計算機アーキテクチャ　121－7
設　計　自　動　化　82－7
（１９９６．１２．１２）

ハードウェア/ソフトウェア・コデザインのための ビット数指定言語 Valen-C とその処理系の開発

冨山 宏之 † 井上 昭彦 † 清水 友人 ‡
神原 弘之 § 安浦 寛人 †

† 九州大学 大学院システム情報科学研究科 情報工学専攻
‡ 京都大学大学院 工学研究科 電子通信工学専攻
§ (財) 京都高度技術研究所

あらまし

本稿ではビット数指定言語 Valen-C を提案し，現在開発中の Valen-C コンパイラの構成と実現法について述べる．Valen-C とは各変数のビット数を陽に記述できるように C 言語を拡張した言語である．Valen-C は特定用途向けディジタル・システムのハードウェア/ソフトウェア・コデザインに有効である．Valen-C を使用することにより，設計者はアプリケーション・プログラムを変更することなく，システムの面積と性能が最適化されるようにプロセッサのデータ語長を変更することができる．また，Gated Clock 機構と Valen-C を使用することにより，プロセッサの低消費電力化も期待される．

キーワード

ハードウェア/ソフトウェア・コデザイン，プログラミング言語，リターゲッタブル・コンパイラ，特定用途向けプロセッサ

Developing Valen-C Language and Its Compiler for Hardware/Software Codesign

Hiroyuki TOMIYAMA † Akihiko INOUE † Tomoto SHIMIZU ‡
Hiroyuki KAMBARA § Hiroto YASUURA †

† Department of Computer Science and Communication Engineering, Kyushu University
‡ Department of Electronics and Communication, Kyoto University
§ ASTEM RI

Abstruct

This paper presents a programming language, called Valen-C, and its compiler under development. Valen-C is an extended C which enables programmers to specify the bit length of each variable explicitly. Valen-C and its compiler are useful for HW/SW codesign of application specific integrated processors, especially for optimization of the word width of processors. Low power of processors is also expected by cooperating the Valen-C compiler with gated clock mechanisms of the data paths.

Keywords

Hardware/Software codesign, Programming language, Retargetable compiler,
Application specific integrated processor

1 はじめに

マイクロプロセッサを中心としたディジタル・システムは，家電製品，通信機器，ゲーム機器など，様々な用途に用いられており，それぞれの用途に応じて最適なシステムを短期間で設計することがますます重要になってきている．一方，近年の CAD 技術や半導体技術の向上により，プロセッサ自身をも用途に応じて最適化することが可能になってきた．

我々はソフトコア・プロセッサを用いたハードウェア/ソフトウェア・コデザイン手法を提案している [10]．ソフトコア・プロセッサとはアプリケーションに応じてシステム設計者が機能や構成を変更することが可能なコア・プロセッサである．システム設計者は，あらかじめ定められたコア・プロセッサの可変項目を，アプリケーションに応じてカスタマイズすることによりシステム設計を行う．

ソフトコア・プロセッサのプロトタイプ BungDLX の可変項目の中にデータ語長[1]がある [11]．プロセッサのデータ語長はシステムの面積や性能に大きな影響を与えるため，アプリケーションに応じてデータ語長を最適に決定することはシステム設計を行う上で重要である．現在まで我々は，アプリケーション・プログラミング言語として C 言語を想定してきた．しかし，C 言語のデータ型のサイズはプロセッサのデータ語長に依存するため，データ語長を変更すると，アプリケーション・プログラムも変更する必要がでてくる．データ語長が可変である場合に，C 言語でアプリケーション・プログラムを記述するのは得策ではない．

本稿では，プログラム中の各変数のビット数を陽に記述できるように C 言語を拡張した言語 Valen-C (C with Variable Length Variables) を提案し，その有効性について議論する．また，現在開発中のコンパイラの構成および実現法についても述べる．Valen-C 言語とそのコンパイラを使用することにより，システム設計者はアプリケーション・プログラムを変更することなく，プロセッサのデータ語長を変更し，システムの性能とコストを評価することができる．

本稿の構成は以下の通りである．第 2章でプログラム中で各変数のビット数を陽に記述する言語の利点について述べる．第 3章および第 3章で我々が開発しているビット数指定言語 Valen-C およびそのリターゲッタブル・コンパイラの概要を説明する．

[1]プロセッサの基本演算語長．汎用レジスタのサイズ．

2 ビット数指定言語の利点

プログラム中に各変数のビット数を陽に指定することには，主として以下の 3 つの利点がある．

- アプリケーションに応じてプロセッサのデータ語長を適切に決定することにより，システムのコストと性能を最適化することが可能となる．
- 移植性が向上するためにプログラムの再利用が促進され，設計期間が短縮され得る．
- コンパイラがオブジェクト・コード中にビット数の情報を付加し，プロセッサはその情報を元にデータパス回路のクロックを制御することにより，低消費電力化を実現でき得る．

本章では上記の 3 つの利点についてもう少し詳しく説明する．

2.1 データ語長の最適化

組み込み用途の比較的小規模な CPU の面積はデータ語長にほぼ比例することが示されいる [6, 11]．プロセッサのデータ語長は CPU の面積を決定する大きな要因であるため，システム設計を行う際に，アプリケーションに応じてプロセッサのデータ語長を適切に決定することは非常に重要である．

プロセッサのデータ語長を小さくすると CPU 面積は小さくなるが，システムの性能が低下する恐れがある．アプリケーション・プログラム中に 16 ビットの精度を持つ演算が存在と仮定する．この時，プロセッサのデータ語長を 16 ビット未満に変更したとすると，変更後のプロセッサは 16 ビットの精度を持つ演算を 1 命令では実行できなくなる．つまり，プロセッサのデータ語長を小さくすることにより，単精度だった演算が多倍精度になるために，実行サイクル数が増加する．ここで，単精度の演算とはプロセッサのデータ語長と同じサイズの精度を持つ演算を指し，N倍精度の演算とはプロセッサのデータ語長の N倍の精度を持つ演算のことを指す．CPU 面積と性能にはトレードオフ関係が存在するため，アプリケーションや設計目標などに合わせて，プロセッサのデータ語長を最適に決定することが重要である．

更に，Shackleford らはデータ語長は CPU 面積だけでなく，ROM(命令メモリ)，ならびに，RAM(データメモリ) の面積に大きな影響を与えることを示している [7]．プロセッサのデータ語長を小さくすると，単精度

だった演算が多倍精度となるため，命令数が増加し，必要なROMの容量が大きくなる．一方，データ語長を小さくすると，必要なRAMの容量は小さくなる傾向にある[2]．これは，RAMにおけるデータの密度が濃くなるためである．例えば，プログラム中のある変数の精度がnビットだと仮定すると，プロセッサのデータ語長がmビット$(n < m)$の時，RAMの$(m-n)$ビットは使用されていないことになる．

システム面積をCPU，ROM，および，RAMの面積の和と定義すると，システムの面積はデータ語長に対して単調に変化しない．以上のように，プロセッサのデータ語長の最適化は非常に複雑な問題であるが，設計者は各変数の精度が何ビット必要であるかを知っていれば，データ語長最適化の指針となるであろう．しかし，C言語などの多くの高級言語では変数の精度をビット単位で記述することができない．もし各変数のビット数がプログラム中に陽に記述されているならば，システム設計者がプロセッサのデータ語長の最適化を行う際に非常に役に立つであろう．

2.2 ソフトウェアの再利用

システムが大規模化し，複雑化するに従い，その設計期間も長期化するであろう．過去に設計した資産を再利用することは，システムの設計期間の短縮に有効である．アプリケーション・プログラム中の各変数のビット数を陽に指定することは，システムの最適化に非常に有効であるというだけでなく，設計の再利用という観点からも有意義である．

例として，C言語を用いてアプリケーション・プログラムの記述を行う場合を想定する．C言語のデータ型のビット数は，そのプログラムが動作するプロセッサに依存する．そのため，データ語長が16ビットであるプロセッサ上で実行されることを想定して記述されたプログラムは，データ語長が32ビットであるプロセッサ上では正常に動作しない可能性がある．先に述べたように，プロセッサのデータ語長は，CPU，RAM，および，ROMの面積，ならびに，システムの性能に対して非常に大きな影響を及ぼす．そのため，アプリケーションに応じてデータ語長を適切に選択することはシステム設計を行う上で重要である．しかし，システム設計の過程において，データ語長を変更するたびに，プログラムを変更することは設計期間の長期化を招くであろう．また，プロセッサのデータ語長が異なると，過去に設計したシステムのソフトウェア資産を使用することも困難である．

プログラム中の各変数のビット数が陽に指定されており，かつ，そのプログラムが各プロセッサ用のコンパイラにより適切にコンパイルされるならば，そのプログラムはプロセッサのデータ語長とは無関係に正常に動作する．つまり，プロセッサのデータ語長が変わっても，過去に設計されたプログラムをそのまま再利用することが可能である．

2.3 低消費電力化

近年の携帯情報機器の普及により，低消費電力化がシステム設計を行う際の最も重要な設計目標の1つとなってきた．消費電力を削減する手法に，使われていない回路のクロックを停止するGated Clockがある．変数のビット数を陽に指定することができる言語，コンパイラ，および，データパスにビット単位のGeted Clockを採用したプロセッサ・アーキテクチャを用いることにより，システムの低消費電力化を実現することができる．

例として，C言語で書かれたプログラムをデータ語長が32ビットであるプロセッサ上で実行する場合を考える．Cプログラム中に`short`型(16ビットとする)のデータに対する演算が存在する時，コンパイラは16ビットのデータを32ビットに符号拡張し，32ビットの演算命令を生成するであろう．その後，32ビットの演算結果の下位16ビットだけが次の演算で使用されるであろう．この場合，プロセッサは将来使われない上位16ビットの計算を無駄に行い，不必要に電力を消費することになる．コンパイラが16ビットのデータを32ビットに符号拡張せず，かつ，プロセッサがGeted Clockにより32ビットのデータパスの上位16ビットのクロックを停止すれば，低消費電力を実現することができる．しかし，その`short`型のデータが本当に必要とするビット数はは16ビット未満だが，プログラミング言語の制約により16ビットのデータとして宣言されたのかも知れない．

プログラム中の各変数のビット数を陽に指定し，コンパイラがそのビット数の情報をアセンブラコード中に埋め込み，プロセッサがその情報を使用してデータパス部の必要最小限の回路にクロックを供給することにより，更なる低消費電力化を実現することができる．

[2]単調に減少するわけではない．

3 Valen-C 言語の概要

3.1 概要

現在我々は，変数のビット数をプログラム中に陽に指定できるようにC言語を拡張したプログラミング言語 Valen-C (C with Variable Length Variables)，および，その処理系を開発している．基本とする言語にC言語を選んだ理由は以下の通りである．

- 現在まで，組み込みソフトウェアを設計する際，アセンブリ言語だけでなくC言語が広く使用されている．C言語を基本とすることにより，過去に設計したソフトウェアの再利用が比較的容易になる．
- C言語のコンパイラや，コンパイラ作成を支援するツールが豊富であるため，コンパイラの作成が比較的容易になる．

Valen-C 言語の条件文やループ文，関数呼び出しの制御構造はC言語と同様であり，構造体や配列のように基本データ型から導かれたデータ型も使用できる．一方，Valen-C はバイト (8 ビット) がメモリアクセスの最小の単位であることを仮定していないため，`sizeof` 演算子の定義がC言語と異なる．以下，基本データ型と `sizeof` 演算子について，Valen-C 言語とC言語の違いを説明する．

3.2 基本データ型

Valen-C の最大の特徴は，プログラム中で宣言される各変数のサイズ (ビット数) を陽に指定できることである．C言語において，整数のデータ型は `int` であり，`short` または `long` の修飾子を適用することができる．つまり，C言語は実質的に，`short`，`int`，ならびに，`long` の3種類の整数のデータ型を持つ．これらのデータ型のサイズはコンパイラの設計者が決定する[3]．多くの計算機において，`short` は16ビット，`int` は16ビットまたは32ビット，`long` は32ビットのサイズを持つ．Valen-C においては，整数のデータ型は `int`n と記述する．ここで，n は正整数であり，変数のビット数を表す．C言語同様に，符号の有無を指定する修飾子 `signed` および `unsigned` を使用することができる．例えば，13ビットのサイズを持つ符号なし整数 `x` を宣言する際には，

```
unsigned int13 x;
```

と記述する．また，C言語同様に変数のビット数を陽に記述しないことも可能である．具体的には，

- `short`
- `int`
- `long`
- `long long`[4]

の4つの整数型を使用することができる．これらの型のビット数はコンパイラ設計者が決定し，Valen-C コンパイラのマシン記述中に記述する．ただし，`short` のサイズを m ビットとすると，`int` は $2 \times m$ ビット，`long` は $3 \times m$ ビット，`long long` は $4 \times m$ ビットでなければならない．

Valen-C における浮動小数点数の扱いは未定である．

3.3 sizeof 演算子

Valen-C においても，任意のオブジェクトのサイズを計算するのに使えるコンパイル時の単項演算子 sizeof が存在する．

```
sizeof オブジェクト
```

および

```
sizeof (型名)
```

は指定したオブジェクトをメモリに格納するのに必要なメモリユニットの数を意味する．ここで，メモリユニットとは，メモリのアクセスの最小単位となる記憶領域を指す．C言語においてはメモリのアクセスの最小単位はバイト (8 ビット) であるが，Valen-C においてメモリのアクセスの最小単位がバイトであるとは限らない．しかし，Cと同様に，`malloc(sizeof` オブジェクト`)` とすることで，そのオブジェクトを格納するのに必要な記憶領域を確保することができる．

[3]通常，コンパイラ設計者は計算機アーキテクチャを元にデータ型のサイズを決定するため，間接的にデータ型のサイズは計算機アーキテクチャにより決定される．

[4]`long long` 型はC言語の ANSI 規格では定義されていない．しかし，既存のいくつかの C コンパイラは `long long` 型をサポートしている．GNU CC [8] は `long long` 型をサポートするコンパイラの1つである．

4 コンパイラの開発

4.1 対応するプロセッサ・アーキテクチャ

Valen-C はシステム設計に使用されることを想定しているため，Valen-C コンパイラはプロセッサ・アーキテクチャに対して，リターゲッタビリティを有していなければならない．本節では，現在開発中の Valen-C コンパイラが対応し得るプロセッサ・アーキテクチャの範囲について説明する．

4.1.1 基本アーキテクチャ

Valen-C コンパイラはプロセッサがロード/ストア・アーキテクチャであることを仮定している．データメモリにアクセスできる命令は，データメモリと汎用レジスタ間のデータ転送命令のみでなければならない．つまり，メモリとレジスタ間の演算命令や，メモリ間の演算命令は許されない．

現在の Valen-C コンパイラは命令スケジューリングを行わない．遅延スロットやプロセッサの並列性を有効に利用するためには，命令スケジューラを新たに作成する必要がある．

4.1.2 レジスタ構成

プロセッサは汎用レジスタ・ファイルを持つことを仮定している．つまり，Valen-C コンパイラは以下の事項を仮定している．

- データメモリへアクセスする際，汎用レジスタによりアドレスの指定を行わなければならない．
- 乗算などの特定の演算のための専用レジスタを持つことは許されない．

以降，汎用レジスタを `rn` と表記する (n は非負の整数)．

プロセッサはゼロレジスタ (値が常に 0 であるレジスタ) を持つことができる．ゼロレジスタが汎用レジスタファイル中に含まれていたとしても，本稿では，ゼロレジスタを汎用レジスタとは呼ばない．

レジスタウィンドウには対応しない．

汎用レジスタを組にして使用することが可能である．汎用レジスタを組にして使用する命令の例として，次の命令を考える．

```
add_l r0, r2, r4
```

この命令は，2 つの汎用レジスタの組 (r0, r1) を連結した値と，レジスタの組 (r2, r3) を連結した値との加算を行い，加算結果をレジスタの組 (r4, r5) に格納する．第 1 ソース・オペランドとして r0 の 1 つの汎用レジスタを指定しているが，2 つの汎用レジスタの組 (r0, r1) が第 1 ソース・オペランドであることを暗に指定している．

汎用レジスタを組にして使用する場合，以下の条件を満たさなければならない．

- 組にして使用できる汎用レジスタ数は 2 または 4 である．3 本の汎用レジスタ，あるいは，5 本以上の汎用レジスタを組にして使用することはできない．
- m 本の汎用レジスタを組にして使用する場合，スタックポインタ等の特定の用途に用いるレジスタ以外の汎用レジスタの数は m の倍数でなければならない．

4.1.3 多倍精度演算に対するハードウェア・サポート

多倍精度演算に対するハードウェア・サポートとして，プロセッサは以下の命令を有していることを Valen-C コンパイラは仮定している．

- 以下の要件を満たす加減算命令
  - 実行の結果，キャリーの有無を保持する
  - 直前に実行した加減算命令のキャリーを含め，加減算を行う
- 以下の要件を満たす 1 ビットのシフト命令
  - 実行の結果，桁あふれ/桁落ちしたビットを保持する
  - シフトを行い，その直前に実行したシフト命令の桁あふれ/桁落ちビットを最下位/最上位ビットに設定する．
- データアドレス長 (A_D) および命令アドレス長 (A_I) がデータ語長 (W_D) より大きい場合，複数の汎用レジスタを組にしてアドレス指定を行い，ロード/ストア，分岐，関数呼出しを行う．

4.1.4 可変項目と制限

変更可能な項目は以下の通りである．

- 汎用レジスタ数

- 命令セット
- データ語長 W_D
- データアドレス長 A_D
- 命令アドレス長 A_I
- データメモリと汎用レジスタ間のデータ転送のサイズ
- データメモリのアラインメント

しかし，上記の項目を自由に変更できるわけではなく，変更可能な範囲に制限が存在する．また，可変項目の間には相関関係が存在し，各々を独立に変更できるわけではない．

命令セット　デスティネーション・オペランドとなる汎用レジスタ (または，汎用レジスタの組) は 1 命令当たり高々1 つでなければならない．命令語長は命令毎に異なっていても良い．

データ語長・データアドレス長・命令アドレス長　データ語長 W_D とは汎用レジスタのビット数，データアドレス長 A_D とはデータメモリのアドレスを指定するために必要なビット数，命令アドレス長 A_I とは命令メモリのアドレスを指定するために必要なビット数をそれぞれ意味する．ただし，A_D ならびに A_I は W_D の最小値の 4 倍以下でなければならない．

4.1.5　マシン記述

コンパイラを各プロセッサ・アーキテクチャに対してカスタマイズするためには，先に説明した可変項目に加え，ビット数を陽に指定していない Valen-C 言語の各データ型のビット数と，その型を持つデータをデータメモリに格納する際のアラインメントを記述する必要がある．

4.2　使用するツール

Valen-C 言語の構文は C 言語と非常に似ているため，コンパイラのフロントエンド[5]は，既存の C 言語のフロントエンドを修正して使用することにした．我々は Stanford 大学で開発された SUIF (Stanford University Intermediate Format) ライブラリ [9][6]を使用すること

[5]Valen-C から中間フォーマットへの変換を行うモジュール．
[6]SUIF ライブラリは主としてコンパイラの研究者のために開発されたものである．

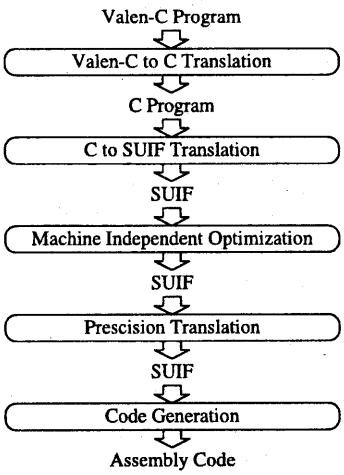

図 1: コンパイル・フロー

にした．SUIF とはプログラミング言語の中間フォーマットであり，SUIF ライブラリとは中間フォーマットに対して処理を行う関数およびクラスのライブラリである．また，SUIF ライブラリのパッケージ中には，C 言語から中間フォーマットへの変換を行うフロントエンド snoot や，マシン非依存の最適化処理を行う porky というツール等が含まれている．

SUIF ライブラリは C++を用いて開発されたため，Valen-C コンパイラも C++を用いて開発することにした．

4.3　コンパイル・フロー

開発するコンパイラは，主として以下の 5 つの処理を行う (図 1参照)．

1. Valen-C から C への変換
2. C から SUIF への変換
3. マシン非依存最適化
4. 演算精度の変換
5. コード生成

以下，各処理の内容および実現法について説明する．

4.4 Valen-C から C への変換

既存のCのフロントエンドを使用するために，Valen-CプログラムをC言語と同じ構文を持つプログラム[7]に変換する．

第3節で述べたように，Valen-C言語とC言語の構文上の違いは型宣言だけである．Valen-Cプログラムから Cプログラムへ変換は，Valen-Cプログラムにおいて intn と宣言された型を，short，int，long，または，long long 型の中で，いずれかに置き換えることにより実現する．その際，intn は n ビット以上である最小の整数型に置き換えられる．例えば，short のサイズを m ビット，int のサイズを $2 \times m$ ビットとすると，$m < n \leq 2 \times m$ を満たす intn が int に変換される．

4.5 C から SUIF への変換

Cの構文を持つプログラムを中間フォーマット SUIF に変換する．SUIF パッケージに含まれる snoot というツールを修正することにより使用する．

修正が必要となる理由は，C言語はプロセッサのデータ語長が int 型のサイズと等しいことを仮定しているためである．snoot はプログラム中に int よりも小さい整数型同士の演算が存在すると，そのオペランドを int 型に変換する．しかし，Valen-C 言語の int 型のサイズはプロセッサのデータ語長よりも大きい場合があり得る．例えば，プロセッサのデータ語長が short 型のサイズと等しい時，コンパイラが short 型の演算を int 型に変換することは，必要以上に高い精度で計算を行うことになる．我々は，オペランド同士の型を一致させる場合や，プログラム中に型変換命令が存在する場合を除き，型変換を行わないように snoot を修正することにした．

4.6 マシン非依存最適化

マシン非依存最適化とは，

- 定数伝搬
- コピー伝搬
- 定数の畳み込み
- 共通部分式の削除

などの特定のプロセッサ・アーキテクチャに依存しない最適化処理を指す．これらの最適化処理は SUIF パッケージ中の porky というツールを使用することにより実現する．

[7]データ型の意味がC言語と異なる．

4.7 精度の変換・コード生成

精度の変換とは，多倍精度の演算を複数の単精度の演算に変換することである．例えば，n 倍精度の加算は n 個の単精度のキャリー付き加算に変換される．多倍精度の乗算，除算，および，剰余算を単精度に変換すると，変換後の命令数が非常に多くなる可能性がある．そこで，多倍精度の乗算，除算，および，剰余算はライブラリ関数として実現する．また，多倍精度のシフト演算はビットシフトを繰り返すことにより実現する．

4.8 プログラミングの注意事項

4.4節で説明したように，Valen-C コンパイラは，プログラム中で intn と宣言された整数値を n ビット以上の整数型の変数として処理する．つまり，intn 型の値を n ビットの整数として処理するわけではない．Valen-C コンパイラは，intn 型の値が n ビットで保持し得る値を越えた時の動作を保証しない[8]．

例として，次の Valen-C プログラム片を考える．

```
signed int8 x = 0xff;
```

プロセッサのデータ語長が8ビットの時，x は8ビットの符号つき整数型の変数として扱われ，x は -1 に初期化される．しかし，プロセッサのデータ語長が10ビットの時，x はコンパイラの内部では10ビットの符号つき変数として扱われるため，x は 255 に初期化される．以上のように，プロセッサのデータ語長が8ビットの時と，10ビットの時とでは，プログラムの意味が変わってくる．上記のプログラム片をプロセッサに対して非依存にするためには，

```
signed int8 x = -1;
```

と記述しなければならない．

5 おわりに

本稿では，ビット数指定言語 Valen-C を提案し，また，Valen-C コンパイラの構成および実現法について

[8]プロセッサのデータ語長により，プログラムの実行結果が変わる恐れがある．

説明した．Valen-C と Valen-C コンパイラは，特定用途向けディジタル・システムのハードウェア/ソフトウェアコデザインに有効である．Valen-C を用いると，アプリケーションに応じてプロセッサのデータ語長を適切に決定することにより，システムのコストと性能を最適化することが容易となる．更に，プログラムの移植性が向上するために設計資産の再利用が促進され，設計期間が短縮され得るという利点もある．また，Valen-C コンパイラがオブジェクト・コード中にビット数の情報を付加し，プロセッサはその情報を元にデータパス回路のクロックを制御することにより，低消費電力化を実現することも期待される．

今後，Valen-C コンパイラの実現を急ぎ，我々が提案するシステム設計支援環境を構築したい．

謝辞

本研究に共同で取り組んで頂いている Hewlett-Packard 研究所の Barry Shackleford 氏，三菱電機 (株) の鈴木文雄氏，安田光宏氏，(財) 九州システム情報技術研究所の伊達博博士，九州大学安浦研究室の Eko Fajar Nurprasetyo 氏，大隈孝憲氏に感謝致します．Valen-C 言語の研究は Barry Shackleford 氏との議論の中で始まった．低消費電力化手法に関して日頃議論して頂き，貴重な御意見を頂いた九州大学安浦研究室の石原亨氏に感謝致します．また，日頃御討論頂く村上和彰助教授をはじめとする九州大学安浦研究室の諸氏に感謝致します．

本研究は一部，情報処理振興事業協会 (IPA) 独創的情報技術育成事業，および，科学研究費補助金 (特別研究員奨励費) の支援による．

参考文献

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools.* Addition-Wesley, 1986.

[2] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann Publishers, Inc., 1990.

[3] I.-J. Huang and A. M. Despain "Synthesis of Application Specific Instruction Set," *IEEE Trans. CAD/ICAS*, 14(6):663–675, June 1995.

[4] B. W. Kernighan and D. M. Ritchie (石田晴久訳). プログラミング言語 *C* 第 *2* 版 —*ANSI* 規格準拠—. 共立出版, 1989 年.

[5] J. Sato, A. Y. Alomary, Y. Honma, T. Nakata, A. Shiomi, N. Hikichi, and M. Imai. "PEAS-I: A Hardware/Software Codesign System for ASIP Development," *IEICE Trans. Fundamentals*, E77–A(3):483–491, March 1996.

[6] B. Shackleford, M. Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, and H. Yasuura. "Satsuki: An Integrated Processor Synthesis and Compiler Generation System," *IEICE Trans. Information and Systems*, E79–D(10):1373–1381, October 1996.

[7] B. Shackleford, M. Yasuda, E. Okushi, H. Koizumi, H. Tomiyama, and H. Yasuura. "Memory-CPU Size Optimization for Embedded System Designs," Submitted to *34th Design Automation Conf.*.

[8] R. M. Stallman. *Using and Porting GNU CC for version 2.7.2*, 1995.

[9] Stanford Compiler Group. *The SUIF Library Version 1.0*, 1994.

[10] H. Yasuura, S. Nakamura, H. Tomiyama, and H. Akaboshi. "Hardware-Software Codesign with a Soft-Core Processor," In *Proc. of Synthesis and System Integration of Mixed Technologies (SASIMI'95)*, pages 79–84, 1995.

[11] エコー ファジャル ヌルプラセティヨー, 井上昭彦, 冨山宏之, 安浦寛人. "ハードウェア/ソフトウェア・コデザインのためのソフトコア・プロセッサの検討," 信学技報, VLD96–13, CAS96–13, DSP96–33, 1996 年 6 月.