卒業論文

OpenMP によるハッシュ関数の記述と
コード生成手法の検討

氏名	:	三谷 誠志
学籍番号	:	2260060106-0
指導教員	:	山崎 勝弘 教授
提出日	:	2010 年 2 月 19 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

本論文では,当初 OpenMP を用いたハードウェア動作合成システムから生成された回路と,手書きで記述した HDL から生成された回路との比較検証を行い，OpenMP ハードウェア動作合成システムの改良と検証を目的としていた．しかし，トランスレータとコードジェネレータに問題点が見つかったため，システムから回路を生成できなかった．そのためシステムの問題点の検出と対応策を検討した．OpenMP ハードウェア動作合成システムは，PC クラスタや SMP クラスタを用いたアルゴリズム検証・評価を行うシミュレーション系，ハードウェアを自動的に合成するハードウェア動作合成系で構成される．シミュレーション系において，OpenMP を用いて対象のアルゴリズムを記述し，クラスタを用いた高速シミュレーションによって，アルゴリズムの正当性や並列化手法の妥当性の評価・検討，及び要求に対する改良を行う．ハードウェア動作合成系では，シミュレーション系から得られたプログラムを OpenMP の構文を利用しながらハードウェアに変換する．

本論文では，ハッシュ関数を用いてシステムの問題点の検出を行った．ハッシュ関数にはデータ依存性があるが，section 文を用いることで並列化を可能とした．この OpenMP プログラムはトランスレータが int 型にしか対応していなかったため，それ以外の型のデータを上位ビット・下位ビットに分けて演算を行うという対処法を用いて記述した．トランスレータを用いて実際に生成した中間表現には誤りがあり，なおかつ膨大な行数になった．

コードジェネレータに関しては，section 文に対応できないことが判明したので，並列化情報解析モジュールの改良方法の検討を行った．

目次

図目次

表目次

1. はじめに

近年，LSI は大規模計算機やパーソナルコンピュータなどの計算機だけではなく，携帯電話や家電，自動車など様々な機器に搭載されており，その用途は多岐に渡っているとともに，もはや日常生活に必要不可欠な要素となっている．常に新しい機能や製品の開発を求められる現代において，電子機器に搭載される LSI にはさらに高い処理性能，多彩な機能，高い信頼性，低い消費電力などが要求されており，LSI の回路規模や複雑さは著しく増加している．しかしその一方で，激しい市場競争により短い期間，低コストでの開発が要求されており，設計規模の増大に設計能力が追いつかないという設計生産性の危機が問題となっている．

こうした要求における開発期間短縮を実現するため，従来の HDL を用いたハードウェアの設計手法ではなく，C ベース言語を用いてシステムをより高い抽象度で記述する手法に設計手法が移行しつつある．LSI の回路動作を C 言語などのプログラミング言語を用いてより抽象的に記述することで HDL による設計に比べてより少ないコード数で機能が記述できるため大幅な設計生産性の向上が期待できる．また様々な仕様の変更も，設計段階において容易に対応できるようになる．

C 言語のような抽象度の高い言語では空間的な概念や並列動作の概念が含まれていないため，自動合成するにあたって回路面積，性能，消費電力の面で最適なハードウェアが生成されるとは限らない．さらに並列動作の概念は，ハードウェア設計において重要な要素であるが，C 言語などの逐次処理の実行モデルでは表現が難しく，並列化手法の有効性の推定や，設計者の意図した並列動作回路を自動で生成することは難しい．[1]～[6]

以上のような問題を解決するために，並列プログラミングに使用される OpenMP を用いたハードウェア動作合成手法の実現を目標としている．本研究では，当初，本システムにより生成された回路と手書きで記述した HDL により生成された回路を比較し，システムの改良と評価を目的としていたが，システムに不具合が生じたためシステムから回路を生成できなかった．そこで，システムの問題点の検出と対応策を検討した．

OpenMP とは，共有メモリ（SMP）環境における並列プログラミングの標準 API である。逐次プログラムに対し，並列部を示す指示文を追加することにより繰り返し処理を並列化する並列リージョン，及び複数の異なる処理を並列化する並列セクションの範囲を指示することができる．そのため，既存の並列プログラミング言語である MPI や PVM のように通信や信号で並列動作を記述しないので，プログラミングが容易である．

OpenMP の並列動作を容易に記述が可能であり抽象度が高いという利点を生かし，本研究で用いる動作合成システムでは，並列動作回路の動作記述に OpenMP を用いる．並列プログラミング言語をハードウェアの設計に用いることで，並列動作の記述や分析，SMP 環境を用いて設計の早期における検証・評価を容易にし，ハードウェアの動作合成における設計者の負担を軽減することが可能である．

昨年度までの研究で，OpenMP で書かれたプログラムから中間表現までを出力するトランスレータ[1][5]，中間表現から並列化された HDL を出力するコードジェネレータ[2][6]まで実装されており，システムとしての枠組みは完成されている．また，このシステムを用いて素数判定プログラムやマンデルブロ集合など複数のプログラムを検証し，コードジェネレータの改良に関する見通しを検討している[3][4]．本研究では，さらに広い視点からシステムを検証するために，ハッシュ関数の SHA-1 と MD5 を対象として，OpenMP による記述から本システムを用いて生成したコードと Verilog-HDL による記述を比較することにより，システムの問題点を検出し，改良方法を検討した．システムの問題点として，データ依存性のあるプログラムやビット数の大きなデータを扱う場合における，OpenMP のプログラム記述の制約に着目し考察した．

本論文では，第 2 章において OpenMP を用いたハードウェア動作合成システムの構成，および中間表現についてとハードウェアモジュールの生成方法を示す．第 3 章では SHA-1 の OpenMP と Verilog-HDL による記述，第 4 章では MD5 の OpenMP と Verilog-HDL による記述を示す．第 5 章ではコードジェネレータの改良方法の検討及び考察を行う．

2. OpenMP を用いたハードウェア動作合成システム

2.1 ハードウェア動作合成システムの構成

ハードウェア動作合成システムの構成を**図 1** に示す．本研究で用いるハードウェア動作合成システムは，並列化の検証・評価を行うアルゴリズム評価系と動作合成を行うハードウェア動作合成系で構成される．[1]～[6]

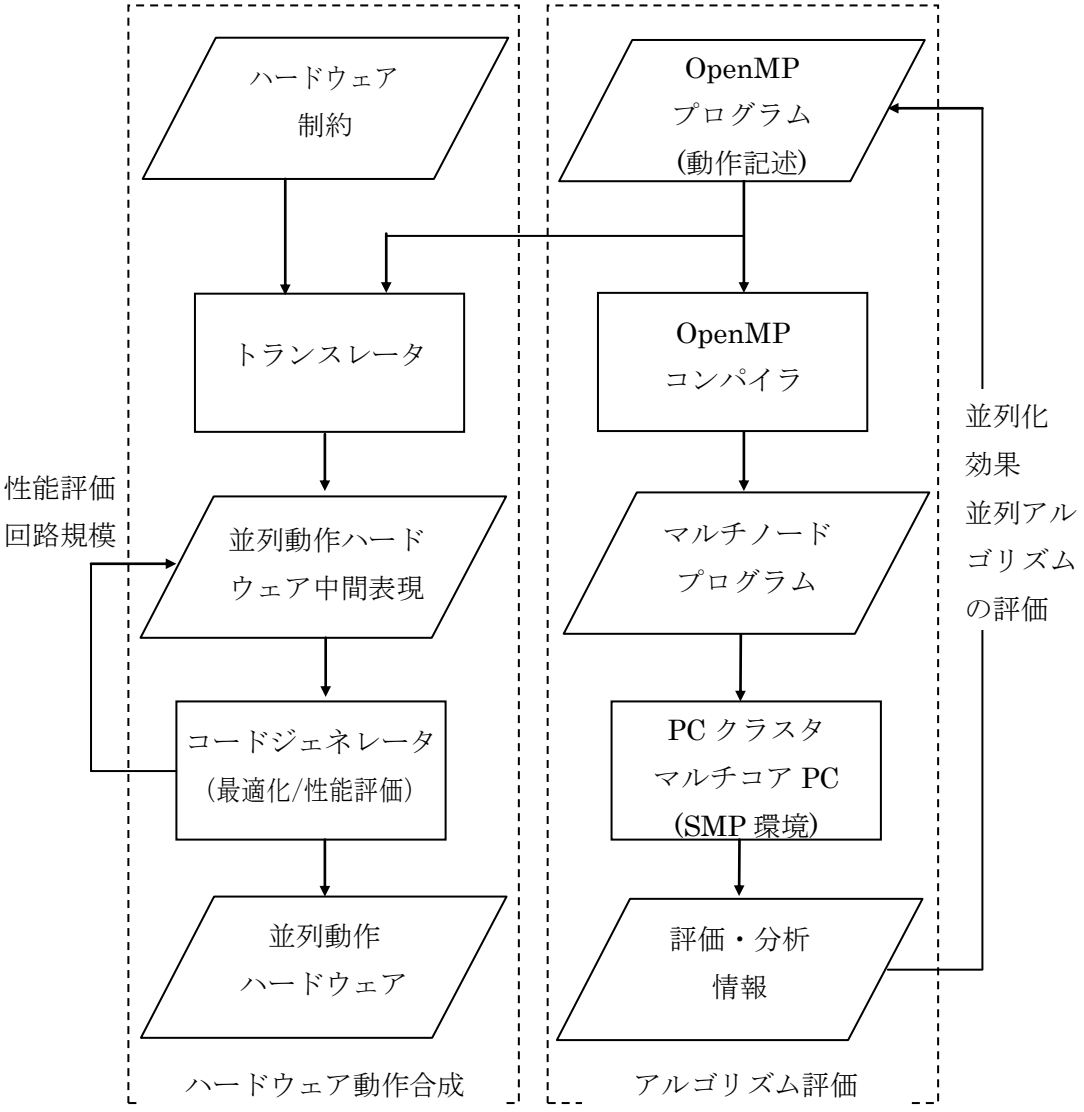

図 1：OpenMP を用いたハードウェア動作合成システム

アルゴリズム評価系では，動作記述として書かれた OpenMP プログラムを，OpenMP コンパイラによってマルチノードプログラムに変換し，PC クラスタやマルチコア PC などの SMP 環境によってアルゴリズムの検証と並列化の評価を行う．すなわち，プロセッサ数を変化させて実行時間を計測し，速度向上を算出して並列化の効果を明らかにする．並列化アルゴリズムの評価・検証を行ない，分析結果を用いて OpenMP プログラムを改善する．

SMP 環境により，高速なソフトウェアシミュレーションを行うことが出来るため，検証時間の短縮と並列化アルゴリズムの評価を設計の早期に行うことが可能である．ハードウェア動作合成系では，アルゴリズム評価系の検証後，得られた OpenMP のソースコードの動作合成を行う．トランスレータを通して中間表現に変換した後，コードジェネレータで並列動作ハードウェアを生成する．トランスレータで出力される中間コードには，OpenMP で指定された並列化情報が含まれており，コードジェネレータではそれらを用いて最適化を行い，並列動作ハードウェアを生成する．

本システムにおけるトランスレータによる中間表現への変換，また中間表現からコード生成を行うコードジェネレータはすでに実装されている．本研究では，本システムの問題点の検出と対処方法の検討を行っている．

2.2 OpenMP プログラムから中間表現への変換

コードジェネレータに入力される中間表現の説明をする．ハードウェア動作合成系におけるトランスレータは，動作記述である OpenMP プログラムを中間表現へと変換する．トランスレータが生成するレジスタ転送方式である RTL 中間表現を用いてハードウェアの生成を行う．RTL の中間表現では処理を表すシンボルテーブルと制御情報を表す状態遷移表が出力され，両方を合わせてコントロールフローグラフ(CFG)を表す．シンボルテーブルは演算される変数や処理，代入先を示しており，状態遷移表によって次に遷移する状態が示される．

OpenMP を用いた C 言語コードを中間コードのシンボルテーブルと状態遷移表に変換した例を図 2 に示す．サンプルのＣ言語コードは単純な for 文に OpenMP におけるプラグマを挿入されており，ループ内では加算と変数への代入を行っている．状態遷移表の#0 で示される状態から, 最初にシンボルテーブルのシンボル 4 で示される定数の代入を表す”: =(1 3)”の処理が行われる．” : =(1 3)”ではシンボル 1 で示される変数 i に対し，シンボル 3 で示される定数 0 の代入を示している．次に for 文の条件式である#2 へ遷移し，条件式の判定を行い分岐する. ここではシンボル 6 が真でなら#3 へ, そうでなければ#1 へと遷移する．シンボル 6 とは条件式を表す”<(1 5)”であり，これはシンボル 1 の変数 i とシンボル 5 の定数”256”との比較，”i<256”を示している．#3 ではシンボルテーブルの 9,10 に該当する加算と代入の演算を行った後，状態をループの先頭に当たる#2 へ遷移する．[3][4]

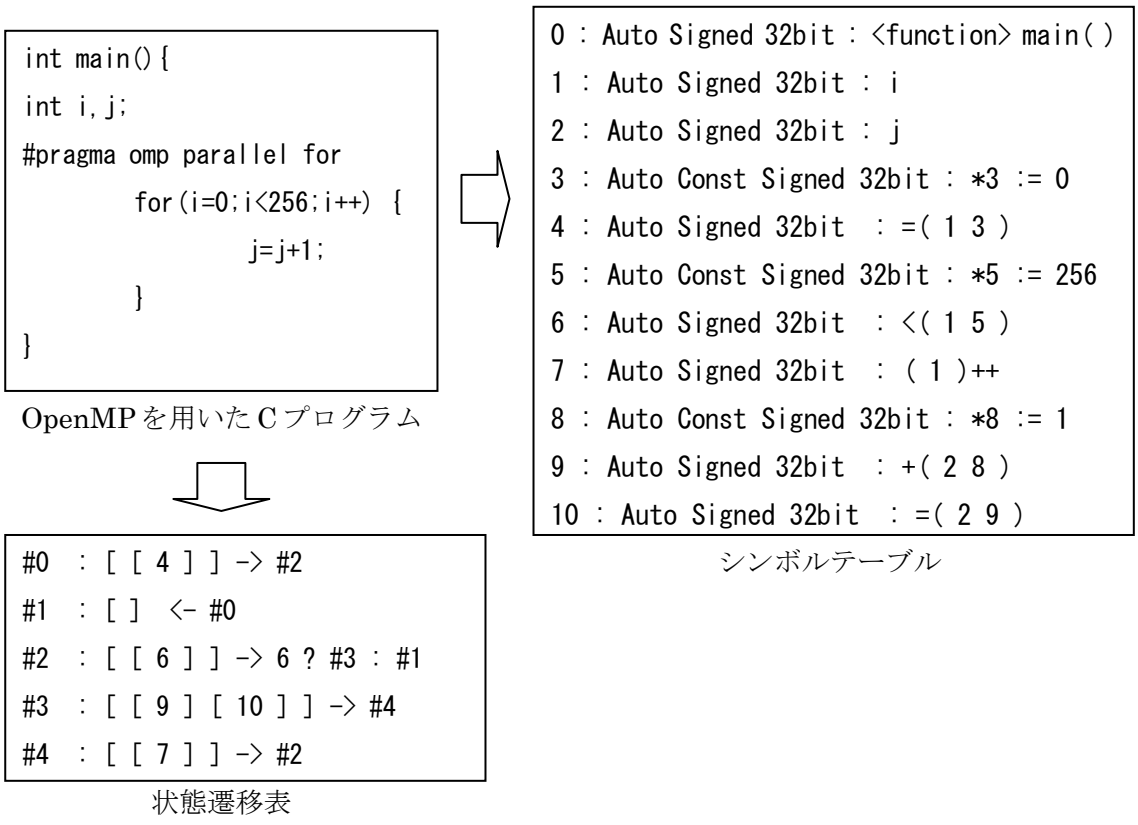

図 2：中間表現のサンプル

2.3 中間表現からのハードウェア生成方法

中間表現からコードジェネレータにより生成されるコードについて説明する．生成されるコードは，演算器部，代入部，状態遷移部に分けて生成される．演算器部と代入部はシンボルテーブル，状態遷移部は状態遷移表を元に生成されている．状態遷移部では CurrentState を状態を表す変数とし，これに P_STATEnumber という別で用意されている複数のパラメータを代入していくことで遷移を行っている．実際はシンボル 5 を実行するときは P_STATE5 というように，number にはその状態で実行されるシンボルの番号が入る．代入部では CurrentState の値によって何を代入するのかを選択し，演算器部も同様に CurrentState の値によって何を加算するのかを選択している．その様子を図 3 に示す．

```
case(CurrentState)
P_STATE4: CurrentState <= P_STATE6;
P_INIT   : if(iSTART==1'b1) CurrentState <= P_STATE4;
        else  CurrentState<= CurrentState;
P_END    : CurrentState <=CurrentState;
P_STATE6: if(i<ConstNum5) CurrentState <= P_STATE9;
        else  CurrentState <= P_END;
P_STATE9: CurrentState <= P_STATE10;
P_STATE10: CurrentState <= P_STATE7;
P_STATE7: CurrentState <= P_STATE6;
default : CurrentState <= CurrentState;
endcase
```

状態遷移部

```
case(CurrentState)
P_END  : oEND <= 1'b1;
P_STATE4 : i <= ConstNum3;
P_STATE7 : i <= ADD_RESULT;
P_STATE9 : REG9 <= ADD_RESULT;
P_STATE10 : j <= REG9;
default : oEND <= 1'b0;
endcase
```

代入部

```
wire [31:0]ADD1_RESULT;
wire [31:0]ADD1_A,ADD1_B;
assign ADD_RESULT = ADD_A + ADD_B;
assign ADD_A = (CurrentState==P_STATE7) ? i :
(CurrentState==P_STATE9) ? j :
j;
assign ADD_B = (CurrentState==P_STATE7) ?
32'd1 :
(CurrentState==P_STATE9) ? ConstNum8 :
ConstNum8;
```

演算器部

図 3：シンボルテーブルと代入部，演算器部の対応

図 4 はハードウェア動作合成によって生成する並列ハードウェアのモデルである．データ並列では同じ処理の繰り返しになるため, 各ノードの DataPath はほとんど同じとなり, FSM は繰り返し範囲に応じて生成される．top_module は各ノードの入力，出力の統合を行う．[1]～[4]

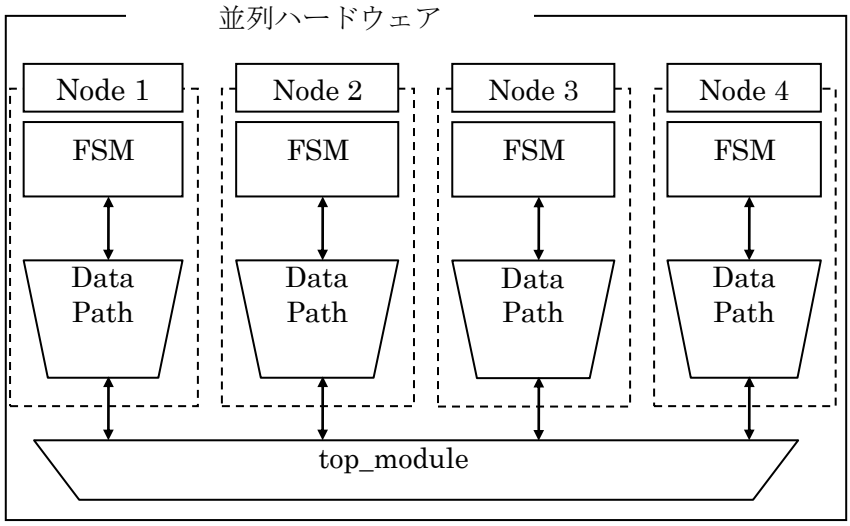

図 4：データ並列のハードウェアモデル

図 5 は並列ハードウェアの各ノードのハードウェアモジュールモデルである．FSM により使用するレジスタと演算器を選択し処理を行う．[1]～[4]

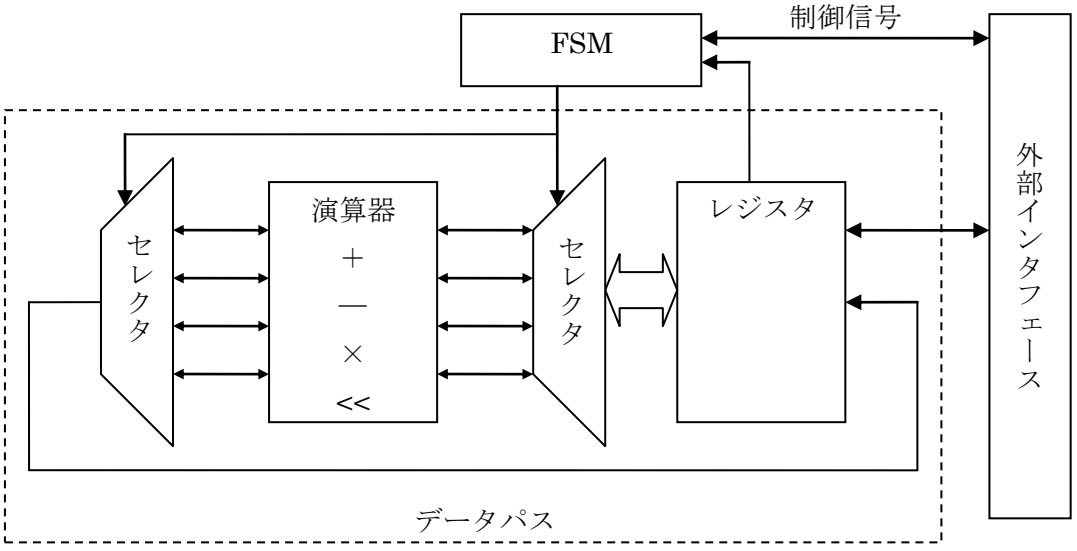

図 5：ハードウェアモジュールモデル

図 6 では実際に HDL 上においてのデータ分割を表している．C ソースコードの for 文の番兵値 256 をもとに，データを 4 つのノードに均等に 64 ずつ分割する処理を表している．この値をもとに各ノードはループ回数や，扱うデータなどを判断する．[3][4]

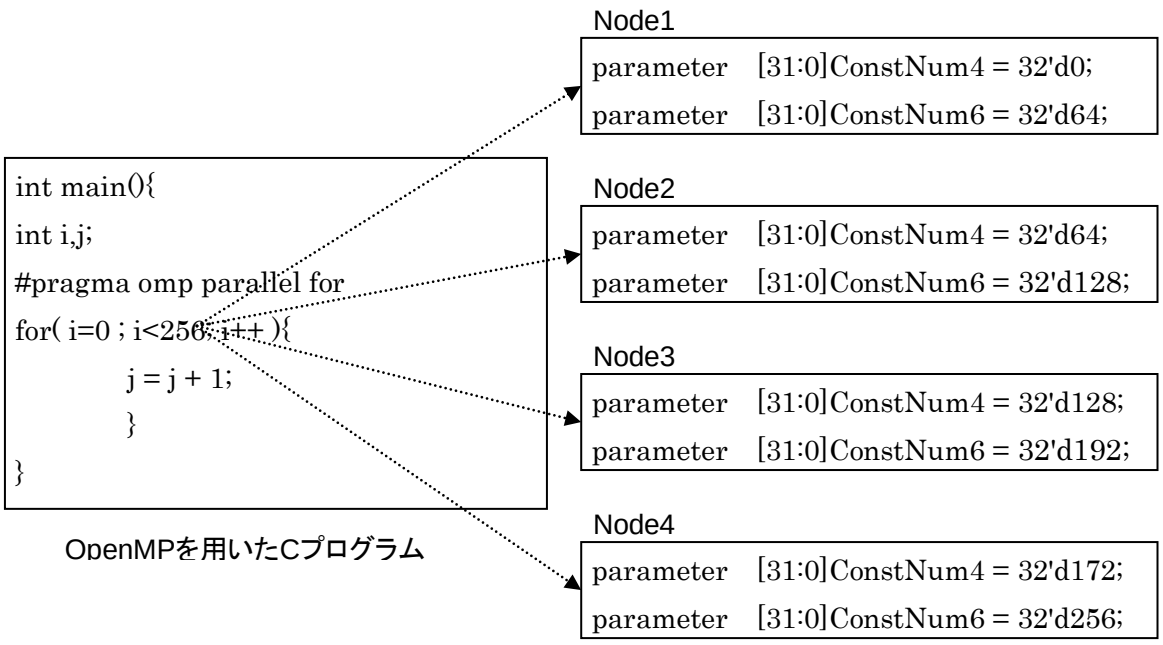

図 6：データ分割による並列化

3. SHA-1 の OpenMP と Verilog-HDL による記述

3.1 SHA-1 のアルゴリズム

SHA-1 は任意の長さの入力メッセージから 160 ビットの疑似乱数（ハッシュ値）を生成するハッシュ関数の一種である．そのアルゴリズムは，まず任意の長さの入力メッセージに「0」を付加し，512 ビットの倍数長に整える（パディング）．パディングしたメッセージを 32 ビットずつ 16 個のブロック W[0]，W[1]，…，W[15]に分けて，W[16]，W[17]，…，W[79]までの W を以下の式で生成する．

W[t] = Rotate_Left(1bit，(W[t-3] ^ W[t-8] ^ W[t-14] ^ W[t-16]))　　16≦t≦79

次に図 7 のように，ハッシュ H0，H1，H2，H3，H4 を決められた値に初期化し，ハッシュ生成に用いる定数配列 K[t]を用意する．

今までに用意した W[t]，H，K[t]を用いてハッシュ生成を行う．ハッシュ値の計算を行う圧縮関数は，図 8 の処理を 80 回行う．なお，図 8 中の <<<n は左方向への n ビット回転シフト（ローテーション）であり，ft は図 7 の式で表わされる原始関数である．[9]

H0 = A = 0x67452301
H1 = B = 0xEFCDAB89
H2 = C = 0x98BADCFE
H3 = D = 0x10325476
H4 = E = 0xC3D2E1F0

ハッシュ H0~H4 の初期化

K[0]~K[19] = 0x5A827999
K[20]~K[39] = 0x6ED9EBA1
K[40]~K[59] = 0x8F1BBCDC
K[60]~K[79] = 0xCA62C1D6

定数配列 K[t]

0≦t≦19：ft(B, C, D) = (B & C) | (~B & D)
0≦t≦19：ft(B, C, D) = B ^ C ^ D
0≦t≦19：ft(B, C, D) = (B & C) | (B & D) | (C & D)
0≦t≦19：ft(B, C, D) = B ^ C ^ D

原始関数 ft

図 7：SHA-1 のハッシュ生成に用いる配列・関数

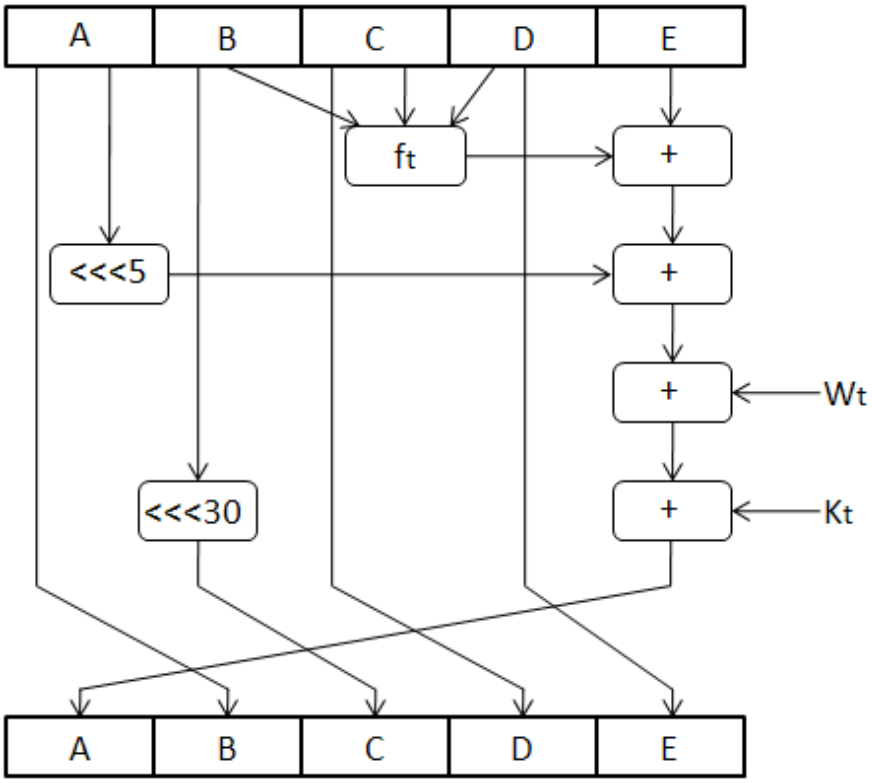

図 8：SHA-1 の圧縮関数のアルゴリズム

図 8 の演算処理を 80 回行った後，最後に出力結果 A，B，C，D，E に初期値 H0，H1，H2，H3，H4 を加算する．

H0 = H0 + A

H1 = H1 + B

H2 = H2 + C

H3 = H3 + D

H4 = H4 + E

これでハッシュ生成が終了する．出力は H0，H1，H2，H3，H4 の上位バイトから順に 16 進数で行う．

3.2 OpenMP による記述

OpenMP は逐次プログラムにプラグマを追加することにより並列化の範囲を指示することができるが，ハッシュ関数はデータ依存性があるため，繰り返し処理を並列化する並列リージョンは使用できない．そのため，複数の異なる処理を並列化する並列セクションを用いて記述する．

図 9 に SHA-1 圧縮関数の OpenMP プログラムを示す。なお，図 9 中の Rotate(A, 5) は A の 5 ビット左ローテーションを表している．

```
for (t=0;  t<80;  t++){
#pragma omp parallel sections
{
#pragma omp section
    buf = E + W[t] + K(t);
#pragma omp section
    tmp = Rotate(A, 5) + f(t, B, C, D);
}
TEMP = buf + tmp;

E = D;
D = C;
C = Rotate(B, 30);
B = A;
A = TEMP;
}
```

図 9：SHA-1 圧縮関数の OpenMP プログラム

図 9 のように，同時に処理できる演算だけを section 文を用いて並列化を行っている．並列化が可能な演算は 2 つしかないが，80 回同様の処理を繰り返すため，最終的に 80 クロック削減することができ，並列効果が望めるはずである．そのため，ハッシュ関数のようなデータ依存性のあるプログラムでも OpenMP での記述，及び OpenMP ハードウェア動作合成システムでの検証が可能である．

ところが，トランスレータではデータの型が int 型にしか対応していないため，必然的に OpenMP プログラムのデータも int 型に統一しなければならない．SHA-1 では t 以外のほとんどの値を 32 ビットで表わしており，決められた初期値 A，B，C，D，E が int 型の扱える範囲を超えてしまうため（初期値の変更は不可能），図 9 のプログラムのままではトランスレータを通すことができないという問題が発生する．

そこで，int 型で対応できないすべての値を 16 ビットずつ 2 つに分けて計算を行うという方法を用いた．例えば，A と B の上位 16 ビットをそれぞれ A1，B1，下位 16 ビットをそれぞれ A2，B2 で表わし，A + B の計算は上位ビット同士の加算(A1 + B1)と下位ビット同士の加算(A2 + B2)を別々に行い，出力も別々に行うという方法である．変更後のプログラムの概要を図 10 に示す．

この方法を用いるにあたり，下位ビットのキャリーを上位ビットに加算するという処理を追加する必要がある．この処理は図 10 では Carry 関数で表記している．例えば，A2 のキャリーを A1 に加算する処理は Carry (A1 , A2)で表記している．

```
int    Carry( int X, int Y ){
    Y = Y >> 16;
    return    X = X + Y;
}
            .
            .

for (t=0;   t<80;   t++){
#pragma omp parallel sections
{
#pragma omp section
    buf1 = E1 + W1[t] + K1(t);      //buf の上位ビット
#pragma omp section
    buf2 = E2 + W2[t] + K2(t);      //buf の下位ビット
#pragma omp section
    tmp1 = Rotate(A1, 5) + f(t, B1, C1, D1);   //tmp の上位ビット
#pragma omp section
    tmp2 = Rotate(A2, 5) + f(t, B2, C2, D2);   //tmp の下位ビット
}
TEMP1 = Carry(buf1 , buf2) + Carry(tmp1 , tmp2);
TEMP2 = buf2 + tmp2;

E1 = D1;
E2 = D2;
D1 = C1;
D2 = C2;
C1 = Rotate(B1, 30);
C2 = Rotate(B2, 30);
B1 = A1;
B2 = A2;
A1 = TEMP1 & 0xFFFF;     // & 0xFFFF は上位 16 ビットを 0 にするため
A2 = TEMP2 & 0xFFFF;
}
```

図 10：トランスレータに対応した SHA-1 圧縮関数の OpenMP プログラム

図 9 に比べて図 10 は処理が倍に増えてしまっているが，section 文で同時に実行できる命令数も増加したため，並列効果が上がるというメリットもある．

以上のような変更ですべてを int 型で表わすことができるため，トランスレータへの対応が可能である．

3.3 Verilog-HDL による記述

Verilog-HDL による記述では組み合わせ回路記述と順序回路記述の 2 つに分けられる．

組み合わせ回路記述では，まず OpenMP と同様にハッシュ H0~H4 を初期化し，assign 文を用いて図 11 のような継続的に実行される代入文を記述する．なお，ハッシュの初期化では，H0~H4 をパラメータとして宣言する．

```
assign   Aout = A + H0;
assign   Bout = B + H1;
assign   Cout = C + H2;
assign   Dout = D + H3;
assign   Eout = E + H4;
```

A～E はすべてレジスタであり，
Aout～Eout は出力ポートである．

図 11：継続的代入文の記述

図 11 の演算は，圧縮関数で求めたハッシュ値が更新されるたびに，出力 Aout～Eout の書き替えができるようにするために，assign 文を用いて継続的に代入している．

次に，図 8 中の配列 Kt や関数 ft といった assign 文では記述できない複雑な論理（回路規模ではない）を function で記述する．function を用いた例を図 12 に示す．

```
function [31:0] K;
    input [7:0] t;
        if(t >= 8'd00 && t <= 8'd19) begin
            K = 32'h5a827999;
        end
        else   begin
            K = 32'hefcdab89;
        end
endfunction
```

図 12：function の記入例

順序回路の記述では always を用いてタイミングごとに異なる命令を記述する．今回の SHA-1 のプログラムでは，3 つのステートに分けて記述した．

1 つ目のステートでは，リセットが 0 のときにすべてのレジスタの値を初期化し，2 つ目のステートで，テストベンチで記述した入力メッセージを呼び出してレジスタに格納している．そして 3 つ目のステートで，図 8 の圧縮関数の演算処理を記述している。3 つ目のステートの処理を図 13 に示す．

```
if(t < 80) begin
    buf = E + W[t] + K(t);
    tmp = { A[26:0], A[31:27] } + f(t, B, C, D);

    TEMP = buf + tmp;
    E = D;
    D = C;
    C = { B[1:0], B[31:2] };
    B = A;
    A = TEMP;
    t = t + 1;
end
```

図 13：SHA-1 圧縮関数の Verilog-HDL 記述

以上のような手順で SHA-1 を記述している．

図 13 は図 9 と同じような構造になっているが，Verilog-HDL での記述は，組み合わせ回路と順序回路に分かれていることやテストベンチが必要であるということを考えても，前述の OpenMP と比較すると抽象度が非常に低く，記述が困難であるといえる．

3.4 記述したプログラムの動作検証

OpenMP ハードウェア動作合成システムを検証するに当たって，作成した OpenMP と Verilog-HDL プログラムの動作を事前に検証する必要がある．まず，実験環境を表 1 に，OpenMP プログラムの実験結果を表 2 に示す．なお，表 2 では OpenMP プログラムをトランスレータ用に変更する前と後の両方のプログラムにおける実行時間を記述している．

表 1：実験環境

アルゴリズム評価	SMP 環境	Quad Xeon 3.0Ghz,Memory 4GB
	OpenMP コンパイラ	Intel コンパイラ 9.1.038
ハードウェア動作合成	論理合成ツール	Xilinx ISE 11
回路シミュレーション	PC 環境	Intel Core2 duo 2.66GHz,Memory 3GB
	シミュレーションツール	ModelSim XE 6.2g

表 2：SHA-1 の OpenMP プログラムの実行時間

スレッド数		1	2	4
実行時間 (ms)	変更前（トランスレータ非対応)	0.027	3.28	7.91
	変更後（トランスレータ対応)	0.040	3.32	9.20

表 2 のように，スレッド数を増やしても実行時間が短くなるどころか，逆に長くなってしまう結果になった．これは，逐次で実行してもほとんど時間がかからないプログラムは，スレッド数を増やすことによって通信時間が大きくなるためだと思われる．

しかし，変更前より命令数が約 2 倍に増えた変更後のプログラムの実行時間は，変更前の実行時間と比較してもあまり大きくなっていない．特に，スレッド数が 2 の場合はほぼ同じ時間で実行できていることがわかる．これは，変更後のプログラム（図 9）は変更前のプログラム（図 10）よりも同時に実行できる命令数が増えたためだと思われる．

次に，Verilog-HDL プログラムの動作検証を行う．この実験では，まずシミュレーションツール(ModelSim)を用いて回路シミュレーションを行い,その後,論理合成ツール(ISE)でハードウェア動作合成を行い，実行時間やクロック数などを求めている．実験結果を表 3 に示す.

表 3：SHA-1 の Verilog-HDL の実験結果

実行時間(ms)	クロック数(clocks)	周波数(MHz)	回路規模(slices)
4.49	163	36.249	7249

4. MD5 の OpenMP と Verilog-HDL による記述

4.1 MD5 のアルゴリズム

MD5 のアルゴリズムについて説明する．MD5 も SHA-1 と同じハッシュ関数の一種であるが，SHA-1 との相違点として，出力されるハッシュ値が 128 ビットであることや圧縮関数での処理の違いなどが挙げられる．

MD5 も SHA-1 と同様に，入力された任意長のメッセージを 512 ビットの倍数長にパディングし，パディングしたメッセージを 32 ビットずつ 16 個の配列 X[0…15]に分けて格納する．

SHA-1 とは違い，MD5 はハッシュが 4 つしかなく，定数配列の求め方や原始関数の処理も異なる．定数配列 T[i]は，i 番目の要素を

$$T[i] = 2^{32} * \mathrm{abs}(\sin(i)) \quad (\mathrm{abs}\cdots\text{絶対値})$$

で表わす．

MD5 のハッシュ生成で用いる配列や関数を図 14 に，圧縮関数の処理内容を図 15 に示す．MD5 では圧縮関数の処理を 64 回繰り返すが，それを 16 回ずつ 4 ラウンドに分けて，1 ラウンドごとに入力メッセージ X[i]を異なる順番で使うという特徴がある．上記のようにパディングした 16 個の配列 X[0…15]を図 14 の k(i)の順番で用いる．1 つのラウンドが 16 ステートなので，16 個の配列をラウンドごとに 1 つずつ使用するということになる．

H0 = A = 0x67452301
H1 = B = 0xEFCDAB89
H2 = C = 0x98BADCFE
H3 = D = 0x10325476

ハッシュ H0~H4 の初期化

1 ラウンド目: k(i) = i
2 ラウンド目: k(i) = (1+5i) mod16
3 ラウンド目: k(i) = (5+3i) mod16
4 ラウンド目: k(i) = 7i mod16

入力 X を用いる順序定数 k(i)

0≦t≦19 : ft(B, C, D) = (B & C) | (~B & D)
0≦t≦19 : ft(B, C, D) = (B & D) | (C & ~D)
0≦t≦19 : ft(B, C, D) = B ^ C ^ D
0≦t≦19 : ft(B, C, D) = B ^ (C | D)

原始関数 g

図 14：MD5 のハッシュ生成に用いる配列・関数

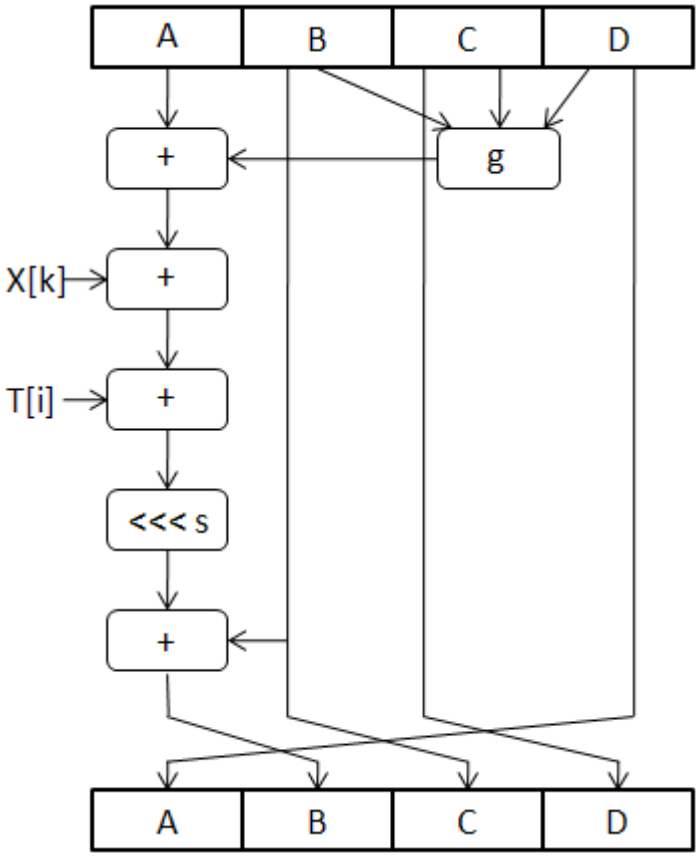

図 15：MD5 の圧縮関数のアルゴリズム

図 15 の演算処理を 64 回行い，SHA-1 と同様に出力結果 A，B，C，D に初期値 H0，H1，H2，H3 を加算する．

H0 = H0 + A
H1 = H1 + B
H2 = H2 + C
H3 = H3 + D

なお，図 15 のように MD5 は左ローテートのビット数が s という定数になっているが，この s は 64 回の演算ごとに割り振られている値である．

4.2 OpenMP による記述

OpenMP による MD5 の記述方法について説明する．MD5 もデータ依存性があるため，SHA-1 と同様に section 文を用いて記述する．このとき，MD5 のハッシュ初期値も int 型の扱える範囲を超えてしまうため，上位 16 ビット・下位 16 ビットに分けて演算を行う手法を用いなければならない．

MD5 は定数配列 T[i] = 2^{32} ＊ abs(sin(i))を求める処理がやや複雑になる．sin 関数は int 型では整数部分しか出力されないが，2^{32} を掛けると 32 ビット左へシフトすることになるので，精度の高い値は得られる．しかし，2^{32} を掛けると int 型の範囲を超えてしまう．そこで，まず 2^{32} の一部を sin に掛け，値を上位 16 ビット・下位 16 ビットに分けた後で 2^{32} の残りを掛けることにした．

sin 関数の精度をできるだけ高くするために，今回は 2^{30} を掛けた後で配列 t1[i]・t2[i]に格納している（2^{30} はオーバーフローを防ぐことのできる最大値）．最後に残りの 2^{2} を掛けるが，すでに上位ビット・下位ビットに分けた後なのでオーバーフローもなく，2 ビット左にシフトするだけなので記述も容易である．

図 15 の圧縮関数のプログラムを図 16 に示す．なお，Carry 関数は図 10 と同じである．

```
for (i=0;  i<64;  i++){
#pragma omp parallel sections
{
#pragma omp section
    buf1 = A1 + g( i, B1, C1, D1 );      //buf の上位ビット
#pragma omp section
    buf2 = A2 + g( i, B2, C2, D2 );      //buf の下位ビット
#pragma omp section
    tmp1 = x1[ k(i) ] + t1[i];      //tmp の上位ビット
#pragma omp section
    tmp2 = x2[ k(i) ] + t2[i];      //tmp の下位ビット
}
TEMP1 = Carry( ( buf1 + tmp1) , ( buf2 + tmp2) );
TEMP2 = buf2 + tmp2;

TEMP1 = Rotate( TEMP1, s[i] ) + B1;
TEMP2 = Rotate( TEMP2, s[i] ) + B2;
TEMP1 = Carry( TEMP1, TEMP2 );

A1 = D1;
A2 = D2;
D1 = C1;
D2 = C2;
C1 = B1;
C2 = B2;
B1 = TEMP1 & 0xFFFF;    // & 0xFFFF は上位 16 ビットを 0 にするため
B2 = TEMP2 & 0xFFFF;
}
```

図 16：トランスレータに対応した MD5 圧縮関数の OpenMP プログラム

4.3 Verilog-HDL による記述

Verilog-HDL による MD5 の記述方法について説明する. MD5 の場合も SHA-1 と同様に組み合わせ回路記述と順序回路記述の 2 つに分かれており，組み合わせ回路記述で継続的に実行される代入文や簡単な関数を記述し，順序回路記述で圧縮関数などの記述を行う.

SHA-1 と大きく異なる点は，sin 関数を使用していることである．Verilog-HDL では sin などの三角関数は扱えないので，2^{32} ＊ abs(sin(i))の計算結果を順次配列に格納するプログラムは作成できない．したがって，0≦ i <64 における計算結果を表したファイルを特別に用意する必要がある.

このファイルでは

assign T[0] = 32'hD76AA478;

assign T[1] = 32'hE8C7B756;

assign T[2] = ・・・

のように合計 64 個の配列すべてを記述しており, 非常に手間のかかるプログラムになった.

図 15 の圧縮関数の演算処理を図 17 に示す.

```
if(t < 64) begin
    buf = A + g(t, B, C, D);
    tmp = X[ k(t) ] + T;

    TEMP = buf + tmp;
    TEMP = Rotate( TEMP, s(t) ) + B;
    A = D;
    D = C;
    C = B;
    B = TEMP;
    t = t + 1;
end
```

図 17：SHA-1 圧縮関数の Verilog-HDL 記述

なお, 配列 T[i]の値を格納したファイルはインスタンスとして呼び出して実行しているが, このファイルにもコンパイルは必要であり，テストベンチも含めて 3 つのプログラムを作成するにはかなりの時間を要する.

4.4 記述した回路の動作検証

MD5 の OpenMP と Verilog-HDL プログラムの動作を検証する．まず，表 4 に OpenMP プログラムの実行結果を示す．ここでも，トランスレータ用に変更する前と後の両方のプログラムにおける実行時間を記述している．

表 4：MD5 の OpenMP プログラムの実行時間

スレッド数		1	2	4
実行時間 (ms)	変更前（トランスレータ非対応）	0.022	2.69	6.23
	変更後（トランスレータ対応）	0.033	2.74	7.33

MD5 も SHA-1 と同様に並列効果は出なかったが，トランスレータ用にプログラムを変更しても実行時間はあまり大きくならないことがわかる．

次に，Verilog-HDL プログラムの実験結果を表 5 に示す．

表 5：MD5 の Verilog-HDL プログラムの実験結果

実行時間(ms)	クロック数(clocks)	周波数(MHz)	回路規模(slices)
2.84	131	46.040	885

表 3 と比較すると，MD5 では回路規模が非常に少ないということがわかる．これは，圧縮関数において，MD5 は SHA-1 よりループの回数や演算に用いるハッシュが少ないためだと思われる．

5.コードジェネレータ改良方法の検討

5.1 コードジェネレータの問題点

トランスレータは int 型以外に対応していないものの，section 文に対応しているためハッシュ関数のようなデータ依存性のある OpenMP プログラムも通すことができた. しかし，コードジェネレータには section 文に対応する記述が全くないないため，生成した中間表現を Verilog-HDL に変換することができなかった.

コードジェネレータには並列化情報解析を行うモジュールがあるが，このモジュールでは pragma parallel for で指定された for ループの定数を変換する機能しか持っていない. 具体的に言うと，中間表現の /Parallel FOR/ という記述が読み込まれたときだけ解析を行うモジュールになっている．したがって，pragma parallel section で記述した部分の解析を行うことはできないという問題点がある.

section 文の解析を行えるようにするためには，並列化情報解析モジュールに /Parallel SECTIONS/ が読み込まれた場合の解析方法を記述する必要がある.

5.2 コードジェネレータの改良案

コードジェネレータの改良案について説明する．section 文を用いた場合に出力される中間表現の例を図 18 に示す.

2.2 章で述べたとおり，中間表現はまず状態遷移表の#0 から処理が始まる．図 18 において，#0 でシンボルテーブルの 4 と 5 の処理を行った後，#1 で/Parallel SECTIONS/に遷移している．この部分の解析を行うプログラムを記述する必要がある.

解析方法としては，/Parallel SECTIONS/ に出力されたシンボルテーブルの値を検出し，その値をシンボルテーブル配列に格納するという作業だけで可能である. なぜなら, section 文には for 文のような条件分岐がないため，状態遷移配列の値は状態遷移表から読み取った値をそのまま格納するだけで対応できる．最後に，シンボルテーブル配列と状態遷移配列の値を出力し，他のモジュールに受け渡している.

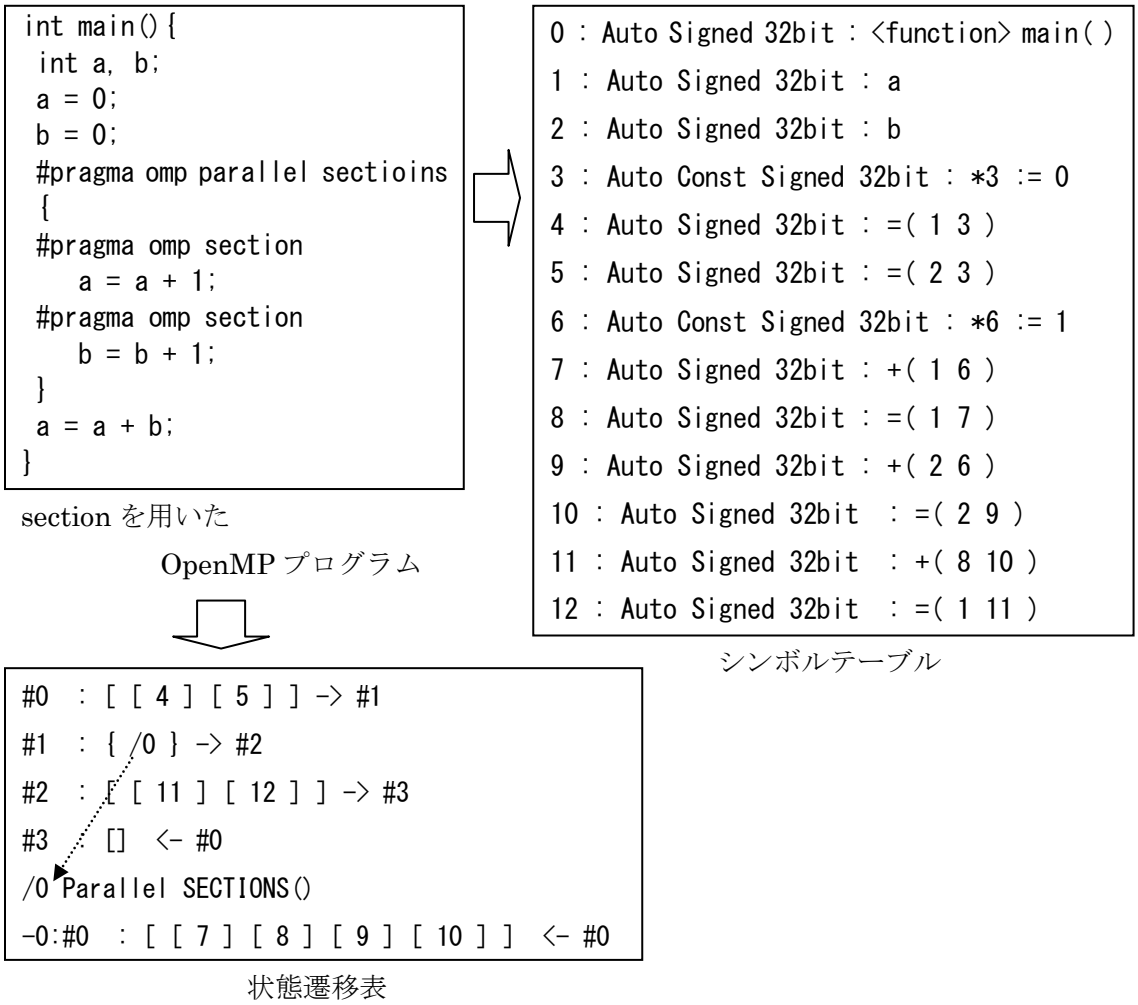

図 18：section 文を用いたときの中間表現のサンプル

ただ，問題はシンボルテーブル配列に格納する値が膨大になるということである．今回作成したハッシュ関数の中間表現では，シンボルテーブルの出力だけで 300 行を軽く超えるという結果になった．これは，ハッシュ関数ではレジスタや定数などが多いためだと思われる．コードジェネレータでもトランスレータと同様にレジスタや定数を一つずつシンボルテーブル配列に格納しているため，Verilog-HDL を生成すると回路規模が膨大になる恐れがある．

この課題の解決策として，並列化情報を解析する際に多次元配列の構造を用いる方法が有効であると思われる．これは section 文に限ったことではなく，for 文の場合も有効である．現在のコードジェネレータは一次元配列にしか対応しておらず，結果的に OpenMP での記述にも制限が生じる．

図 19 に多次元配列の例を示す．多次元配列を用いて解析を行う場合は，その他の解析情報の配列を受け取るモジュールへの変更が必要になるなど課題もあるが，中間表現が膨大になってしまう以上，中間表現を解析するモジュールの多次元化は一つの改良案として挙げられる．

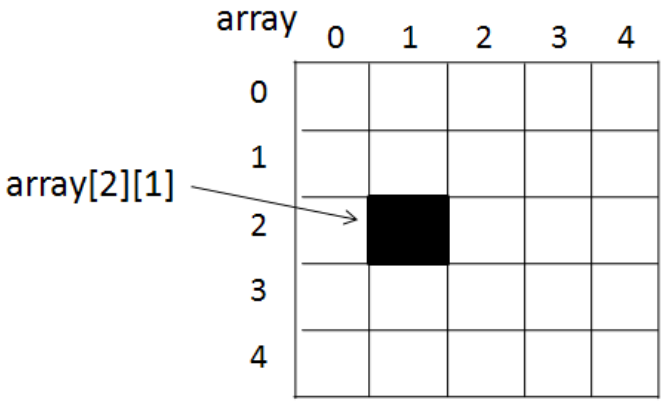

図 19：多次元配列の例

5.3 考察

OpenMP を用いたハードウェア動作合成システムの問題点の検出と対処方法の検討を行った．まずトランスレータに関しては，ビット数の非常に大きな値や浮動小数値を扱うことができず OpenMP での記述に制約があることが問題点として挙げられる. 対応策として，トランスレータで扱えないデータを上位ビット・下位ビットに分けて演算を行うという手法で記述した．

また，コードジェネレータに関しては情報解析を行う際，一次元配列やレジスタのみを用いているため，多次元配列を用いた OpenMP 記述には対応できないと思われる．また，レジスタや定数の多いハッシュ関数のようなプログラムでは，回路規模が膨大になる恐れがある．こういった課題を解決する上で，多次元配列の構造を用いて並列化情報を解析する方法は有効であると思われる．

6. おわりに

ハッシュ関数を OpenMP と Verilog-HDL で記述し，手書きの回路と OpenMP を用いたハードウェア動作合成システムにより生成した回路の比較を試みたが，トランスレータとコードジェネレータに問題点が見つかったため，その対応策を検討した．

トランスレータに関しては，int 型にしか対応していなかったため，それ以外の型のデータを上位ビット・下位ビットに分けるという対処法を用いた．しかし，生成された中間コードに誤りがあり，手書きで修正する必要があるなど課題が残った．

コードジェネレータに関しては，section 文に対応する記述が欠けており，Verilog-HDL のコードを生成することができなかった．並列化情報解析モジュールの改良が必要である．また，多次元配列を用いての解析機能拡張・回路規模の縮小も今後の目標といえる．

謝辞

本研究の機会を与えてくださり，貴重な助言，ご指導を頂きました山崎勝弘教授に深く感謝いたします．また，本動作合成システムを立ち上げ，事あるごとに相談に乗って頂き，貴重な助言を頂いた中谷嵩之氏に深く感謝いたします．

最後に，共同研究者である住井大介氏をはじめ高性能計算研究室の皆様に心より感謝いたします．

参考文献

[1] 中谷嵩之：OpenMP によるハードウェア動作合成システムの設計と検証，立命館大学大学院理工学研究科修士論文，2007.

[2] 松崎裕樹：OpenMP によるハードウェア動作合成システム　コードジェネレータの実装と画像処理による評価，立命館大学大学院理工学研究科修士論文，2008.

[3] 金森史樹：OpenMP ハードウェア動作合成システムの検証（Ⅰ），立命館大学理工学部電子情報デザイン学科卒業論文，2009

[4] 苅屋徹：OpenMP ハードウェア動作合成システムの検証（Ⅱ），立命館大学理工学部電子情報デザイン学科卒業論文，2009

[5] 中谷嵩之，松崎裕樹，山崎勝弘，“OpenMP によるハードウェア動作合成システムの設計と検証”，FIT2007，C-006，2007.

[6] 松崎裕樹，中谷嵩之，山崎勝弘，“OpenMP によるハードウェア動作合成システム：コードジェネレータの実装と画像処理による評価”，FIT2008，C-008，2008.

[7] 田口景介：実習 C 言語　新装版，アスキー，2002

[8] 安田絹子，小林林広，飯塚博道，阿部貴之，青柳信吾：マルチコア CPU のための並列プログラミング，秀和システム，2006

[9] ウイリアム・スターリングス著，石橋啓一郎，三川荘子，福田剛士訳：暗号とネットワークセキュリティー　理論と実際，ピアソン・エデュケーション，2001

[10] 北山洋幸：OpenMP 入門　マルチコア CPU 時代の並列プログラミング，秀和システム，2009

付録 A　SHA-1 の OpenMP プログラム（トランスレータ対応）

```
#include<stdio.h>
#include<string.h>
#include<sys/time.h>
#include<omp.h>

double second()
{
      struct timeval tv;
            gettimeofday(&tv, NULL);
            return    tv.tv_sec + tv.tv_usec / 1000000.0;
}

int K1(int t){
            if(0 <= t && t < 20)
                        return    0x5A82;
            else if(20 <= t && t < 40)
                        return    0x6ED9;
            else if(40 <= t && t < 60)
                        return    0x8F1B;
            else
                        return    0xCA62;
}

int K2(int t){
            if(0 <= t && t < 20)
                        return    0x7999;
            else if(20 <= t && t < 40)
                        return    0xEBA1;
            else if(40 <= t && t < 60)
                        return    0xBCDC;
            else
                        return    0xC1D6;
}

int f(int t, int B, int C, int D){
            if(0 <= t && t < 20)
                        return (B & C) | (~B & D);
            else if(20 <= t && t < 40)
                        return    (B ^ C ^ D) & 0xFFFF;
            else if(40 <= t && t < 60)
                        return    (B & C) | (B & D) | (C & D);
            else
                        return    (B ^ C ^ D) & 0xFFFF;
}

int S(int x, int n){
            return    x = ( x << n ) + ( x >> (32 - n) );
}

int Carry(int X, int Y){
            Y = Y >> 16;
            return    X = X + Y;
}

int Rotate(int X, int Y, int n){
            if(n<16){
                        Y = Y >> (16 - n);
```

```
                        X = X << n;
                        X = X & 0xFFFF;
                        return    X = X + Y;
                }
                else{
                        Y = Y << (n - 16);
                        Y = Y & 0xFFFF;
                        X = X >> (32 - n);
                        return X = X + Y;
                }
}

int main()
{
                int t;
                int A1 = 0x6745;
                int A2 = 0x2301;
                int B1 = 0xEFCD;
                int B2 = 0xAB89;
                int C1 = 0x98BA;
                int C2 = 0xDCFE;
                int D1 = 0x1032;
                int D2 = 0x5476;
                int E1 = 0xC3D2;
                int E2 = 0xE1F0;
                int H1[5] = {A1, B1, C1, D1, E1};
                int H2[5] = {A2, B2, C2, D2, E2};
                int W[80], w1[80], w2[80];
                int TEMP1, TEMP2, tmp1, tmp2, buf1, buf2;
                double time, start;

                W[0]    = 0x0;
                W[1]    = 0x1;
                W[2]    = 0x2;
                W[3]    = 0x3;
                W[4]    = 0x4;
                W[5]    = 0x5;
                W[6]    = 0x6;
                W[7]    = 0x7;
                W[8]    = 0x8;
                W[9]    = 0x9;
                W[10] = 0xA;
                W[11] = 0xB;
                W[12] = 0xC;
                W[13] = 0xD;
                W[14] = 0xE;
                W[15] = 0xF;

                for (t=16;    t<80;    t++) {
                W[t] = S( (W[t-3] ^ W[t-8] ^ W[t-14] ^ W[t-16]) , 1);
        }

        for(t=0;    t<80;    t++){
                w1[t] = W[t] >> 16;
                w2[t] = W[t] & 0xFFFF;
        }

        start = second();

        for (t=0;    t<80;    t++){
```

```
        #pragma omp parallel sections
        {
        #pragma omp section
            buf1 = E1 + w1[t] + K1(t);
        #pragma omp section
            buf2 = E2 + w2[t] + K2(t);
        #pragma omp section
            tmp1 = Rotate(A1, A2, 5) + f(t,B1,C1,D1);
        #pragma omp section
            tmp2 = Rotate(A2, A1, 5) + f(t,B2,C2,D2);
        }

        TEMP1 = Carry(buf1, tmp1) + Carry(buf2, buf2);
        TEMP2 = (buf2 & 0xFFFF) + (tmp2 & 0xFFFF);

        TEMP1 = Carry(TEMP1, TEMP2) & 0xFFFF;
        TEMP2 = TEMP2 & 0xFFFF;

        E1 = D1;
        E2 = D2;
        D1 = C1;
        D2 = C2;
        C1 = Rotate(B1, B2, 30);
        C2 = Rotate(B2, B1, 30);
        B1 = A1;
        B2 = A2;
        A1 = TEMP1;
        A2 = TEMP2;
    }
    time = second() - start;

        H1[0] = Carry( (H1[0] + A1) , (H2[0] + A2) ) & 0xFFFF;
        H1[1] = Carry( (H1[1] + A1) , (H2[1] + A2) ) & 0xFFFF;
        H1[2] = Carry( (H1[2] + A1) , (H2[2] + A2) ) & 0xFFFF;
        H1[3] = Carry( (H1[3] + A1) , (H2[3] + A2) ) & 0xFFFF;
        H1[4] = Carry( (H1[4] + A1) , (H2[4] + A2) ) & 0xFFFF;

        H2[0] = H2[0] & 0xFFFF;
        H2[1] = H2[1] & 0xFFFF;
        H2[2] = H2[2] & 0xFFFF;
        H2[3] = H2[3] & 0xFFFF;
        H2[4] = H2[4] & 0xFFFF;

    printf("%04X%04X %04X%04X %04X%04X %04X%04X %04X%04X¥n", H1[0],
                                H2[0], H1[1], H2[1], H1[2], H2[2], H1[3], H2[3], H1[4], H2[4]);
    printf("time= %lf ¥n", time);

    return 0;
}
```

付録 B　SHA-1 の Verilog-HDL プログラム

```
module sha_test( clk, xrst, Y, Aout, Bout, Cout, Dout, Eout, ef );

input                   clk, xrst;
input       [511:0]     Y;
output [31:0] Aout, Bout, Cout, Dout, Eout;
output      ef;
reg     ef;
reg         [1:0]       state;
reg         [7:0]       count;
reg         [31:0]      W [79:0];
reg         [31:0]      A, B, C, D, E, tmpW, TEMP, buffer1, buffer2;

parameter               H0 = 32'h67452301;
parameter               H1 = 32'hefcdab89;
parameter               H2 = 32'h98badcfe;
parameter               H3 = 32'h10325476;
parameter               H4 = 32'hc3d2e1f0;

assign Aout = A + H0;
assign Bout = B + H1;
assign Cout = C + H2;
assign Dout = D + H3;
assign Eout = E + H4;

function [31:0] K;
      input [7:0] count;
                        if(count <= 8'd19 && count >= 8'd00)begin
                                K = 32'h5a827999;
                        end
                        else if(count <= 8'd39 && count >= 8'd20)begin
                                K = 32'h6ed9eba1;
                        end
                        else if(count <= 8'd59 && count >= 8'd40)begin
                                K = 32'h8f1bbcdc;
                        end
                        else begin
                                K = 32'hca62c1d6;
                        end
endfunction

function [31:0] F;
      input [7:0] count;
      input [31:0] B, C, D;
            if(count <= 8'd19 && count >= 8'd0)begin
                                F = (B & C) | (~B & D);
                        end
                        else if(count <= 8'd39 && count >= 8'd20)begin
                                F = B ^ C ^ D;
                        end
                        else if(count <= 8'd59 && count >= 8'd40)begin
                                F = (B & C) | (B & D) | (C & D);
                        end
                        else begin
                                F = B ^ C ^ D;
                        end
endfunction
```

```
always @(posedge clk or negedge xrst)
begin
	if(!xrst) begin
		tmpW = 0;
		TEMP = 0;
		W[0][31:0] = 0;
		W[1][31:0] = 0;
		W[2][31:0] = 0;
		W[3][31:0] = 0;
		W[4][31:0] = 0;
		W[5][31:0] = 0;
		W[6][31:0] = 0;
		W[7][31:0] = 0;
		W[8][31:0] = 0;
		W[9][31:0] = 0;
		W[10][31:0] = 0;
		W[11][31:0] = 0;
		W[12][31:0] = 0;
		W[13][31:0] = 0;
		W[14][31:0] = 0;
		W[15][31:0] = 0;
		A = 0;
		B = 0;
		C = 0;
		D = 0;
		E = 0;
		state = 0;
		ef = 1;
		count = 0;
		buffer1 = 0;
		buffer2 = 0;
	end

	else if(state == 0) begin
		W[0][31:0] = Y[511:480];
		W[1][31:0] = Y[479:448];
		W[2][31:0] = Y[447:416];
		W[3][31:0] = Y[415:384];
		W[4][31:0] = Y[383:352];
		W[5][31:0] = Y[351:320];
		W[6][31:0] = Y[319:288];
		W[7][31:0] = Y[287:256];
		W[8][31:0] = Y[255:224];
		W[9][31:0] = Y[223:192];
		W[10][31:0] = Y[191:160];
		W[11][31:0] = Y[159:128];
		W[12][31:0] = Y[127:96];
		W[13][31:0] = Y[95:64];
		W[14][31:0] = Y[63:32];
		W[15][31:0] = Y[31:0];
		A = H0;
		B = H1;
		C = H2;
		D = H3;
		E = H4;
		state = 1;
		ef = 1;
	end

	else if(state == 1) begin
```

```
            if(count < 80) begin
                if(count > 15) begin
                        tmpW = ( W[count-16] ^ W[count-14] ^ W[count-8] ^ W[count-3] );
                        W[count] = { tmpW[30:0], tmpW[31] };
                end

                buffer1 = E + W[count] + K(count);
                buffer2 = { A[26:0], A[31:27] } + F(count, B, C, D);

                TEMP = buffer1 + buffer2;
                E = D;
                D = C;
                C = { B[1:0], B[31:2] };
                B = A;
                A = TEMP;
                count = count + 1;
            end

            else begin
                ef = 0;
                state = 0;
            end
        end
end
endmodule
```

付録 C　MD5 の OpenMP プログラム（トランスレータ対応）

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<sys/time.h>
#include<omp.h>

double second()
{
      struct timeval tv;
            gettimeofday(&tv, NULL);
            return    tv.tv_sec + tv.tv_usec / 1000000.0;
}

int g(int t, int B, int C, int D){
            if(0 <= t && t < 16)
                        return (B & C) | (~B & D);
            else if(16 <= t && t < 32)
                        return    (B & D) | (C & ~D);
            else if(32 <= t && t < 48)
                        return    (B ^ C ^ D) & 0xFFFF;
            else
                        return    C ^ (B | ~D) & 0xFFFF;
}

int k(int i){
            if(0 <= i && i < 16)
                        return i;
            else if(16 <= i && i < 32)
```

```
                return   (1 + 5*i) % 16;
        else if(32 <= i && i < 48)
                return   (5 + 3*i) % 16;
        else
                return   (7*i) % 16;
}

int Carry(int X, int Y){
        Y = Y >> 16;
        return   X = X + Y;
}

int Rotate(int X, int Y, int n){
        if(n<16){
                Y = Y >> (16 - n);
                X = X << n;
                X = X & 0xFFFF;
                return   X = X + Y;
        }
        else{
                Y = Y << (n - 16);
                Y = Y & 0xFFFF;
                X = X >> (32 - n);
                return X = X + Y;
        }
}

int main()
{
        int i, n;
        int A1 = 0x6745;
        int A2 = 0x2301;
        int B1 = 0xEFCD;
        int B2 = 0xAB89;
        int C1 = 0x98BA;
        int C2 = 0xDCFE;
        int D1 = 0x1032;
        int D2 = 0x5476;
        int s[64];
        int X[16], x1[64], x2[64];
        int T[64], t1[64], t2[64];
        int H1[4] = {A1, B1, C1, D1};
        int H2[4] = {A2, B2, C2, D2};
        int TEMP1, TEMP2, buf1, buf2, tmp1, tmp2;
        double time, start;

        X[0]   = 0x0;
        X[1]   = 0x1;
        X[2]   = 0x2;
        X[3]   = 0x3;
        X[4]   = 0x4;
        X[5]   = 0x5;
        X[6]   = 0x6;
        X[7]   = 0x7;
        X[8]   = 0x8;
        X[9]   = 0x9;
        X[10] = 0xA;
        X[11] = 0xB;
        X[12] = 0xC;
        X[13] = 0xD;
```

```
X[14] = 0xE;
X[15] = 0xF;

for(i=0;   i<64;   i++){
            x1[i] = X[i] >> 16;
            x2[i] = X[i] & 0xFFFF;
}

for(i=1;   i<=64;   i++) {
    T[i-1] = sin(i * (3.14159265358979323846264338) / 180) * 1073741824;
}

for(i=0;   i<64;   i++){
            t1[i] = T[i] / 0x10000;
            t2[i] = T[i] & 0xFFFF;
            t1[i] = t1[i] * 4;
            t2[i] = t2[i] * 4;
            t1[i] = Carry(t1[i], t2[i]);
            t2[i] = t2[i] & 0xFFFF;
}
//ローテートに用いる定数 s
for(n=0;   n<4;   n++){
            s[n*4]     =   7;
            s[n*4 + 1] = 12;
            s[n*4 + 2] = 17;
            s[n*4 + 3] = 22;
}
for(n=4;   n<8;   n++){
            s[n*4]     =   5;
            s[n*4 + 1] =   9;
            s[n*4 + 2] = 14;
            s[n*4 + 3] = 20;
}
for(n=8;   n<12;   n++){
            s[n*4]     =   4;
            s[n*4 + 1] = 11;
            s[n*4 + 2] = 16;
            s[n*4 + 3] = 23;
}
for(n=12;   n<16;   n++){
            s[n*4]     =   6;
            s[n*4 + 1] = 10;
            s[n*4 + 2] = 15;
            s[n*4 + 3] = 21;
}

start = second();

for(i=0;   i<64; i++){
            #pragma omp parallel sections
            {
            #pragma omp section
                        buf1 = A1 + g(i, B1, C1, D1);
            #pragma omp section
                        buf2 = A2 + g(i, B2, C2, D2);
            #pragma omp section
                        tmp1 = x1[ k(i) ] + t1[i];
            #pragma omp section
                        tmp2 = x2[ k(i) ] + t2[i];
            }
```

```
                    TEMP1 = Carry( (buf1 + tmp1) , (buf2 + tmp2) ) & 0xFFFF;
                    TEMP2 = (buf2 + tmp2) & 0xFFFF;

                    TEMP1 = Rotate( TEMP1, TEMP2, s[i] ) + B1;
                    TEMP2 = Rotate( TEMP2, TEMP1, s[i] ) + B2;
                    TEMP1 = Carry(TEMP1, TEMP2) & 0xFFFF;
                    TEMP2 = TEMP2 & 0xFFFF;

                    A1 = D1;
                    A2 = D2;
                    D1 = C1;
                    D2 = C2;
                    C1 = B1;
                    C2 = B2;
                    B1 = TEMP1;
                    B2 = TEMP2;
                }
        time = second() - start;

        H1[0] = Carry( (H1[0] + A1) , (H2[0] + A2) ) & 0xFFFF;
        H1[1] = Carry( (H1[1] + A1) , (H2[1] + A2) ) & 0xFFFF;
        H1[2] = Carry( (H1[2] + A1) , (H2[2] + A2) ) & 0xFFFF;
        H1[3] = Carry( (H1[3] + A1) , (H2[3] + A2) ) & 0xFFFF;

        H2[0] = H2[0] & 0xFFFF;
        H2[1] = H2[1] & 0xFFFF;
        H2[2] = H2[2] & 0xFFFF;
        H2[3] = H2[3] & 0xFFFF;

    printf("%04X%04X  %04X%04X  %04X%04X  %04X%04X¥n", H1[0], H2[0],
                                                H1[1], H2[1], H1[2], H2[2], H1[3], H2[3]);
    printf("time= %lf ¥n", time);

    return 0;
}
```

付録 D　MD5 の Verilog-HDL プログラム

```
module md5( clk, xrst, Y, A_out, B_out, C_out, D_out, ef );

input               clk, xrst;
input     [511:0]   Y;
output    [31:0]    A_out, B_out, C_out, D_out;
output    ef;
reg                 ef;
reg       [1:0]     state;
reg       [6:0]     t;
reg                 [31:0]    A, B, C, D;
reg                 [31:0]    TEMP, buffer1, buffer2;
reg                 [31:0]    X[31:0];
wire      [31:0]    T;

parameter           H0 = 32'h67452301;
parameter           H1 = 32'hefcdab89;
parameter           H2 = 32'h98badcfe;
parameter           H3 = 32'h10325476;

assign    A_out = A + H0;
assign    B_out = B + H1;
assign    C_out = C + H2;
assign    D_out = D + H3;

function    [31:0]  G;
          input   [6:0]    t;
          input   [31:0]   B, C, D;
                  if( t>=7'd0   &&   t<7'd16 ) begin
                          G = (B & C) | (~B & D);
                  end
                  else if( t>=7'd16   &&   t<7'd32 ) begin
                          G = (B & D) | (C & ~D);
                  end
                  else if( t>=7'd32   &&   t<7'd48 ) begin
                          G = B ^ C ^ D;
                  end
                  else begin
                          G = C ^ (B | ~D);
                  end
endfunction

function    [31:0]  k;
          input   [6:0]    t;
                  if( t>=7'd0   &&   t<7'd16 ) begin
                          k = t;
                  end
                  else if( t>=7'd16   &&   t<7'd32 ) begin
                          k = (1 + 5*t) % 16;
                  end
                  else if( t>=7'd32   &&   t<7'd48 ) begin
                          k = (5 + 3*t) % 16;
                  end
                  else begin
                          k = (7*t) % 16;
                  end
endfunction
```

```
function    [5:0]   s;
            input   [6:0]    t;
                        if( t>=7'd0   &&   t<7'd16 ) begin
                                    if( t[1:0] == 2'b00 ) begin
                                                s = 6'd7;
                                    end
                                    else if( t[1:0] == 2'b01 )begin
                                                s = 6'd12;
                                    end
                                    else if( t[1:0] == 2'b10 )begin
                                                s = 6'd17;
                                    end
                                    else begin
                                                s = 6'd22;
                                    end
                        end
                        else if( t>=7'd16   &&   t<7'd32 ) begin
                                    if( t[1:0] == 2'b00 ) begin
                                                s = 6'd5;
                                    end
                                    else if( t[1:0] == 2'b01 )begin
                                                s = 6'd9;
                                    end
                                    else if( t[1:0] == 2'b10 )begin
                                                s = 6'd14;
                                    end
                                    else begin
                                                s = 6'd20;
                                    end
                        end
                        else if( t>=7'd32   &&   t<7'd48 ) begin
                                    if( t[1:0] == 2'b00 ) begin
                                                s = 6'd4;
                                    end
                                    else if( t[1:0] == 2'b01 )begin
                                                s = 6'd11;
                                    end
                                    else if( t[1:0] == 2'b10 )begin
                                                s = 6'd16;
                                    end
                                    else begin
                                                s = 6'd23;
                                    end
                        end
                        else begin
                                    if( t[1:0] == 2'b00 ) begin
                                                s = 6'd6;
                                    end
                                    else if( t[1:0] == 2'b01 )begin
                                                s = 6'd10;
                                    end
                                    else if( t[1:0] == 2'b10 )begin
                                                s = 6'd15;
                                    end
                                    else begin
                                                s = 6'd21;
                                    end
                        end
endfunction
```

```
function    [31:0]   Rotate;
            input   [31:0]   x;
            input   [5:0]    n;
                        assign Rotate = (x << n) + (x >> (32-n) );
endfunction

abs_sin   inst_abs_sin( .t(t), .OUT(T) );

always @(posedge clk or negedge xrst)
begin
            if(!xrst) begin
                        TEMP = 0;
                        X[0][31:0]   = 0;
                        X[1][31:0]   = 0;
                        X[2][31:0]   = 0;
                        X[3][31:0]   = 0;
                        X[4][31:0]   = 0;
                        X[5][31:0]   = 0;
                        X[6][31:0]   = 0;
                        X[7][31:0]   = 0;
                        X[8][31:0]   = 0;
                        X[9][31:0]   = 0;
                        X[10][31:0] = 0;
                        X[11][31:0] = 0;
                        X[12][31:0] = 0;
                        X[13][31:0] = 0;
                        X[14][31:0] = 0;
                        X[15][31:0] = 0;
                        A = 0;
                        B = 0;
                        C = 0;
                        D = 0;
                        state = 0;
                        ef = 1;
                        t = 0;
                        buffer1 = 0;
                        buffer2 = 0;
            end

            else if(state == 0) begin
                        X[0][31:0]   = Y[511:480];
                        X[1][31:0]   = Y[479:448];
                        X[2][31:0]   = Y[447:416];
                        X[3][31:0]   = Y[415:384];
                        X[4][31:0]   = Y[383:352];
                        X[5][31:0]   = Y[351:320];
                        X[6][31:0]   = Y[319:288];
                        X[7][31:0]   = Y[287:256];
                        X[8][31:0]   = Y[255:224];
                        X[9][31:0]   = Y[223:192];
                        X[10][31:0] = Y[191:160];
                        X[11][31:0] = Y[159:128];
                        X[12][31:0] = Y[127:96];
                        X[13][31:0] = Y[95:64];
                        X[14][31:0] = Y[63:32];
                        X[15][31:0] = Y[31:0];
                        A = H0;
                        B = H1;
                        C = H2;
                        D = H3;
```

```
                        state = 1;
                        ef = 1;
                end

                else if(state == 1) begin
                        if( t<64 ) begin
                                buffer1 = A + G(t, B, C, D);
                                buffer2 = X[ k(t) ] + T;

                                TEMP = buffer1 + buffer2;
                                TEMP = Rotate( TEMP, s(t) ) + B;
                                A = D;
                                D = C;
                                C = B;
                                B = TEMP;
                                t = t + 1;
                        end
                        else begin
                                state = 0;
                                ef = 0;
                        end
                end
end
endmodule
```