

プロセッサ設計演習

ウン クアン イー

2021.06.22

1 はじめに

現代のマイクロプロセッサの動作原理を理解することを目的とし、基礎である命令パイプライン構造を持つプロセッサを設計・実装した。プロセッサの設計・実装は、Verilog-HDL というハードウェア記述言語を用いて行った。更に、プロセッサの最大遅延時間と面積を削減するために、分岐予測を導入して、パイプラインの各ステージの役割分担を見直した。分岐予測の導入によって、テストプログラムの実行クロックサイクル数が (数字を入れる) 減少した。パイプラインの各ステージの役割分担の見直しによって、ボトルネックとなるステージの処理が他のステージに行われるようにすることで、最大遅延時間が (数字を入れる) 減少した。

本稿では、第 2 章でプロセッサの仕様について述べる。第 3 章で、プロセッサの機能検証の方法について説明し、第 4 章で、プロセッサの性能評価と論理合成の結果を示す。第 5 章で、分岐予測とパイプラインステージの役割分担の見直しについて述べる。最後に、第 6 章でまとめを行う。

2 プロセッサの仕様

2.1 外部インタフェース

図 1 にプロセッサの外部インタフェースを示す。信号線名の後ろに # が記述されている信号線は負論理であり、# が記述されていない信号線は正論理であることを示している。

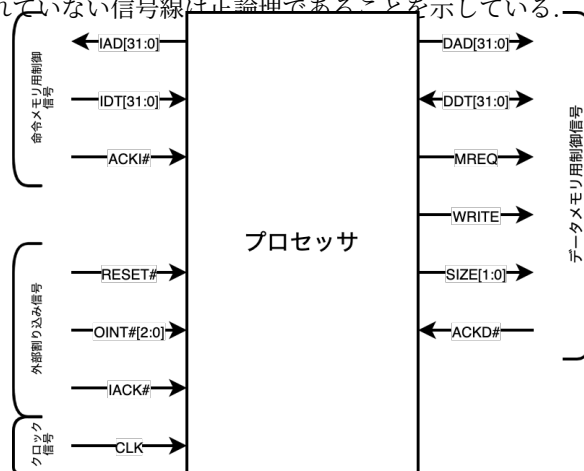


図 1 プロセッサの外部インタフェース

図 1 にあるそれぞれの信号線の説明は以下の通りである。

- IAD (Instruction ADDRESS Bus)
命令メモリへの 32 [bit] のアクセスアドレスバス。
- IDT (Instruction DaTa Bus)
命令メモリからの 32 [bit] のデータバス。
- ACKI# (ACKnowledge from Instruction memory)
命令メモリへのアクセスに対するアクノリッジ信号。命令メモリにアクセスして、この信号がインアクティブであれば、読み出し・書き込みが完了していないことを意味する。
- RESET#
リセット信号。
- OINT#
外部からの割り込みを示す信号。外部からの割り込みがあった時に、対応するビットがアクティブになる。
- IACK# (Interrupt ACKnowledge)
外部の割り込みを処理している時にアクティブになる信号。
- CLK (CLOCK)
クロック信号
- DAD (Data ADDRESS bus)
データメモリへの 32 [bit] のアクセスアドレスバス。
- DDT (Data DaTa bu)
データメモリからの 32 [bit] のデータバス。
- MREQ
データメモリに対するアクセス (読み出し・書き込み) をリクエストするための信号。データメモリへアクセスする前にリクエスト信号をアクティブにする必要がある。
- WRITE
データメモリへの書き込みをリクエストする信号。データメモリにデータを書き込む時にこの信号をアクティブにする必要がある。
- SIZE
データメモリへのアクセスサイズを示す。
 - word アクセス: SIZE = 00
 - halfword アクセス: SIZE = 01
 - byte アクセス: SIZE = 10
- ACKD# (ACKnowledge from Data memory)
データメモリへのアクセスに対するアクノリッジ信号。データメモリにアクセスして、この信号がインアクティブであれば、読み出し・書き込みが完了していないことを意味する。

命令メモリとデータメモリはシミュレーション環境で用意されているため、プロセッサの中に実装しない。ただし、メモリはリトルエンディアン方式を採用する。メモリのアドレスは 1 [byte] ごとに振り分けられ、1 つのアドレスに 32 [bit] のデータを保持することができる。

また、シミュレーション環境において、外部からの割り込みが発生しないため、OINT 入力信号の処理と、

IACK 出力信号への出力を行わない。

2.2 命令セット

設計したプロセッサがサポートしている命令の一覧は表 1 に示している。この命令セットは、RISC-V 32I 命令セットの一部である。RISC-V 32I にあり、このプロセッサがサポートしていない命令は、`fence`, `fence.i`, `sfence.vma`, `ebreak`, `uret`, `sret`, `wfi` である。機能検証と性能評価のプログラムでは、フェンスを使うことがないので、`fence`, `fence.i`, `sfence.vma` を実装しない。また、今回のプロセッサは、デバッグ、ユーザモード、スーパーバイザモード、外部割り込みをサポートしていないため、`ebreak`, `uret`, `sret`, `wfi` を実装しない。

命令	内容	形式	命令	内容	形式
lui	load upper immediate	U	add	add	R
auipc	add upper immediate to pc	U	sub	sub	R
jal	jump and link	J	sll	shift left logical	R
jalr	jump and link register	J	slt	set less than	R
beq	branch equal	B	sltu	set less than unsigned	R
bne	branch not equal	B	xor	exclusive or	R
blt	branch less than	B	srl	shift right logical	R
bge	branch greater than or equal	B	sra	shift right arithmetic	R
bltu	branch less than unsigned	B	or	or	R
bgeu	branch greater than or equal unsigned	B	and	and	R
lb	load byte	I	ecall	environment call	I
lh	load halfword	I	csrrw	csr read and write	I
lw	load word	I	csrrs	csr read and set	I
lbu	load byte unsigned	I	csrrc	csr read and clear	I
lhu	load halfword unsigned	I	csrrwi	csr read and write immediate	I
sb	store byte	S	csrrsi	csr read and set immediate	I
sh	store halfword	S	csrrci	csr read and clear immediate	I
sw	store word	S	mret	machine-mode exception return	R
addi	add immediate	I			
slti	set less than immediate	I			
sltiu	set less than immediate unsigned	I			
xori	exclusive or immediate	I			
ori	or immediate	I			
andi	and immediate	I			
slli	shift left logical immediate	I			
srli	shift right logical immediate	I			
srai	shift right arithmetic immediate	I			

表 1 命令セット

2.3 例外・割り込み処理

このプロセッサが対応している例外・割り込み処理を優先順位の高い順に以下に示す。

1. リセット
2. 不正命令
3. 命令アクセス・ミスアライメント
4. ECALL 命令

2.4 パイプライン処理

今回設計したプロセッサは、1つの命令を5つのパイプラインステージに分けて、実行される。それぞれのステージの名前と役割は以下の通りである。

1. IF ステージ
次に実行する命令を命令メモリから読み出す。
2. ID ステージ
命令を解釈して、EX ステージで行われる演算に必要な入力を用意する。
3. EX ステージ
命令で必要な演算を行う。
4. MEM ステージ
データメモリへのアクセス (読み出し・書き込み) を行う。
5. WB ステージ
汎用レジスタへデータを書き込む。

2.5 データハザードとその解決法

パイプライン処理では、異なるステージにおいて、異なる命令が実行されている。2つの命令の間にデータ依存性が存在する時に、先に実行される命令が汎用レジスタを更新する前に、後で実行される命令が同じレジスタの古い値を読み出してしまうことがある。これによって、正確な演算結果を得ることができない。この現象を、RAW (Read After Write) ハザードという。以下、RAW ハザードが発生する場合を記述する。

1. 命令 m はレジスタ x_n を更新し、命令 $m+1$ はレジスタ x_n の値を用いた演算を行う場合。
2. 命令 m はレジスタ x_n を更新し、命令 $m+2$ はレジスタ x_n の値を用いた演算を行う場合。

RAW ハザードを解決するために、データフォワーディングとパイプラインストールの2つの方法がある。

2.5.1 データフォワーディング

データフォワーディングとは、EX (または、MEM) ステージにある命令 m の演算結果を ID (または、EX) ステージにある命令 $m+1$ (または、命令 $m+2$) に渡し、命令 $m+1$ (または、命令 $m+2$) の EX (または、MEM) ステージで使用方法である。この時に、命令 $m+1$ (または、命令 $m+2$) はレジスタ x_n の最新値

を用いて演算を行うため、正確な結果が得られる。

データフォワーディングによって、上記の RAW ハザードが発生する場合の中で、命令 m がロード命令以外の場合の対処ができる。データフォワーディングで解決できなかった場合は、パイプラインストールを用いる。

2.5.2 パイプラインストール

パイプラインストールとは、パイプラインの各ステージにある命令を次のステージに進まないようにする方法である。データフォワーディングに必要なデータが用意できるまでに、それ以降の命令を待たせる。

パイプラインストールによって、データフォワーディングが解決できる状況が作られるため、RAW ハザードが解決できる。

RAW ハザードが発生する状況とその解決法を表 2 にまとめる。

レジスタ x_n を更新する命令	レジスタ x_n を用いる命令	解決法
ロード命令以外	ストア命令	データフォワーディング (EX/MEM \rightarrow ID)
ロード命令以外	ストア命令以外	データフォワーディング (EX/MEM \rightarrow ID)
ロード命令	ストア命令	データフォワーディング (MEM \rightarrow EX)
ロード命令	ストア命令以外	パイプラインストール

表 2 RAW ハザードに関わる命令とその解決法

2.6 プロセッサの名前

3 プロセッサの機能検証

今回設計したプロセッサの機能検証を以下のプログラムを用いて行った。

- アセンブリプログラム
 - load: ロード命令の動作検証
 - store: ストア命令の動作検証
 - p2: 演算と分岐命令の動作検証
 - trap: ecall, mret 命令の動作検証
- C プログラム
 - hello: Hello World! をコンソールに表示するプログラム
 - napier: Napier's Constant の値を 64 桁の精度で計算するプログラム
 - pi: pi の値を 64 桁の精度で計算するプログラム
 - prime: 2 を含め、40 個の素数を昇順に見つけるプログラム
 - bubblesort: 100 個の整数を Bubble Sort でソートするプログラム
 - insertsort: 100 個の整数を Insert Sort でソートするプログラム
 - quicksort: 100 個の整数を Quick Sort でソートするプログラム

機能検証は、Verilog-HDL で記述したプロセッサに対して、論理シミュレーター xmvverilog と波形ツール SimVision を用いて、シミュレーションを行った。上記のプログラムが正しく実行され、正確な出力が得られたことを確認した。

4 プロセッサの性能評価と論理合成

設計したプロセッサの性能を、プログラム実行のクロックサイクル数、最小動作クロック周期、面積、と消費電力という面で評価する。

4.1 評価方法

MiBench ベンチマークプログラムを用いて、プログラム実行のクロックサイクル数を求めた。MiBench の中に、stringsearch, bitcnts, dijkstra といったプログラムが含まれている。次に、論理合成ツール Design Compiler を用いて、論理合成を行い、最小動作クロック周期、面積、と消費電力を測定した。

最小動作クロック周期の求め方について説明する。

1. タイミング制約を $10[ns]$ と設定して論理合成を行い、論理合成の結果にある slack (与えたタイミング制約と最大遅延時間との差) が正であることを確認する。
2. タイミング制約を $9[ns], 8[ns], \dots$ のように $1[ns]$ ずつ下げ、slack が負になるタイミング制約を見つける。

例えば、タイミング制約を $5[ns]$ に設定した時に slack が負になったら、作成したプロセッサの最小動作クロック周期は $6[ns]$ になる。

4.2 評価結果

各ベンチマークのプログラムの実行に必要なクロックサイクル数を表 3 に示す。また、プロセッサの最小動作クロック周期、面積、と消費電力の測定結果を表 4 に示す。

ベンチマークプログラム	クロックサイクル数
stringsearch	10594
bitcnts	56040
dijkstra	4079473

表 3 ベンチマークプログラムの実行クロックサイクル数

最小動作クロック周期 $[ns]$	面積 $[\mu m^2]$	消費電力 $[mW]$
6	357534.722794	7.5732

表 4 論理合成の結果

5 改善点

改善点の内容

6 まとめ

まとめの内容

参考文献

参考文献の内容