愛知工業大学研究報告
第39号B　平成16年

MIPS CPU のFPGA化

Implementation of MIPS CPU in FPGA

杉野 晃洋†　，　堀田 厚生††

Akihiro SUGINO　，　Atuo HOTTA

Abstract　The multi-cycle system and the pipelined architecture designed 32-bit CPU which used the MIPS architecture in order to study the design technique of CPU. It is made to implement in the product 'EP1S10F780C7ES' of the stratix series which is one of the highly efficient FPGA devices of ALTERA. The design of a multi-cycle system and a pipelined architecture was performed, and both performance comparison was performed. MIPS CPU of a multi-cycle system operated by 61.60MHz. MIPS CPU of a pipelined architecture operated by 42.90MHz. A general performance ratio is considered that an about 2.60-time performance ratio is obtained.

1．はじめに

現在、コンピュータだけではなく車や家電などあらゆるものが高度に電子制御化され、その制御に使用される CPU はあらゆるシステムの頭脳としてとても重要な役割を持つ。CPU（Central Processing Unit）の性能はシステム全体の処理能力に多大な影響を与える。CPU の中でも RISC(Reduced Instruction Set Computer)アーキテクチャを採用したものは、ワークステーション、ゲーム機などさまざまな製品に広く使用されている。RISC は従来の CISC(Complex Instruction Set Computer)と異なり、回路を極力簡単にし、高性能なコンパイラとの相乗効果により処理効率を高めている。

また、FPGA（Field Programmable Gate Array）という論理回路を自由にプログラミングすることができるLSIが広く使用されるようになってきた。この需要の増加により FPGA の性能が飛躍的に上がり、１つの CPU 程度ならチップ１つに集積化できるようになった。そして、FPGA の種類によってはCPU 以外にもさまざまな機能を持った回路を実現できるため、チップ１つでシステムを構成できるSOC も可能である。

本研究では、CPU RISC アーキテクチャである MIPS アーキテクチャを利用してマルチサイクル方式、パイプライン方式の MIPS CPU を Verilog - HDL および回路図エディタにて設計し、FPGA に実装し、動作検証を行い、正常動作を確認した。

設計に使用したソフトウェアは、ALTERA 社の『Quartus II　Version 3.0』である。設計、論理合成、配置配線、シミュレーション、検証までの全てのプロセスを『Quartus II　Version 3.0』で行うことができる。

2．FPGA

2・1　FPGA（Field Programmable Gate Array）の概要

FPGA とはプログラミングすることができる主に S-RAM ベースの LSI である。現在のところ数百万ゲートレベルの規模を持つデバイスが市販されており、CPU や図形/画像処理あるいは音声処理用の回路など大規模な回路を容易に実現することが可能である。専用 LSI（ASIC）より動作が遅く高価だが、少数の使用ならば FPGA の方が断然低コストである。1985 年に Xilinx 社によって初めて製品化された。（ここで言う FPGA の定義は、組み合わせ論理を実現するための LUT と順序論理を実現するためのレジスタを組み合わせた構造を持っていると考えている。）

† 愛知工業大学大学院　工学研究科
電気電子工学専攻（豊田市）

†† 愛知工業大学電子工学科（豊田市）

２・２　Stratix の概要

Stratix ファミリは電源電圧 1.5V、0.13μm、全層銅配線 SRAM プロセスを採用している。
今回は CQ 出版の Stratix 評価キットに設計した CPU を書込み、動作させた。キットに搭載されている ALTERA FPGA Stratix EP1S10F780C7ES は LE 10570 個、RAM 920,448 bit、DSP 6 個、PLL 6 個などをサポートしている高性能なデバイスである。

３．MIPS アーキテクチャ

MIPS とは 1981 年～1984 年にスタンフォード大学のジョン・L・ヘネシーらのグループにより開発された CPU アーキテクチャーであり，RISC タイプで限定された数の命令を高速実行するという設計方針が採用されている。MIPS の名前は「Microprocessor without Interlocked Pipeline Stages Chip」から由来し、性能を評価する MIPS「Million Instruction Per Second」との掛け言葉でもある。MIPS は元々ハードウェア的にインターロック機構を持たず、コンパイラの動的スケジューリングによりストール処理を行っていた。ただし、今ではパイプライン段数の増加によりあまり高速化にならないためハードウェア的にストール処理を行っている。

３・１　命令形式

MIPS はパイプラインに適したアーキテクチャになっているため命令形式が少ない。実際には**表２**に示す 3 つの命令形式がある。
各フィールドの意味を以下に示す。

OP（Operation code）
　　：操作の種類
Rs（Source Register1）
　　：参照先レジスタ
Rt（Source Register2）
　　：参照先レジスタ、格納先レジスタ
Rd（Destination Register）
　　：格納先レジスタ
Shamt（Shift Amount）
　　：シフト量
Func（Function）
　　：R 形式命令用の操作の種類
Immediate　：即値
Address　：アドレス値

３・２　命令セット

MIPS 命令セットは整数演算を中心に 25 命令に限定した物を用いた。詳細は**表１**と**表３**に示す。halt 命令は本来 MIPS の命令にはないが、プログラム停止の都合上付加した。

４．マルチサイクル方式 MIPS CPU の設計

４・１　マルチサイクル方式

マルチサイクル方式では、１つの命令を命令フェッチ（FETCH）、命令デコード＆レジスタフェッチ（DECODE）、演算実行（EXE）、メモリアクセス（MEM）、ライトバック（WB）などの複数のステップに分け、それぞれのステップが１クロックサイクルで実行される。１命令を複数サイクルに分割することで動作周波数をあげることができる。命令の種類によってステップ数は異なる。例を**図１**に示す。また、具体的な各サイクルの処理内容を**表４**に示す。

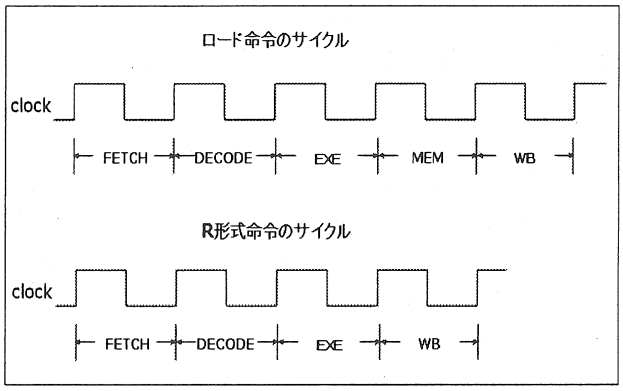

図１．命令のサイクル

４・２　ステートマシン

ステートマシン（state machine）とはあらかじめ決められた複数の状態を，決められた条件にしたがって，決められた順番で遷移していく制御回路である。ステートマシンには 2 種類ある。

・ミーリ型ステートマシン
出力信号を内部状態と入力信号によって生成する。回路の構成は簡単だが入力ノイズが回路に直接影響する。入力信号とフリップフロップの出力の先にある組み合わせ回路の分だけ出力信号が遅れるので，高速回路には不向きである。

表1. MIPS のアセンブリ言語

区分	命令	例	意味	備考
算術演算	add	add $s1,$s2,$s3	$s1=$s2+$s3	3オペランド,オーバフローあり
	subtract	sub $s1,$s2,$s3	$s1=$s2-$s3	3オペランド,オーバフローあり
	add immediate	addi $s1,$s2,100	$s1=$s2+100	定数を加算,オーバフローあり
	add unsigned	addu $s1,$s2,$s3	$s1=$s2+$s3	3オペランド,オーバフローなし
	subtract unsigned	subu $s1,$s2,$s3	$s1=$s2-$s3	3オペランド,オーバフローなし
	add imm. unsign.	addiu $s1,$s2,100	$s1=$s2+100	定数を加算,オーバフローなし
論理演算	and	and $s1,$s2,$s3	$s1=$s2 & $s3	レジスタ間の AND
	or	or $s1,$s2,$s3	$s1=$s2 | $s3	レジスタ間の OR
	and immediat	andi $s1,$s2,100	$s1=$s2 & 100	レジスタと定数間の AND
	or immediate	ori $s1,$s2,100	$s1=$s2 | 100	レジスタと定数間の OR
	shift left logical	sll $s1,$s2,10	$s1=$s2 ≪10	定数分左へ論理シフト
	shift right logical	srl $s1,$s2,10	$s1=$s2 ≫10	定数分右へ論理シフト
データ転送	load word	lw $s1,100($s2)	$1=MEM[$s2+100]	メモリからレジスタへ転送
	store word	sw $s1,100($s2)	MEM[$s2+100]=$1	レジスタからメモリへ転送
	load upper imm.	lui $s1,100	$s1 =100×$2^{16}$	定数を上位 16 ビットへロード
条件分岐	branch on equal	beq $s1,$s2,25	if($s1==$s2) goto PC+4+100	等しいときに PC 相対分岐
	branch on not eq.	bne $s1,$s2,25	if($s1!=$s2) goto PC+4+100	等しくないときに PC 相対分岐
	set on less than	slt $s1,$s2,$s3	if($s2<$s3) $s1=1 ;else $s1=0	大小比較,レジスタ間 2の補数用
	set less than imm.	slti $s1,$s2,100	if($s2<100) $s1=1 ;else $s1=0	大小比較,レジスタと定数間 2の補数用
	set less than uns.	situ $s1,$s2,$s3	if($s2<$s3) $1=1 ;else $s1=0	大小比較,レジスタ間 自然数用
	set l.t.imm.uns.	sltiu $s1,$s2,100	if($s2<100) $s1=1 ;else $s1=0	大小比較,レジスタと定数間 自然数用
無条件ジャンプ	jump	j 2500	go to 10000	目的のアドレスへジャンプ
	jump register	jr $ra	go to $ra	サブルーチン RET 命令
	jump and link	jal 2500	$ra=PC+4; go to 10000	サブルーチン CALL 命令
制御	halt	halt		処理の停止

表2. MIPS の命令形式

名前	例						備考
フィールド長	6ビット	5ビット	5ビット	5ビット	5ビット	6ビット	MIPS 命令の長さは 32 ビット
R 形式	op	rs	rt	rd	shamt	funct	算術命令の形式
I 形式	op	rs	rt	address/immediate			データ転送，分岐，即値命令の形式
J 形式	op	target address					ジャンプ命令の形式

表3. MIPS の機械語

名前	形式	例						備考
		6ビット	5ビット	5ビット	5ビット	5ビット	6ビット	
add	R	0	2	3	1	0	32	add $s1, $s2, $s3
sub	R	0	2	3	1	0	34	sub $s1, $s2, $s3
addi	I	8	2	1	100			addi $s1, $s2, 100
addu	R	0	2	3	1	0	33	addu $s1, $s2, $s3
subu	R	0	2	3	1	0	35	subu $s1, $s2, $s3
addiu	I	9	2	1	100			addiu $s1, $s2, 100
and	R	0	2	3	1	0	36	and $s1, $s2, $s3
or	R	0	2	3	1	0	37	or $s1, $s2, $s3
andi	I	12	2	1	100			andi $s1, $s2, 100
ori	I	13	2	1	100			ori $s1, $s2, 100
sll	R	0	0	2	1	10	0	sll $s1, $s2, 10
srl	R	0	0	2	1	10	2	srl $s1, $s2, 10
lw	I	35	2	1	100			lw $s1, 100($s2)
sw	I	43	2	1	100			sw $s1, 100($s2)
lui	I	15	0	1	100			lui $s1, 100
beq	I	4	1	2	25			beq $s1, $s2, 25
bne	I	5	1	2	25			bne $s1, $s2, 25
sit	R	0	2	3	1	0	42	sit $s1, $s2, $s3
slti	I	10	2	1	100			slti $s1, $s2, 100
sltu	R	0	2	3	1	0	43	situ $s1, $s2, $s3
sltiu	I	11	2	1	100			sltiu $s1, $s2, 100
j	J	2	2500					j 2500
jr	R	0	31	0	0	0	8	jr $ra
jal	J	3	2500					jal 2500
halt	?	63	0	0	0	0	0	halt

注. 通常$s0－$s7 は MIPS の仕様上$17－$24 に定義されているが今回は理解の簡易性のため$0－$7 として扱う。

表4. 各サイクルの処理内容

サイクル	処理内容
FETCH	PC の更新、メモリから命令を読出す
DECODE	命令の種類の解読、レジスタから必要なオペランドの値を出力
EXE	演算処理
MEM	メモリ内容の更新、または読出し
WB	レジスタファイルの更新

・ムーア型ステートマシン

出力信号は内部状態のみから生成される。入力信号の出力信号への影響は直接的にはない。
状態を示すフリップフロップだけで出力が決まるので，結果的に回路規模が大きくなります．しかし，出力に組み合わせ回路がないので高速に動作する。

今回設計したステートマシンはムーア型である。
各命令の各サイクルにステートを割り当て、各ステート時にどのような制御を行うかを決定しハードウェア化する。

4・3 設計内容

マルチサイクル方式 MIPS CPU の設計の特徴

1．FETCH ステージは 2 分割

今回、メモリの実現には実装する stratix の内部ＲＡＭを使用する。しかし、内部ＲＡＭには入力ポートにＦＦが強制的にデフォルトで組み込まれているのでステージを２つに分割されてしまう。また、ＦＦを並列化させるデータパスを作ることで解決する可能性はあるがハーバードアーキテクチャを取り入れていないため並列化だけではタイミングが間に合わなくなる。
具体的にいうと、FETCH ステージの前のサイクル（各命令の最終ステート）に PC を確定させる信号がセットアップされなければならない。その信号は PC_sel<=0、M_ADDR_sel<=0 である。しかし、ストア命令（sw）は最終ステートで M_ADDR_sel<=1 にしないとメモリにデータを書き込むことができない。よって、命令メモリをデータメモリを独立させるハーバードアーキテクチャを取り入れない限り FETCH ステージを 1 サイクルにできないと考えられる。今回はマルチサイクルの設計が主目的ではなく参考としている文献にもハーバードアーキテクチャを取り入れられていないため、FETCH ステージは 2 分割とした。2 分割した場合の FETCH ステージは**表5**のようになる。

表5. FETCH ステージ分割による変更内容

サイクル	処理内容
FETCH 1	PC の更新
FETCH 2	メモリから命令を読み出す

また、DECODE ステージ内のレジスタファイルの実現にも実装する stratix の内部ＲＡＭを使用する。よって、FETCH ステージと同様の問題が起こる。しかし、こちらは工夫次第で簡単に解決できる。

2．演算ユニット

演算ユニットは 32 ビットの ALU、バレルシフタ、SLT（Set Less Than）を使用する。

3．命令デコーダの仕様

命令デコーダはステートマシンにより実現され、そのステートマシンは命令レジスタ（IR）から得た命令内容を解読し、データパス内の各モジュールを制御するための信号を出力する。

4．テストプログラムの実行方法

テストプログラムは『Quartus Ⅱ』の mif ファイルにより stratix の内部ＲＡＭに初期設定しておき、評価キットの電源投入＆リセット解除後に実行される。その後 PC モニタ上で動作確認を行う。

5．パイプライン方式 MIPS CPU の設計

5・1　パイプライン方式

パイプラインとは

・複数の命令を並列に実行することでスループットを上げる一般的な高速化手法である。
・ただし、1 命令の処理時間が短縮されるわけではない。
・ハードウェアの資源を効率的に使用することができる。
・パイプライン方式の理想処理能力は 1 CPI だが実際にはハザードによって命令処理の待ち時間がかかるので速度向上比は落ちる。

パイプラインの動作は基本的に**図2**のようになっている。

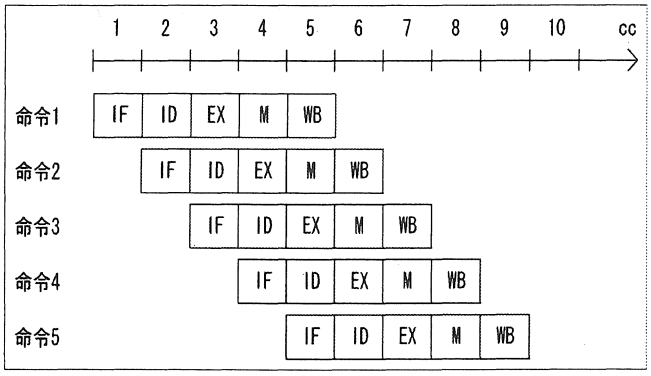

図2. パイプラインの動作のイメージ

5・2　設計内容

パイプライン方式 MIPS CPU の設計の特徴

1．IF ステージは 1 クロックサイクル
　・パイプラインは元々クロックサイクル毎にメモリアドレス制御を行う必要がある。
　・構造ハザードを防ぐためハーバードアーキテクチャを採用する。
　・メモリの入力段 FF を PC と同じ機能を持たせて並列化する。

これら 3 つの事から 5 段パイプラインにすることが可能になった。

2．分岐制御ユニットは EXE ステージで行う

参考文献 1）では分岐ハザードでの性能低下を防ぐため、ID ステージで行うが、これを採用するとデータハザードが起こり得る。この場合無理にフォワーディングするとクリティカルパスが大きくなり、最大動作周波数が相当低下するため意味がない。代わりにストールすると ID ステージにした意味がない。それは、プログラムの構造上たいていの場合ストールすることになるからである。（ただし、コンパイラで動的スケジューリングを行えば多少改善されると思われる。）よって、分岐制御ユニットは EXE ステージで行う。

3．ストール制御とフラッシュ制御の方法

RF の入力段 FF には we 信号がないため RF 単独でフラッシュ制御できない。また、rst、clken 入力を使用してストール制御、フラッシュ制御をしようとすると前の命令の WB ステージが実行できなくなるので使用できない。よって、RF のストール制御を hazard_sel で対応し、フラッシュ制御を Flush_sel1、Flush_sel 2 で対応する。
具体的にストール制御とフラッシュ制御の方法を説明すると次のようになる。

ストール･･･IF、ID ステージ内の命令を保持する。
　IF ステージ･･PC、裏 PC の we 信号を‘0’にする。
　ID ステージ･･IF_ID_FF1,2、IR の we 信号を‘0’にする。
RF の入力段 FF は hazard_sel<=1 で対応する。

フラッシュ･･･IF、ID ステージ内の命令を無効化する。
　IF ステージ･･･Flush_sel1<=1 により NOP 命令を RF の入力段 FF に送る。NOP 命令とは sll $0,$0,0 である。（$0 はハードウェア的に常に‘0’）
　ID ステージ･･･Flush_sel2<=1 による制御信号の無効化により命令の無効化を行う。

4．フォワーディングの方法

参考文献 1）の P.445～P.447 の考え方を参照した。

6．結果

6・1　論理合成結果

『Quartus II　Version 3.0』による論理合成結果を**表6、表7**に示す。

表6. マルチサイクル方式 MIPS CPU 論理合成結果

	ロジアナ IP なし	ロジアナ IP あり
最大動作周波数	69.78MH z	61.60MH z
使用 LE（全 10570 個）	1298（12%）	3164（29%）
使用内部メモリ（全 920448bit）	33792（3%）	262144（28%）
使用 PLL（全 6 個）	1 個	1 個

表7. パイプライン方式 MIPS CPU 論理合成結果

	ロジアナ IP なし	ロジアナ IP あり
最大動作周波数	51.28MH z	42.90MH z
使用 LE（全 10570 個）	1440（13%）	3599（34%）
使用内部メモリ（全 920448bit）	43008（4%）	350208（38%）
使用 PLL（全 6 個）	1 個	1 個

対象デバイスは ALTERA FPGA Stratix EP1S10F780C7ES である。
論理合成結果から得られた最大動作周波数は『Quartus II Version 3.0』内に持つデータベースから算出されたものなのでシミュレーション上の目安となる値である。
実機動作検証のためロジアナ IP（Intellectual Property）を追加する場合、ロジアナ IP なしよりも少し最大動作周波数が低下する。ロジアナ IP は本来、必要がない回路なので両方の結果を示しておく。（ロジアナ IP なしの場合は詳しい内部動作が確認できないが、ロジアナ IP ありの状態で正常動作していれば FPGA の特性上、正常動作していると思われる。）

『Quartus II Version 3.0』の内部動作検証機能「Signal Tap II」は FPGA 内の余った LE、RAM を利用して ALTERA が提供しているロジックアナライザ IP（Intellectual Property）をインプリメントする。RAM の容量に比例してサンプリングできる回数が変化する。よって、stratix レベルの FPGA でないとこの機能を利用するのは難しい。

ここで言う IP（Intellectual Property：知的財産権ファイル）とは、個人や企業が LSI のソフトウェア，およびハードウェア機能ブロックを設計したものであり、無償、または有償で他の人が使用できるようにしたものである。

各方式のクリティカルパスと最大遅延時間は
マルチサイクル方式 MIPS CPU（ロジアナ IP あり）
命令デコーダ→ALU ゼロ出力→分岐制御→PC
＝（セル遅延 7.369 ns）＋（配線遅延 7.945 ns）
＝ 15.314 ns

パイプライン方式 MIPS CPU（ロジアナ IP あり）
フォワーディング回路 2 → BP1_sel → BP2_sel → ALU 出力 → ALUOUT FF
＝（セル遅延 8.392ns）＋（配線遅延 11.560ns）
＝ 19.952ns

6・2 テストプログラム

設計した CPU の検証のために次のテストプログラムを作成した。
テストプログラム 1・・・32 ビットデータ内のビットが‘1’である個数を数えるプログラム
テストプログラム 2・・・乗算プログラム（16 ビット×16 ビット）
テストプログラム 3・・・ユークリッドの互除法
テストプログラム 4・・・演算ユニットの機能チェックプログラム
テストプログラム 5・・・データハザード時のフォワーディングチェックプログラム（6 つ）

6・3 シミュレーション

各テストプログラムのシミュレーションは正常動作した。その際の動作周波数はマルチサイクル方式 72.60MHz、パイプライン方式 57.70MHz である。
ここで、マルチサイクル方式の動作周波数が論理合成結果の**表6**に対して高いのはクリティカルパスを通ることがなかったために起こったと考えられる。

表8. テストプログラム1の結果 （入力データ 5555AAAA）

	実行クロックサイクル数	命令数	CPI	MIPS
マルチサイクル方式	665	145	4.59	13.42
パイプライン方式	184	145	1.27	33.78

表9. テストプログラム2の結果 （入力データ 49×13）

	実行クロックサイクル数	命令数	CPI	MIPS
マルチサイクル方式	135	28	4.82	12.78
パイプライン方式	40	28	1.43	30.00

表10. テストプログラム3の結果 （入力データ 18、12）

	実行クロックサイクル数	命令数	CPI	MIPS
マルチサイクル方式	1947	399	4.88	12.62
パイプライン方式	476	399	1.17	36.67

（単位　実行クロックサイクル数：クロックサイクル、命令数：ステップ）

6・4　実機検証

各テストプログラムの実機検証は正常動作した。その際の動作周波数はマルチサイクル方式97.50MHz、パイプライン方式77.00MHzである。

ここでもシミュレーション検証同様、マルチサイクル方式の動作周波数が論理合成結果の**表6**に対して高かった。その原因はクリティカルパスを通ることがなかった事、またはFPGAに限らずLSIには1つ1つ性能のばらつきがある事という2つの原因が考えられる。今回の場合は　シミュレーション検証時にクリティカルパスを通ることがなかったと考えられるため実機検証でも同様の原因だと考えられる。

マルチサイクル方式とパイプライン方式に対してテストプログラム1～3を処理させ、その結果から性能比較をしたものを**表8～表10**に示す。

また、性能は次のような数式で表すことができる。

$$CPI = \text{実行クロックサイクル数} \div \text{命令数} \quad (1)$$

$$MIPS = fmax\ (MHz) \div CPI \quad (2)$$

数式（2）が示すとおり、MIPS（Million Instruction Per Second）値はCPI（Clock cycle Per Instruction）が低下してもfmax（最大動作周波数）が低下してしまうとあまり向上しない。よって、パイプライン方式のマルチサイクル方式に対するMIPS比はそれぞれ2.53倍、2.36倍、2.92倍となった。

テストプログラムによってMIPS比が異なるのはほとんど分岐ハザードの回数による差である。一般的なプログラムを実行した場合のMIPS比は平均の約2.60倍と思われる。

7．結言

マルチサイクル方式MIPS CPUの設計、パイプライン方式MIPS CPUの設計をVerilog－HDLおよび回路図エディタにて実現し、5つのテストプログラムに対してStratix EP1S10F780C7ESによる実機動作を確認した。シミュレーション同様に正しく動作した。動作周波数はマルチサイクル方式61.60MHz、パイプライン方式42.90MHzである。

参考文献

1）パターソン＆ヘネシー,コンピュータの構成と設計第2版,上下巻,日経BP,東京,1999
2）深山 正幸 他三名.HDLによるVLSIの設計 共立出版株式会社,東京,2002
3）CQ出版,DESIGN WAVE MAGAZINE 10,2003
4）CQ出版,Interface 9,10 2003

（受理　平成16年3月19日）