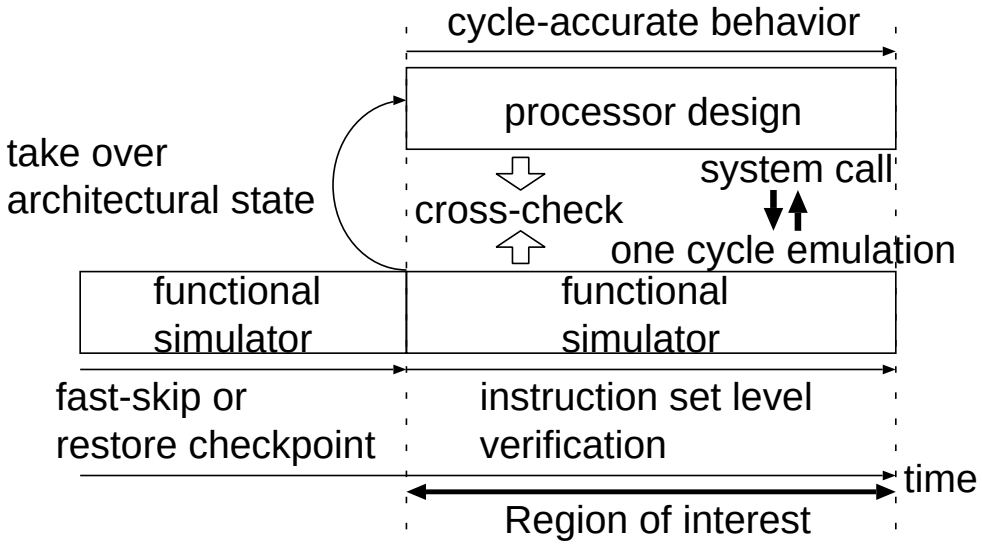

図 5.5: Co simulation environment.

る．HDL シミュレータは図 5.7 に示すリセットルーティンを用いて，内

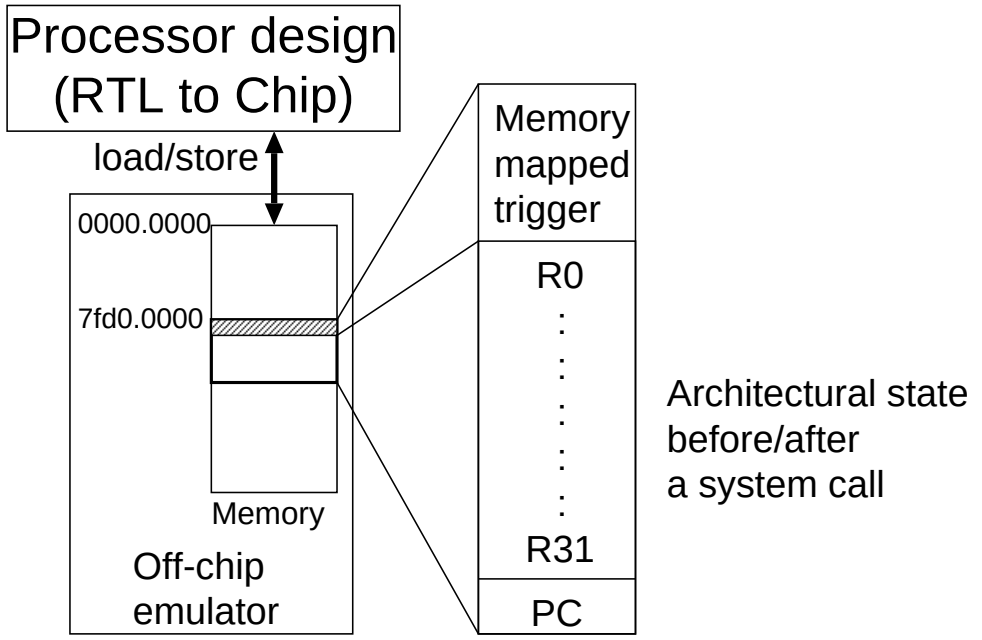

図 5.6: Off-chip system call emulation mechanism.

部情報をコピーすることで，プログラムを再開する．HDL シミュレータは起動した直後に，プログラムカウンタが bfc00000 に初期化される．これは一般的なリセットルーティンのスタートアドレスである．このルーティンでは，特定のメモリ空間に書き込まれたレジスタファイルの値とプログラムカウンタをロードする．このプログラムカウンタは，HDL の検証を行いたい部分，あるいは性能評価を行いたい部分の開始位置のアドレスを示す．

```
bfc00000 <__reset_handler>:
        lui k0,  0x7fd0
        lw  $1,  0x104(k0)
                 :
        lw  $31, 0x17c(k0)
        /* load program counter */
        lw  k1,  0x278(k0)
        jr  k1
```

図 5.7: Reset routine

このメカニズムは設計データの中に特別な回路を実装する必要がなく，純粋なデザインを維持する事ができる．また，このフレームワークでは HDL シミュレータ，機能シミュレータ間の相互通信にロード及びストア命令を使用しているため，RTL から LSI 設計まで一貫して同じメカニズムを利用できる．

6　OSの移植

SakuraProcessor は MIPS32R2 の命令セットに基づいているため，既存の OS やアプリケーションを比較的容易に移植できる．しかしながら，SakuraProcessor はプロセッサコアのみであるため，実際に組込みシステムで利用する場合には，周辺機器を追加し，必要なドライバ等を追加する必要がある．そのため，SakuraProcessor には，OS の動作確認に必要となる UART を実装している．

6.1　Linux

Linux は Linus が開発したオープンソースの OS であり，スーパーコンピュータからデスクトップパソコン，組込み機器まで幅広く利用されている．Linux はリアルタイム性は保証していないが，高精度のタイマーを保持していることや，ドライバなどの拡張が容易なことから，厳密なリアルタイム処理を求められない分野でも利用されている．

Linux は元々MIPS32R2 の命令セットをサポートしているため，カーネルの主要な部分の変更は不要である．しかし，SakuraProcessor はメモリコントローラや UART 等の周辺機器を実装した CPU ボードを持っていないため，そのままでは Linux を動作させることはできない．そこで，Linux カーネルに前述の UART のドライバのみを追加した仮想ボードを定義した．また，Linux を動作させる場合，タイマー割り込みが必要になるが，MIPS32R2 はタイマーを内蔵しているため，プロセッサ内のタイマーを利用するように設定を行った．

Linux の Ver.3.3.8 をベースに，UART のみを追加した SakuraProcessor 用に移植を行った．追加したコードは 786 行で，変更した部分は arch/mips の下のみである．

6.2　uC/OS-II

uC/OS-II は Micrium 社の開発したリアルタイム OS であり，以下の特徴を持つ．すなわち，

- ポータビリティが高い，
- ROM 化できる，

- プリエンティブである，
- マルチタスクをサポートしている，

ことである．uC/OS-II はポータビリティは高いが，すべて C 言語で記述されており，割り込み制御やコプロセッサ，周辺機器へのアクセス等のアーキテクチャに依存する部分は設計者が用意する必要がある．本研究では，Atlas 社が MIPS32 用に移植したコードをベースに SakuraProcessor に再移植した．変更した部分は mips_ucos/atlas/inc/atlas.h 及び mips_ucos/common/src/ の中から os_cpu_c.c，os_cpu_a.S である．atlas.h は，シリアル通信の部分で変更を加えた．os_cpu_c.c，os_cpu_a.S は，例外処理の部分に変更を加えた．

表 7.2: EDA environment for physical design

Phase	EDA tool
Functional verification	Synopsys VCS, Ver. 2009.06
Logical synthesis	Synopsys Design Compiler, Ver. 2010.03-SP5
Place & route	Synopsys IC Compiler 2010.12

表 7.3: Design environment for FPGA

Phase	EDA tool
Functional verification	Synopsys VCS, Ver. 2009.06
Logical synthesis	Synopsys SynplifyPro Ver.2013.09
Place & route	Altera QuartusII Ver.13.0SP1

7 評価

SakuraProcessor の性能検証を行うために，CMOS プロセス，及び Altera 社製 FPGA に SakuraProcessor を実装した．表 7.2 に論理検証，論理合成，及び配置配線に使用したツール，図 7.8 に SakuraProcessor を合成した際のチップ写真を示す．動作周波数，回路規模を算出した結果，動作周波数は 200MHz，回路規模は $1,640\mu m \times 1,200\mu m$ となり，組込み機器として十分な性能であり，回路規模についても，妥当な値であることを確認した．

図 7.3 に SakuraProcessor を Altera 社製 FPGA に実装した際に使用したツールを示す．SakuraProcessor を Stratix II にマッピングした結果，Combinational ALUTs が 19,297，Dedicated logic registers が 12,040，Embedded block memory が 331,516bit，使用した DSP ブロックは 24 個であった．

SakuraProcessor 上で Linux 及び uC/OS-II を実行した際に，コンソールに出力した結果を図 7.9，図 7.10 に示す．これにより SakuraProcessor 上で Linux，uC/OS-II が正常に動作していることを確認した．

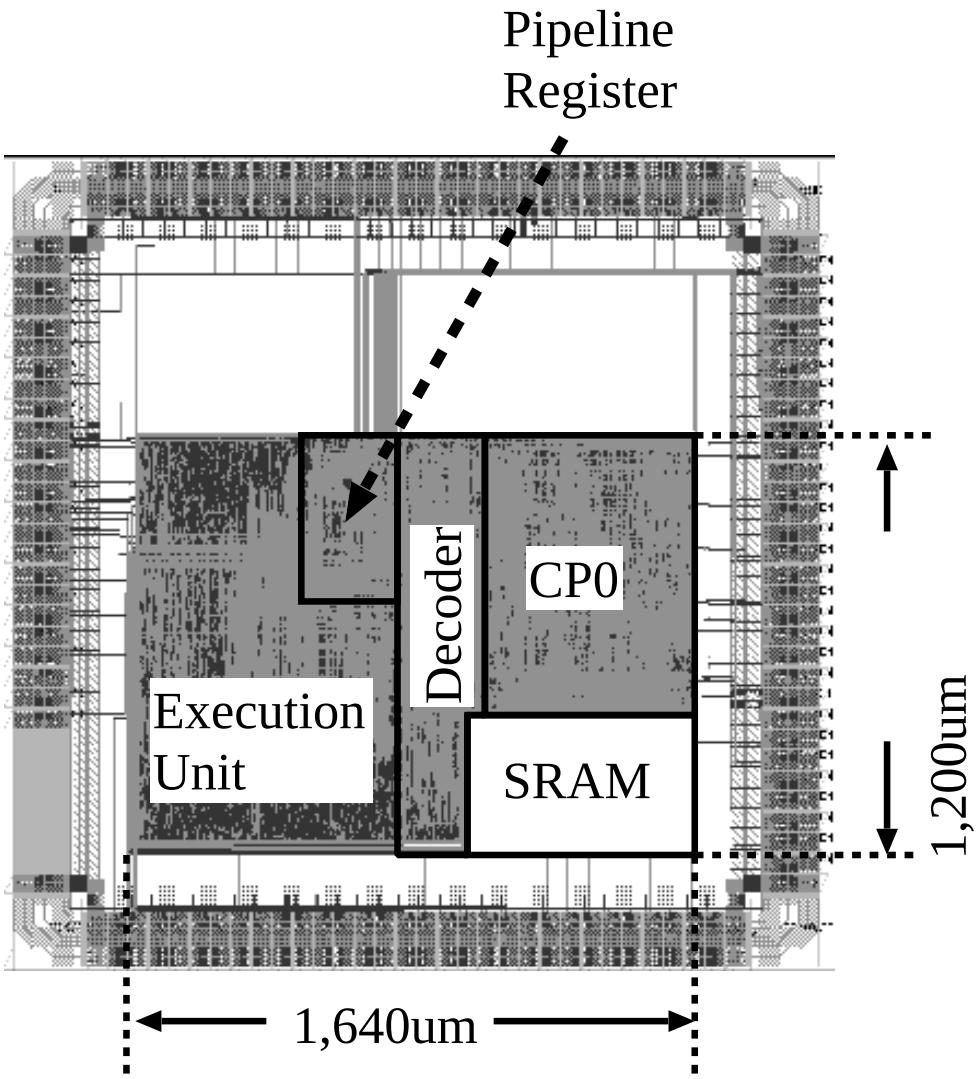
Pipeline
Register
Decoder
CP0
Execution
Unit
SRAM
1,200um
1,640um

図 7.8: チップ写真

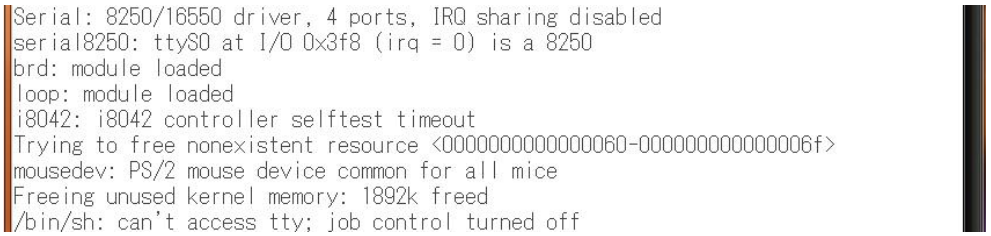

図 7.9: Linux 動作結果

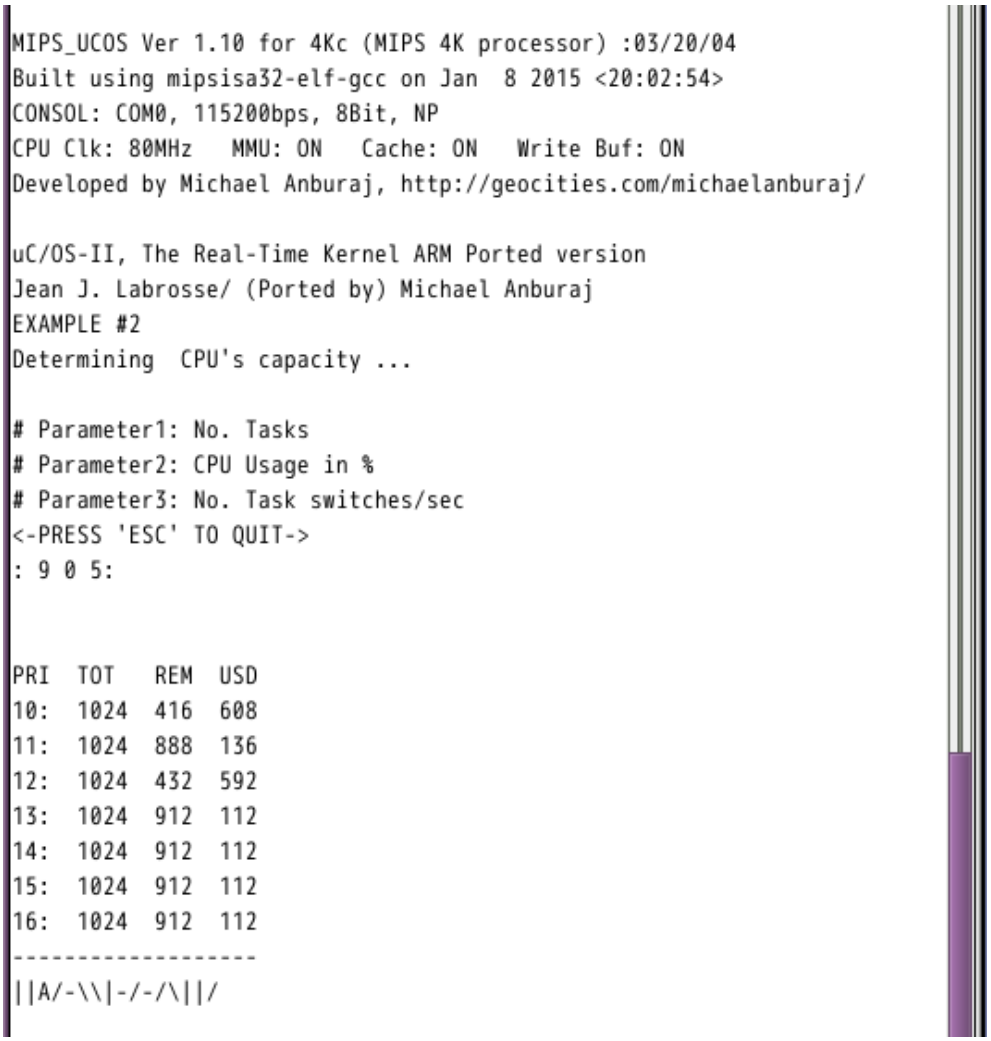

図 7.10: uC/OS-II 動作結果

8 SakuraProcessorへのスケジューラ実装方法の提案

本研究の最終目標は，RTOSで行われているスケジューリング機構の一部をハードウェアで実装し，電力効率の高いリアルタイムシステムを実現する事である．これを実現するためにリアルタイムタスクの実行処理時間を細分化し，図8.11に示すチェックポイントを各タスクに設定する．チェックポイントごとにデッドライン制約を満たせるかどうかを判断し，デッドラインに余裕がある場合はLow-end core，余裕がない場合はHigh-end coreを用いる．本研究では，Low-end coreとしてSakuraProcessorを用いる．

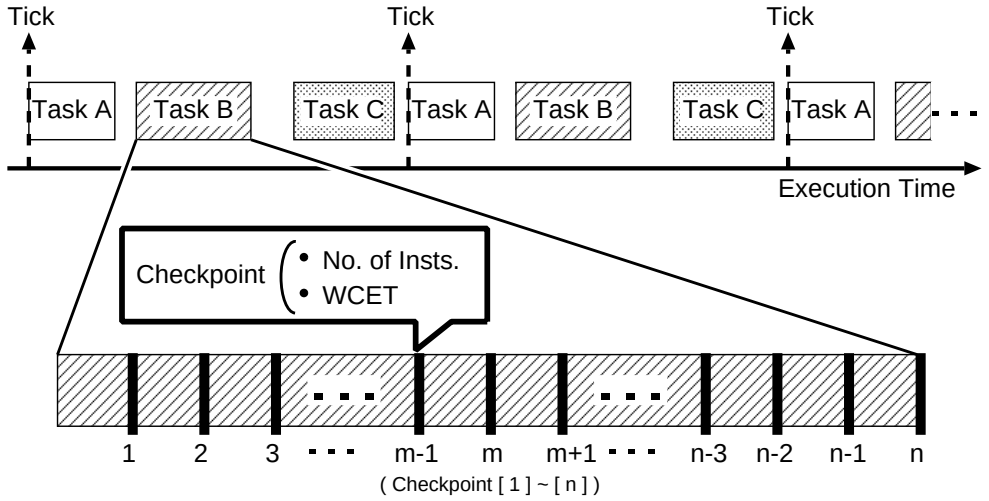

図 8.11: Check Point

提案するSakuraProcessorへのスケジューラ実装方法の概要を図8.12に示す．Dynamic Scheduling Unitは，動的スケジューリングおよび優先度の高いタスクのプロセッサへの割り当てを行う．Deadline Monitoring Unitはタスクの実行状態を監視するために用いられる．Context Switch Support Unitはタスクの切り替えを行うユニットである．ユニット間の通信にはScheduler Busを介して行う．これらのユニットについて以下で詳細に説明する．

まず，図8.13に示すScheduling Unitについて述べる．Scheduling Unitは，主にレジスタ，スケジューラ，FIFOで構成され，レジスタにはDead-

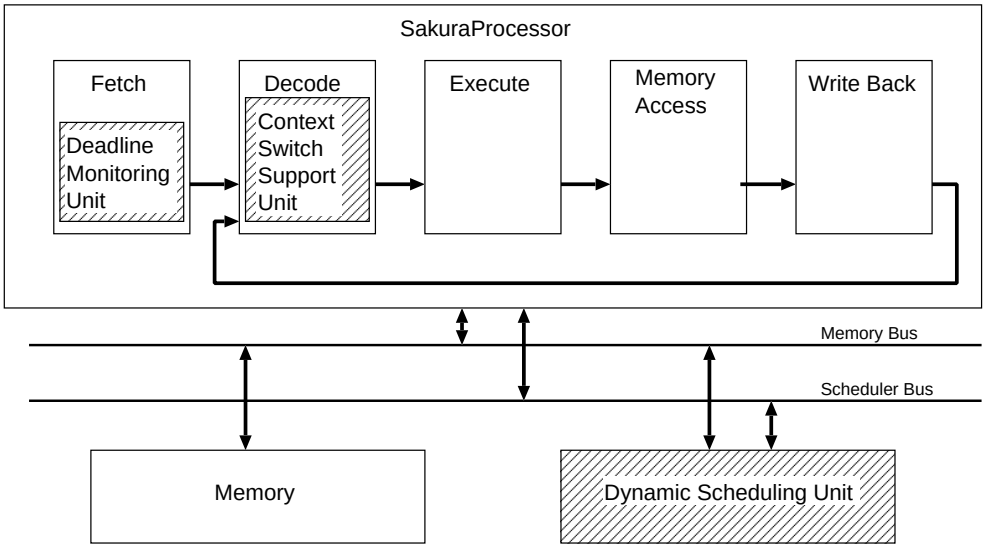

図 8.12: ハードウェアスケジューラ

Line，WCET，Check Point といったタスクの情報を保持し，プロセッサとメモリマップド IO で接続される．これらの機構はプロセッサの一部として組込むため，専用命令を用いてアクセスすることも可能であるが，アセンブラやコンパイラを改良する必要がある．そこで既存の SDK を用いることが可能なメモリマップド IO を採用した．Dyanmic Scheduling Unit では，これらのタスク情報と Monitoring Unit から受け取った実行中のタスク情報を元に各タスクのデッドラインまでの時間を計算する．Priority Controller でタスクを優先度順に並べ変え，FIFO に格納する．

次に，図 8.14 に示す Context Switch Support Unit について述べる．Context Switch Support Unit はタスク切り替えのオーバーヘッドの問題を解決する機構である．次に実行すべきタスクのコンテキスト情報を格納する Context Queue，コンテキストスイッチの制御を行うための Task Controller，プロセッサ間でのコンテキストスイッチの制御を行うための Core Switching Controller から成る．

次に，図 8.15 に示す Deadline Monitoring Unit について述べる．Deadline Monitoring Unit はタスクの実行状態を常に監視するための機構である．Deadline Controller では，SakuraProcessor 内部にある Program-Counter の値を受け取り，実行命令数のカウントを行う．

Context Switch Support Unit，Deadline Monitoring Unit はレジスタ

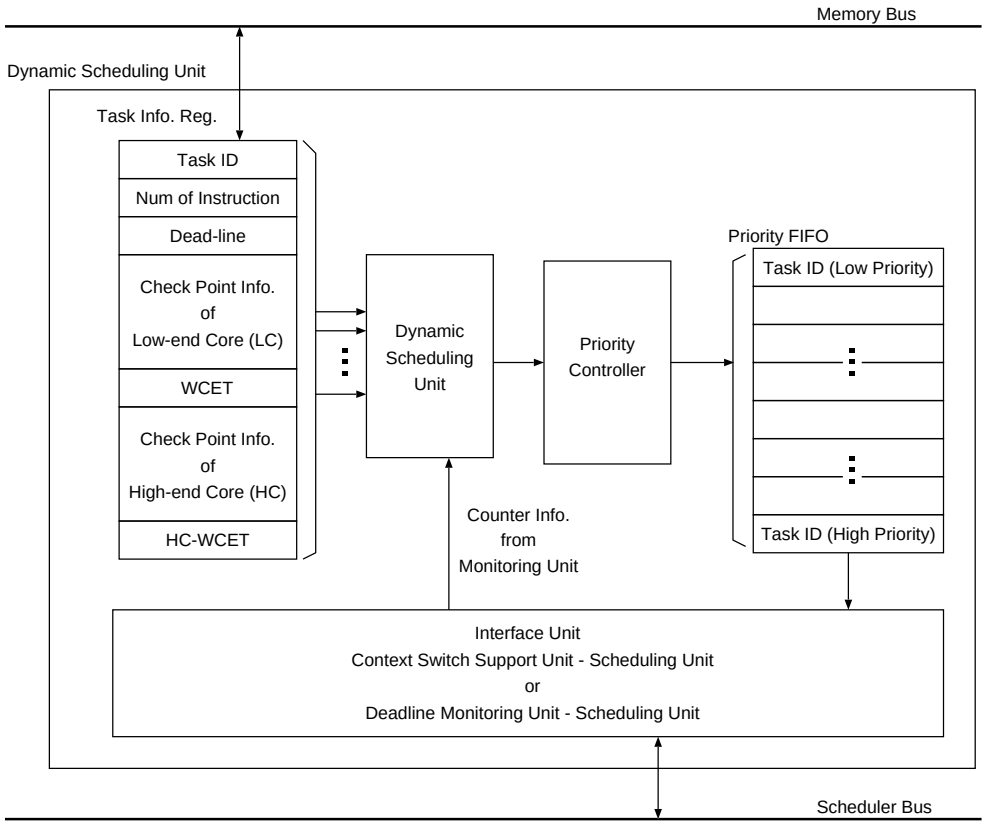

図 8.13: Scheduling Unit

ファイル，プログラムカウンタの値を直接操作する必要がある．SakuraProcessor は階層構造になっており，これらのユニットを適切な位置に配置する事で容易に改良することが可能である．そのため，Context Switch Support Unit はレジスタファイルのあるデコードステージに配置され，Deadline Monitoring Unit はプログラムカウンタのあるフェッチステージに配置される．現在，SakuraProcessor に前述のハードウェア機構を追加実装している．

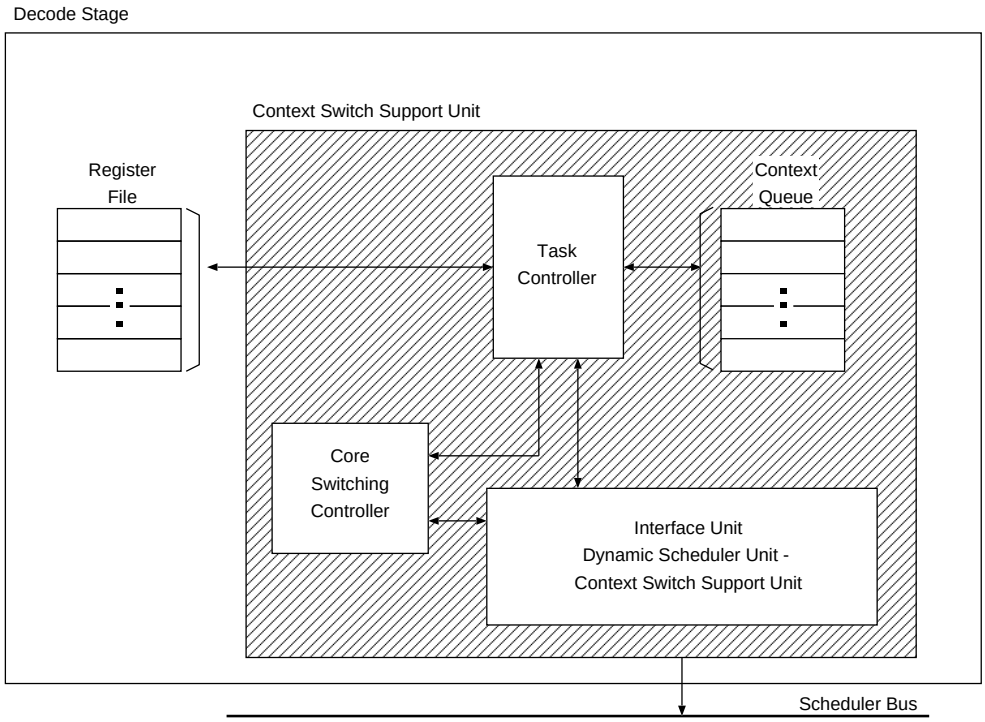

図 8.14: Core Switch Unit

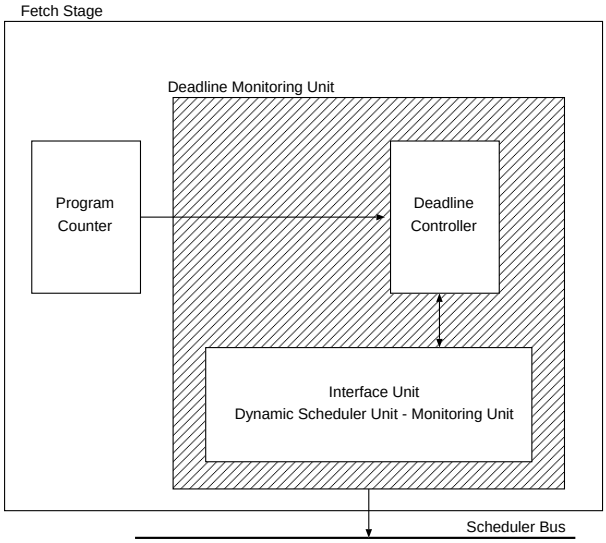

図 8.15: Deadline Monitoring Unit

9　まとめ

本論文では，既存のソフトマクロプロセッサの中から，組込みシステムの開発に最適なプロセッサの選定を行った．しかし，十分な性能および機能性を持つ開発環境が存在しないことからSakuraProcessor及び機能シミュレータを含むコシミュレーション環境の構築を行った．

SakuraProcessorはシンプルな5段パイプラインプロセッサとし，階層設計を用いる事で，十分な性能を持ち容易に改良が可能なプロセッサの実現を目指した．評価結果より動作周波数，回路規模を算出し，組込みシステムとして妥当な回路規模，動作周波数である事を確認した．また，Linux及びuCOS-IIを用いて正常に動作する事を確認した．機能シミュレータは高速スキップを用いることで，開発効率の向上を実現した．これらの結果より，開発環境として十分な機能をもつシミュレータの開発を達成した．

謝辞

本研究の機会を与えて頂いた近藤利夫教授，並びにご指導，ご助言頂いた佐々木敬泰助教，深沢研究員に深く感謝いたします．また，様々な局面でご助力頂いた計算機アーキテクチャ研究室の皆様にも心より感謝いたします．

参考文献

[1] 瀬戸 勇介, 中林 智之, 佐々木 敬泰, 近藤 利夫: ハードウェアスケジューラを用いたリアルタイムマルチコアプロセッサの省電力化, 第15回 組込みシステム技術に関するサマーワークショップ予行集, pp.1-6 (2013)．

[2] Giorgio C. Buttazzo, “Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications”, Third Edition, Springer (2011).

[3] Yunsup Lee, Andrew Waterman, Rimas Avizienis, Henry Cook, Chen Sun, Vladimir Stojanovc, Krste Asanovic, “A 45nm 1.3GHz 16.7 Double-Precision GFLOPS/W RISC-V Processor with Vector Accelerators”, Proceedings of the European Solid-State Circuits Conference (2014).

[4] Tomoyuki SUGIYAMA, Takahiro SASAKI, Toshio KONDO, “Development of C++/RTL co-simulation environment for accelerating VLSI design of an embedded processor”, Proceedings of the International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC2013), pp.281-284 (2013).

[5] Tomoyuki Nakabayashi, Tomoyuki Sugiyama, Takahiro Sasaki, Eric Rotenberg, Toshio Kondo: “Co-simulation framework for streamlining microprocessor development on standard ASIC design flow”, Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC2013), pp. 400-405 (2014).

[6] 荒木 英夫，久津輪 敏郎，原嶋 勝美: FPGA による組込み制御を目的としたプロセッサシステムの実装と評価, 電子情報通信学会論文誌, Vol.J-86-C, No. 8, pp.299-807 (2003)．

[7] 丸山 修孝, 一場 利幸, 本田 晋也, 高田 広章: マルチコア対応RTOSのハードウェア化による性能向上, 電子情報通信学会論文誌, Vol.J-96-D, No. 10, pp.2150-2162 (2013).

[8] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "FabScalar: Composing synthesizable RTL designs of arbitrary cores within a canonical superscalar template", Proceedings of the 38th IEEE/ACM International Symposium on Computer Architecture (ISCA-38), pp. 11-22 (2011).