設計自動化 24－8
（1984. 11. 20）

ALUの制御コード割当法

高木茂
日本電信電話公社 武蔵野電気通信研究所

1. まえがき

大規模論理装置，ＬＳＩ設計のボトルネックは論理設計工程になりつつある．このボトルネックの解決には，論理合成技術の確立が最も重要と考えられる．論理合成技術確立の一環として，多機能演算器（以後，ＡＬＵと略す）の論理合成法を研究している．

演算機能の種類と，各演算機能の選択を指示する制御コードが機能仕様として与えられたとき，これに基づきＡＬＵを自動合成する手法については既に報告した[1]．合成されたＡＬＵのデコーダ回路のゲート量は，機能仕様における制御コードの割当方に依存する．従来，ＡＬＵの制御コード割当問題を扱った報告は，見当たらない．本論文は，デコーダのゲート量を（準）最小とする制御コード割当法を示す．

本手法の特徴を以下に示す．

（１）合成すべきＡＬＵの仕様を，演算機能の種類と，各機能の選択を指示する機能識別子の一覧表で与える（以後，機能仕様と呼ぶ）．

（２）機能仕様よりＡＬＵのデコーダの論理式（複数の式からなる）を導く．得られたデコーダの論理式は，機能識別子の論理和となっている．

（３）（２）で得られた論理式に含まれる機能識別子を制御コードで置換し，論理の簡約化を行った場合に，最も積項数が小さくなるよう，制御コードを割り当てる．

（４）制御コード割当問題はＮＰ完全な問題であるのでヒューリスティクな手法（２分割法）で（準）最適解を求める．

以下，第２章では機能仕様の記述法を，第３章ではデコーダ論理式の導出法を，第４章では制御コード割当アルゴリズムを，第５章では実験例を示す．

2．機能仕様の記述法

本論文では２項ＡＬＵを扱う．ＡＬＵは，一般に，制御入力端子とデータ入出力端子を持ち，制御入力端子に制御信号（制御信号のビットパターンを制御コードと呼ぶ）を受けると入力データに演算を施し，その結果を出力端子に出力する．制御コードが未定の段階では，機能種別を識別するために，ラベル（機能識別子と呼ぶ）を使用するのが自然である．従ってＡＬＵの機能仕様を機能識別子とその時のデータ入出力端子間の関数関係で記述する．例えば，データ入力端子Ａ，Ｂ，データ出力端子Ｆを持ち，機能識別子がＥＯＲの時，排他的論理和を実行するＡＬＵの機能仕様を，次の様に記述する．

$$\mathrm{EOR} \qquad F = A \oplus B$$

一般にＡＬＵは複数の機能を有するので，上記，記述例の集合となる．

図１に本論文で使用する演算子の種類と機能仕様の構文を示す．また，本文中で使用する演算子も特にことわらない限り図１の意味で使用する．

図２に機能仕様の記述例を示す（実際の処理システムでは，ＬＩＳＰのＳ式の構文を使用し，演算子はプレフィクス記法で記述している）． 図２は，本論文で例題として扱う３２の機能を有するＡＬＵの機能仕様である．C0～C31 が機能識別子である．

3．デコーダ論理式の導出

機能仕様が与えられた時，それを一度に２レベルの積和論理式に展開し，ＡＬＵを合成すること

OP Symbol	Function
·	Logical AND
+	Logical OR
⊕	Exclusive OR
−	INVERT
PLUS	Arithmetic ADD

<Function SPEC> : := {<A Function SPEC>} ;
<A Function SPEC> : := Unique-label <Assignment exp> ;
<assignment exp> : := Output-terminal = <Expression> ;
<Expression> : := <Arithmetic EXP> | <Logic EXP> ;
<Arithmetic EXP> : := <Logic EXP> PLUS Carry-input-terminal |
<Logic EXP> PLUS <Logic EXP>
PLUS Carry-input-terminal ;
<Logic EXP> : := Input-terminal | Constant | $\overline{\langle\text{Logic EXP}\rangle}$ |
<Logic EXP> <Logic Operator> <Logic EXP> ;

Fig. 1. − Operators and Function Specification Syntax

Label	Function
C0	F = A PLUS Cin
C1	F = (A + B) PLUS Cin
C2	F = (A + $\overline{B}$) PLUS Cin
C3	F = 1 PLUS Cin
C4	F = A PLUS (A · $\overline{B}$) PLUS Cin
C5	F = (A + B) PLUS (A · $\overline{B}$) PLUS Cin
C6	F = A PLUS $\overline{B}$ PLUS Cin
C7	F = $\overline{(A \cdot \overline{B})}$ PLUS Cin
C8	F = A PLUS (A · B) PLUS Cin
C9	F = A PLUS B PLUS Cin
C10	F = (A + $\overline{B}$) PLUS (A · B) PLUS Cin
C11	F = $\overline{(A \cdot B)}$ PLUS Cin
C12	F = A PLUS A PLUS Cin
C13	F = (A + B) PLUS A PLUS Cin
C14	F = (A + $\overline{B}$) PLUS A PLUS Cin
C15	F = $\overline{A}$ PLUS Cin
C16	F = $\overline{A}$
C17	F = $\overline{A + B}$
C18	F = $\overline{A} \cdot B$
C19	F = 0
C20	F = $\overline{A \cdot \overline{B}}$
C21	F = $\overline{B}$
C22	F = $A \oplus B$
C23	F = $A \cdot \overline{B}$
C24	F = $\overline{A} + B$
C25	F = $\overline{A \oplus B}$
C26	F = B
C27	F = A · B
C28	F = 1
C29	F = A + $\overline{B}$
C30	F = A + B
C31	F = A

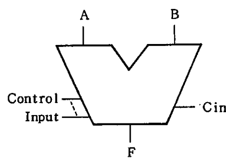


Fig. 2 − ALU function specification example

は，非現実的である[1]．本論文では，文献（1）で示したALUの合成手法を使用する．即ち，

（1）ALU回路構成方式の設定

ALUを多段の論理回路で合成する事を狙いとし，全体の論理を適切に分割，合成する回路構成方式を設定する．

（2）サブ目標の達成

回路構成方式で設定された各サブ問題に対し，機能仕様より，順次，必要な詳細論理を生成し，変形，簡単化，等の処理を行って，解決して行く

ALUのデコーダの論理式は，ALUの回路構成方式に依存する．従って，最適な，制御コードもALU回路構成方式に依存する．本論文では，図3に示す回路構成方式に沿って以後の説明を行う．なお，他の回路構成方式にも，4章の割当アルゴリズムは適用できる． 以後本文中に現れるA，B，F，Carry-gate, G，P等の記号は，図3中のそれに対応するものとする． 図3の回路構成方式の基本思想を示す．

（1）ALUを（イ）出力段に排他的論理和を配し，（ロ）キャリー予測論理，（ハ）G信号生成論理，（ニ）P信号生成論理，（ホ）デコーダ論理，から構成する．Gは，nビット目からのキャリー生成を，Pは，nビット目の部分和を意図している．

（2）算術和実行時には，G及びP信号よりキャリーを予測し，出力段の排他的論理和に，部分和Pと共に印加する．

（3）論理演算実行時には，Carry-gate 信号を“0”とする事によりキャリー予測論理の出力を“0”に抑止する．Pとして，指定された論理演算実行結果が出てくるようP生成論理を構成する．

（4）G及びP生成論理は，万能論理生成回路方式（Universal Logic Implementer）[2] で実現する．

この段階では，デコーダ，G，P等生成論理の

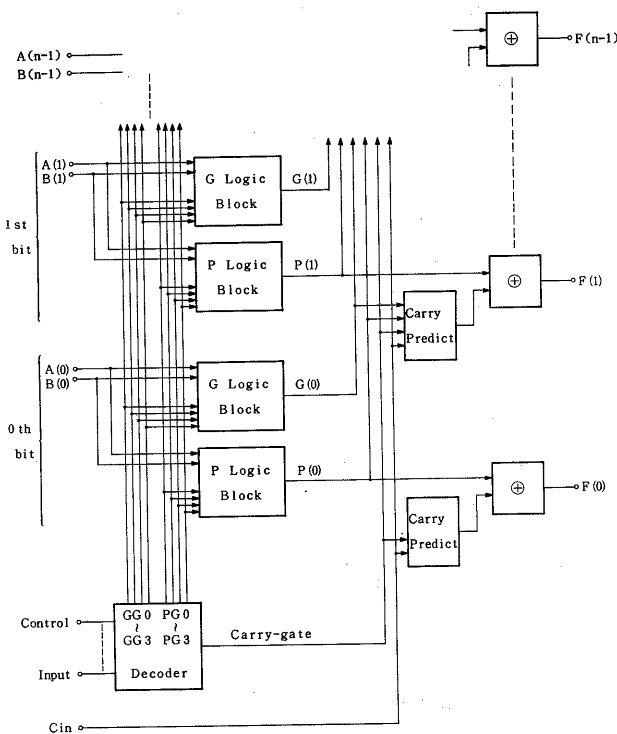


Fig.3- ALU Architecture Example

Carry-gate = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7 +
C8 + C9 + C10 + C11 + C12 + C13 + C14 + C15

GG0 = C8 + C9 + C10 + C11 + C12 + C13 + C14

GG1 = C4 + C5 + C6 + C12 + C13 + C14

GG2 = 0

GG3 = 0

PG0 = C0 + C1 + C2 + C3 + C4 + C5 + C6 + C7 +
C24 + C25 + C26 + C27 + C28 + C29 + C30 + C31

PG1 = C0 + C1 + C2 + C3 + C8 + C9 + C10 + C11 +
C20 + C21 + C22 + C23 + C28 + C29 + C30 + C31

PG2 = C1 + C3 + C5 + C7 + C9 + C11 + C13 + C15 +
C16 + C18 + C20 + C22 + C24 + C26 + C28 + C30

PG3 = C2 + C3 + C6 + C7 + C10 + C11 + C14 + C15 +
C16 + C17 + C20 + C21 + C24 + C25 + C28 + C29

Ci : Label

Fig. 4 — Decoder Boolean expression example

詳細は未定である．具体的機能仕様が与えられて始めて，これら中身が合成される．これら論理の合成手法の詳細は，文献（１）を参照されたい．

図３の中で機能識別子が含まれる論理はデコーダのみである．デコーダで生成される制御信号は，Carry-gate 信号，Ｐ生成回路を制御するPG0～PG3，Ｇ生成回路を制御するGG0～GG3 の合計９本である．PG0~PG3 及GG0～GG3 の論理式は，Ｐ及びＧ生成論理を構成する万能論理生成回路方式に依存する．万能論理生成回路方式は複数種類存在するが，本論文では，文献(1)と同じ形式のものを使用する．

文献（１）の手法を使い，図２の機能仕様を図３の回路構成方式に適用すると，デコーダの論理式として，図４が得られる．図４の論理式がいずれも機能識別子の論理和となっている事に注意されたい．文献（１）の手法によれば，デコーダの論理式は機能識別子の論理和で得られる（文献（１）では，デコーダの論理式簡約化手法としてＰＲＥＳＴＯ[3] を使用している．もし，ＰＲＥＳＴＯのかわりに，ＭＩＮＩ[4]，或いは，ＥＳＰＲＥＳＳＯ[5] 等論理式の否定をとる処理を含む簡約化手法を使用すると，この性質が損なわれることがある）．

従って，制御コードの割当問題は，“機能識別子の論理和で表される複数の論理式が最も簡約になるような制御コードを求めること”と定式化できる．

以下，具体的手法を示す．

４．制御コード割当アルゴリズム

制御コード割当問題はＮＰ完全な問題であるため，ヒューリスティックな手法を用いる．複数の手法を試みたが，その中で最も良い結果の得られた手法（２分割手法）を示す．

4．1　コード割当木

ｍ種類の機能識別子は，次式で与えられるｎビットでコード化できる．

$$n = \lceil \log_2 m \rceil$$

ここで　「　」はガウス関数である

このｎビットをベクトル

$(S_0, S_1 -- S_{n-1})$ で表す事とする．

また，コードの割当状態を図５に示すような２分木（コード割当木）で表すことにする．コード割当木の構成方法は，図５より明らかであろう．即ち，

（１）葉に機能識別子を対応付ける．

（２）ノードにレベル付けを行う．根のノードをレベル０とし，レベルｉのノードの下にぶらさがるノードをレベル $i+1$（この＋は算術和を意味する）とする．

（３）レベルｉのノードの左の枝は $S_i = 0$ であるコード割当に対応させ，右の枝は，$S_i = 1$ であるコード割当に対応させる．

（４）コード $(S_0, -- S_{n-1})$ を持つ機能識別子は，（３）の対応付け規則を，レベル０からレベル $n-1$ まで適用することによりレベルｎの葉に対応付けられる．

従って，コード割当木をたどることにより，コードを容易に判別できる．例えば，図５のｂのコードは（００１）であり（論理式表現では $\overline{S_0} \cdot \overline{S_1} \cdot S_2$ である），ｈのコードは（１１１）である（論理式表現では $S_0 \cdot S_1 \cdot S_2$）である．一般に一番左の葉のコードはａｌｌ“０”であり，一番右の葉は，ａｌｌ“１”となっている．以下，特にことわらないかぎり，コードのビットパターン表現と論理式表現は同一意味とする．

コード割当木について次の性質がある．

（コード割当木の性質）　レベルｉの或るノードからぶらさがるすべての葉に対応付けられた機能識別子の制御コード（の論理式表現）を各々 L_1，L_2 ――，L_j（ｊは２の巾乗）とすると，次式が成立する．

$$L_1 + L_2 + --- L_J = F(S_0) \cdot F(S_1) \cdot -- F(S_{i-1})$$

ここで $F(S_K)$ は根のノードよりこのノードに到達するためのレベルｋにおけるコード割当であり，$S_K = 0$ ならば $\overline{S_K}$ であり，$S_K = 1$ ならば S_K である．即ち L_1，――，L_J は，Ｐ次元（$P = \log_2 j$）キューブの全頂点となっている．従って上記ｊ個の積和論理式は，１つの積項に簡約化できる．

３章で示した様に，デコーダの論理式は，機能識別子の論理和で表されている．これと，コード割当木の性質を考慮すると，制御コード割当法として次の指針が得られる．

（コード割当指針）デコーダの論理式中で，互いに論理和がとられる機能識別子は，なるべく，コード割当木の同一ノードの下にまとまるよう割り当てる．

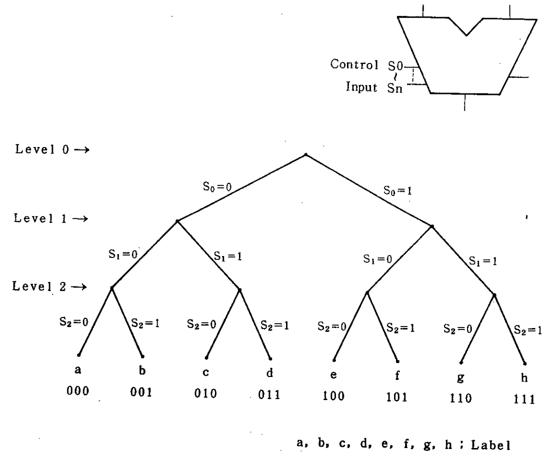


Fig. 5 − Control code assignment tree concept

4．2　コード割当アルゴリズム

4．1で得られたコード割当指針に基づき，機能識別子の全集合をトップダウン的に2分割しながら制御割当木を構成し，制御コードを生成する手法を示す．

図6に処理アルゴリズムを示す．主な特徴を記す．

（1）機能識別子のリストをV-listとする．デコーダの各論理式を構成する機能識別子のリストのリストをIncodeとする．関数Binary-divide にV-listとIncodeを適用することにより，制御構造木を表現するリストを得る．得られた制御構造木と4.1 節の対応付け法に従って制御コードを生成する．

（2）Binary-divide は，入力V-listをLeftとRight の2つのリストに2分割した後，LeftとRight の各々について再帰的にBinary-divide を適用する事により処理を進める．Leftは，制御割当木に於ける左の枝に対応し，Rightは，右の枝に対応する．

（3）機能の種別数mが2 の巾乗に一致しない場合も考慮してある．

$$\text{Left-list-length} = 2^{(\lceil \log_2 m \rceil - 1)}$$

$$\text{Diff} = 2 * \text{Left-list-length} - m \quad \text{とする}$$

mが2の巾乗に一致するならばDiff＝0であり，さもなくば，Diffは正整数である．

（4）もし，Incodeのある要素リストEL-list に対し，V-list とEL-list の積集合が次の条件を満たすならば，この積集合をLeftとし，Right をV-listとLeftの差集合とする．求まったRight,及びLeftにBinary-divide を再帰的に適用する．

<条件> $\text{Left-list-length} \geq |\text{EL-list} \cap \text{V-list}| > \text{Left-list-length} - \text{Diff}$

（5）（4）の条件が成立しないならば，

$$\big|\,|\text{V-list}| - 2 * |\text{EL-list} \cap \text{V-list}|\,\big|$$

を最小とするEL-list を選び

$$\text{Left} = \text{EL-list} \cap \text{V-list}$$
$$\text{Right} = \text{V-list} \sim \text{Left} \quad \text{とする．}$$

ここで∪は和集合をとる演算，∩ は積集合をとる演算，～ は差集合をとる演算，| | は集合の大きさ，或いは，数値の絶対値をとる演算である．

求まったLeftの機能識別子数L が

L ＞ Left-list-length　であるならば，

$$\text{Left-list-length} \geq L \geq \text{Left-list-length} - \text{Diff}$$

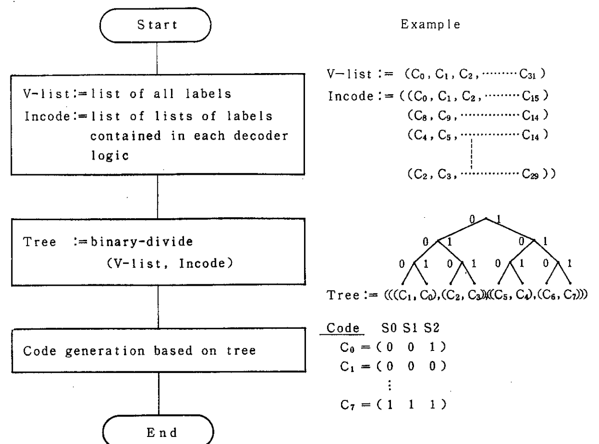
を満たすように，Left の中から機能識別子を抜き出し（関数Select-vで行う)Rightに追加する．

もし，Left-list-length - Diff ＞ L であるならば，

$$\text{Left-list-length} \geq L \geq \text{Left-list-length} - \text{Diff}$$

を満たすように,Rightの中から機能識別子を抜き出し（関数Select-vでおこなう)Left に追加する．

（6）Select-vは，Right 或いはLeftの中からIncodeの各要素リストのなるべくサブセットとなるような機能識別子の集合を選び出す関数である．



```
Function binary-divide (V-list, Incode)

-IF |V-list|<2
 └ THEN
    └Return V-list
-  Left-list-length:=2^(⌈log2|V-list|⌋-1)
-  Diff:=2×Left-list-length − |V-list|
-IF there exist some EL-list in Incode (EL-list ∈ Incode)
 | that satisfy
 | Left-list-length≧|EL-list ∩ V-list|
 |                  ≧Left-list-length − Diff
 ├ THEN
 |  ├ Left :=EL-list ∩ V-list
 |  ├ Right:=V-list ~ Left
 |  └ Return list (binary-divide(Left, Incode),
 |                 binary-divide(Right, Incode))
 └ ELSE
    | for some EL-list in Incode that minimize
    |   ||V-list|−2×|EL-list ∩ V-list||
    ├ Left :=EL-list ∩ V-list
    ├ Right:=V-list ~ Left
    ├ N :=|Left|− Left-list-length
    └ IF N<0
       ├ THEN
       |  ├ Left :=Left ∪ Salect-V (−N, Diff, Right, Incode)
       |  ├ Right:=V-list ~ Left
       |  └ Return list (binary-divide (Left, Incode),
       |                 binary-divide (Right, Incode))
       └ ELSE
          ├ Left :=Left ~ Select-V (−N, Diff, Left, Incode)
          ├ Right:=V-list ~ Left
          └ Return list (binary-divide (Left, Incode),
                         binary-divide (Right, Incode))


Function Select-V (N, Diff, V-list, Incode)
-IF |V-list|<2
 └Return V-list
-IF there exist some EL-list in Incode (EL-list ∈ Incode)
 | that satisfy
 | N−Diff≦|EL-list ∩ V-list|≦N
 ├ THEN
 |  └Return EL-list ∩ V-list
 └ ELSE
    └ IF there exist some EL-list in Incode that satisfy
       |  |EL-list ∩ V-list|≦N-Diff
       |  and that maxmize |EL-list ∩ V-list|
       ├ THEN
       |  ├ X:=EL-list ∩ V-list
       |  └ Return X∪Select-V (N-|X|, Diff,
       |                       V-list~X, Incode)
       └ ELSE
          └Return first Nth of V-list

∪ ; Intersection    ∩ : Union    | | : Length of list/Absolute value
⌈⌋ ; Gauss function
```

Fig.6- Control Code Assignment Algorithm

5. 実験例

図2で示した例題に本手法を適用して得られた制御コードを図7に示す.

表1に, 自動生成されたコードと, 人手設定コード, 任意設定コード(仕様を見ずに適当に設定したコード)との比較をしめす. 8機能を有するALUと32機能を有するALUの2例について比較した. 表1は, 人手設定コード, 自動生成コード, 任意設定コード, の各々について図3の回路構成方式に従って, ALUのデコーダ論理を自動生成し, その, 積項数, 平均ファンイン/積項を求めたものである. いずれの場合においても, 自動生成コードの積項数, 平均ファンイン/積項が最も小さくなった. ただし, 人手設定コードは, 図3に示した回路構成方式を前提にして設計されたものではないため, 本論文の手法が人手設定より常にすぐれているという事は, 結論できない. しかし, 自動生成されたコードが人手設定に匹敵する質を持つと言えよう.

制御コード生成時間は, 2MIPSの計算機を使い, 32機能ALUで160秒, 8機能ALUで4秒であった. なお, 制御コード生成時間の大部分は, 機能仕様よりデコーダの論理式を導く時間であった.

人手設定コード, 自動設計コード, 任意設定コードに基づき合成された32機能ALUのデコーダの論理式を図8に示す.

6. むすび

ALUの制御コードを自動割当する手法を示した.

(1) ALUの機能仕様を機能識別子とデータ入出力端子間の関数関係で与える,

(2) ALUの回路構成方式を設定する,

(3) 機能仕様と, 回路構成方式に基づきALUのデコーダの論理式を導く,

Label	Synthesized code (S0 S1 S2 S2 S3)	Manualy set code	Arbitrary set code
C0	(0 1 1 0 1)	(0 0 0 0 0)	(0 0 0 0 0)
C1	(0 1 0 0 1)	(0 0 0 0 1)	(0 0 0 1 1)
C2	(0 0 1 0 1)	(0 0 0 1 0)	(0 0 0 1 0)
C3	(0 0 0 0 1)	(0 0 0 1 1)	(0 0 0 0 1)
C4	(0 1 1 1 1)	(0 0 1 0 0)	(1 0 0 0 0)
C5	(0 1 0 1 1)	(0 0 1 0 1)	(1 0 0 1 1)
C6	(0 0 1 1 1)	(0 0 1 1 0)	(1 0 0 1 0)
C7	(0 0 0 1 1)	(0 0 1 1 1)	(1 0 0 0 1)
C8	(0 1 1 0 0)	(0 1 0 0 0)	(0 0 1 0 1)
C9	(0 1 0 0 0)	(0 1 0 0 1)	(0 0 1 1 1)
C10	(0 0 1 0 0)	(0 1 0 1 0)	(0 0 1 1 0)
C11	(0 0 0 0 0)	(0 1 0 1 1)	(0 0 1 0 1)
C12	(0 1 1 1 0)	(0 1 1 0 0)	(1 1 1 1 1)
C13	(0 1 0 1 0)	(0 1 1 0 1)	(0 1 0 0 0)
C14	(0 0 1 1 0)	(0 1 1 1 0)	(0 1 0 1 0)
C15	(0 0 0 1 0)	(0 1 1 1 1)	(0 1 1 1 0)
C16	(1 0 0 1 1)	(1 0 0 0 0)	(0 1 0 1 1)
C17	(1 0 1 1 1)	(1 0 0 0 1)	(0 1 1 0 0)
C18	(1 1 0 1 1)	(1 0 0 1 0)	(0 1 1 1 1)
C19	(1 1 1 1 1)	(1 0 0 1 1)	(0 1 0 0 1)
C20	(1 0 0 0 1)	(1 0 1 0 0)	(0 1 1 0 1)
C21	(1 0 1 0 1)	(1 0 1 0 1)	(1 0 1 0 0)
C22	(1 1 0 0 1)	(1 0 1 1 0)	(1 0 1 0 1)
C23	(1 1 1 0 1)	(1 0 1 1 1)	(1 0 1 1 0)
C24	(1 0 0 1 0)	(1 1 0 0 0)	(1 0 1 1 1)
C25	(1 0 1 1 0)	(1 1 0 0 1)	(1 1 0 0 0)
C26	(1 1 0 1 0)	(1 1 0 1 0)	(1 1 0 0 1)
C27	(1 1 1 1 0)	(1 1 0 1 1)	(1 1 0 1 0)
C28	(1 0 0 0 0)	(1 1 1 0 0)	(1 1 0 1 1)
C29	(1 0 1 0 0)	(1 1 1 0 1)	(1 1 1 0 0)
C30	(1 1 0 0 0)	(1 1 1 1 0)	(1 1 1 0 1)
C31	(1 1 1 0 0)	(1 1 1 1 1)	(1 1 1 1 0)

Fig.7- Control Code Assignment Example

$\text{Carry-gate} = \overline{S0}$
$GG0 = S1 \cdot \overline{S4} + S2 \cdot \overline{S4}$
$GG1 = S1 \cdot S3 + S2 \cdot S3$
$PG0 = S0 \cdot \overline{S4} + \overline{S0} \cdot S4$
$PG1 = \overline{S3}$
$PG2 = \overline{S2}$
$PG3 = \overline{S1}$

(i) Decoder logic based on synthesized code

$\text{Carry-gate} = \overline{S0}$
$GG0 = S1 \cdot \overline{S4} + S1 \cdot \overline{S3}$
$GG1 = S2 \cdot \overline{S3} + S2 \cdot \overline{S4}$
$PG0 = \overline{S0} \cdot \overline{S1} + S0 \cdot S1$
$PG1 = \overline{S0} \cdot \overline{S2} + S0 \cdot S2$
$PG2 = \overline{S0} \cdot S4 + S0 \cdot \overline{S4}$
$PG3 = \overline{S0} \cdot S3 + S0 \cdot \overline{S3}$

(ii) Decoder logic based on manualy set code

$\text{Carry-gate} = S0 \cdot S3 \cdot \overline{S4} + \overline{S0} \cdot \overline{S2} \cdot \overline{S4} + S0 \cdot S1 \cdot S3 \cdot S4 + \overline{S0} \cdot \overline{S1} \cdot S4 + \overline{S0} \cdot \overline{S1} \cdot S3 + \overline{S1} \cdot \overline{S2}$
$GG0 = S2 \cdot S4 + \overline{S1} \cdot S2 \cdot S3 + S0 \cdot S1 + S1 \cdot \overline{S2}$
$GG1 = S0 \cdot \overline{S4} + S0 \cdot S3 + S0 \cdot S1 + S1 \cdot \overline{S2}$
$PG0 = S0 \cdot S1 \cdot \overline{S4} + S0 \cdot S1 \cdot \overline{S3} + S0 \cdot \overline{S1} \cdot S3 \cdot S4 + S0 \cdot \overline{S2} + \overline{S1} \cdot \overline{S2}$
$PG1 = S0 \cdot S2 \cdot \overline{S3} + S0 \cdot S1 \cdot \overline{S2} \cdot S3 \cdot S4 + S0 \cdot S2 \cdot \overline{S4} + S2 \cdot \overline{S3} \cdot S4 + \overline{S0} \cdot \overline{S1} \cdot S3 + \overline{S0} \cdot \overline{S1} \cdot S4 + \overline{S0} \cdot \overline{S1} \cdot \overline{S2}$
$PG2 = S0 \cdot \overline{S3} \cdot S4 + S0 \cdot \overline{S2} \cdot S4 + S0 \cdot S2 \cdot S4 + \overline{S0} \cdot S1 \cdot S2 \cdot S3 + \overline{S2} \cdot S3 \cdot S4 + \overline{S0} \cdot S1 \cdot \overline{S2} \cdot \overline{S3} \cdot \overline{S4} + \overline{S1} \cdot S4$
$PG3 = S0 \cdot S2 \cdot \overline{S3} \cdot \overline{S4} + S1 \cdot \overline{S2} \cdot S3 \cdot S4 + S0 \cdot S1 \cdot \overline{S3} \cdot \overline{S4} + S0 \cdot \overline{S1} \cdot S2 \cdot S3 \cdot S4 + \overline{S0} \cdot S2 \cdot \overline{S3} \cdot S4 + \overline{S0} \cdot S1 \cdot S2 \cdot \overline{S3} + \overline{S0} \cdot S1 \cdot S2 \cdot \overline{S4} + \overline{S0} \cdot S1 \cdot \overline{S2} \cdot S3 + \overline{S0} \cdot \overline{S1} \cdot \overline{S3} \cdot \overline{S4} + \overline{S0} \cdot S3 \cdot \overline{S4} + \overline{S1} \cdot \overline{S2} \cdot \overline{S3} \cdot S4 + \overline{S1} \cdot \overline{S2} \cdot S3 \cdot \overline{S4}$

(iii) Decoder logic based on arbitrary set code

Fig.8- Decoder Logic Example

表１．ＡＬＵ制御コードの比較

例題	人手設定コード		自動生成コード		ランダム設定コード	
	積項数	Fan-in/ 積項	積項数	Fan-in/ 積項	積項数	Fan-in/ 積項
32機能 ALU	１２	2	6	2	４５	4
8 機能 ALU	１６	2．2	6	2．1	１５	2．2

（４）デコーダの論理式に含まれる機能識別子を制御コードで置換し簡約化を行った時の積項数が最小になるようコード割当を行う，

（５）コード割当は，ＮＰ完全な問題であるため，発見的手法（２分割手法）を採用し準最適解を求める，ことにより，人手設計に匹敵する制御コードを生成する事ができた．

３２機能を有するＡＬＵの制御コード生成時間は，約１６０秒であり，十分，実用的な範囲に納まっていと考える．

今後，より多くの例題について２分割法の有効性を確認するとともに，制御回路の合成における状態割当問題等に本手法の適用を検討していく予定である．

＜文献＞

（１）髙木：“ＡＬＵの論理合成法”設計自動化研究会資料２４—７
（２）Winkel,D.,Prosser,F.:"The Art of Digital Design",pp.84-86 Prentice-Hall,N.J.(1980)
（３）BROWN,D.W.:"A STATE-MACHINE SYNTHESIZER" ,Proc.18th DA Conference, PP.301-305(1981)
（４）Hong,S.J.,Cain,R.G.,Ostapko,D.L.:"MINI : A Heuristic Approach for Logic Minimization", IBM J.RES.DEVELOP., PP.443-458(1974)
（５）Brayton,R.K.,Hachtel,G.D.,Hemachandra,L.A.,Newton,A.R., Sangiovanni-Vincentelli.A.L.M. :"A Comparison of Logic Minimization Strategies Using ESPRESSO:An APL Program Package for Partitioned Logic Minimization2", Proc. ISCAS Rome,pp.42-48(1982)