数理解析研究所講究録 1375 巻 2004 年 137-143

決定グラフを用いた二線式単一磁束量子回路の論理設計法

Logic Design Method of Dual-Rail RSFQ Circuits Using Decision Diagrams

小畑 幸嗣 Koji Obata　高木 一義 Kazuyoshi Takagi　高木 直史 Naofumi Takagi

名古屋大学大学院情報科学研究科情報システム学専攻
Department of Information Engineering, Nagoya University

概要

単一磁束量子 (SFQ) 回路は半導体回路を上回る高速性、低消費電力性を持った、超電導体を用いた次世代デバイスである。本稿では、SFQ 回路向けの回路方式の一つである、二線式を用いた回路の新しい論理設計法を提案する。基本回路として 2x2-Join と呼ばれる回路を用いる。二分決定グラフを用いた一出力回路の論理設計法について述べた後、決定グラフを変形し、多出力回路への拡張を行なう。本稿のアルゴリズムを適用すると、2 個の 2x2-Join 回路で全加算器を構成することが可能である。

1 はじめに

単一磁束量子 (SFQ) 回路 [1] は、超高速動作可能、低消費電力の超電導体を用いた次世代デバイスである。現状の半導体回路では実現不可能な超高速動作回路を作成することが可能となる。SFQ 回路の研究は 1990 年代に本格化した。現在までの研究は製造に関するものが多く、論理設計に関する研究は少ない。

SFQ 回路は、パルス論理回路である。そのため、一本の信号線で "0" と "1" の二つの論理値を表現することができない。論理値の表現法として主に、同期クロックを用いるクロック同期式、二本の信号線を用いる二線式が考えられている。二線式はクロック信号が必要ないため、クロック同期式よりも高速な回路が設計できる可能性がある。

二線式 SFQ 回路向けの基本論理回路として、2x2-Join 回路 [5] がある。一つの 2x2-Join 回路といくつかの二入力合流回路 (cb) [1] を用いて、AND や OR などの複数の論理演算を同時に行なうことができる。そのため、2x2-Join 回路を効果的に用いることにより、二入力 AND や OR を用いた回路よりも、小面積で高速な回路を構成することが可能となる。従って、2x2-Join 回路を効果的に用いることが可能な論理設計法の開発が重要である。

本稿では、2x2-Join 回路を基本論理回路とした、二線式 SFQ 回路の新しい論理設計法を提案する。まず、二分決定グラフ (BDD) を用いた一出力回路の論理設計法について述べた後、BDD の枝に新たな属性を付加し、変形した決定グラフを用いた、多出力回路の設計法について述べる。これまでに提案されている BDD を用いた二線式 SFQ 回路の論理設計は、BDD の一つの節点を一つの基本論理回路で置き換える手法であった [6]。一方、本稿の手法では、最大二つの節点を一つの基本論理回路で置き換えることが可能となる。本稿の論理設計法を用いることにより、2x2-Join 回路を効果的に使用した二線式回路が得られる。

以下の章では、まず、SFQ 回路や 2x2-Join 回路についての説明を行なう。次に、BDD を用いた一出力回路の論理設計法について説明し、それを多出力回路へ拡張する。また、それらのアルゴリズムを実際の論理関数に適用した例を示す。最後に、まとめと今後の課題について述べる。

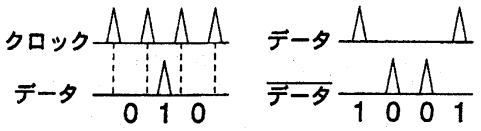
2 準備

2.1 SFQ 回路

SFQ 回路はジョセフソン接合とインダクタンスから構成される。ジョセフソン接合がスイッチする際に発生する電圧パルスを用いて処理が進められる。このため、パルス論理が用いられる。電圧パルスの制御は、ジョセフソン接合に流れる電流と回路中のインダクタンスの値を調整することによって行なわれる。

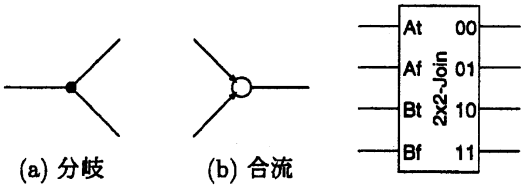
2.2 回路方式

SFQ 回路ではパルス論理が用いられるため、一本の信号線で "0" と "1" の二つの論理値を表現することができない。これは、パルスが存在する場合を論理値の "1" とした時に、パルスがない状態が論理値の "0" であるのか、そもそも信号が存在しないのかが区別できないためである。このため、二つの論理値の表現に



(a) クロック同期式　　(b) 二線式

図 1: 回路方式



(a) 分岐　　(b) 合流　　(c) 2x2-Join

図 2: 基本回路

は一本より多くの信号線が必要になる。具体的な論理値の表現法としては主に、クロック同期式と二線式が考えられている。

クロック同期式 データ信号に加えて、クロック信号を供給する方式である。クロックパルスとクロックパルスの間に、データ線上にパルスが存在すれば論理値の "1" を、存在しなければ論理値の "0" を表す (図 1 (a))。すべての基本論理回路にクロックを供給する必要がある。

二線式 論理値の "1" を表す信号線と論理値の "0" を表す信号線を用いる方式である。"1" を表す信号線上にパルスが存在すれば論理値の "1" を、"0" を表す信号線上にパルスが存在すれば "0" を表す (図 1 (b))。一つの論理値を表現するのに、必ず二本の信号線が必要になる。

本稿では、回路方式として二線式を用いる。

2.3 基本回路

本稿のアルゴリズムを適用して得られる回路は以下の基本回路から構成される [1, 5]。

分岐 (spl) 一つのパルスを二つに複製する基本回路である。回路図中では、図 2 (a) のように黒丸で表す。

合流 (cb) 異なる二つの方向から入力されたパルスの合流を行なう基本回路である。なお、二つの入力から同時にパルスを入力することは禁止されている。回路図中では、図 2 (b) のように丸と矢印で表す。

At & Bt → 11

At & Bf → 10

Af & Bt → 01

Af & Bf → 00

図 3: 2x2-Join 回路の入出力の関係

2x2-Join 論理演算が実行可能な基本回路である。回路図中では図 2 (c) のように表す。入力 A, B にはそれぞれ二線式のデータが入力され、入力の組合せに応じて四個の出力 (00, 01, 10, 11) のうちただ一つにパルスを生じさせる。入力と出力の関係を図 3 に示す。

この 2x2-Join 回路と前述の spl, cb とを組み合わせることにより、AND ,OR, XOR などの単独の演算だけでなく、一つの 2x2-Join 回路で AND と OR など複数の演算を同時に行なうことができる。

2.4 BDD

BDD は論理関数を表す有向非巡回グラフであり、節点集合は変数をラベルとして持つ非終端節点と定数 (0 または 1) をラベルとして持つ終端節点から構成される。また、枝集合は、各非終端節点から出る二本の枝から構成される。非終端節点は変数節点とも呼ばれ、変数節点の中に根節点と呼ばれる節点が存在する。また、終端節点を葉節点とも呼ぶ。非終端節点の出次数は 2 であり、終端節点の出次数は 0 である。非終端節点から出ている二本の枝はそれぞれラベルを持ち、0 のラベルを持つ枝を 0 枝、1 のラベルを持つ枝を 1 枝と呼ぶ。BDD は論理関数のシャノン展開をグラフで表したものと考えることができる。

変数に 0, 1 の値割り当てが与えられると、BDD の根から順にラベルの変数の 0, 1 に従ってグラフを辿っていくことができる。グラフを辿っていった時に、最終的に到達する終端節点の値がその論理関数の値となる。

BDD の例として $X \cdot Y$ の BDD を図 4 に示す。本稿では 0 枝を破線で、1 枝を実線で表す。X と Y の値が共に 1 の場合のみ関数の値は 1 になり、その他の場合は 0 になる。本稿では根節点をレベル 0 とし、順次葉節点に向かって 1, 2, $\cdots$ とする。

BDD では、変数順序を固定し、冗長な節点の削除と等価な節点の統合を行なうことによって、論理関数を一意に表現することが可能となる。このような BDD を ROBDD と呼ぶ。本稿では、この ROBDD を用いて一出力回路の論理設計を行ない、変形した ROBDD を用いて多出力回路の論理設計を行なう。

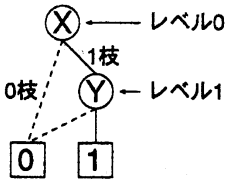


図 4: $X \cdot Y$ の BDD

3 BDD を用いた一出力回路の論理設計法

3.1 アルゴリズム

BDD を用いた二線式 SFQ 回路の論理設計アルゴリズムは以下の通りである。このアルゴリズムでは、BDD の根から葉に向かって順に回路を構成していき、BDD の根から葉に向かって信号が流れる回路が構成される。

[アルゴリズム]

1. 論理関数から ROBDD を作成する。
2. レベル 1 の節点で、根の 0 枝、1 枝が入力されているものにそれぞれ番号 0, 1 を付ける。
3. 2x2-Join 回路を一つ用意し、根の変数を入力 A に、レベル 1 の変数を入力 B に接続する。根とレベル 1 の節点を処理済みとする。
4. $i = 2$ からレベルの最大値まで以下を繰り返す。
   (a) レベル i で未処理の節点が存在する間以下を繰り返す
      i. レベル i で最左の未処理節点 α を選択し、番号 1 を付加、処理済みとする。
      ii. 新たな 2x2-Join 回路の入力 B に α の変数を接続する。
      iii. 入力 At には、α に入力されている枝を cb で一つの信号にして接続する。対応する信号線は次の通りである。
         - α の親が根の場合は、枝が 0 枝ならば根の変数の False 信号、1 枝ならば根の変数の True 信号に対応する。
         - α の親が根以外の場合は、その親に対応する 2x2-Join 回路の出力 (節点の番号 枝の番号) に対応する。
      iv. 入力 Af には、α に入力されている枝を、根まで逆にすべての経路を辿った時に、その経路中にある節点の枝で経路に含まれないものを接続する。
      v. Af に入力した枝と、レベル i の未処理の節点に入力されている枝が同一の場合、その節点の番号を 0 とし、処理済みとする。
5. 葉の "0" を回路の出力の False に、"1" を True にして、回路の出力を作成する。

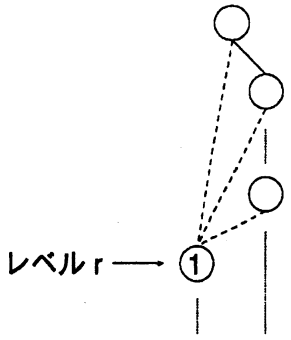


図 5: At に接続する信号線中の cb の数が最大になる例

上記のアルゴリズムを用いると、回路中の 2x2-Join 回路の数は ROBDD のレベル 1 の節点と葉を除いた節点数以下になる。また、2x2-Join 回路の段数は変数の数よりも 1 小さくなる。

レベル l の節点を処理している時、At に接続する信号線を構成する際に必要となる cb の数は最大 $l-1$ 個である。図 5 の 1 番の節点のように、l よりも小さなレベルの節点の枝がすべて入力されている場合は cb が $l-1$ 個必要になる。同様に Af に接続する信号線中にも、最大 $l-1$ 個の cb が必要になる。

上記のアルゴリズム全体の計算量は、(4. の部分の計算量) × (節点数) となる。4. の中で最も計算時間の必要な部分は (a) iv. である。アルゴリズムでは、処理している節点の祖先をすべて調べる必要があるが、前のレベルまでの結果を保存しておくことによって、計算量を減らすことができる。前のレベルまでの結果が保存されていれば、親のみ調べることによって Af に接続する信号線を求めることができる。従って、上記のアルゴリズム全体の計算量は、(個々の節点の親の数) × (節点数) となる。ROBDD 中の個々の節点の親の数の合計は、ROBDD 中の枝の数の合計と等しいため、全体の計算量は ROBDD の節点数の二乗に比例する。

3.2 一出力回路の設計例

一出力回路の設計例として、全加算器の和出力 ($S = X \oplus Y \oplus Z$) を計算する回路を示す。全加算器の和の ROBDD を図 6 に示す。この BDD のレベル 0 と 1

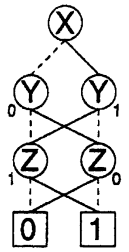
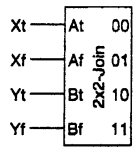
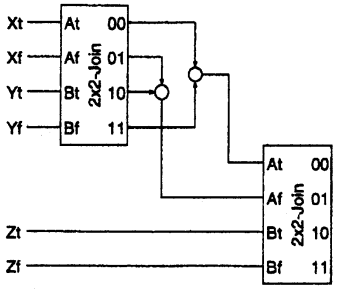


図 6: 全加算器の和 (S) の ROBDD

の変数 (X, Y) から一段目の 2x2-Join 回路が構成される (図 7 (a))。一段目の 2x2-Join 回路の出力と BDD のレベル 2 の変数 (Z) から二段目の 2x2-Join 回路が構成される (図 7 (b))。最後に葉に入力されている枝から出力が構成され、回路が完成する。全加算器の和を計算する回路を図 8 に示す。回路中の 2x2-Join 回路の数は 2 個、2x2-Join 回路の段数は 2 段となっている。



(a) 一段目の 2x2-Join 回路まで設計



(b) 二段目の 2x2-Join 回路まで設計

図 7: 全加算器の和 (S) を計算する回路の設計過程

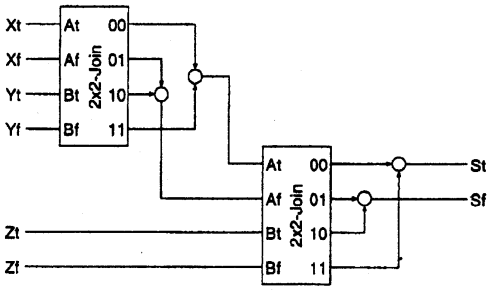


図 8: 全加算器の和 (S) を計算する回路

4 多出力回路への拡張

4.1 決定グラフの拡張

3 章のアルゴリズムでは、BDD を根から葉に向かって処理し、回路を構成していった。多出力回路を考える場合、回路の共有は入力側から行なわれる。従って、3 章のアルゴリズムを適用する場合は、各関数を表す BDD の根の側を共有しなくてはならない。具体的には、根の側から見て形の一致している部分グラフを共有する。BDD を根の側から共有すると、枝や節点がどの関数に属しているのかが分からなくなる。そのため、BDD の枝に関数名を付加することによって、枝や節点の属している関数が分かるようにする。

このような枝を実現するデータ構造として、図 9 のようなものが考えられる。図中の節点番号は節点を一意に識別する番号を表し、入次数、レベルはそれぞれ節点の入次数とレベルを表す。0, 1 枝リストには、それらの枝を辿った先の節点とその枝の属している関数名を格納しておく。枝リストには最大、処理を行なう関数の数だけのデータが格納される。従って、グラフ全体では (節点数) × (関数の個数) に比例する記憶領域が必要になる。

グラフの作成は次のようにして行なう。処理を行なう関数の順番は任意である。根の側から見て形の一致している部分グラフを共有するので、関数の順番によって生成されるグラフの形が変わることはない。

1. 一つ関数を選択し、通常の ROBDD を作成する。
2. 残りの関数に関して、以下の操作を行なう。

節点番号	0 枝リスト	1 枝リスト	入次数	レベル

図 9: 決定グラフのデータ構造

- 根節点の変数が既に作成したグラフと同じである場合は、以下の操作を行なう。
  (a) 根節点は共有可能なので、共有する。
  (b) グラフの作成過程で入次数が変化する節点、入次数が変化しなくても親が別の節点に変わる節点に対して、以下の操作を行なう。
    i. 入次数が等しく、同じレベルの節点で親がすべて同じものが存在するかを調べる。

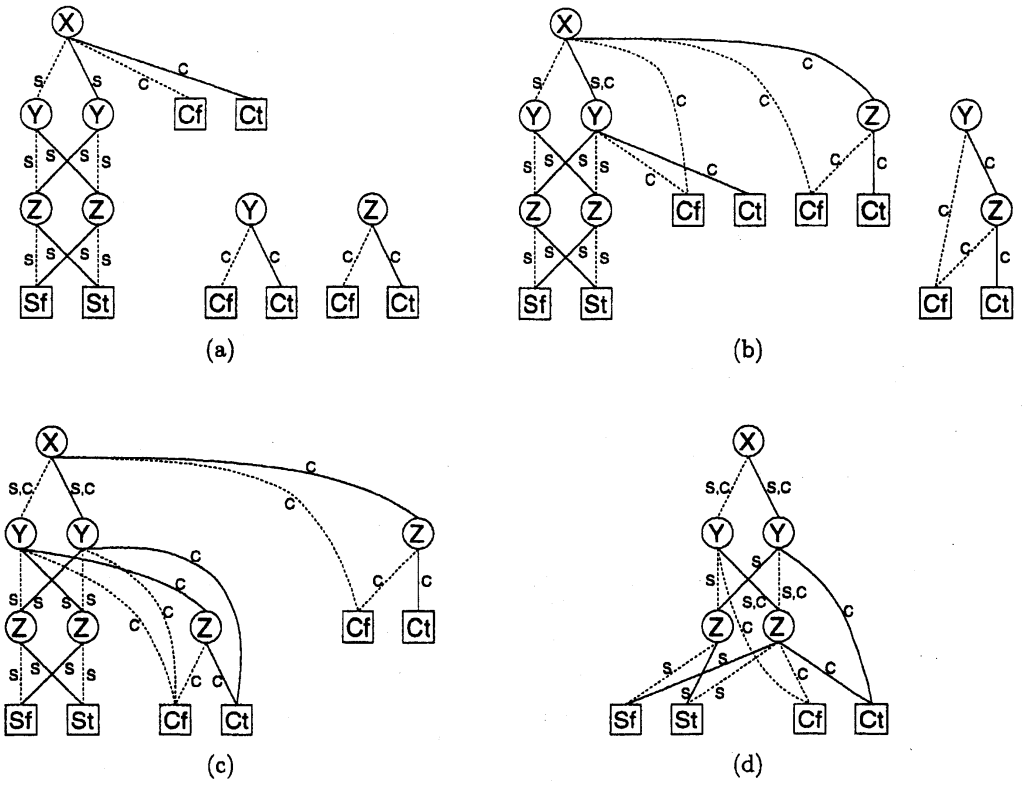


図 10: 全加算器のグラフの作成過程

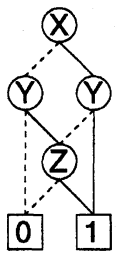


図 11: 全加算器の桁上げ (C) の ROBDD

ii. (a) で見つけた節点に対して、すべての親が同じ関数で共有されているかを調べる。

iii. すべての親が同じ関数で共有されているならば、親との枝の "0", "1" が一致するかを調べる。

iv. (a), (b), (c) が成り立つならば、枝の関数名リストを更新し節点を共有する。

v. 節点を共有したならば、共有した節点の子供についても共有可能か調べる。

- 根節点の変数が既に作成したグラフと異なる場合は共有は不可能であるので、通常の ROBDD を作成する手順でグラフを作成する。

根の変数が同じ場合、必ずその根は共有できる。従って、多くのグラフに現れる変数の節点をグラフのレベルの小さい部分にしたほうが、グラフを共有できる可能性は高くなる。

例として、全加算器 ($C = XY + YZ + ZX, S = X \oplus Y \oplus Z$) のグラフの作成課程を図 10 に示す。グラフを見やすくするため、葉の "0", "1" はそれぞれ対応する関数名 + "f", "t" に変更してある。S のグラフを作った後 C のグラフを作るとする。また、変数順序は X, Y, Z とする。図 10 (d) が完成したグラフである。完成したグラフのうち、S の関数名が付いている枝とそれに接続している節点のみを取り出せば、通常の S の ROBDD (図 6) と同一になる。同様に C の枝と節点を取り出せば、通常の C の ROBDD (図 11) と同一になる。

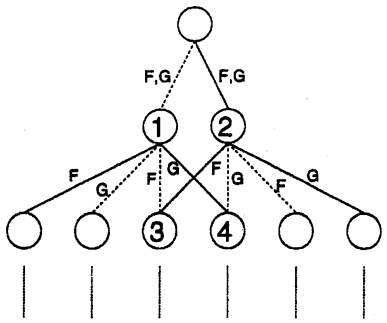


図 12: アルゴリズムの拡張が有効な例

4.2 アルゴリズムの拡張

多出力関数の際の論理設計は、一出力関数のアルゴリズムを関数毎に適用することによって行なうことができる。ここでは、更に効率の良い回路を生成するためにアルゴリズムを拡張する。具体的には、処理中の節点 α の親が他の関数と共有されている場合、3 章のアルゴリズムの 4(a)v. の次に以下の操作を行なう。

- Af に入力した枝の関数名を親の共有相手の関数名に変更した場合に、それらの枝が入力されている α と同じレベルの未処理の節点が存在するかを調べ、そのような節点が存在した場合、その節点の番号を 0 として処理済みとする。

図 12 に拡張されたアルゴリズムが有効な例を示す。図中の関数 F、3 番の節点を処理中であるとする。この場合、3 番の節点に対応する 2x2-Join 回路の入力 Af には、1 番の節点の 1 枝と 2 番の節点の 0 枝が入力される。4 番の節点には、関数 G の関数名が付いた、それらの枝が入力されている。1 番と 2 番の節点は関数 F と G で共有されているため、上記の条件を満たす。従って、4 番の節点の番号を 0 とし、処理済みとすることができる。

多出力回路の設計は、回路を構成している関数それぞれに、上記の拡張したアルゴリズムを適用して行なう。このようにして設計した回路は、一出力関数の場合と同様に、回路中の 2x2-Join 回路の数はグラフ中のレベル 1 の節点と葉を除いた節点数以下になる。また、回路中の 2x2-Join 回路の段数は変数の数よりも 1 小さくなる。

4.3 多出力回路の設計例

多出力回路の設計例として、全加算器の設計を示す。全加算器のグラフは図 10 (d) に示した。設計は関数 S, C の順で行なうものとする。関数 S の回路は一出力回路の設計例で示した回路図 (図 8) と同一の回路が

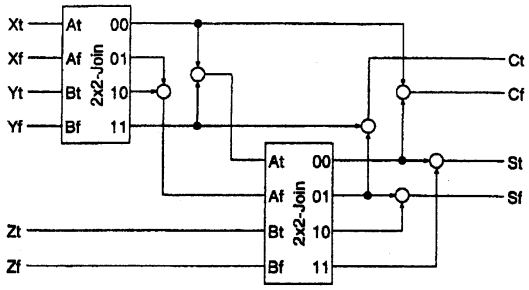


図 13: 全加算器の回路図

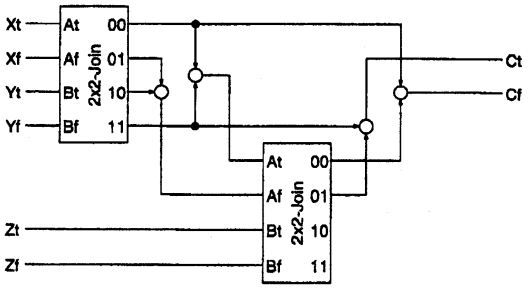


図 14: 全加算器の桁上げ (C) を計算する回路図

生成される。続いて関数 C について設計すると、図 13 が得られる。関数 C の回路を単体で設計すると図 14 のようになり、図 13 の回路は、この C の回路と S の回路 (図 8) において、入力側から一致している部分を共有した回路になっていることがわかる。図 13 の回路中の 2x2-Join 回路の数は 2 個であり、2x2-Join 回路の段数も 2 段である。図 8 の S 回路と比較して、cb が 2 個、spl が 4 個増加するだけで、全加算器の和を計算する回路に桁上げを計算する回路を追加することが可能であった。

5 アルゴリズムの評価

本稿で示したアルゴリズムを、いくつかの基本的な論理関数に適用して評価を行なった。比較対象として、二入力 AND, OR, XOR で回路を構成する手法 (手法 1) 、文献 [6] で紹介されている、グラフの節点に一つの基本論理回路 (Bina) を割り当てていく手法 (手法 2) を用いた。回路中で必要になる基本論理回路 (二入力 AND, OR, XOR 、Bina、2x2-Join) の数で比較を行なった。基本論理回路の大きさには最大で約 2 倍の違いがある。また、それらの論理関数から共有二分決定グラフ (SBDD) を作成した際の節点数も計算し、提案手法で用いるグラフの節点数との比較を行なった。

表 1: アルゴリズムの評価結果

	提案手法	手法 1	手法 2	グラフの節点数	SBDD の節点数
1 ビット比較器	2	4	3	4	4
4 ビット比較器	8	16	12	13	13
8 ビット比較器	16	32	24	25	25
全加算器	2	5	4	5	6
4 ビット順次桁上げ加算器	8	20	16	17	21
8 ビット順次桁上げ加算器	16	40	32	33	41

なお、SBDD の節点数は否定エッジを用いた場合の節点数である。

表 1 の結果より、提案手法は手法 1 、手法 2 に比べて、非常に少ない数の基本論理回路で回路を構成可能であることがわかる。また、加算器の場合、通常の SBDD を作成するよりも、提案手法で用いるグラフの方が節点数が少なくなる。

6 まとめ

本稿では、決定グラフを用いた二線式 SFQ 回路の論理設計法について述べた。まず、ROBDD を用いた一出力回路の設計法について述べ、続いて ROBDD に変更を加えることで多出力回路への拡張を行なった。

今後は、様々な複雑な論理関数に対して本稿で提案したアルゴリズムを適用し、アルゴリズムの評価を行なう予定である。また、グラフの共有に関して、より良い方法がないかの検討も行なう。

謝辞

本研究は、低消費電力型超電導ネットワークデバイスの開発の研究として、ISTEC を通じて、新エネルギー・産業技術総合開発機構の委託により実施したものである。

参考文献

[1] K.K.Likharev, V.K.Semenov "RSFQ Logic/Memory Family: A New Josephson-Junction Technology for Sub-Terahertz-Clock-Frequency Digital Systems", IEEE Trans. Appl. Supercond., vol.1, no.1, pp.3-28, March 1991.

[2] Kris Gaj, Eby G.Friedman, Marc J.Feldman "Timing of Large RSFQ Digital Circuits", Ext. abs. 6th Int. Supercond. Electr. Conf., ISEC'97, Berlin, Germany, 1997.

[3] M.Maezawa, I.Kurosawa, M.Aoyagi, H.Nakagawa, Y.Kameda, T.Nanya "Rapid Single-Flux-Quantum Dual-Rail Logic for Asynchronous Circuits", IEEE Trans. Appl. Supercond., vol.7, no.2, pp.2705-2708, June 1997.

[4] Y.Kameda, S.Polonsky, M.Maezawa, T.Nanya "Primitive-Level Pipelining Method on Delay-Insensitive Model for RSFQ Pulse-Driven Logic", Proc. ASYNC-98, pp.262-273, March 1998

[5] Y.Kameda, S.V.Polonsky, M.Maezawa, T.Nanya "Self-Timed Parallel Adders based on DI RSFQ Primitives", IEEE Trans. Appl. Supercond., vol.9, no.2, pp.4040-4045, June 1999.

[6] N.Yoshikawa, J.Koshiyama "Top-Down RSFQ Logic Design Based on a Binary Decision Diagram", IEEE Trans. Appl. Supercond., vol.11, no.1, pp.1098-1101, March 2001.

[7] R E. Bryant "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. Comput., Vol.C-35, No.8, pp.677-691, 1986