社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2003－ＳＬＤＭ－112　(20)
2003／11／28

エンジン制御システムのHW/SWコデザイン

夏目　貴将†　飯山　真一†　本田　晋也†　冨山　宏之††　高田　広章††

† 豊橋技術科学大学情報工学系　〒441-8580 豊橋市天伯町雲雀ヶ丘1-1
†† 名古屋大学大学院情報科学研究科　〒464-8603 名古屋市千種区不老町
E-mail: †{natsu,shin,honda,tomiyama,hiro}@ertl.jp

あらまし　本稿では，エンジン制御システムに対するハードウェア/ソフトウェア・コデザインの適用事例を報告する．エンジン制御において，マップ引きと呼ばれる処理が頻繁に実行される．マップ引き処理をハードウェア化することで，処理の高速化を実現した．論理合成により面積と遅延時間の評価を行なうと同時に，ハードウェア/ソフトウェア・コシミュレーションを通じて機能の検証を行なった．

キーワード　HW/SWコデザイン，マップ引き，エンジン制御システム

HW/SW Codesign of an Engine Control System

Takayuki NATSUME†, Shinichi IIYAMA†, Shinya HONDA†,

Hiroyuki TOMIYAMA††, and Hiroaki TAKADA††

† Department of Information and Computer Sciences, Toyohashi University of Technology
Hibarigaoka 1-1, Tenpaku-cho, Toyohashi, Aichi, 441-8580 Japan
†† Graduate School of Information Science, Nagoya University
Furo-cho, Chikusa-ku, Nagoya, 464-8603 Japan
E-mail: †{natsu,shin,honda,tomiyama,hiro}@ertl.jp

Abstract　This paper describes a case study on hardware/software codesign of an engine control system. Table look-up is frequently executed for engine control. We have implemented the table look-up in hardware for improving the system performance. The designed circuit was successfully synthesized and validated through hardware/software cosimulation.

Key words　HW/SW Codesign, Table Look-Up, Engine Control Systems

1. はじめに

近年，組込みシステムのソフトウェアの複雑化や大規模化が著しいが，エンジン制御システムもその例外ではない．低燃費化，走行性の向上，排気ガスや安全性に関する厳しい規則をクリアするためには，エンジン制御の品質および処理精度を向上させる必要がある[1]．その結果，エンジン制御システムのソフトウェアは年々複雑化，大規模化しており，それに伴い，プロセッサに対する性能向上の要求が高まっている．しかしながら，より高性能なプロセッサを新規に導入することは，コスト面だけでなく，耐環境性能や信頼性の面からも現実的でないことが多い．

性能を高める他の方法として，従来ソフトウェアとして実行していた処理の一部をハードウェア化する方法がある．一般的に，ハードウェア(LSI)はソフトウェアと比較して実行が高速であるが，機能の変更や追加が困難である．また，生産量が少ないとコストが高くなるという欠点がある．そのため，今後変更が見込まれず，かつ，全車種共通の処理をハードウェア化の対象とすべきである．

本稿では，エンジン制御システムを対象としたハードウェア/ソフトウェア・コデザイン(以下，単にコデザイン)の適用事例について報告する．エンジン制御システムにおいて，マップ引きと呼ばれる処理は，プロセッサ負荷が高く，全車共通の処理であり，かつ，今後も使用される基本的な処理である．本研究では，マップ引き処理をハードウェア化することで，エンジン制御の高速化を狙った．まず，マップ引き回路のアーキテクチャを検討し，Verilog-HDLを用いて設計記述した後，設計した回路の性能を評価した．また，ハードウェア/ソフトウェア・

コシミュレーション (以下，単にコシミュレーション) により，動作確認を行なった．

以下，2 章において本研究で対象とするマップ引き処理について説明する．3 章ではマップ引き回路のアーキテクチャの検討と設計を行ない，4 章で評価する．

2. エンジン制御システム

エンジン制御は，エンジン回転数，アクセル開度や冷却水の温度等，センサから得られる入力信号から，燃料の噴射量や噴射時間等を決定し，各アクチュエータを駆動することで実現される．燃料の噴射量や噴射時間等を決定するために，補正係数の計算が頻繁に行なわれている．通常，マップと呼ばれるテーブルに格納された値を用いた補正係数の計算が行なわれる．この処理は一般に「マップ引き」と呼ばれる(注1)．

このマップ引きは，全車共通の処理であり，かつ，今後も使用される基本的な処理である．マップ引きは，主としてデータ探索と補間計算の 2 つの処理から構成され，補間計算の際に乗除算が行なわれている．現在，マップ引きは ECU (Electronic Control Unit) と呼ばれるマイクロコントローラで行なわれることが多い．しかし，現在の ECU の多くは高速な乗除算器を有しておらず，かつ，マップ引きは頻繁に実行されるため，エンジン制御におけるマップ引きの負荷は非常に高い．そこで，本研究では，マップ引き処理をハードウェア化することにより，エンジン制御の高速化を狙う．

本章ではマップ引き処理の概要を説明する．

2.1 マップ引き処理

マップ引き処理は，マップと呼ばれるテーブルを用いて，エンジン制御に必要な補正係数を求める処理である．センサから得た入力値から補正係数を求める計算は非線形となり，数式での表記が不可能な場合や，たとえ数式での表記が可能であっても非常に大きな計算量を必要とする場合が多い．そのため，あらかじめ入力に対する補正計算の結果を，離散的な点で構成されるマップへ格納する．その間の入力値に対する出力は，その入力値をはさむ 2 点の値から補間して求められる．具体的には，図 1 に示すようなマップを用いて，センサから与えられる冷却水温から暖気増量補正係数を求める．例えば冷却水温が -20℃ (A 点) の場合，点 A_0 および点 A_1 の値を使用し，中間の値を計算により補間することで暖気増量補正係数を求める [2]．本稿では，図 1 のように，変化する 1 つの入力値に対して，それに対応する出力値が 1 つ決まるようなマップを 1 次元マップと呼び，これを補間することを 1 次補間と呼ぶ．

より精度の高い制御を行なうために，2 つ以上のセンサ入力から 1 つの補正係数を求める場合がある．例えば，図 2 に示すような 2 次元マップを用いて，エンジン回転速度 (入力 X) と吸気管圧力 (入力 Y) から基本噴射時間を求める(注2)．1 次補間と同様に，2 つの入力 X, Y を囲む 4 点 Z_{00}，Z_{01}，Z_{10}，Z_{11}

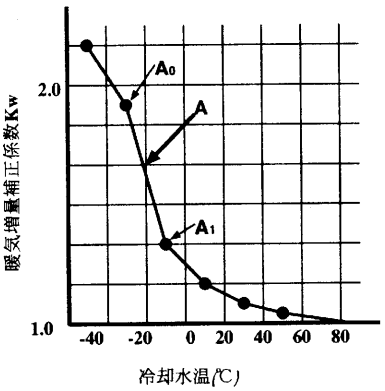

図 1　1 次元マップ

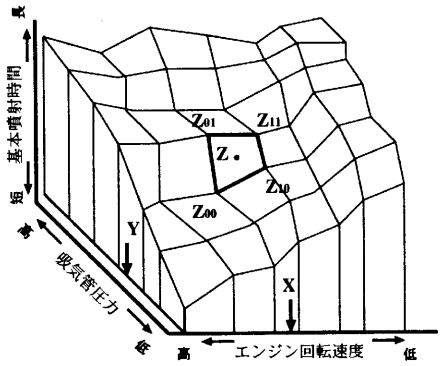

図 2　2 次元マップ

を求めた後，補間計算を行なって出力値 Z を得る．

このように，マップ引き処理は，入力値に対して，マップ中のどのデータを用いるかを決定するデータ探索と，そのデータを用いて近似的な出力値を得る補間計算からなる．

2.2 補間計算

マップ引き処理で行なわれる補間計算について説明する．

1 次補間は，図 3 に示すように，マップの連続する 2 点間 $[x_0, x_1]$ の入力 x に対して，2 点間 $[y_0, y_1]$ で対応する y を求める計算であり，その補間 (1 次補間) の計算式は式 (1) で示される．

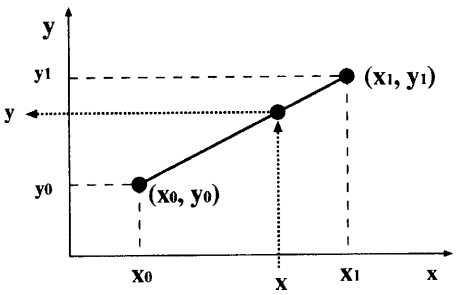

図 3　1 次補間

(注1)：マップをテーブルと呼ぶこともあるが，ここではマップと呼ぶ．

(注2)：本稿では，変化する 2 つの入力値に対して，それに対応する出力値が 1 つ決まるようなマップを 2 次元マップと呼び，これを補間することを 2 次補間と呼ぶ．

$$y = \frac{(y_1 - y_0)(x - x_0)}{(x_1 - x_0)} + y_0 \quad (1)$$

2 次補間は，式 (2)〜(4) に示すように，1 次補間計算を 3 回行なうことで実現可能である．具体的に，図 4 を用いて，x 軸方向に連続する 2 点間 $[x_0, x_1]$ の入力値 x と，y 軸方向に連続する 2 点間 $[y_0, y_1]$ の入力値 y がある場合について説明する．まず，マップから与えられる線分 AB を用いて，x に対する 1 次補間計算を行ない E の座標を求める．同様に，線分 CD を用いて，x に対する 1 次補間計算を行ない F の座標を求める．さらに，線分 EF から，y に対する 1 次補間計算を行ない出力値 z を得る．

$$z_0 = \frac{(z_{10} - z_{00})(x - x_0)}{(x_1 - x_0)} + z_{00} \quad (2)$$

$$z_1 = \frac{(z_{11} - z_{01})(x - x_0)}{(x_1 - x_0)} + z_{01} \quad (3)$$

$$z = \frac{(z_1 - z_0)(y - y_0)}{(y_1 - y_0)} + z_0 \quad (4)$$

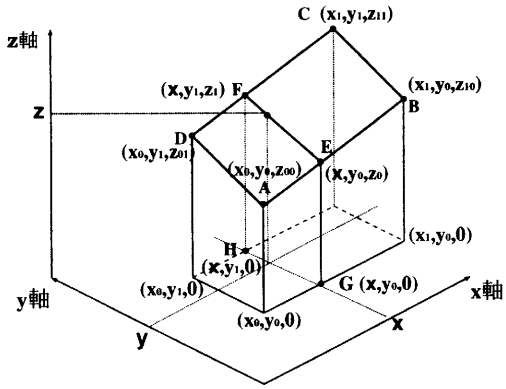

図 4　2 次 補 間

さらに高次のマップを用いた補間計算は，2 次補間と同様に，1 次補間の計算を繰り返すことで実現可能である．具体的には，n 次補間を行なうには，$2^n - 1$ 回の 1 次補間計算を行なえばよい．より精度の高い制御を行なうためには，より高次の補間計算を行なう必要があるが，その次数が高くなるに伴い，指数関数的にプロセッサの負荷が高くなる．従って，専用化されたハードウェアによる補間計算の処理は必須になると考えられる．

3. アーキテクチャの検討

2 章で述べたように，マップ引き処理は探索処理と補間計算処理で構成される．本研究では，1 次補間を対象としたマップ引き回路を設計し，この回路にハードウェアとソフトウェアのインタフェースを追加した．

図 5 に本研究で設計したマップ 引き回路のアーキテクチャを示す．

ハードウェア化の対象はマップ引き処理のみで，その他の処理は，従来通りソフトウェアで行なうものとした．HW/SW 間でデータ通信を行なうために，以下の I/O レジスタを用意した．

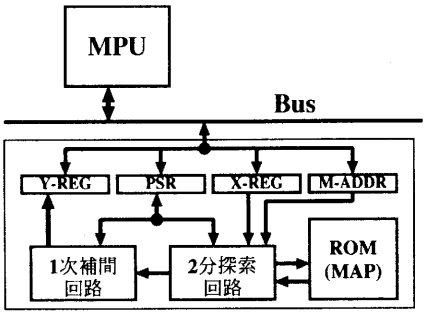

図 5　マップ引き回路

- X-REG(X value REGister)
- M-ADDR(Map top ADDRess)
- Y-REG(Y value REGister)
- PSR(Peripheral Status Register)

X-REG は，入力値 X をソフトウェア側の MPU(Micro Processing Unit) から受け取り，探索回路に渡すためのレジスタである．M-ADDR は探索で使用するマップの先頭アドレスである．マップの先頭番地には，そのマップの点の数 (以下，マップ点数と呼ぶ) が格納されており，マップの X 値，Y 値はその後に連続して格納されている (3.1 節参照)．そのため，マップの先頭番地のみをハードウェアに与えることで，ハードウェアはマップの値を得ることが可能である．Y-REG は補間計算結果である Y 値を出力するレジスタである．PSR は，マップ引き回路の状態を示す状態フラグ，ソフトウェアから要求される計算リクエストフラグ，および，マップ引き処理が終了したことを示す終了フラグ等，マップ引き回路のステータスを示すレジスタである．

探索回路への入力値，メモリ内のデータ，および，出力値のデータは 16 ビット整数型である．従って，補間計算回路への入力データは全て 16 ビットである．マップ引き回路本体である探索回路および 1 次補間回路の詳細については，3.1 節および 3.2 節にて説明する．

3.1 探 索 回 路

代表的な探索法の一つに逐次探索法がある．逐次探索法はリストを先頭から順に探索する方法で，リストの要素数を n とすると計算量は $O(n)$ である．また，リストが予めソートされている場合に有効な探索法として 2 分探索法がある．2 分探索法はリストを二つのグループに分け，目的のデータが属するグループが判明した後，そのグループをさらに 2 分し，目的のデータが属するグループを調査するという操作を繰り返す．計算量は $O(\log n)$ である．

本研究では 2 分探索法を応用したアルゴリズムを使用する．図 6 の例を用いて，そのアルゴリズムを説明する．

マップ点数が 10 のマップ (XMAP) があり，この配列要素を M_i とする．これに，$(M_2 \leqq X_{in} < M_3)$ を満たす入力値 X_{in} を与え，その値に対応するデータ (この場合は M_2 と M_3) を探索する場合を考える．

探索に使用する変数として，i，j，および，K を用意する．i

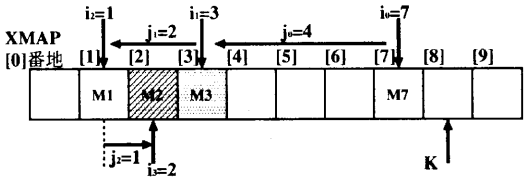

図 6　2 分探索法

は XMAP 中のデータの番地を指すインデックスで，j の値により移動する場所が決められる．また，K はマップ点数以下で最大の 2^m(m は正の整数) である．図 6 の場合 $K = 8$ である．i の初期値 i_0 は $K-1$，j の初期値 j_0 は $K/2$ とする．図 6 の場合，$i_0 = 7$，$j_0 = 4$ である．

次に XMAP の i 番地のデータ (M_i) と入力データ X_{in} を比較して，($X_{in} \leqq M_i$) または i 値がマップ点数以上の場合，i の値を $i-j$ とする．この条件以外の場合は i の値を $i+j$ とする．また，この演算と同時に，j の値を $j/2$ とする．図 6 の場合は，$i_1 = 7-4 = 3$ と，$j_1 = 4/2 = 2$ の計算を同時に行なう．

以上の計算を繰り返し，$j = 0$ となった時にループを終了する．その時点での M_i と入力値 X_{in} を比較して，($M_i \leqq X_{in}$) の場合は，M_i と M_{i+1} を出力し，($X_{in} < M_i$) の場合は，M_{i-1} と M_i を出力する．図 6 の場合は，($M_2 \leqq X_{in}$) であるため，M_2 と M_3 を出力する．また図 7 に示すように，M_2，M_3 に対応する YMAP の値をインデックス値 i を使用して出力する．

このように，出力された M_2，M_3 は，2.2 節の式 (1) で示される x_0，x_1 に相当し，YMAP の中から選出された 2 つの値も，y_0，y_1 に相当する．

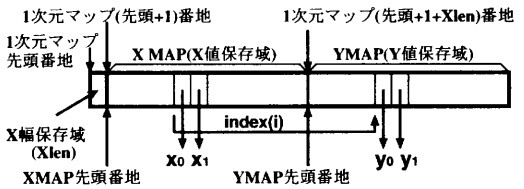

図 7　Y 値の検出

このようなアルゴリズムを持つ 2 分探索回路を Verilog-HDL を用いて設計した．図 8 に設計した 2 分探索回路を示す．

X 値が XV-REG へ，探索するマップの先頭番地が MT-ADDR-REG へ渡され，探索リクエスト信号 (図中では省略) が送られると，探索回路は動作を開始する．まず，マップの先頭番地からマップ点数を読み出し，Xlen-REG へ格納する．次に，マップ点数から K 値を算出し，その値を使用して i と j の初期値を算出し，I-REG と J-REG へ格納する．また，i の初期値をアドレスデコーダへ送り XMAP の i 番地の値を M_i-REG へ格納する．この次のステージでは，比較器により I-REG，Xlen-REG，XV-REG，および，M_i-REG の値を比較し I-REG の値を決定する．また，この値をアドレスデコーダへ送り，次に M_i-REG へ格納する値を XMAP から読み出す．この処理を J-REG が 0 になるまで繰り返す．J-REG が 0 になると，その時点のインデックス (I-REG) 値を使用し，XMAP の探索結果を X0-REG と X1-REG に，YMAP の探索結果を Y0-REG と Y1-REG に格納し，処理を終了する．

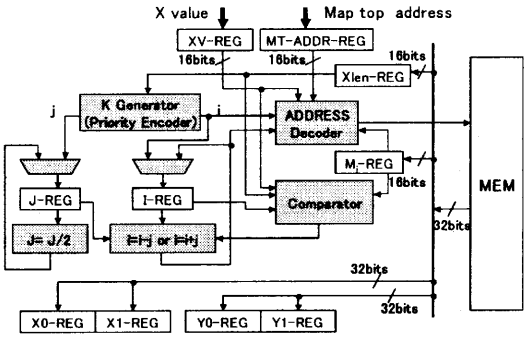

図 8　2 分探索回路

3.2　補間計算回路

2.2 節の式 (1) に示す 1 次補間計算式には，加算 1 回，減算 3 回，乗算 1 回と除算 1 回の演算が含まれる．このうち減算は演算器を共有することが可能であるが，減算器のコストは比較的小さいため性能向上を優先して，減算器を 3 つ用意する．

補間計算回路は 2 分探索回路と同様に，Verilog-HDL を用いて設計した．加減算器は比較的小規模であることや，論理合成ツールのライブラリ環境が整っているため，特定のライブラリに依存しないよう記述し，演算器へのマッピングは論理合成ツールに任せた．乗算器は文献 [3] で設計している乗算器をベースとし，除算器については，演算器アーキテクチャの検討から行なった．

乗算器は，乗算の部分積の和を求めるために，Wallace ツリーを用いる．また，最終的に残る各桁 2 ビットの加算には，キャリ・スキップ・アダーを用いる．

次に，除算器について説明する．除算は，通常，部分剰余と除数を比較し，部分剰余が大きければ除数を引いて新たな部分剰余を求める操作を繰り返すことで実現される [4]．この手法 (基数 2) をそのままハードウェアで行なう場合，通常，減算 1 回とシフト 1 回を 1 クロックで行なう．例えば，本研究で使用する除算器は被除数が 32 ビット，除数が 16 ビットであるため，この手法をそのまま適用した場合 16 クロックを要する．

本研究では，基数を 4 とすることで，除算器の高速化を狙う．この手法は，基数 2 の場合において 2 クロック必要な処理を 1 クロックで実現する手法である．すなわち，(除数 $\times 1$)，(除数 $\times 2$)，および，(除数 $\times 3$) と，部分剰余の比較を並行して行なう．その結果，{ 部分剰余 $\geqq$ (除数 $\times 3$)} ならば $(11)_2$ を，{(除数 $\times 3$) $>$ 部分剰余 $\geqq$ (除数 $\times 2$)} ならば $(10)_2$ を，{(除数 $\times 2$) $>$ 部分剰余 $\geqq$ (除数 $\times 1$)} ならば $(01)_2$ を，{(除数 $\times 1$)$>$ 部分剰余 } ならば，$(00)_2$ を，商の一部として返す．

図 9 に，基数 2 の場合の除算を元に，基数 4 で除算を行なった場合の例を示す．

はじめのステップでは，{(除数 $\times 3$) $>$ 部分剰余 $\geqq$ (除数 $\times 2$)} の条件を満たすため，$(10)_2$ を商の一部として返す．次のステッ

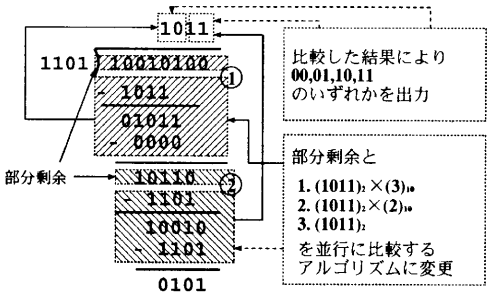

図 9　基数 4 の除算

プでは，{ 部分剰余 ≥(除数 ×3)} の条件を満たすため $(11)_2$ を返す．除数の桁数が更に大きな場合でも，この処理を繰り返すことで，基数 4 の除算が実現される．基数 4 の除算器を用いると，被除数が 32 ビットで除数が 16 ビットの除算処理は，8 クロックで実行することが可能である．

本研究では，まず基数 4 の除算器を設計し，この除算器を用いて，2.2 節の式 (1) で示した 1 次補間計算を行なう回路を設計した．

図 10 に 1 次補間計算回路を示す．

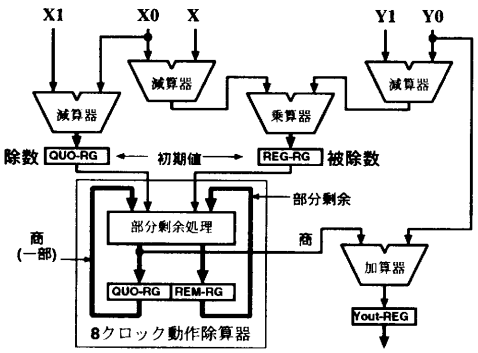

図 10　1 次補間計算回路

はじめに，探索されたデータ等が各演算器へ入力され $(y_1 - y_0)(x - x_0)$ および $(x_1 - x_0)$ を計算し，除算器への入力値 (除数，被除数) を求める．被除数は，クロックの変化とともに部分剰余として値を変化させ，除算器の中を 8 回ループする．ループ後に商が算出され，Y_0 を加算した値が，補間計算回路の出力値として，Yout-REG へ格納される．従って，この回路は，合計 9 クロックで補間計算を行なうことが可能である．

4. 評　　価

設計したマップ引き回路を以下の方法で評価した．まず，RTL で記述したマップ引き回路とソフトウェアとのコシミュレーションを行なった．次に，マップ引き処理をソフトウェアで行なった場合とハードウェアで行なった場合のサイクル数を比較した．更に，設計したマップ引き回路を論理合成し，回路の面積および遅延時間を見積もった．

4.1 協調検証

本研究では，VaST System Technology 社の CoMET [5] というコシミュレーション環境を使用して，ハードウェアとソフトウェアの協調検証を行なった．CoMET には，プロセッサの命令レベルのシミュレーションモデルと，Virtual-Bus と呼ばれる通信インタフェースが用意されている．元々CoMET では C/C++言語を用いてハードウェアのシミュレーションモデルを記述し，コシミュレーションを行なうことが可能である．しかし本研究では実際に Verilog-HDL で記述したマップ引き回路をそのまま用いてコシミュレーションを行なった．CoMET では Virtual-Bus を通じてソフトウェア (プロセッサのシミュレーションモデル) とハードウェアとの通信が行なわれる．ソフトウェアとハードウェアとのインタフェースを定義するために，元々用意されている Virtual-Bus の機能を用いた．具体的には，ソフトウェアから見たハードウェアのメモリマップやハードウェアの入出力ポート等を記述した．記述したソフトウェアプログラムはプロセッサのオブジェクトコードに変換された後，CoMET 上のプロセッサのシミュレーションモデルにロードされ実行される．

マップ引き回路は，ソフトウェアからマップ先頭番地，センサからの入力値を受け取り，要求信号を合図としてその動作を開始する．ソフトウェアは，マップ引き回路の処理で得られる結果が格納されたレジスタを読むことで，マップ引き処理の結果を得る．設計したマップ引き回路とソフトウェアが正しく協調して動作することを，コシミュレーションを行なうことで確認した．

4.2 サイクル数

マップ引き処理をソフトウェアで行なった場合とハードウェアで行なった場合のサイクル数を比較した．探索と補間計算に要するサイクル数，および，マップ引き処理全体に要するサイクル数を表 1 に示す．サイクル数はマップ点数に依存するため，マップ点数を変化させて評価した．尚，ソフトウェアで実行した場合の結果は，除算器を持たないプロセッサを想定している．

表 1　サイクル数の比較

マップ点数	ソフトウェア			ハードウェア		
	探索	補間	マップ引き	探索	補間	マップ引き
2	126	61	182	5	9	14
3-4	126-133	61	182-189	6	9	15
5-8	168-179	61	224-235	7	9	16
9-16	210-219	61	266-275	8	9	17
17-32	252-262	61	308-318	9	9	18

探索処理のサイクル数は，マップ点数が多くなるに従って増加する．表 1 で示すマップ点数の範囲では 25 倍から 28 倍高速に動作させることを可能にした．また，補間計算は，ハードウェアで処理した場合は約 6.7 倍の速度で処理が可能である．

マップ引き処理に必要な時間は，探索に必要な時間と補間計算に必要な時間を加算することで求められる．従って，マップ引き処理全体のハードウェア化により 12 倍から 17 倍高速な処理を行なうことが可能である．

4.3 論理合成結果

設計したマップ引き回路の面積と遅延を評価するため，以下の条件を与えて論理合成を行なった．

- ライブラリ：ROHM 0.35 μm
- 制約条件：遅延 16，20，25，30 ナノ秒

表 2 に上記の条件で論理合成を行なった結果を示す．

表 2　論理合成結果

	遅延制約	16 ns	20 ns	25 ns	30 ns
探索回路	遅延時間 (ns)	19.28	20	24.28	25.53
	面積 (mm^2)	0.3130	0.2237	0.1774	0.1662
補間計算回路	遅延時間 (ns)	15.84	19.07	24.62	28.74
	面積 (mm^2)	0.5785	0.5085	0.5027	0.5098

補間のクリティカルパスは，減算器と乗算器が接続された箇所であったが (図 10 参照)，16 ナノ秒の遅延制約を満足することができた．一方，探索回路のクリティカルパスは，Xlen-REG から比較器等を通ってメモリアクセスを行なうパスであり，16 ナノ秒の制約を満たすことができなかった．このパスがマップ引き回路全体のクリティカルパスであり，その遅延は 19.2 ナノ秒である．よって，設計したマップ引き回路は 50MHz で動作することが可能である．

4.4 考　　察

現在の設計では，探索回路の遅延が補間計算回路の遅延より長いため，探索回路のアーキテクチャの変更を検討中である．具体的には，3.1 節の図 8 に示した回路図において，アドレスデコーダとメモリの間にレジスタを導入することを検討している．これにより，探索に必要となるクロック数が増加するが，遅延時間を短縮することが可能である．

別の方法として，補間計算回路の除算器のアーキテクチャを改良することで，高速化を検討している．除算器の基数を上げると，遅延時間が長くなるが，必要なクロック数は減少する．今回設計した除算器は，基数 4 であるが，基数を 8 または 16 に変更することで，クロック数を削減することが可能である．基数を 8 または 16 にした場合の補間計算回路の遅延時間が，探索回路の遅延時間以内に収まれば，マップ引き回路全体のクロック周波数を下げることなく，サイクル数を削減することが可能である．

ハードウェア・アーキテクチャに改良を加えることで高速化を図る方法の他に，計算の負荷が低くなるように補間計算式自体を変形する方法があげられる．この方法では，補間計算において，除算の負荷が高いことに注目する．2.2 節の式 (1) に示す 1 次補間計算式の場合，$(y_1 - y_0)/(x_1 - x_0)$ の値は既知数であるため，予め計算を行ないメモリに保存しておくことが可能である．これにより，補間計算式は乗算 1 回，減算 1 回，加算 1 回の式で形成され，除算は不要となる．しかし，メモリ容量が増加するという問題がある．また，$(y_1 - y_0)/(x_1 - x_0)$ の値を 16 ビットで保持する場合，桁落ちが生じ，計算精度が低下するという問題も生じる．

本稿では主に 1 次補間の高速化について言及したが，2 次補間の高速化も検討中である．

2.2 節の式 (2)，式 (3) および式 (4) で示されるように，2 次補間は 1 次補間を 3 回繰り返すことで行なわれる．よって，現在設計した 1 次補間回路を使用することにで，2 次補間計算も高速に行なうことが可能である．

2 次補間計算を更に高速化する方法として，1 次補間回路をもう 1 つ追加し，式 (2) と式 (3) を並列に計算する方法が考えられる．これにより，2 次補間計算に要する時間が約 2/3 に短縮される．

また，2 次補間を高速化する他の手法として，計算式を変形することにより除算の回数を 1 回に削減する方法や，更には除算を全く行なわない方法が存在する．しかしながら，計算精度の低下や，メモリ量の増加等の問題を伴う．

今後，これらの問題点と性能向上とのトレードオフを図りながら，2 次補間回路の設計を行なう予定である．また，3 次以上の補間計算についても検討する．

5. おわりに

本稿では，エンジン制御システムのハードウェア/ソフトウェア・コデザインの事例研究の報告を行なった．エンジン制御において頻繁に行なわれるマップ引き処理をハードウェアで実装することにより，高速化を実現した．

今後は，設計した 1 次元マップ引き回路の改良を行なうとともに，2 次元以上のマップ引き回路の設計を行なう予定である．

謝辞　エンジン制御システムについてご助言頂いたトヨタ自動車 (株) 第 2 電子技術部の岡村竜路氏に感謝します．また，ご協力頂いた豊田工業高等専門学校情報工学科の仲野巧 助教授に感謝します．

文　　献

[1] 飯山真一，高田広章，菅沼英明：エンジン制御システムのためのリアルタイム性検証手法，情報処理学会論文誌，Vol.43，No.6，pp.1715–1724，2002.
[2] 淺原宏：電子制御エンジンの基礎・応用，CQ 出版社，1995.
[3] Desigh Wave Magazine，CQ 出版社，2003 年 7 月号.
[4] 富田眞治，中島浩：コンピュータハードウェア，昭晃堂，1995.
[5] http://www.vastsystems.com/