Synthesis of Multiple Constant Multiplication Circuits Using GA with Chromosomes Composed of Stack Type Operators

Yosuke Isoo and Hisamichi Toyoshima

Department of Electrical Engineering
Kanagawa University
3-27-1 Rokkakubashi, Kanagawa-ku
Yokohama 221-8686 Japan
Tel : +81-45-481-5661, Fax : +81-45-491-7915
E-mail : toyo@cc.kanagawa-u.ac.jp

Abstract

The purpose of this paper is to find an efficient solution for multiple constant multiplication (MCM) problem. Since the circuit structure can be represented as a directed acyclic graph, evolutionary computing is considered as an effective tool for optimization of circuit synthesis.

In this paper, we propose a stack type operator as a chromosome element to synthesize a directed acyclic graph efficiently. This type of chromosome can represent a graph structure with a set of simple symbols and so we can employ the similar method to a conventional GA.

1 Introduction

There exist many applications about multiple constant multiplication (MCM) in DSP, telecommunications, graphics, and so on. In MCM circuits, it is possible to reduce computation by replacing multipliers with adders and shifts, however, synthesis of MCM so as to minimize computational complexity is known as an NP complete problem. For MCM problem, several synthesis methods have been proposed [1-4]. Among them, representation of MCM circuit is categorized by two methods. One is by a binary array and another is by a directed acyclic graph (DAG).

The method that synthesizes MCM circuit with binary array is the algorithm that extracts the common bit pattern to reduce the number of additions and shifts [2-4]. This method is simple and fast, however, its optimization is local and so the solution is not optimum.

On the other hand, in the synthesis method using the representation of DAG [1], partial results formed in any one constant-sample multiplication can be reused to assist in the formation of other product terms. As a result, the whole multiplication block can be simplified with additions and shifts instead of multiplications. For optimization of graph structure, genetic programming (GP) is considered as an effective tool, and Ref.[5] shows the GP based algorithm to synthesize graph structure. However GP has the problem that the chromosome structure and genetic operators are so complicated and time-consuming.

In this paper, we propose a stack type operator as a chromosome element to synthesize DAG efficiently. This type of chromosome can represent DAG with a set of simple symbols and so we can employ the similar method to a conventional GA.

2 MCM as a directed acyclic graph

Multiple constant multiplication (MCM) is expressed as

$$y_i = a_i x, \quad i = 0, 1, \cdots, N-1, \tag{1}$$

where each a_i is a constant coefficient. Figure 1 shows a flow graph of MCM as a multiplication block.

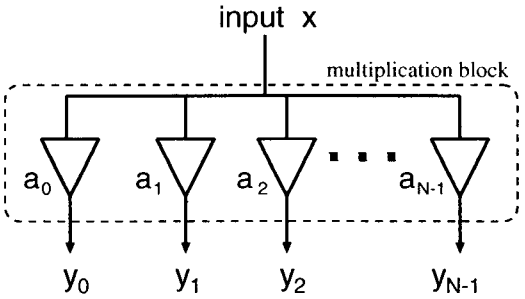

Figure 1: Multiplication block

One of the expression methods for MCM circuit, a directed acyclic graph (DAG) has been presented [1]. In MCM circuit, partial results formed in any one constant-sample multiplication can be reused to assist in the formation of other product terms. As a result, the whole multiplication block can be simplified with additions and shifts instead of multiplications. For example, the constant set {1,7,16,21,33} can be expressed by DAG as shown in Figure 2. Ref.[1] also shows the synthesis method of MCM circuits, however, this is a straightforward method and is not considered to find the optimum solution.

As an optimization approach for synthesizing graph structure, genetic programming (GP) is to be considered. Some GP algorithms with graph structure chromosomes have been introduced in Ref.[5]. However, GP

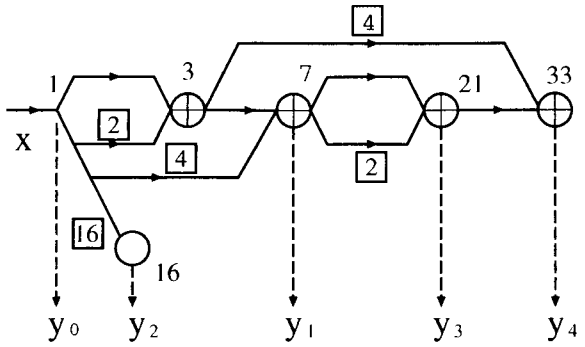

Figure 2: Example of constant set{1,7,16,21,33}

has some problems that a chromosome structure is complicated and computation is time-consuming. Therefore, we will show another simple chromosome structure for representing DAG to be applied by GA in the following sections.

3 Genetic Algorithms with chromosomes composed of stack type operators

3.1 Stack type operators

Stack is a mechanism that last-in data is first-out and computation is performed using a reverse polish notation. Stack type operators are symbols for the operation on stack elements. The sequence of stack type operators is executed one by one to operates stack elements.

In this work we employ the following operators:

- dup : duplicates the top of stack value.
- add : pops the top and the second top of stack values, adds them and pushes the result.
- sft : shifts the top of stack value to the left by one bit.
- rol : rolls every stack value so as to move the bottom of stack to the top of stack.

Each operator's behavior is shown in Figure 3.

Combined the above operators, the equivalent operation to DAG can be performed. In this paper, a combination of stack type operators is considered as a chromosome. It should be noted that each chromosome is generated by the following rules to avoid missing stack elements:

(1) Initial value of stack is $x = 1$.

(2) 'add' is not pushed if the number of stack elements is less than two.

(3) 'rol' is not pushed if the number of stack elements is less than two.

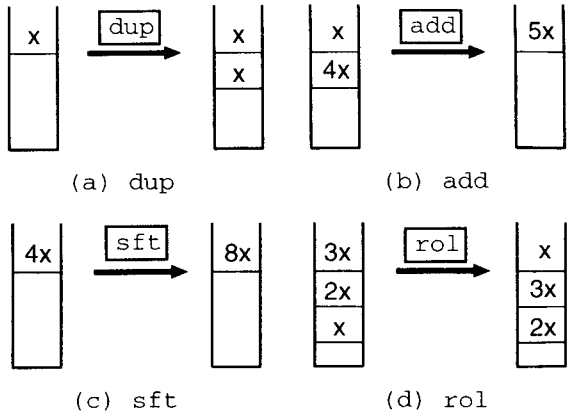

Figure 3: Movement of stack type operators

(4) 'add' is not pushed after 'dup'.

An example of chromosome to express the constant set {1,5,14} is shown in Figure 4.

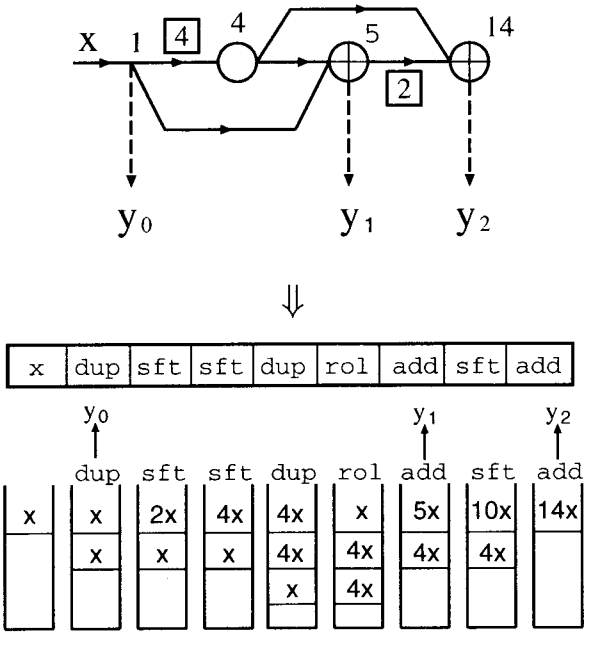

Figure 4: Example of stack type operator

3.2 Optimization using GA

Usually each chromosome in GA is encoded as a bit string, and so it is difficult to represent a graph structure such as DAG. However, as the proposed type of chromosome is considered as a symbol string, conventional GA is easily applied to optimize a graph structure. GA operations handled in this work are shown in the following:.

Selection : Elitism is used to prevent losing the best found solution and to increase the performance of GA.

Crossover : Single point crossover is used provided that a crossover point is selected so that the number of stack elements is the same each other. An example of crossover behavior is shown in Figure 5.

Mutation : Generate a new chromosome by a certain mutation rate.

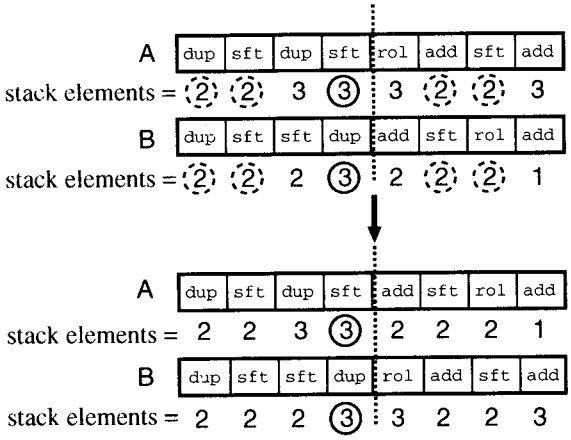

Figure 5: Crossover

To evaluate the fitness of each chromosome, we have the following fitness function. Let n_m be the number of matched coefficients, n_a be the number of additions, and n_s be the number of shifts, respectively. The fitness is defined as

$$\begin{aligned} fitness = \; & k_m \times n_m/N \\ & +k_a \times N/(n_a+1) \\ & +k_s \times N/(n_s+1), \end{aligned} \tag{2}$$

where N is the number of constants, and k_m, k_a, or k_s denotes each weight.

A flowchart of GA is shown in Figure 6.

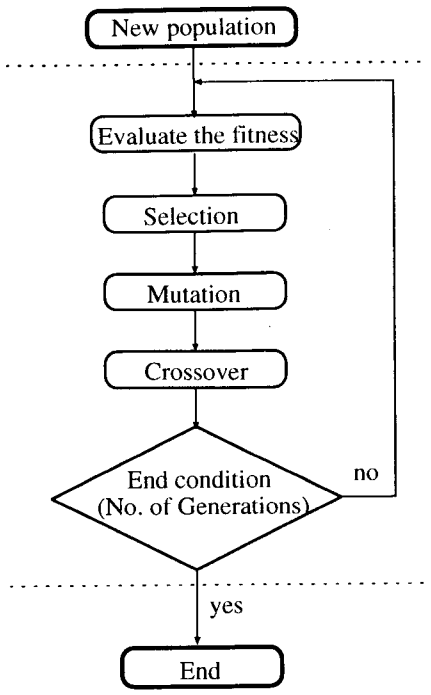

Figure 6: GA flowchart

4 Experimental Results

In this section, some examples of optimization result using the proposed method are shown. Also the performance of the proposed method and comparisons with a conventional method are presented.

4.1 Optimization examples

We have the following simulations on the condition that the constant set a_i is given as Eqn.(3) or Eqn.(4).

$$\{a_i\} = \{1, 7, 16, 21, 33\} \tag{3}$$

$$\{a_i\} = \{1, 21, 43, 93, 128, 179, 237\} \tag{4}$$

Each parameter of GA is shown in Table 1. In case of Eqn.(3), the obtained structure is shown in Figure 7. This result requires 4 additions and 8 shifts, whereas Ref.[1] requires 4 additions and 10 shifts. In case of Eqn.(4), the DAG circuit of Figure 8 is obtained. This structure consists of 8 additions and 11 shifts, whereas Ref.[1] consists of 9 additions and 17 shifts.

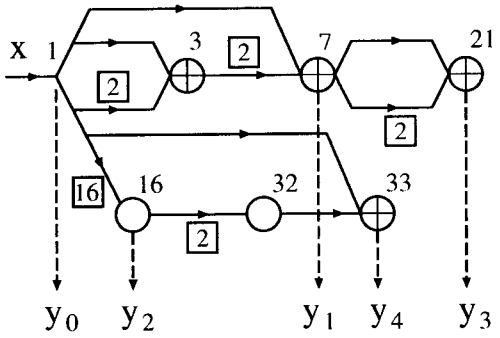

Figure 7: Resultant circuit for the constant set {1,7,16,21,33}

4.2 Performance considerations and comparisons

To evaluate the statistical performance of the proposed method, we show the following simulations. For several patterns of the number of bit of each constant and the number of constant set, such as six 7-bit, five 8-bit,

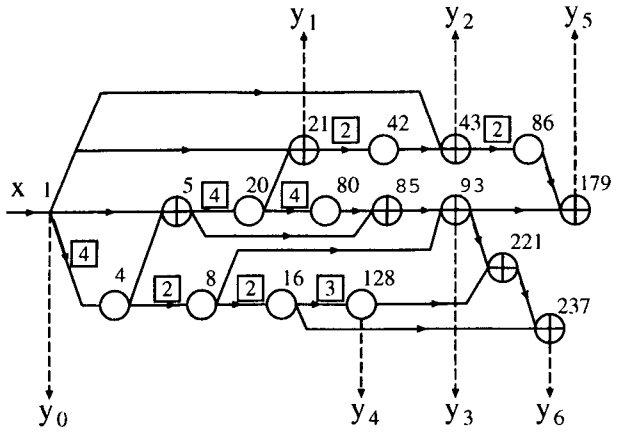

Figure 8: Resultant circuit for the constant set {1,21,43,93,128,179,237}

Table 1: GA parameters in optimization examples

	Eqn.(3)	Eqn.(4)
No. of generations	1500	1500
No. of populations	3000	3000
gene length	23	80
mutation rate	20%	7%
crossover rate	60%	80%
k_a	20	20
k_s	1	1

or four 9-bit constants, ten simulations were made respectively. In each simulation, each parameter of GA is shown in Table 2, and every constant set to be optimized is generated at random. Figure 3 summarizes the results of average reduction rate of additions and shifts compared with Ref.[1]. As seen from this table, the number of additions is reduced by 10% – 20% and the number of shifts is reduced by about 50%.

Table 2: GA parameters in random simulations

	7 bit	8 bit	9 bit
No. of generations	1200	1500	2000
No. of populations	3000	3000	4000
gene length	70	80	100
mutation rate	7%	10%	20%
crossover rate	80%	80%	90%
k_a	20	20	20
k_s	1	1	1

5 Conclusions

In this paper, we have proposed the novel chromosome structure to synthesize DAG. The proposed chromosome is composed of stack type operators and it is expressed by the simple sequence of symbols. This concept enabled us to employ GA as an optimization method and we have shown that MCM circuit is synthesized with smaller number of additions and shifts than the existing method.

Table 3: Experimental results

No. of bit	No. of constants	Reduction rate in 'add'	Reduction rate in 'sft'
9 bit	4	11.5%	48.9%
8 bit	5	16.3%	57.0%
7 bit	6	21.1%	52.8%

As a result of simulations, for up to 9 bit constant set, the proposed method is very effective for MCM problem. However, some problems still remain. For example, to find the optimum solution, multiple parameters have to be set appropriately. Furthermore, it is difficult to optimize the MCM with long word-length constants. Solving these problems will be the further study.

References

[1] David Bull, David Horrocks, "Primitive operator digital filters", IEE Proceedings-g Vol.138, pp.401–411, June 1991.

[2] A.Matsuura, M.Yukishita, and A.Nagoya, "A Hierarchical Method for the Multiple Constant Multiplication Problem", IEICE Trans. Fundamentals vol. E80-A, No.10, pp.1767–1773, Oct. 1997.

[3] M.Potkonjak, M.B.Srivastava, and A. P. Chandrakasan, "Multiple constant multiplication: Efficient and versatile framework and algorithms for exploring common subexpression elimination", IEEE Trans. Computer-Aided Des. Integrated Circuits & Syst., vol.15, no.2, pp.151-165, Feb. 1996.

[4] Keshab K.Parhi, "VLSI Digital Signal Processing Systems", Wiley-Interscience, pp.559-590, 1999.

[5] Zbigniew Michalewicz, "Genetic Algorithms + Data Structures = Evolution Programs Second, Extended Edition", Springer-verlag, 1994.