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

2006－SLDM－127（6）
2006／11／28

マルチメディアコアの展開機能を利用したテストデータ圧縮・展開

瀬戸原志典† 中島 佑介†† 市原 英行†† 井上 智生††

†広島市立大学大学院情報科学研究科
††広島市立大学情報科学部
〒731-3194広島市安佐南区大塚東3-4-1
E-mail: {setohara, nakashima}@dsgn.im.hiroshima-cu.ac.jp, {ichihara, tomoo}@im.hiroshima-cu.ac.jp

あらまし テストデータ圧縮・展開手法は，LSIのテストに要するコストを削減するための重要な手法の1つである．一般的にこの手法では，LSI内部に展開器を必要とする．本論文では，マルチメディアコアを持つシステムLSIに対して，マルチメディアコアの展開機能を利用したテストデータ圧縮・展開手法を提案する．マルチメディアコアを持つシステムLSIでは，動画像展開などのデータ展開機能を有するため，これらを利用することで圧縮テストデータ専用の展開器を必要としない効率的なテストデータ圧縮・展開を行うことができる．また，JPEGアルゴリズムの可変長符号(VLC)に着目し，VLCデコーダを利用したテストデータ圧縮・展開手法の提案を行い，さらに，この圧縮・展開手法に適したテストデータ生成法を提案する．
キーワード テスト圧縮・展開，埋込み展開器，マルチメディアコア，符号化

Test Compression/Decompression with the Decoding Function in Multimedia Cores

Yukinori SETOHARA†, Yusuke NAKASHIMA††, Hideyuki ICHIHARA††, and Tomoo INOUE††

† Graduate School of Information Science, Hiroshima City University
†† Faculty of Information Sciences, Hiroshima City University
3-4-1 Ozuka-higashi, Asaminami-ku, Hiroshima 731-3194
E-mail: {setohara, nakashima}@dsgn.im.hiroshima-cu.ac.jp, {ichihara, tomoo}@im.hiroshima-cu.ac.jp

Abstract Test data compression / decompression scheme for LSI testing can reduce test application cost. A drawback of this scheme is that an decompressor must be embedded in the LSI under test. In this paper, we propose a scheme of test compression / decompression with the decoding function in multimedia cores. Today's system LSI with multimedia cores has decoders for images, sounds and videos, so that test data decompression without additional decomopressors can be achieved . In addition, we focus on the variable-length coding (VLC) of JPEG algorithm, and propose a test compression / decompression method using the VLC decoder. We also propose a test generation method for this method.
Key words Test compression / decompression, embedded decompressors, multimedia core, and encoding.

1. はじめに

近年，LSIの大規模化，複雑化に伴い，LSIの設計にかかる時間やLSIが正しく動作するかどうかを確かめるテストに要する時間が増大している．中でも，テストデータ量の増加によるテスト実行時間の増加や，テスタに必要なメモリサイズの増大が問題となっている．これらの問題を解決するため，テストデータ圧縮・展開手法が提案されている[1].

テストデータ圧縮・展開手法は，生成したテストデータを，あらかじめ圧縮しておく．テスト時には，被LSI内部に埋め込まれた展開器により展開する手法である．このように圧縮・展開を行うことで，被テストLSI内部に展開器を付加する必要はあるものの，テスタメモリの削減，テスタから被テストLSIへの転送時間の短縮が期待できる．

テストデータ圧縮・展開手法は，被テストLSI内部に圧縮したテストデータを展開する展開器が必要である．展開器は，ハフマンデコーダ[2]，GOLOMBデコーダ[3]，FDRデコーダ[4]，再構成可能デコーダ[5]など，様々であるが，これらは全て本来のLSIの機能に必要ない機能であり，テストデータ展開のために新たに加えられたものである．

一方で今日，携帯電話やカーナビゲーションシステムなどのアプリケーションでは，マルチメディアコアを有するLSIが用いられるようになった．マルチメディア用LSIでは，画像や音声データの圧縮・展開を行うためにデータ展開機構を備えているため，このようなデータ展開機構をテストデータの展開に利用できることが期待できる．

本論文では，マルチメディアコアを有するLSIに対し，マルチメディアコアの展開機能を利用したテストデータ圧縮・展開手法を提案する．マルチメディアコアの展開機能を利用することで，展開器を必要としない効率的なテストデータ圧縮・展開を行うことができる．また，マルチメディアコアを利用した1つの例として，JPEGアルゴリズムにおける可変長符号（VLC）に着目し，このVLCデコーダを利用したテストプラン圧縮・展開手法を提案する．VLCは圧縮後のデータを圧縮前のデータに完全に復元することができる可逆圧縮であるため，比較的テスト圧縮に利用しやすい．さらに，この展開手法に適したテストデータの生成法も提案する．提案するテストデータ生成法では，与えられたドントケアビットを含むテスト集合に対して，(1)テストベクトルの並べ替え，(2)ドントケアビットの0または1へのマッピング，そして(3)テストベクトルの反転を行うことで，VLCによる圧縮に適したテストベクトル集合を生成する．

以下に本論文の構成を述べる．まず，第2節では，マルチメディアコアを有するLSIに対し，マルチメディアコアの展開機能を利用したテストデータ圧縮・展開手法について述べる．第3節では，マルチメディアコアの展開機能として，JPEGアルゴリズムにおける可変長符号化(VLC)を用い，テストデータ圧縮・展開を行う方法を述べる．第4節では，VLCに適したテストデータ生成法を述べる．第5節で評価実験について述べ，第6節でまとめと今後の課題について述べる．

2. マルチメディアコアにおける埋め込み展開器を利用したテストデータ圧縮・展開

マルチメディアコアが埋め込まれたLSIに対し，マルチメディアコアの画像展開機能や，動画像展開機能などを利用したテストデータ圧縮・展開を行う方法を提案する．図1に示すように，あらかじめテストデータを音声や画像等の圧縮法を用いて圧縮(T')を行い，圧縮されたテストデータをテスタに蓄える．被テスト回路のテストを行う場合は，テスタから圧縮されたテストデータを送り，被テストLSIが有しているマルチメディアコアの展開器を利用し，テストデータを展開し，回路に印加する．なお，図に示したように，マルチメディアコアに対して，適切な補助回路(テストコントローラ)を加えることにより，必要に応じてマルチメディアコアの展開機能の一部を利用できると考えている．このように，被テストLSIが有している機能を利用するため，新たに展開器を埋め込む必要がなく，テストコントローラや，テストデータを適切に被テスト回路に与えるためのテストアクセス機構（TAM）などの比較的小さいハードウェアオーバーヘッドで，テストデータの展開を行うことができる．

マルチメディアコアが利用する圧縮・展開手法は，一般的に，

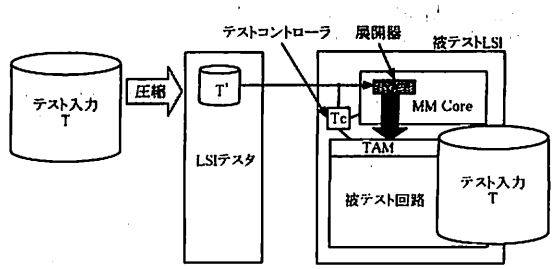

図1　マルチメディアコアを有するLSIのテスト

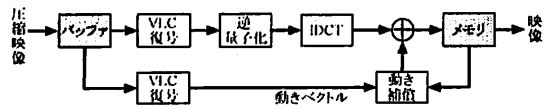

図2　MPEGアーキテクチャ(展開器)

圧縮したデータを完全に元のデータに復元できる可逆な圧縮・展開手法と，完全に元のデータには戻すことができない不可逆な圧縮・展開手法からなる．図2は動画像圧縮・展開手法の1つであるMPEGの展開器アーキテクチャを示している．この動画像展開アーキテクチャにおいて，可変長符号化(VLC)に対する復号処理と動き補償は可逆な展開処理であり，また，逆量子化は，圧縮時の量子化処理により扱うデータに依存して決められた不要な情報を削減しているため，不可逆な展開処理に相当する．

このように，1つの圧縮・展開手法であっても，様々な可逆・不可逆な処理を含んでおり，テストデータ圧縮・展開に用いる場合にはそれぞれの特長に応じた利用を行う必要がある．例えば，可逆な圧縮を行うVLCなどは，これまで提案されてきたテストデータ圧縮・展開手法[1]~[5]に類似しているため，比較的容易に利用できると考えられる．また，不可逆な圧縮である量子化を用いてテストデータを圧縮する場合は，圧縮によって削減される情報が存在するため，圧縮によってテストデータの故障検出能力が低下しないような工夫（例えばテストデータを適切に変換するなど）が必要となる．

次節では，マルチメディアコアの展開機能を利用したテストデータ圧縮・展開手法の1つとして，JPEGアルゴリズムにおける可変長符号化（VLC）を利用したテストデータ圧縮・展開を提案する[6][7]．

3. JPEGアルゴリズムにおける可変長符号化を利用したテストデータ圧縮・展開

VLCでは8ビットのブロック列を入力とするため，テストベクトル列を8ビットブロック列に変換する．図3に示す長さ10ビットの4つのテストベクトル集合$S=(v_0,v_1,v_2,v_3)$（総ビット数40）が与えられたとする．このテストベクトル列Sに対して8ビットでブロック化を行い，図4に示すブロック系列

(注1)：なお，MPEGアルゴリズムでも同様な圧縮・展開が用いられている（図2のVLC復号がその展開部分）ため，本手法は，比較的容易に応用できる．

v_0 = 0000000000
v_1 = 0101110000
v_2 = 0000000001
v_3 = 0000001101

図3　テストベクトル集合 S

b_0 = 00000000
b_1 = 00010111
b_2 = 00000000
b_3 = 00000100
b_4 = 00001101

図4　S をブロック化して得られる 8 ビットブロック列 B

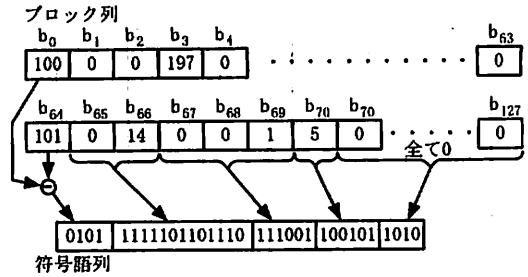

図5　VLC の符号化

$B=(b_0,b_1,b_2,b_3,b_4)$ を得る．

テストベクトル列 $S=(v_0,v_1,\ldots,v_n)$ からブロック系列 $B=(b_0,b_1,\ldots,b_m)$ への写像は次のように表せる．ここで，v_i は i 番目のテストベクトルを表し，$v_i=q_{i,0}q_{i,1}\cdots q_{i,w}$ とする．なお，$w+1$ はベクトル長を表し，$q_{i,j}\in\{0,1\}$ は，テストベクトル内の要素を表している．一方，g 番目のブロックは $b_g=p_{g,0}p_{g,1}\cdots p_{g,7}$ と表すこととする．なお $p_{g,j}\in\{0,1\}$ である．この時，テストベクトル v_i の j 番目の要素 $q_{i,j}$ に対応する，ブロック b_g の要素を $p_{g,h}$ とするとき，g と h は，

$$g=\lfloor(i\times(w+1)+j)/8\rfloor \quad (1)$$

$$h=(i\times(w+1)+j)\ mod\ 8 \quad (2)$$

と表わされる．

変換したブロック列 $b_0,b_1,\ldots b_m$ は，予め用意された符号化表に基づいて符号化される．ここでは，文献[8] の Annex K を基にした符号化表を用いる．

ブロック系列において，$b_0,b_{64},b_{128},\ldots$ は DC ブロックと呼ばれ，それ以外のブロックは AC ブロックと呼ばれる(注2)．DC ブロックと AC ブロックは以下のように別々に符号化される．

DC ブロック $b_{64\times l}(l=0,1,\ldots)$ は，1 つ前の DC ブロック $b_{64\times(l-1)}$ との値の差分 $d_l=b_{64\times l}-b_{64\times(l-1)}$ の絶対値に基づいて符号化される．符号語は，表 1 に示すように，接頭語と d_l を 2 進数で表現した接尾語（長さは表 1 の対応する各グループ番号と同じ）からなる．表 1 の最後の列には差分 d_l に対応する語長を示しいてる．図 5 は，b_0 から b_{127} までのブロック列と対応する符号語列を示している．各ブロックに示した数字は 10 進表現で各ブロックの内容を示している．この例において，DC ブロック b_{64} は，b_0 との差分が $101-100=1$ であるため，表 1 に示したように 010 という接頭語に差分の $1_{(2)}$ を付加した符号語 0101 に割り当てられる．

一方，AC ブロックはランレンスグス符号を用いて符号化される．図 5 において，$b_1,\ldots,b_{63},b_{65},\ldots,b_{127}$ は AC ブロックである．まず，AC ブロック列はゼロブロックランに分割する．1 つのゼロブロックランは連続するゼロブロックの列と 1 つの終端ブロックからなるブロック列である．ここで，8 ビットすべてのビットが 0 であるブロックをゼロブロック，いずれかのビットに 1 を含むブロックを終端ブロックと呼ぶ．図 5 において，b_{67},b_{68},b_{69} の部分列はゼロブロックランである．それぞれのゼロブロックランはその大きさ（ゼロブロックの数）と終端ブロックの値によって，表 2 に示したような接頭語と，終端ブロックを接尾語（ただし，終端ブロックの最上位の 1 よりも上位部分は除去する）とした符号語に割り当てられる．例えば，図 5 のゼロブロックラン b_{67},b_{68},b_{69} はゼロランの大きさが 2，終端の終端ブロックの値が 1 であるため，接頭語が 11100，接尾語が 1 である長さ 6 ビットの符号語 111001 に符号化される．また，ある AC ブロック以降の AC ブロックが全てゼロブロックならば（図 5 の b_70 から b_127 に相当），4 ビットの符号語 1010 に割り当てられる．

なお，図 4 のブロック列を AC ブロックだと仮定し，VLC を用いて圧縮すると 40 ビットのブロック列は 34 ビットに圧縮できる．これは長さ 1，終端ブロックのグループが 5 のゼロブロックラン b_1,b_2 が 16 ビットに，長さ 1，終端ブロックのグループが 3 のゼロブロックラン b_3,b_4 が 10 ビットに，長さ 0，終端ブロックのグループが 4 のゼロブロックラン b_5 が 8 ビットに符号化されるためである．

表 2 からわかるように，終端ブロックが同じならばゼロブロックランが大きいほど符号語が長くなるものの，ゼロブロックランが大きくなる割合に比べると符号語長の増加の割合は小さいため，相対的にゼロブロックランが大きくなるほど，圧縮率は高くなる．また，同じ大きさのゼロブロックランならば，終端ブロックの値が大きくなるほど符号語は長くなるため，終端ブロックの大きさが小さいほど圧縮率が高くなる事がわかる．また，ブロックのほとんどは AC ブロックとなるため，この AC ブロックの符号化の圧縮率が重要となる．このため，次節で述べるテストデータ生成手法では，AC ブロックのみを対象とした手法で説明する．なお，実験は DC ブロックを含めた圧縮率を評価している．

4. JPEG アルゴリズムの可変長符号化に適したテストデータ生成法

本節では VLC に適したテストデータの生成法を提案する．提案するテストデータ生成法では，ドントケアビットを含むテストベクトル集合を受けとり，以下に述べる 3 つの処理を行うことで，完全に 0 または 1 にマッピングされた入力順の決まった

(注2)：JPEG アルゴリズム全体から見れば，DC ブロックは画像全体のオフセットを表す（周波数成分を含まない）成分を表現しており，AC ブロックは周波数成分を表すブロックである．ここでは本来の意味は重要ではないため説明を割愛する．

表1 DC成分用エントロピー符号化表

グループ	差分 $\|di\|$	接頭語	接尾語	符号長
0	0	00		2
1	1	010		4
2	2, 3	011		5
3	4−7	100	diの最上位の1よりも下位部分	6
4	8−15	101		7
5	16−31	110		8
6	32−63	1110		10
7	63−127	11110		12
8	128−255	111110		14

表2 AC成分用エントロピー符号化表(接頭語のみ)

		グループ					・・・	
		0	1	2	3	4	・・・	8
終端ブロック		0	1	2,3	4−7	8−15	・・・	128−255
ゼロブロックラン長	0	1010	00	01	100	1011	・・・	1111 1101 0
	1		1100	11011	1111001	1111 1011 0	・・・	1111 1011 0000 0100
	2		11100	11111001	1111 1101 1	1111 1111 0100 0111	・・・	1111 1111 0000 0100
	3		111010	11111011	1111 1111 0101 1	1111 1111 1000 0111	・・・	1111 1111 1001 0011
	⋮		⋮	⋮	⋮	⋮	⋱	⋮
	15	1111 1111 001	1111 1111 0101 1	1111 1111 1101 1	1111 1111 1110 1	1111 1111 1111 0000	・・・	1111 1111 1111 0100

テストベクトル系列を生成する．生成されたテストベクトル系列はVLCにより高い圧縮率を得ることができる．本生成法で用いる3つの手法は，(1) テストベクトルの並べ替え，(2) ドントケアの0，1マッピング，(3) テストベクトルの反転である．それぞれの手法について説明した後に，テストデータ生成アルゴリズム全体について説明する．

4.1 テストベクトルの並べ替え

テスト対象回路は，組み合わせ回路またはフルスキャン設計された順序回路を想定しているため，テストベクトルは並べ替え可能であると仮定している．そこで，テストベクトルを並べ替えることでVLCに適した並びを考える．

図3に示したテストベクトル列は先ほど述べたように，図4のようにブロック化され，結果としてVLCで34ビットに圧縮される．一方，ベクトルの並べ替えを行ったテストベクトル列を$S'=(v_4,v_2,v_3,v_1)$ように並べ替え，ブロック化を行うと，$B'=(b'_1,b'_2,b'_3,b'_4,b'_5)=((00000011),(01010111),(00000000),(00000000),(00000000),(00000001))$となる．このブロック列にVLCを適用した場合，26ビットとなる．これはテストベクトルを並べ替えることで，ブロックのビット列が変化し，b'_3,b'_4,b'_5のように，ゼロブロックラン長が延び，終端ブロックの値が小さくなったことが原因である．

3節の最後に述べたように，基本的にはなるべくゼロブロックランが生じるように，かつ，終端ブロックの値がなるべく小さくなるように並べることが望ましいと考えられる．ただし，最適なテストベクトルの並びを見つけるためには，すべてのベクトルの並びを試みることが必要になると考えられる．よって，4.4節ではグリーディ法でテストベクトルの順列を決める方法を提案する．

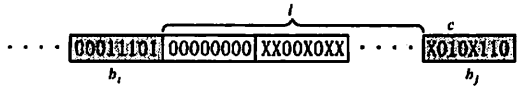

図6 テストベクトルの並べ替え

4.2 ドントケアマッピング

本テストデータ生成法では，対象としているテストデータにはXで示されるドントケアビットが存在すると考えている．VLCで圧縮を行う場合，ゼロブロックランレングスが長い方が圧縮率が高くなる傾向があるため，ドントケアビットを0にマッピングをすることが，有効であると考えられる．しかし，ある条件下ではドントケアビットを単に0にマッピングするのではなく，適切に1にマッピングすることで，符号化後のデータ量を少なくできる場合がある．

例えば，ドントケアビットを含んだブロック列$B=(b_1,b_2,b_3)=(0XXX0XX0,00XX0XXX,XXX011X1)$を考える．このブロック列において，ゼロブロックランが大きくなるようにすべてのドントケアビットを0にマッピングすると，$(b_1,b_2,b_3)=(00000000,00000000,00001101)$となり，大きさ2，終端ブロックが1101のゼロブロックランとなる．これは，表2に従えば，16ビットに圧縮される．一方で，$(b_1,b_2,b_3)=(00000000,00000001,00001101)$のように，大きさ1，終端ブロック1のゼロブロックランと，大きさ0，終端1101のゼロブロックランとなるようにマッピングを行うと，それぞれ5ビット，8ビットの符号語となり，合計13ビットに圧縮できる．

この例のように，ドントケアビットを含むブロック列が与えられた場合，最も圧縮率が高くなるドントケアビットのマッピングは，表2に従って一意に決めることができる．提案テストデータ生成法ではドントケアビットは常に最適なマッピングを行うものとする．ドントケアビットのマッピング手順を以下に示す．

図6に示したブロック列においてb_jを終端ブロックとするゼロブロックランについて考える．b_{i+1}からb_{j-1}までのブロックは，ゼロブロックまたは，0またはXだけからなるブロックであるとする．また，b_iはドントケアビットを含まない終端ブロックとする．このとき，$l=j-i-1$は，b_jを終端ブロックとするゼロブロックランの最大長となる．次に，b_jの最も最上位(図では最も左)に存在する1の要素の番号をcとする．つまり，$p_{j,c}=1$かつ$\forall 0\leq x<c, p_{j,x}\neq 1$である．

最適なドントケアマッピングはこのlとcによって決定する．表3は表2に従い，それぞれのlとcの組に対して，1にすべきドントケアビットの候補の一部を示している．候補は，b_jに対する相対的なブロックの位置kと，そのブロック内の要素位置dの2項組の系列で表される．例えば$(k,d)=(-1,6)$であれば，ブロックb_{j-1}の要素$p_{j-1,6}$を意味する．アルゴリズムでは表3に示した候補順を左から順に試み，候補のビットがドントケアビットであればそれを1にマッピングし，その後，b_{i+1}からb_jまでの全てのドントケアを0にマッピングする．候補がドントケアビットでなければ，次の候補を調べ，候

表3 ゼロランレングスを適切な長さに区切る組み合わせ

l	c	候補 (k,d)
1	5	$(-1,7)$
	4	$(-1,7),(-1,6)$
	3	$(-1,7),(-1,6)$
	2	$(-1,7),(-1,6),(-1,5),(-1,4)$
	⋮	⋮
2	5	$(-1,7)$
	⋮	⋮

補が1にすべきドントケアビットが見つかるか，候補が無くなるまで繰り返す．候補がなくなった場合は，b_{i+1} から b_j までのすべてのドントケアビットを0にマッピングする．例えば，$(b_{i-1},b_i)=(0000XXX0,0X1XX01X)$ となるゼロブロックランが存在した場合，$l=1$，$c=2$ となり，表3の $l=1$，$c=2$ の候補よりまず，$(k,d)=(-1,7)$ の候補 b_{i-1} の要素 $p_{i-1,7}$ の候補の1マッピングを試みる．しかし，b_{i-1} の要素 $p_{i-1,7}$ は0であるため，次の候補である $(k,d)=(-1,6)$ の意味する $p_{i-1,6}$ のドントケアを1にマッピングを行う．そして，残りの b_{i-1},b_i に存在するドントケアを0にマッピングを行う．これにより，最適なドントケアマッピング行ったブロック $(b_{i-1},b_i)=(00000010,00100010)$ を得ることができる．

上記の処理を番号の小さい終端ブロックから順に行うことで，最適なドントケアマッピングを行うことができる．

4.3 テストベクトルの反転

テストベクトルの各ビットに対して，0は1に，1は0に置き換えることをベクトルの反転と呼ぶ．反転を行うことで，ゼロブロックラン長が長くなり，終端ブロックの値が小さくなる場合は，テストベクトルを反転することを行う．これによりテスト系列は，反転を行うものと行わないものに分けて，生成されることになる．

反転したテストベクトルは被テスト回路に入力する際，元に戻す必要があるため，被テストLSIに対し，反転を行うためのテスト容易化設計が必要となる．なお，このために必要な付加回路はNOTゲートから構成される簡単な回路である．

4.4 VLCに適したテストデータ生成アルゴリズム

以上で述べた3つの手法を利用した，VLCに適したテストデータ生成アルゴリズムについて説明する．

基本となる戦略は，与えられたテストベクトルの並べ替えである．並べ替えはテストベクトル集合 T から1つテストベクトル t を選択し，既に並べられたテストベクトル系列 S_p(初期値は $S_p=(\Phi)$) に連結することで行う．そして，t は T から除去し，この処理を T が空になるまで繰り返す．テストベクトルの選択時には，そのテストベクトルを反転するべきか否かも同時に決定し，反転を行う場合は，系列 SI_p(初期値は $SI_p=(\Phi)$) に連結する．よって，テストデータ生成の結果，テストベクトルの反転を行わない系列 S_p と反転を行う系列 SI_p の2つのテスト系列が生成される．

選択されるテストベクトル t は，以下の手順によって決定する．今，テストベクトルの反転を行わない部分系列 S_p と反転

t_1 : 111XX101XX11110
t_2 : 0X0XX00X000XX01
t_3 : 111XX110XXX1110

図7 テスト集合

表4 テストベクトル t_2 におけるブロック化

ブロック化	b_0	b_1	b_2	符号化後データサイズ
1		0X0XX00X	000XX01	17.0
2	0	X0XX00X0	00XX01	23.5
3	0X	0XX00X00	0XX01	32.0
4	0X0	XX00X000	XX01	28.0
5	0X0X	X00X000X	X01	26.0
6	0X0XX	00X000XX	01	27.0
7	0X0XX0	0X000XX0	1	23.5
8	0X0XX00	0X000XX01		20.5
平均値				24.688

を行う部分系列 SI_p が与えられているとする．テストベクトル集合 T から1つテストベクトルを選択し，S_p の最後尾に連結し，4.2節におけるドントケアマッピングを行った場合の S_p の圧縮率と，反転したテストベクトル $\bar{t}$ を SI_p に連結し，ドントケアマッピングを行った場合の SI_p の圧縮率両方を調べる．これをテストベクトル集合 T にある全てのベクトルについて調べ，符号化後の圧縮率が最大になるベクトル t を選択する．ここで，圧縮率は，

$$\text{圧縮率}=\frac{\text{符号化前データ量}-\text{符号化後データ量}}{\text{符号化前データ量}} \qquad (3)$$

とする．選ばれた t は，S_p と SI_p のいずれかに連結される．

符号化後の圧縮率が最大になるベクトルが複数存在する場合，各ベクトルに対して，考えられる全てのブロック化を考慮して，それぞれのブロック化による平均値を計算し，その平均値(符号化後サイズの期待値)が最も大きいベクトルを選択する．これは，複数の候補がある場合は，現在のブロック化が最も有効なものを選びたいという考え方に基づくものであり，最も平均値が大きいベクトルがそれにあたるためである．

図7のベクトルを用いてブロック化による平均値を求める手順を示す．考えられるブロック化は，表4に示すように8通りある．それぞれのブロック化に対して，符号化後のデータサイズが最大の場合と最小の場合を考えて，中間値をそのブロック化の符号化後データサイズとする．このブロック化での最大サイズとなるパターンは $(b_0,b_1,b_2)=(11111111,0X0XX00X,000XX011)$ で，最小のパターンは $(b_0,b_1,b_2)=(00000000,0X0XX00X,000XX011)$ である．それぞれ，25と9となるので，符号化後データサイズは17となる．それぞれのブロック化によるデータサイズは，表4の符号化後データサイズに示す．考えられる全てのブロック化の符号化後データサイズの平均値をベクトルの符号化後サイズの期待値とする．t_2 では，24.688が平均値となる．

さらにオプションとして，圧縮率が最大になるベクトルだけ

表5 テストベクトル並べ替え時の圧縮率

テストベクトル		期待値	ベクトル $t_i(\bar{t}_i)$ を選択した場合の $S_p(SI_p)$ の圧縮率		
			1 回目	2 回目	3 回目
t_1	111XX101XX11110	43.375	-0.771	-0.625	-0.625
t_2	0X0XX00X000XX01	24.688	0.313	-	-
t_3	111XX110XXX1110	45.250	-0.771	-0.542	-
$\bar{t}_1$	000XX010XX00001	31.062	0.230	0.230	0.167
$\bar{t}_2$	1X1XX11X111XX10	44.250	-0.771	-	-
$\bar{t}_3$	000XX001XXX0001	31.125	0.188	0.188	-
S_p の内容			t_2	t_2	t_2
SI_p の内容			-	$\bar{t}_3$	$\bar{t}_3.\bar{t}_1$

表6 提案手法を行った実験結果

回路名	総データ量 [bit]	0 マッピング		テストデータ生成	
		圧縮後データ量	圧縮率	圧縮後データ量	圧縮率
b12	14,994	10,065	0.329	9,854	0.343
b15	218,735	80,829	0.631	76,788	0.649
b15_1	520,890	174,817	0.664	166,030	0.681
b17	1,639,308	384,174	0.766	373,326	0.772
b17_1	1,640,760	538,285	0.672	502,277	0.694
b18	4,216,392	1,428,810	0.661	1,372,597	0.675
b18_1	4,216,392	1,358,795	0.682	1,304,397	0.694
平均			0.564		0.551

を上記の平均値を用いた選択の候補にするのではなく，各ベクトルの圧縮率が最大の圧縮率-α%以上であるベクトルも候補として，平均値を用いた選択を行うことも考える．α はパラメータであり，α が 0 の場合は，上記の符号化後のデータサイズが最小になるベクトルが複数存在する場合の選択となる．

図 7 に示すテスト集合についてテストデータ生成を行う．まず，反転を行わない系列 S または，反転を行う系列 SI の先頭ベクトルを決定する．表における $\bar{t}_i$ は，t_i の反転ベクトルである．ベクトルの選択の様子を表 5 に示す．パラメータ α は 5 とする．圧縮率の最大値は t_2 の 0.313 である．圧縮率が $0.313-0.05=0.263$ 以上のベクトルは，t_2 のみであるため，t_2 のベクトルが選択され，反転を行わない部分系列 S_p に連結される．次に，T から t_2 を除いたテスト集合 $T=\{t_1,t_3\}$ に対し，S_p，SI_p への連結を考える．テストベクトル t_1，t_3 をそれぞれ S_p，SI_p に連結した場合の圧縮率を表 5 の 2 回目の圧縮率に示す．圧縮率の最大値は $\bar{t}_1$ の 0.230 であり，圧縮率が $0.230-0.05=0.180$ 以上のテストベクトルは $\bar{t}_1$ と $\bar{t}_3$ がある．このベクトルにおいて，期待値が最も大きいベクトル $\bar{t}_3$ が選択され，0，1 反転を行う部分系列 SI_p に連結される．最後に，T から t_3 を除いたテスト集合 $T=\{t_1\}$ に対し，S_p，SI_p への連結を考える．表 5 の 3 回目の圧縮率よりテストベクトル t_1 は，反転を行う部分系列 SI_p に連結される．上記の結果，テストベクトルの反転を行わない系列 $S=(t_2)$ と 0，1 反転を行う系列 $SI=(\bar{t}_3,\bar{t}_1)$ が生成される．

5. 評価実験

3 節で提案したテストデータ生成法を用い生成したテストデータに対し，JPEG アルゴリズムの VLC を利用し，符号化を行った結果を表 6 に示す．テストベクトル集合は，ITC99 ベンチマーク回路 [10] に対し，Synopsys 社での TetraMAX を利用して生成を行ったものを使用した．ここで，圧縮率は式 (3) より導出した．表 6 において，「0 マッピング」は，提案手法を用いず，テストデータ生成されたテストベクトルを全てドントケアを 0 にマッピングした後，VLC を用いて圧縮した．「テストデータ生成法」は，提案したテストデータ生成法を用いた結果である．ここで，パラメータ α は，5 とした．

表 6 から，JPEG アルゴリズムにおける VLC を用い，テストデータ圧縮・展開を行うことで，データ量を平均で 56.4%，最高で 76.6%削減することができた (b17)．一方，提案したテストデータ生成法を利用した場合，平均で 1.3%，最高で 2.2%データ量の削減をすることができた．

6. まとめ，今後の課題

本論文では，マルチメディアコアの展開機能を利用した，テストデータ圧縮・展開手法，及び，テストデータ生成法を提案した．評価実験の結果から，JPEG アルゴリズムにおける VLC を用いテストデータ圧縮・展開を行うことにより，テストデータ量の削減が可能であることがわかった．また，VLC に適したテストデータ生成を行うことにより，わずかながらテストデータ量の削減が可能であった．

今回は，マルチメディアコアの可逆な処理のみを用いて，テストデータ圧縮・展開を行ったが，不可逆な処理を利用することで，音声や画像等の圧縮法の特長を活かしたより効率の良い圧縮ができると考えられる．また，不可逆性を考慮したテストデータ生成も提案する予定である．

参考文献

[1] 樋上喜信，梶原誠司，市原英行，高松雄三，"論理回路に対するテストコスト削減法 -テストデータ量および実行時間の削減-，" 電子情報通信学会論文誌 D-I，Vol. J87-D-I，No. 3，pp. 291-307，2004 年 3 月．

[2] Gonciari, P. T., Al-Hashimi, B. and Nicolici, N. (2003) Variable-Length Input Huffman Coding for System-on-a-Chip Test. IEEE Transactions on Computer-Aided Design 22(6):pp.783-783.

[3] A. Chandra and K. Chakrabarty, Test Data Compression for System-on-a-Chip Using Golomb Codes," Proc. VTS, pp.113-120, 2000.

[4] A. Chandra and K. Chakrabarty, "Frequency-Directed Run-Length(FDR) Codes with Application to System-on-a-Chip Test Data Compression," Proc. VTS, pp.42-47, 2001.

[5] 佐伯友之，市原英行，井上智生，"LSI テストにおける再構成可能な埋込み展開器について，" 信学技報，Vol. 105，No. 42，pp. 1-6，2005 年 5 月．

[6] 半谷精一郎，杉山賢二，JPEG・MPEG 完全理解，コロナ社，ISBN4-339-00778-1，2005.

[7] G. J. Sullivan, and T. Wiegand, "Video Compression - From Concepts to the H.264/AVC Standard," IEEE proc, Vol.93, No.1, pp.18-31, 2005.

[8] ISO/IEC 10918-1:1994, Information technology - Digital compression and coding of continuous-tone still images: Requirements and guidelines, 情報技術 - 連続階調静止画像のディジタル圧縮及び符合処理 - 第 1 部 要件及び指針．

[9] 小野定康，鈴木純司，わかりやすい JPEG/MPEG2 の技術，オーム社，ISBN4-274-07917-1，2001.

[10] http://www.cerc.utexas.edu/itc99-benchmarks/bench.html