設計自動化 24－7
（1984. 11. 20）

ALUの論理合成法
(設計手法ベース論理合成法)

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

論理設計工数削減のため，長年，組合せ回路自動合成技術が研究されてきた．

従来の組合せ回路自動合成技術では，合成すべき回路の仕様をブール式，或いは，真理値表等のプリミティブなレベルで与え，これをキューブ等の内部表現に変換し，論理の簡約化[1][2][3]，因子化を行った後，回路に変換する[4][5][6]手法が主流である．しかし，この手法で扱えるのは，最終解の積項数が高々数百程度となる，いわゆる，浅い論理関数である．制御系（ランダムロジック）の仕様は浅い論理関数になることが多いのに対し，データパス系の機能ブロックは多くの場合深い論理関数となる．即ち，データパス系機能ブロックの仕様を表す真理値表，或いはブール式の量は，データ幅増加と共に，爆発的に増大し，メモリ量，計算量ネックとなる．このため，従来技術は主として，制御系の合成のみにしか使えないのが現状である．従来技術は，『問題を最もプリミティブなレベルに分解し，膨大なデータを，ある1つのヒューリスティクな手順に従って解く』アプローチと近似できる．

一方，論理設計者はこれとは異なった方法で設計を行っている．設計者は『長年の設計事例，経験を知識として蓄積し，これを活用することにより効率的に問題を解決する』アプローチをとっている．設計対象の多くは，以前に解いた（或いは，似た問題を解いた）ことのある問題である．設計事例の蓄積，活用のアプローチは2通り考えられる．

（1）設計事例のインスタンス（詳細な設計済み回路）と設計修正知識を記憶しておく．仕様が与えられたとき，似た設計事例を探し，仕様に差異が存在する場合は，設計修正知識を活用し，その差異をうめるようインスタンスを変更する．

（2）抽象的な回路の概略構成（回路構成方式）と必要に応じその細部を再現（設計）できる手続きの組合せを記憶しておく．仕様が与えられたとき，これら知識を活用しトップダウン的に設計を進める．

本論文では，（イ）設計インスタンスを記憶する事は，メモリ量が増加すること，及び，（ロ）設計修正知識を系統的に蓄積する事は難しそうなこと，より後者の立場に立って設計知識を定式化し，これを活用することを提案する．

データパス系の機能ブロックについては，これら設計知識の体系化も進み，多数の論理回路構成方式が教科書的に蓄積されてきている．これは，データパス系の機能ブロックは，ハードウェアの性能（論理段数），ゲート量の支配項である事，及び，比較的その機能のレパートリィが限られている（専用的である）ものの，その機能を実現する論理回路のバリエーションは極めて多く，またバリエーションにより，ゲート量，論理段数等は大きく異なるためである．例えば，加算器についても，リップル加算器，キャリィセーブ加算器，キャリー予測加算器等，様々な回路構成方式が存在する．さらに，例えば，キャリー予測加算器であっても，キャリーの予測の範囲に関してバリエーションがある．これらを選択，或いは組み合わせて設計を行うことにより，様々な性能，ゲート量の加算回路を実現できる．

本論文のアプローチによれば，少ない計算量で，所与の仕様を満たす回路を合成しうることをしめす．以後，これを設計手法ベース論理合成法と呼ぶ．また，設計手法ベース論理合成法では，設計者が考案（発明）した新設計手法を蓄積することによりＤＡシステムの問題解決能力をインクリメンタルに増加できる利点がある．

第2章では，設計手法ベース論理合成法の概念を示す．設計手法は機能ブロックの種類毎に異なる．3章では，ＡＬＵを具体例に本手法の具体的内容を示す．

2．設計手法ベース論理合成法の概念

機能ブロックの仕様は，広義には，入出力端子間の関数関係（機能仕様と呼ぶ）のほかに，論理段数，ゲート数，更には，評価尺度のはっきりしない試験容易性，論理の規則性，等様々な側面から規定されうる．これら，仕様を規定する項目の

中には，互いに相反の関係のものも多い．どの項目にどの程度重きを置くかによって，様々な回路構成方式が考案されて来ている．

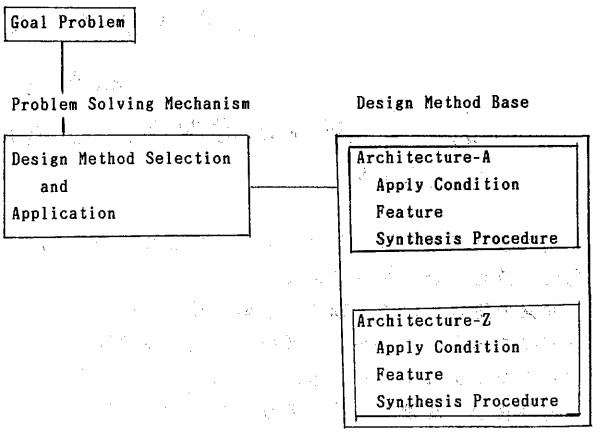
設計手法ベース論理合成システムの概念構成は，各回路構成方式に沿って論理回路を生成する手続き（以後設計手法と呼ぶ）の集合である設計手法ベースと，与えられた仕様に対し，適切な生成手続きを選択し，適用する問題解決機構でモデル化できる（図１）

（１）設計手法

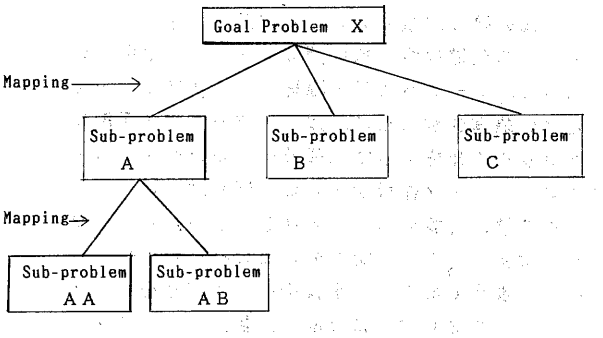
設計手法は，（イ）回路構成方式と（ロ）論理回路変換・合成手続きの組合せとして表現される．回路構成方式は，サブ問題化技法の一種と捉えられる．即ち，大きな問題は複数のより小さな問題（サブ問題）に分解して解く事が良く行われる（図２）．最初の問題Ｘが複数のサブ問題（Ａ，Ｂ，Ｃ）に分解され，また，サブ問題が更に小さなサブ問題（ＡＡ，ＢＢ）に分解される．最下位のサブ問題は，プリミティブな要素の組合せで実現される．各サブ問題を解き，結合する事により全体の解が得られる．論理回路変換・合成手続きは，元の問題からサブ問題への変換手続き，或いは，最下位のサブ問題をプリミティブな要素で合成する手続きに対応する．

設計手法は，仕様を入力とし，論理式を出力する．最も簡単な例として，リップル加算器の設計手法を付録に示す．リップル加算器設計手法は，最下位の問題（サブ問題が存在しない）であり，従って，論理合成手続きのみからなる．この合成手続きは，任意のデータ幅のリップル加算器の論理式を生成できる．設計手法は一般には，階層的に構成される．例えば，乗算器の設計手法は，加算器の設計手法を内部で参照する．

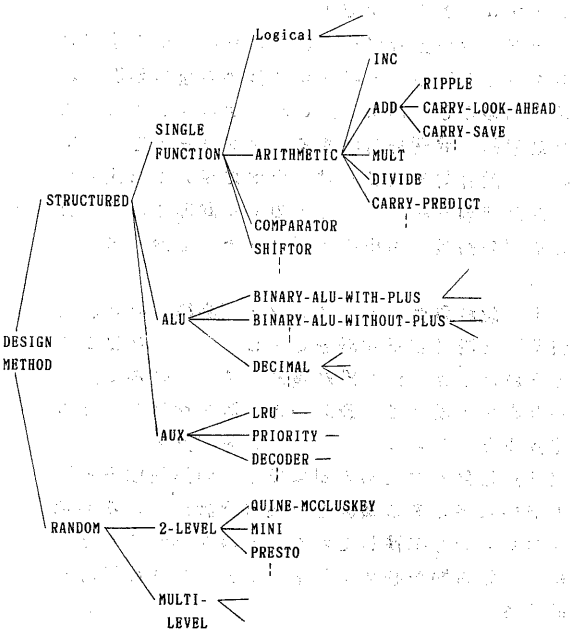
設計手法の分類例を図３に示す．図３では，機能ブロックを，Random Logic,Structured Logic,に分類している．ＭＩＮＩ[1]，ＰＲＥＳＴＯ[2]等の従来の合成手法を，ランダム論理の１設計手法に分類している．Structured Logic は更に，加算器，乗算器，に代表される単機能ブロックと，ＡＬＵに代表される複合機能ブロック，及び，その他，に，大別される．単機能ブロックは，付録に示したような，比較的単純な論理式合成手続きで実現される場合が多い．これに対し，ＡＬＵでは，論理式変形操作，或いは，従来の発見的な合成手法を部分的に適用する等，様々な処理を必要とすることが多い．ＡＬＵは更に加算機能を含むか，シフト機能を含むか等により細分される．



Ｆｉｇ．１　Design Method Base Logic Synthsis System Concept



Ｆｉｇ．２　Problem Reduction Concept



Ｆｉｇ．３　Design Method Example

ＡＬＵの細分類の１つであるBinary-alu-with-plus（加算機能は含むが，シフト機能は含まない）にも複数の設計手法が存在する．更にこれらＡＬＵの設計手法は，内部でＭＩＮＩ，ＰＲＥＳＴＯ等のRandom Logic設計手法，或いはCarry-predict 設計手法等を参照する．

一般に，各サブ問題を解決する手法は，複数存在する．解決手法を組み合わせることにより，同一機能仕様に対し様々なゲート量，性能の回路を生成しうる．

（２）問題解決機構

１つの設計手法は，ある限られた範囲の機能仕様を実現することを前提としている（受理できる機能仕様範囲と呼ぶ）．また，１つの設計手法は，論理設計空間（ゲート量，論理段数等の評価パラメータを軸に持つ空間）のある領域を離散的にカバーする手法を与えるに過ぎず，望ましい仕様を正確に実現できるとは限らない．設計者は，目標である論理段数，ゲート数を実現するため，経験の設計手法を組み合わせて使用することもよく行っている．従って，与えられた仕様を正確に満たす回路を合成しようとすると次のことを実現する必要がある．

（イ）設計手法毎に，その手法が受理しうる機能仕様の範囲，論理設計空間内でカバーしそうな範囲，重点を置いている項目を記述できること．

（ロ）仕様が与えられた時に，（イ）の情報に基づき最適な（と予想される）設計手法を選択し適用する．適用結果が仕様を完全には満たさない場合には，他の設計手法を選択する．それでも満たさない場合には，複数の設計手法を組合せて実現する，などの試行錯誤メカニズムを設ける．

設計手法ベース論理合成システム構築に必要な，問題解決メカニズムについては別途検討の上報告する．したがって，本論文では，仕様としては，機能仕様のみを考慮する．更に，設計手法の種類は多数存在するため，これらを効率良く記述出来る言語の設定が好ましいが，これについても別途報告する事としたい．

本論文は，ＡＬＵの合成問題を例として，回路構成方式に沿った設計手法が確かに存在し，かつ，論理合成に極めて有効であることをしめす．ＡＬＵはデータパス系機能ブロックの中では，機能仕様も複雑であり，また，設計手法の設定の難しい問題と考えられる．従って，この問題を例にとることにより，他の機能ブロックの設計手法の実現法も容易に類推できると期待される．

３．設計手法の具体例：ＡＬＵの合成

３．１　機能仕様の記述法

以後，２項ＡＬＵの合成問題を例題とする．ＡＬＵの仕様をブール式，或いは，真理値表のような低レベルな表現法で記述すると，データ幅の増加と共に，これらブール式，或いは，真理値表の量は爆発的に増大し，現実的でない，よりハイレベルな機能仕様の記述法をしめす．

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，従ってＡＬＵの機能仕様は，制御信号のパターン（制御コードと呼ぶ）と，その時のデータ入力端子間の関数関係で記述できる．例えば，データ入力端子Ａ，Ｂ，データ出力端子Ｆを持ち，制御入力端子Ｓ０，Ｓ１に各々“０”，“１”の制御信号が加えられた時，排他的論理和を実行するＡＬＵの機能は次の様に記述できる．

$$\overline{S0}\cdot S1 \qquad F = A \oplus B$$

ここでデータ入力端子Ａ，Ｂ，データ出力端子Ｆはｎビット幅を持つのに対し，制御信号端子は各々１ビット幅とする．一般にＡＬＵは複数機能を有するので上記，記述例の集合となる．また，仕様の一部として，ゲート量，速度，等についての制約条件が付けられるのが普通であるが，本論文の段階では，これらは，考慮しない．

一般に，受理できる機能仕様の範囲を広くする程，回路構成方式の設定は，困難になる．適切な範囲を設定することが重要である．図４に，設定例を示す．図４は，受理できる演算子の種類と機能仕様の構文を示す．本論文では比較演算子，シフト演算子は考慮していない．これらを含むよう拡張するためには，第３．２章の回路構成方式設定を変更する必要がある．

また，算術減算は，算術加算で表現し直されていると仮定した．算術減算を直接扱えるようにするには，算術加算に変換する前処理を施す必要がある．

図５に，機能仕様の例を示す（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５は，本論文で合成の例題に使う３２種類の機能を有するＡＬＵの機能仕様である．

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

<Function SPEC> : : = {<A Function SPEC>} ;
<A Function SPEC> : : = <Control Code><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 | $\overline{\text{<Logic EXP>}}$ |
<Logic EXP><Logic Operator><Logic EXP> ;
<Control Code> : : = Product-term-of-control-input ;

Fig.4 Operators and Function Specification Syntax

Control Code	Function
$S4 \cdot S3 \cdot S2 \cdot S1 \cdot S0$	$F = A$
$S4 \cdot S3 \cdot S2 \cdot S1 \cdot \overline{S0}$	$F = A + B$
$S4 \cdot S3 \cdot S2 \cdot \overline{S1} \cdot S0$	$F = A + \overline{B}$
$S4 \cdot S3 \cdot S2 \cdot \overline{S1} \cdot \overline{S0}$	$F = 1$
$S4 \cdot S3 \cdot \overline{S2} \cdot S1 \cdot S0$	$F = A \cdot B$
$S4 \cdot S3 \cdot \overline{S2} \cdot S1 \cdot \overline{S0}$	$F = B$
$S4 \cdot S3 \cdot \overline{S2} \cdot \overline{S1} \cdot S0$	$F = \overline{A \oplus B}$
$S4 \cdot S3 \cdot \overline{S2} \cdot \overline{S1} \cdot \overline{S0}$	$F = \overline{A} + B$
$S4 \cdot \overline{S3} \cdot S2 \cdot S1 \cdot S0$	$F = A \cdot \overline{B}$
$S4 \cdot \overline{S3} \cdot S2 \cdot S1 \cdot \overline{S0}$	$F = A \oplus B$
$S4 \cdot \overline{S3} \cdot S2 \cdot \overline{S1} \cdot S0$	$F = \overline{B}$
$S4 \cdot \overline{S3} \cdot S2 \cdot \overline{S1} \cdot \overline{S0}$	$F = \overline{A \cdot B}$
$S4 \cdot \overline{S3} \cdot \overline{S2} \cdot S1 \cdot S0$	$F = 0$
$S4 \cdot \overline{S3} \cdot \overline{S2} \cdot S1 \cdot \overline{S0}$	$F = \overline{A} \cdot B$
$S4 \cdot \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0$	$F = \overline{A + B}$
$S4 \cdot \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot \overline{S0}$	$F = \overline{A}$
$\overline{S4} \cdot S3 \cdot S2 \cdot S1 \cdot S0$	$F = \overline{A}$ PLUS Cin
$\overline{S4} \cdot S3 \cdot S2 \cdot S1 \cdot \overline{S0}$	$F = (A + \overline{B})$ PLUS A PLUS Cin
$\overline{S4} \cdot S3 \cdot S2 \cdot \overline{S1} \cdot S0$	$F = (A + B)$ PLUS A PLUS Cin
$\overline{S4} \cdot S3 \cdot S2 \cdot \overline{S1} \cdot \overline{S0}$	$F = A$ PLUS A PLUS Cin
$\overline{S4} \cdot S3 \cdot \overline{S2} \cdot S1 \cdot S0$	$F = \overline{(A \cdot B)}$ PLUS Cin
$\overline{S4} \cdot S3 \cdot \overline{S2} \cdot S1 \cdot \overline{S0}$	$F = (A + \overline{B})$ PLUS $(A \cdot B)$ PLUS Cin
$\overline{S4} \cdot S3 \cdot \overline{S2} \cdot \overline{S1} \cdot S0$	$F = A$ PLUS B PLUS Cin
$\overline{S4} \cdot S3 \cdot \overline{S2} \cdot \overline{S1} \cdot \overline{S0}$	$F = A$ PLUS $(A \cdot B)$ PLUS Cin
$\overline{S4} \cdot \overline{S3} \cdot S2 \cdot S1 \cdot S0$	$F = \overline{(A \cdot \overline{B})}$ PLUS Cin
$\overline{S4} \cdot \overline{S3} \cdot S2 \cdot S1 \cdot \overline{S0}$	$F = A$ PLUS $\overline{B}$ PLUS Cin
$\overline{S4} \cdot \overline{S3} \cdot S2 \cdot \overline{S1} \cdot S0$	$F = (A + B)$ PLUS $(A \cdot \overline{B})$ PLUS Cin
$\overline{S4} \cdot \overline{S3} \cdot S2 \cdot \overline{S1} \cdot \overline{S0}$	$F = A$ PLUS $(A \cdot \overline{B})$ PLUS Cin
$\overline{S4} \cdot \overline{S3} \cdot \overline{S2} \cdot S1 \cdot S0$	$F = 1$ PLUS Cin
$\overline{S4} \cdot \overline{S3} \cdot \overline{S2} \cdot S1 \cdot \overline{S0}$	$F = (A + \overline{B})$ PLUS Cin
$\overline{S4} \cdot \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot S0$	$F = (A + B)$ PLUS Cin
$\overline{S4} \cdot \overline{S3} \cdot \overline{S2} \cdot \overline{S1} \cdot \overline{S0}$	$F = A$ PLUS Cin

Fig.5 ALU Function Specification Example

3．2 回路構成方式と論理回路変換・合成手続き

図４の範囲で記述された任意の機能仕様を受理できる回路構成方式と，それに沿った論理回路合成・変換手続きの例をしめす．

3．2．1 回路構成方式の設定

図４の機能仕様の範囲を受理しうる，ＡＬＵの回路構成方式は，多数考えられる．すべての，機能仕様に対し，最適解を合成しうる回路構成方式が存在するとは考えづらい．　機能仕様が与えられた時に，最適な回路構成方式を選択する方法はまだ得られていない．ここでは，いくつかの実験を通じ，比較的良い結果の得られた回路構成方式を例に示す．

図６にＡＬＵの回路構成方式の例を示す．図６は，図３中の設計手法Binary-Alu-with-Plusの１種類である．以後，本文中に現れるＡ，Ｂ，Ｆ，Ｇ，Ｐ，Carry-gate等の記号は，図６中のそれに対応するものとする．

この回路構成方式は，機能仕様に算術和の機能が含まれている時に適用できる方式である．この回路構成方式の基本的思想を次に示す．

（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およびＰ生成論理は，万能論理生成回路方式（Universal Logic Implementer）[7] で実現する．

回路構成方式の段階では，デコーダ，Ｇ，Ｐ等生成論理の詳細はブラックボックスである．具体的機能仕様に基づき，これらの中身を合成してゆく．

図６に示したBinary-Alu-with-Plusのバリエーションとして，例えば，次の様な回路構成方式も考えられる．

（１）Ｇ（及びＰ）生成論理とデコーダを分離せず，１つの論理として合成する回路構成方式

（２）出力段の排他的論理和の前段に更に排他的論理和を配する回路構成方式

しかし，図５の例題では，これらバリエーションより図６の回路構成方式に基づいて合成したほうがゲート量は，少なくなった．

３．２．２　論理回路合成・変換手続き

具体的機能仕様が与えられたときに，図６に示した回路構成方式に沿って各サブブロックの論理仕様を導き，解決し，全体の論理を生成する論理回路合成・変換手続きを図７に示す．図７では，各サブ問題と合成・変換手続きの対応付けも示している．

以下，各手続きについて順を追って詳述する．

（１）中間論理生成手続き

Ｇ，Ｐ，及び，Carry-gate 信号の論理を生成する手続きである．Ｇ及びＰは，データのどのビット位置でも論理の形は同じであるので１ビット分だけ考慮する．

まず，Ｇ，Ｐ，及びCarry-gateを“0”に初期設定する．次に，機能仕様の各々について，その構文のタイプを調べ，タイプ毎に以下に示す規則に従い，Ｇ，Ｐ，Carry-gateの論理を合成してゆく．

（ｉ）算術和タイプ

２項ＡＬＵを想定した場合，２種のタイプにわかれる（図４参照）．

＜算術和タイプ１＞　次の構文を持つタイプである

制御コード　　Ｆ＝∝ ＰＬＵＳ　Ｃin

この構文に対して次の論理合成規則を適用する．尚，ここで制御コードは図５で示したように，制御入力の論理積で表されていることに注意されたい．

「論理合成規則１」

Ｐ＝Ｐ　＋　制御コード・∝

Carry-gate ＝ Carry-gate＋制御コード

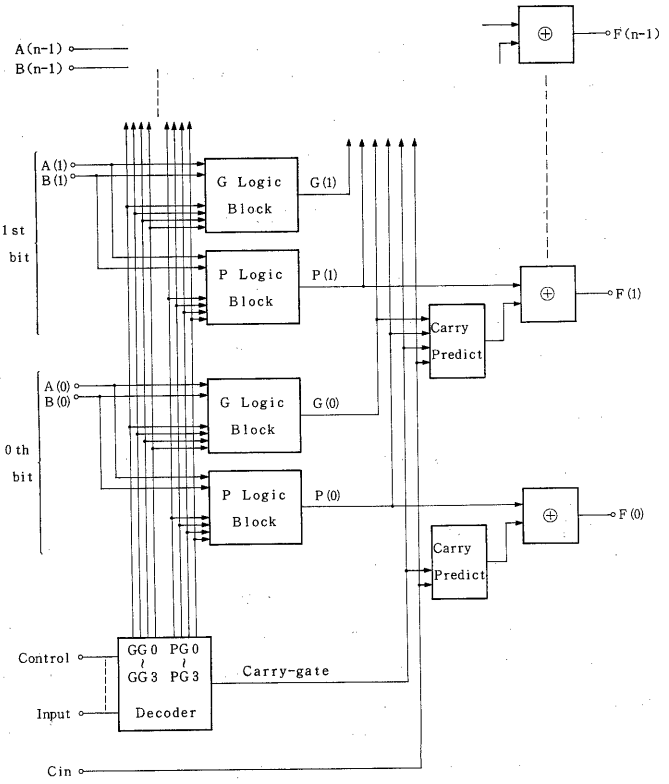


Fig.6　ALU Architecture Example

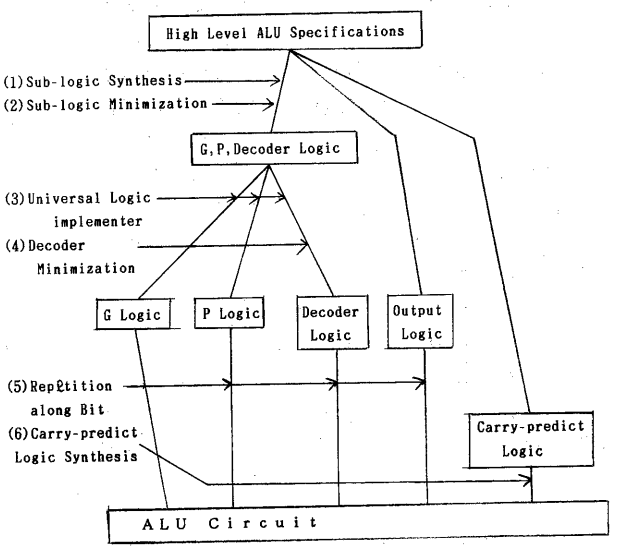


Fig.7　ALU Synthesis Procedure

ここでαは，PLUSを含まない式である．
また，「P＝P＋制御コード・α」は，論理式Pと制御コード・α(制御コードとの論理積）の論理和をPとすることを意味する．

＜算術和タイプ２＞　次の構文を持つタイプとする

制御コード　F＝α PLUS β PLUS　Cin

『論理合成規則２』

P＝P　＋　制御コード・($\alpha \oplus \beta$)

G＝G　＋　制御コード・(α ・β)

Carry-gate=Carry-gate ＋ 制御コード

ここでα,β はPLUS を含まない式である．

(ii) 論理演算タイプ

算術和を含まないタイプである．

制御コード　　F＝α

『論理合成規則３』

P＝P　＋　制御コード・α

G＝don't care　When　制御コード

ここで，「G=don't care When 制御コード」は制御コードが"1" の値をとるとき，Carry-gateの値は，don't careであることを意味する．

（２）中間論理の簡約化手続き

（１）で得られた，G，P，Carry-gate の論理を簡約化する．簡約化手法としてＰＲＥＳＴＯ[2] を使用する．

（３）万能論理生成形式への変換手続き

（２）の処理の結果，P，Gの積和標準形のブール式が得られる．このステップにおけるP，Gの処理方法は同じであるのでPのみについて示す．P生成回路を図８に示す様な万能論理生成回路と，それへの制御信号PG0～PG3　という回路形式で実現する（万能論理生成回路は複数種類存在し，図８はその１種類である．他の種類については，文献（７）を参照されたい）．そのため，PG0～PG3　の論理を求める．

まず，PG0～PG3　を"0" に初期設定する．次に，Pのブール式の各積項毎に入力データA，B，或いは，$\overline{A}$，$\overline{B}$の含まれ方を調べ，含まれかたによって，表１に示す規則に従って，PG0～PG3　の論理を合成してゆく．表１においてＲＥＳＴとは，A，B，$\overline{A}$，$\overline{B}$，を含まない積項である．

（４）デコード論理の簡約化手続き

（３）で求めた制御信号PG0～PG3　の簡約化を行う事によりデコーダの論理式を得る．簡約化手法としてＰＲＥＳＴＯを使用する．

（５）ビット列方向への繰り返し手続き

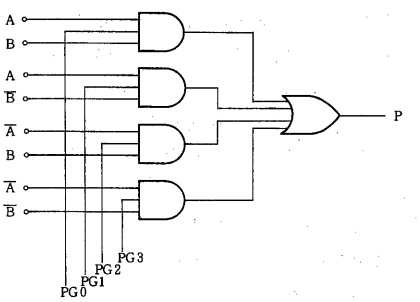


Fig.8　P Logic by Universal Logic Implementer

表１．万能論理生成回路制御信号合成規則

積項の形式	PG0～PG3　合成規則
A・B・REST	PG0 ＝ PG0 ＋ REST
A・$\overline{B}$・REST	PG1 ＝ PG1 ＋ REST
$\overline{A}$・B・REST	PG2 ＝ PG2 ＋ REST
$\overline{A}$・$\overline{B}$・REST	PG3 ＝ PG3 ＋ REST
A・REST	PG0 ＝ PG0 ＋ REST PG1 ＝ PG1 ＋ REST
B・REST	PG0 ＝ PG0 ＋ REST PG2 ＝ PG2 ＋ REST
$\overline{A}$・REST	PG2 ＝ PG2 ＋ REST PG3 ＝ PG3 ＋ REST
$\overline{B}$・REST	PG1 ＝ PG1 ＋ REST PG3 ＝ PG3 ＋ REST
REST	PG0 ＝ PG0 ＋ REST PG1 ＝ PG1 ＋ REST PG2 ＝ PG2 ＋ REST PG3 ＝ PG３＋ REST

G，P，出力論理をデータ幅ｎビット分繰り返す．

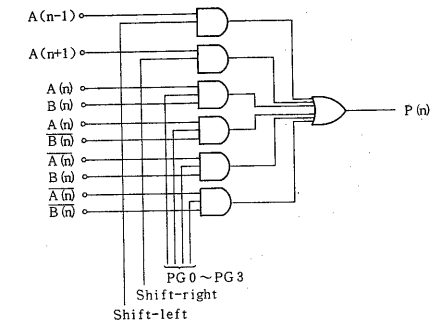
（６）キャリー予測論理の作成手続き

Carry-predict 回路構成方式を使用する．Carry-gate 付のキャリー予測論理は次の漸化式により，G，P，Carry-gateの論理式として表される（０はLSB であり，nの大きいほどMSB 側とする）

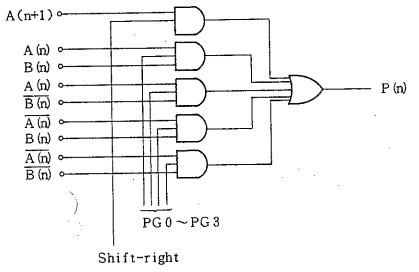
Carry(0) ＝ Cin・Carry-gate

Carry(n) ＝ G(n-1) ・Carry-gate
＋P(n-1)・Carry(n-1)

キャリー予測の範囲をパラメータとして設定可能なキャリー予測論理生成手続きにより，上記論理と等価なブール式を生成する．



(i) Special ports for both Shift-right and Shift-left function



(ii) Special port for Shift-right function

Fig.9 P Logic Expansion for Shift Function

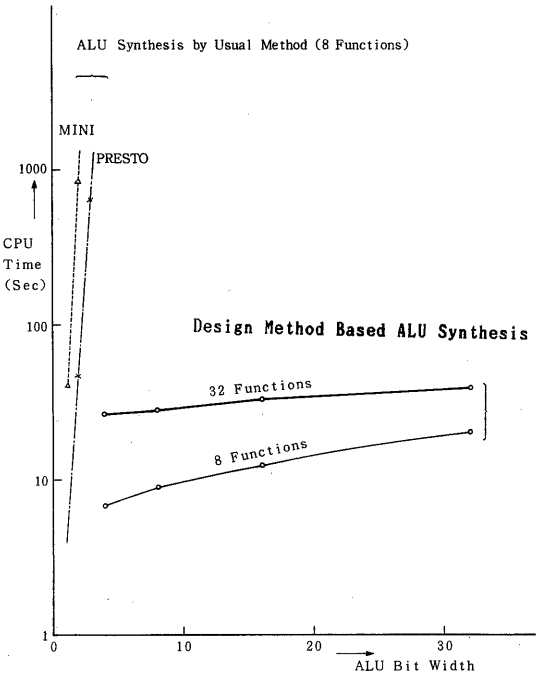


Fig.10 ALU Synthesis Time

図６～図８では，比較演算，シフト演算を考慮しなかったが，これら受理しうるよう設計手法を拡張する例を示す．

（１）シフト演算への拡張

バレルシフトは，ＡＬＵとはそぐわず，それ専用のハードウェアを設けることが普通である．従って，１ビットの右，或いは左シフトのみを考慮する．１ビットシフトは，Ｐ生成回路の万能論理生成回路に１ないし２個のＡＮＤ回路を追加し，上位，或いは下位ビットのデータ入力信号と，制御信号を印加する（図９（ｉ））．また，１ビット左シフトは，加算（A PLUS A あるいは B PLUS B）でも実現可能である（図９（ii））．

（２）比較演算への拡張

比較演算時には，ＡＬＵに減算（A PLUS $\overline{B}$ PLUS Cin）を行わせるよう構成するとともに，出力回路に“０”検出回路，符号検出回路等を必要に応じ付加する．

３．３　実験例

図１０に論理合成時間を示す．プログラムはＬＩＳＰで記述されており，約２ＭＩＰＳの計算機を使用した．実験は，８機能を有するＡＬＵと図５で示した３２の機能を有するＡＬＵについて行った．本論文で提案した手法，及び，従来手法，即ち，ＡＬＵの機能仕様を直接ブール式に展開し，MINI，PRESTO，で論理合成を行った時のCPU タイムを合わせて示す．なお，従来手法では，32 機能を有するＡＬＵの合成はメモリネックで実行困難であったため，８機能を有するＡＬＵのみについて示す．従来手法では，データ幅の増加と共に，ＣＰＵタイムが急激に増大することがわかる．これに対し，本論文の手法によれば，ＣＰＵタイムの増加率は，データ幅ｎの１乗以下である．

合成されたブール式に因子化等の処理を施さず，そのまま，ＡＮＤ，ＯＲ回路で表現したものを図１１に示す．Ｐ生成回路は，万能論理生成回路通りの構成となっているが，Ｇ生成回路は，万能論理生成回路の全回路を必要とせず，ＡとＢの論理積およびＡと$\overline{B}$の論理積を制御する回路のみから構成されている．

ＬＳＩにおいてはセル面積が重要な評価因子となる．セル面積はほぼ回路の総ファンイン数に比例することが知られている．従来手法，設計手法

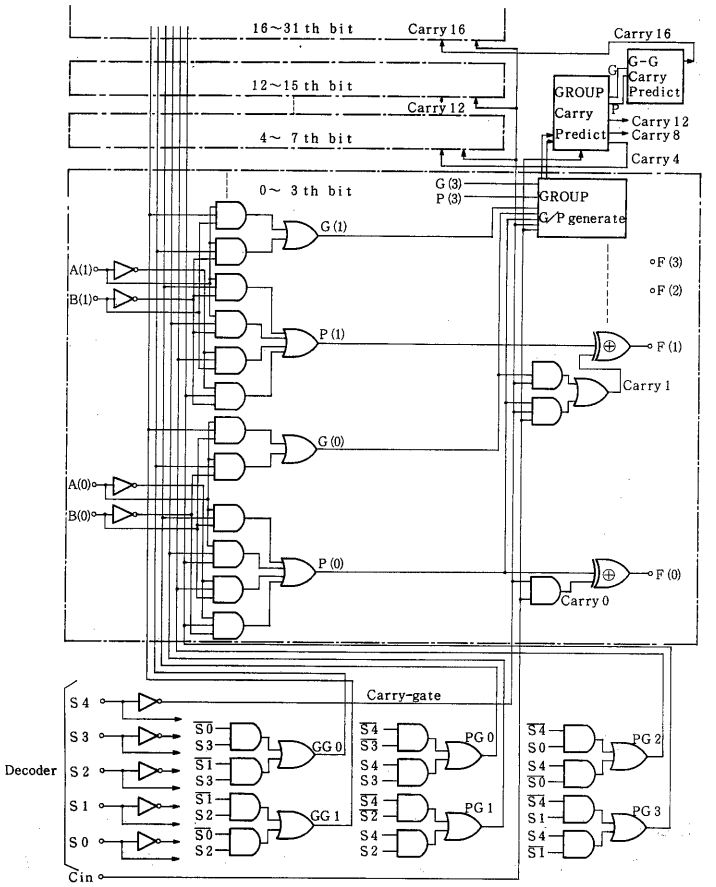


Fig.11 ALU Synthesis Example

ベース論理合成法，人手設定による回路の総ファンイン数の比較を図１２に示す．このデータは回路をＡＮＤ，ＯＲ回路で構成した時の値を示す．従来手法では，ビット幅の増加に伴いファンイン数の増加が著しい．設計手法ベース論理合成法によれば，８機能，４ビット幅ＡＬＵの総ファンイン数は人手設定のそれの５％減程度であり，３２機能，４ビット幅ＡＬＵの総ファンイン数は人手設定のそれの１割増程度であった．３２機能ＡＬＵの総ファンイン数の方が８機能ＡＬＵより総ファンイン数が少なくなっていることは興味深い．これは，ＡＬＵ設計では，機能仕様の与えかたが適切であると，機能種別数が増えても，総ファンイン数が増えないという性質を示している．また，総ファンイン数はビット幅に比例（比例常数はほぼ１である）する．

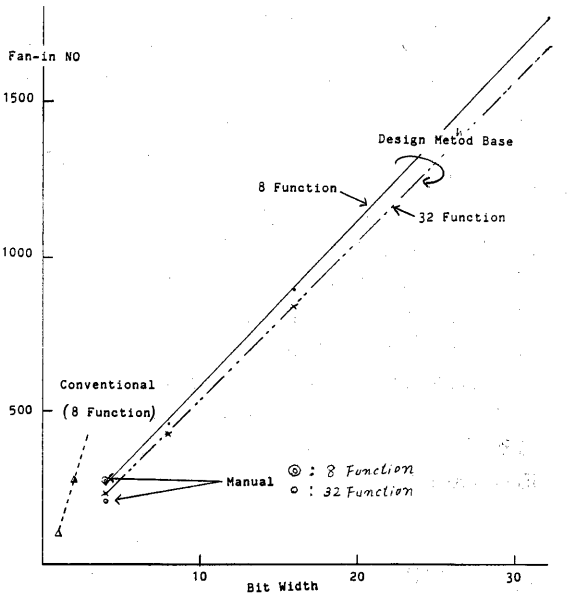


Fig.12 Fan-in Number

4．むすび

設計者の間では，所与の機能を，狙いとした論理段数，ゲート量で実現する指針を与えるものとして，様々な設計手法が考案されている．本論文は，これら設計手法を，回路構成方式とこれに沿った論理回路合成・変換手続きの組合せと捉え，これをアルゴリズム化し，設計手法ベースとして蓄積，活用すること事を提案した．本手法によれば，少ない計算量で狙いとした回路を合成しうることを示した．

３２機能を有する３２ビットのＡＬＵの合成時間は，２ＭＩＰＳの計算機で，４０秒であった．また，合成された論理は，因子化などの処理が不要な程度に，高品質であった．

設計手法の種類は多数存在する．今後，（１）これらを，効率良く表現出来る記述言語の設定，（２）機能仕様，及び，ゲート量，速度等の制約条件が与えられた時に最適な設計手法を選択し適用する機構，を検討していく予定である．また，設計手法ベース論理合成法の，順序回路への拡張性を検討していく予定である．

＜文献＞

（１）Hong,S.J.,Cain,R.G.,Ostapko,D.L. :
"MINI:A Heuristic Approach for Logic Minimization", IBM J.RES.DEVELOP.,
PP.443-458(1974)

（２）BROWN,D.W.:
"A STATE-MACHINE SYNTHESIZER", Proc.18th DA Conference, PP.301-305(1981)

（３）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 Minimization",
Proc. ISCAS Rome, PP.42-48(1982)

（４）榎本，中島，村井，笹尾：" 組合せ回路合成システム-COMPO-",
設計自動化研究会資料 20-1(1984)

（５）Darringer,J.A. :"A New Look at Logic Synthesis",Proc.17th DA Conference,
PP.543-549(1981)

（６）Shinsha,T.,Kubo,T.,Hikosaka,M.,Akiyama ,K.,Ishihara,k. :"POLARIS:PORARITY PROPAGATION ALGORITHM FOR COMBINATIONAL LOGIC SYNTHESIS",Proc.21th DA Conference ,pp.322-328(1984)

（７）Winkel,D.,Prosser,F.:"The Art of Digital Design", pp.84-86 Prentice-Hall,N.J. (1980)

付録　論理式生成手続きの例 (Ripple-adder)

```
Ripple-adder  bit-width

  DO I= 0 TO  bit-width-1
   Collect T(I)=A(I) + B(I),
   Collect G(I)=A(I) ・ B(I),
   IF I=0
     THEN  Collect C(I)=Cin,
     ELSE  Collect C(I)=G(I) + T(I)・C(I-1),
     END-IF,
   Collect S(I)=T(I) + G(I),
   END-DO,

 * Collect ; function  to collect expressions
                into the result buffer
```