Design Methods for Polymorphic Combinational Logic Circuits based on the Bi_Decomposition Approach

Zhifang Li$^{1,2}$, Wenjian Luo$^{1,2}$, Lihua Yue$^{1,2}$ and Xufa Wang$^{1,2}$

1 School of Computer Science and Technology, University of Science and Technology of China, Hefei, 230027, Anhui, China
2 Anhui Key Laboratory of Software in Computing and Communication, University of Science and Technology of China, Hefei, 230027, Anhui, China
Email: zhifangl@mail.ustc.edu.cn, {wjluo, llyue, xfwang}@ustc.edu.cn

Abstract—Polymorphic circuits are a special kind of digital logic components, which possess multiple build-in functions. In different environments, a polymorphic circuit would perform different functions. Evolutionary Algorithms, Binary Decision Diagrams (BDD) and the multiplex method have been adopted to design polymorphic circuits. However, the evolutionary methods face the scalable problem. The BDD method consumes too much gate resource. The polymorphic circuit built by the multiplex method rarely contains polymorphic gates. In this paper, based on the traditional Bi_Decomposition circuit design approach, two methods, i.e. the Poly_Bi_Decomposition method and the Transformation&Bi_Decomposition method, are proposed for designing polymorphic circuits. The Poly_Bi_Decomposition method can design relatively large and gate-efficient polymorphic circuits with a higher percentage of polymorphic gates. The experimental results show the good performance of the proposed methods.

Keywords- Polymorphic electronic, Polymorphic gates, Polymorphic circuits, Bi_Decomposition

I. INTRODUCTION

Polymorphic Electronics are a new research field proposed by Stoica in 2001 [1]. A polymorphic electronic component is sensitive to the environmental signals, and it behaves differently in different environments. For example, a polymorphic NAND/NOR gate controlled by VDD would perform the NAND function when the voltage is 3.3V and perform the NOR function when the Voltage is 1.8V [2]. Current work about Polymorphic Electronics focuses on polymorphic digital logic components, including polymorphic digital logic gates and polymorphic combinational digital logic circuits.

Some polymorphic gates have been designed and fabricated in silicon [1-5], such as the NAND/NOR [2, 4] gate controlled by VDD and the AND/OR [1] gate controlled by temperature. Polymorphic gates are the basic building blocks of polymorphic circuits. Due to the characteristic of multi-functional and sensivity to environment signals, polymorphic logic circuits have the potential application in security, verification, multi-functional circuits and smart systems [1].

However, there is no effective method for building large scale polymorphic circuits. Evolutionary Algorithms [3, 6, 7] have been adopted for generating area-efficient polymorphic circuits, but it can not be scaled to large circuits. Up to now, “3×4 multiplier / 7 bit sorting-net” is the largest polymorphic circuit designed by the evolutionary method [8]. Binary Decision Diagrams (BDD) and the multiplex methods have also been used to design polymorphic circuits [9]. However, the BDD method consumes too much gate resource, and the multiplex method hardly utilizes the build-in multi-functional characteristics of polymorphic components.

In this paper, based on the Bi_Decomposition approach for traditional circuit design, the Poly_Bi_Decomposition method for the polymorphic circuit synthesis is proposed. The Poly_Bi_Decomposition method can design gate-efficient and large scale polymorphic circuits with a high percentage of polymorphic gates. In addition, some transformation rules are given for designing polymorphic circuits through existing circuits design method and tools, e.g. Bi_Decomposition [10]. By combining these rules and the Bi_Decomposition method [10], the Transformation&Bi_Decomposition method is proposed.

The rest of this paper is organized as follows. Section II introduces related works briefly. Section III gives a short introduction to the Bi_Decomposition [10]. Section IV explains the proposed methods. Section V demonstrates the experimental results. Section VI gives some discussions. Finally, Section VII concludes the whole paper.

II. RELATED WORKS

Polymorphic electronic is a novel research field, and several researchers have conducted some pioneer work. In this section, firstly, the works about synthesis of polymorphic circuits through evolutionary methods are summarized. Secondly, the Binary Decision Diagrams (BDD) and polymorphic multiplex methods for designing polymorphic circuits [9] are briefly introduced.
A. Evolutionary Design of Polymorphic Digital Circuits

Polymorphic gates are the basic building blocks for designing polymorphic digital circuits. The polymorphic gate possesses multiple intrinsically build-in functions. In each mode, the polymorphic gate would perform exactly the same as a traditional logic gate. In [1], Stoica and his colleagues designed the polymorphic gates AND/OR and AND/OR/XOR controlled by VDD, and AND/OR controlled by temperature. Two kinds of NAND/NOR polymorphic gates have been designed and fabricated with the 0.5 and 0.7 CMOS technology in [2] and [4], respectively. The NAND/NOR/NXOR/AND have been reported in [5].

Up to now, there is little theory for guiding the design of polymorphic circuits. In [11] and [12], the definition of complete polymorphic gate sets and the algorithms for judging the completeness of a polymorphic gate set are given, respectively. However, there is no efficient method for guiding the design of large scale polymorphic circuits.

Recently, evolutionary methods are widely adopted for designing polymorphic circuits. In Table I, some polymorphic circuits designed by Evolutionary Algorithms are listed. Currently, the largest polymorphic circuit obtained is “3×4 multiplier / 7 bit sorting-net” [8], which is composed of about 100 polymorphic gates. It can be observed from Table I that it is hard to design large scale polymorphic circuits through evolutionary methods.

| Polymorphic circuits    | gates     | Generations | Num. of gates | Reference |
|-------------------------|-----------|-------------|---------------|-----------|
| 5-semmetry / 5-median   | NAND/NOR, XOR | –           | 13            | [13]      |
| 2×3 multiplier / 5-sorting-net | NAND/NOR, AND, OR, XOR | 854,900  | 30            | [8]       |
| 3×3 multiplier / 6-sorting-net | NAND/NOR | 26,972,648 | 52            | [8]       |
| 3×3 multiplier / 6-adder      | NAND/NOR, OR/XOR | 2,514,043 | 89            | [6]       |
| 3×4 multiplier / 7-sorting-net | NAND/NOR, AND | 62,617,151 | 113           | [8]       |

B. Binary Decision Diagrams and Multiplex Methods for Designing Polymorphic Digital Circuits

In [9], the Binary Decision Diagrams (BDD) and polymorphic multiplexes are adopted for designing polymorphic circuits.

As for the PolyBDD method [9], firstly, the original polymorphic function is transmitted to another function. The variable number of the new obtained function is one less than the original function, and its output value is an integer ranged from 0 to 15. Each integer (from 0 to 15) corresponds to a polymorphic component in \{g_1, g_2 \mid g_1, g_2 \in \{1, 0, \text{NOT, WIRE}\}\}. Finally, a polymorphic circuit implementing the original function is built. The BDD method consumes too much gate resource.

The polymorphic multiplex method [9] is a combination of traditional circuit design method and polymorphic multiplexes. Firstly, traditional circuit design methods, such as ABC [14] and Espresso [15], are adopted to design the single function circuit in each mode. Then those single function circuits are connected to the corresponding input pin of the polymorphic multiplex. The polymorphic multiplex switches one of its inputs to the output according to the environment. The multiplex method could generate gate-efficient results. However, the inherent multifunctional properties of polymorphic gates are not considered, and the circuits designed by this method have no essential different from the traditional multifunctional circuits. It is noted that the polymorphic multiplex is firstly proposed by Sekanina in [13]. Additionally, different kind of polymorphic multiplex based on the complete gate set is given in [11].

III. INTRODUCTION TO THE Bi_DECOMPOSITION APPROACH

The Bi Decomposition method [10, 16] is an effective approach for designing traditional logic circuits. In [10], Steinbach and Lang give an detailed introduction of the Bi Decomposition method. In this section, a brief introduction of the Bi Decomposition circuit design method is given. Figure 1 is an illustration of the principle of the Bi Decomposition method. The Boolean function f(A, S, B) is the combination of the logic gate g (g \in \{\text{AND, OR, NOT}\})

 otherwise, it is a weak bi_decomposition.

Figure 2 gives an example of the bi_decomposition. The Boolean function f(x_1, x_2, x_3, x_4) in Figure 2(a) is decomposed to r(x_2, x_3, x_4) in Figure 2(b) and h(x_3) in Figure 2(c) through the logic gate AND, i.e. f(x_1, x_2, x_3, x_4) = AND(r(x_2, x_3, x_4), h(x_3)). In this instance,

Figure 1. The illustration of the bi_decomposition, where f(A, S, B) = g( r(A, S), h(B, S) ) [10].
A = \{x_2, x_3, x_4\}, B = \{x_1\} and S = \Phi.

The similar decomposition can be conducted to \(r(A, S)\) and \(h(B, S)\) until the variable number of the boolean function is not greater than 2. When the decomposition process ends, a circuit implementing the function \(f(A, S, B)\) is obtained.

IV. THE PROPOSED METHODS

In this section, based on the bi decomposition approach [10], two kinds of methods are proposed for designing polymorphic circuits with polymorphic gates as the basic building blocks.

A. The Poly_Bi_Decomposition method

A polymorphic Boolean function \(f\) can be presented as \(f_1/f_2\). In mode 1, the function is \(f_1\) and in mode 2, the function is \(f_2\). For example, the polymorphic Boolean function “4×4 multiplier / 8 bit sorting-net” performs the function 4×4 multiplier in mode 1 and function 8 bit sorting-net in mode 2.

Similar to the Bi Decomposition method in [10], Figure 3 is an illustration of the principle of the Poly_Bi Decomposition. The polymorphic Boolean function is the combination of a polymorphic gate \(g_1/g_2\), polymorphic Boolean functions \(r(A, S)\) and \(h(B, S)\), where \(g_1, g_2 \in \{\text{AND, OR, XOR}\}\), \(r(A, S) = r_1(A, S) \cup r_2(A, S)\) and \(h(B, S) = h_1(B, S) \cup h_2(B, S)\). The variable sets A and B would not be empty simultaneously. \(f_1(A, B) = g_1(r_1(A, S), h_1(B, S))\) and \(f_2(A, B) = g_2(r_2(A, S), h_2(B, S))\).

![Figure 3. The illustration of the Poly_Bi Decomposition, where \(f_1(A, B) = g_1(r_1(A, S), h_1(B, S))\) and \(f_2(A, B) = g_2(r_2(A, S), h_2(B, S))\).](image)

Figure 4 gives an example of the Poly_Bi Decomposition. The four variable polymorphic Boolean function \(f_1/f_2\) is decomposed to a three variable polymorphic Boolean function \(r_1/r_2\) and a two variable polymorphic Boolean function \(h_1/h_2\). In this example, \(A = \{x_2, x_4\}, B = \{x_1\}\) and \(S = \{x_3\}\).

![Figure 4. An example of the Poly_Bi Decomposition, where \(f_1(A, B) = \text{XOR}(r_1(A, S), h_1(B, S))\) and \(f_2(A, B) = \text{OR}(r_2(A, S), h_2(B, S))\).](image)
“find_initial_variable(...)”. Secondly, a better polymorphic bi-decomposition is generated by the “for” loop started at step 6. Finally, according to the measurement at step 16, a polymorphic bi-decomposition is selected.

```
find_initial_variable(f, g) in f/g is a two modes polymorphic Boolean function, g ∈ {AND, OR, XOR}, F is the input variable set of f/g. out The initial polymorphic bi-decomposition
1 for each x1 ∈ F, each x2 ∈ F and x1 ≠ x2 do
2 S ← V = {x1, x2}
3 if there exist r and h satisfy f(x1, x2) = g(r(x1), S), h(x2, S) then
4 if there exist g2, r2 and h2 satisfy f(x1, S, x2) = g2(r2(x1), S), h2(x2, S) then
5 return (g2, r2, h2)
6 return (NULL, φ, φ)
```

Figure 6. find the initial polymorphic bi-decomposition of f/g.

However, for some polymorphic Boolean functions, the algorithm in Figure 5 cannot give a decomposition. For example, there does not exist a polymorphic bi-decomposition for the polymorphic Boolean function “4 bit parity / 4 bit majority” shown in Figure 7.

```
| x1 | x2 | x3 | x4 | 4 bit parity | 4 bit majority |
|----|----|----|----|-------------|--------------|
| 0  | 0  | 0  | 0  | 0           | 0            |
| 0  | 0  | 0  | 1  | 0           | 1            |
| 0  | 0  | 1  | 0  | 0           | 1            |
| 0  | 1  | 0  | 0  | 0           | 1            |
| 0  | 0  | 1  | 1  | 0           | 1            |
| 0  | 1  | 0  | 1  | 0           | 0            |
| 0  | 1  | 1  | 0  | 0           | 0            |
| 1  | 0  | 0  | 0  | 0           | 0            |
| 1  | 0  | 0  | 1  | 0           | 1            |
| 1  | 0  | 1  | 0  | 0           | 1            |
| 1  | 1  | 0  | 0  | 0           | 0            |
```

Figure 7. The polymorphic Boolean function “4 bit parity / 4 bit majority”

```
Poly_Design(f/g)
1 if | F | > 2 then
2 {p, r, h} ← Merge&Decomposition(f/g)
3 if p = NULL then
4 {g, r, h} ← Merge&Decomposition(f/g)
5 if r is a single mode Boolean function then
6 The bi_decomposition method in [10] is adopted to design a circuit implementing r
7 else
8 Poly_Design(r)
9 if h is a single mode Boolean function then
10 The bi_decomposition method in [10] is adopted to design a circuit implementing h
11 else
12 Poly_Design(h)
13 else
14 Poly_Design(r, h)
15 Poly_Design(h, r)
```

Figure 10. The algorithm to design polymorphic circuits through the polymorphic bi_decomposition method.
For any polymorphic Boolean function, it can be decomposed through the algorithm in Figure 5 or Figure 8. Firstly, it is checked that whether a polymorphic gate can decompose the polymorphic Boolean function to two simpler polymorphic Boolean functions (Figure 5). If the answer is yes, a polymorphic bi_decomposition is obtained. Otherwise, the original polymorphic Boolean function is merged to a single mode Boolean function, and the traditional bi_decomposition approach [10] is adopted to decompose the single mode Boolean function. And the new obtained two single mode Boolean functions are transformed to polymorphic Boolean functions (Figure 8).

Similar to the circuit design process through bi decomposition in [10], Figure 10 gives the algorithm to design polymorphic circuits through the Poly_Bi_Decomposition.

Figure 11 shows the “4 bit parity / 4 bit majority” polymorphic circuits designed by the algorithm in Figure 10.

![Figure 11](https://via.placeholder.com/150)

**Figure 11.** The “4 bit parity / 4 bit majority” polymorphic circuit designed by the polymorphic bi_decomposition method

B. The Transformation&Bi Decomposition method

In fact, through a transformation process, the bi decomposition method in [10] can be used to design polymorphic circuits directly. The steps to design a polymorphic circuit are given below.

(1) The polymorphic Boolean function \(f_1/f_2(x_1, \ldots, x_n)\) is transformed to a single mode Boolean function \(f'(x_0, x_1, \ldots, x_n)\), where \(f'(x_0 = 0, x_1, \ldots, x_n) = f_1(x_1, \ldots, x_n)\) and \(f'(x_0 = 1, x_1, \ldots, x_n) = f_2(x_1, \ldots, x_n)\).

(2) The bi decomposition method in [10] is adopted to design the circuit Cir implementing \(f'(x_0, x_1, \ldots, x_n)\).

(3) For every gate g of Cir, if \(x_0\) or \(\overline{x_0}\) is the input of g, let \(Var_g\) denote the input variable set which influence the output of g. Suppose \(Cir_g\) is the subcircuit which is composed of g and all the logic gates in Cir that would influence the output of g. Let \(Cir_g(x_0 = 1)\) denote the function of \(Cir_g\) when the value of \(x_0\) is logic-1, and \(Cir_g(x_0 = 0)\) denote the function of \(Cir_g\) when the value of \(x_0\) is logic-0.

\[(3.1) \ |Var_g| \leq 3.\]  
Clearly, \(Cir_g(x_0 = 0)\) and \(Cir_g(x_0 = 1)\) perform as some logic gates, and they are denoted as \(g_1\) and \(g_2\), respectively. The subcircuit \(Cir_g\) is replaced by the polymorphic gate \(g_1/g_2\).

\[(3.2) \ |Var_g| > 3.\]  
Without loss of generality, suppose \(x_0\) is connected to input pin A of g. Suppose the subcircuit connected to the input pin B of g is Cir\_g\_B.

\[(3.2.1) \text{If } g = \text{AND, } g(0, Cir\_g\_B) = 0 \text{ and } g(1, Cir\_g\_B) = Cir\_g\_B. \text{ Therefore, } g \text{ is replaced by the polymorphic gate WIRE/WIRE.}\]

\[(3.2.2) \text{If } g = \text{OR, } g(0, Cir\_g\_B) = Cir\_g\_B \text{ and } g(1, Cir\_g\_B) = 1. \text{ Therefore, } g \text{ is replaced by the polymorphic gate WIRE/ONE.}\]

\[(3.2.3) \text{If } g = \text{XOR, } g(0, Cir\_g\_B) = Cir\_g\_B \text{ and } g(1, Cir\_g\_B) = \overline{Cir\_g\_B}. \text{ Therefore, } g \text{ is replaced by the polymorphic gate WIRE/NOT.}\]

After the operations in Step 3, the circuit Cir is transformed to a polymorphic circuit implementing the function \(f_1/f_2(x_1, \ldots, x_n)\).

The rest part of this section gives the process of constructing the “4 bit parity / 4 bit majority” polymorphic circuit.

Firstly, the polymorphic Boolean function in Figure 7 is transformed to a single mode Boolean function in Figure 9(a). Secondly, the bi decomposition method in [10] is adopted to design a traditional logic circuit implementing the function in Figure 9(a), and the structure of the circuit is shown in Figure 12. Thirdly, those parts in dashed rectangles are replaced according to the rules given in Step 3. Finally, the polymorphic circuit wanted is obtained, and it is shown in Figure 13.

![Figure 12](https://via.placeholder.com/150)

**Figure 12.** The circuit implementing the Boolean function in Figure 9(a)

![Figure 13](https://via.placeholder.com/150)

**Figure 13.** The “4 bit parity / 4 bit majority” polymorphic circuit designed by the Transformation&Bi Decomposition method
In fact, with the transformation method in Step 1 and Step 3, any traditional circuit design methods (such as ABC [14] and Espresso [15]) can be used to design polymorphic circuits.

For example, a polymorphic circuits implementing \( f(x_1, \ldots, x_n) / f_2(x_1, \ldots, x_n) \) can be designed by ABC with the following steps. (1) \( f(x_1, \ldots, x_n) / f_2(x_1, \ldots, x_n) \) is transformed to a single mode Boolean function \( f'(x_0, x_1, \ldots, x_n) \), where \( f'(0, x_1, \ldots, x_n) = f_1(x_1, \ldots, x_n) \) and \( f'(1, x_1, \ldots, x_n) = f_2(x_1, \ldots, x_n) \). (2) The ABC method is used to design the circuit \( C \) implementing \( f'(x_0, x_1, \ldots, x_n) \). (3) Part of the circuit \( C \) is replaced by the corresponding polymorphic gates. At the end, the polymorphic circuit implementing \( f(x_1, \ldots, x_n) / f_2(x_1, \ldots, x_n) \) is obtained.

**V. Experiments**

In this section, “parity / majority”, “multiplier / sorting-net”, and some polymorphic Boolean function constructed by the traditional Boolean function selected from the MCNC [17] library are adopted to test the performance of the proposed methods. The results of the proposed methods are compared with both PolyBDD and polymorphic multiplex methods introduced in [9].

In the design process, both traditional logic gates and polymorphic logic gates are adopted. When a polymorphic Boolean function is decomposed, the traditional logic gates adopted are AND, OR and XOR, and the polymorphic gates are \{AND/OR, AND/XOR, OR/AND, OR/XOR, XOR/AND, XOR/OR\}.

### TABLE II. THE EXPERIMENTAL RESULTS ON “MULTIPLIER / SORTING-NET”

| The number of inputs | 2x3 / 5bit | 3x3 / 6bit | 3x4 / 7bit | 4x4 / 8bit | 5x5 / 10bit | 6x6 / 12bit |
|----------------------|------------|------------|------------|------------|-------------|-------------|
| PolyBDD [9]          | –          | –          | –          | 1028 (–)   | –           | –           |
| Multiplex method based on ABC [9] | 61 (–) | 119 (–) | 198 (–) | 359 (–) | – | – |
| Multiplex method based on Espresso [9] | – | – | – | 2309 (–) | – | – |
| Poly_Bi_Decomposition | 49 (8.6%) | 145 (35.8%) | 248 (27.0%) | 570 (43.8%) | 2507 (36.5%) | 10130 (25.1%) |
| Transformation&Bi_Decomposition | 65 (20.0%) | 170 (22.3%) | 263 (11.0%) | 630 (13.5%) | 2667 (7.7%) | 10329 (5.1%) |

### TABLE III. THE EXPERIMENTAL RESULTS ON “PARITY / MAJORITY”

| The number of inputs | 7 | 9 | 11 | 13 | 15 |
|---------------------|---|---|----|----|----|
| Multiplex method based on ABC [9] | 39 (–) | 58 (–) | 79 (–) | 112 (–) | – |
| Poly_Bi_Decomposition | 41 (1) | 59 (2) | 90 (1) | 128 (2) | 186 (1) |
| Transformation&Bi_Decomposition | 64 (5) | 71 (2) | 181 (5) | 144 (2) | 999 (126) |

### TABLE IV. THE EXPERIMENTAL RESULTS ON 8 POLYMORPHIC CIRCUITS

|                      | majority10 / sao24 (10 → 1) | parity10 / sao24 (10 → 1) | 4x4mul / f51m (8 → 8) | sorting-net8 / f51m (8 → 8) |
|----------------------|----------------------------|---------------------------|-----------------------|-----------------------------|
| Poly_Bi_Decomposition | 206 (10.6%) | 54 (40.7%) | 354 (16.1%) | 175 (25.1%) |
| Transformation&Bi_Decomposition | 208 (5.7%) | 119 (11.7%) | 375 (5.3%) | 235 (8.0%) |
| ex1010 / sorting_net10 (10 → 10) | 2,789 (23.8%) | 98 (60.2%) | 3,587 (40.2%) | 4,571 (48.5%) |
| Transformation&Bi_Decomposition | 3,022 (6.1%) | 152 (13.8%) | 3,716 (9.1%) | 4,682 (7.5%) |

Table II shows the experimental results of “multiplier / sorting-net”. In Table II, the number outside the bracket is the number of gates consumed, and the number inside the bracket is the percentage of the polymorphic gates. It can be observed from Table II that the Poly_Bi_Decomposition method consumes less gate resource than the Transformation&Bi_Decomposition method. The difference of the two proposed methods is not large in terms of gate resource. But, the polymorphic gate percentage of the circuits designed by the Poly_Bi_Decomposition method is much higher. The “Multiplex method based on ABC” can build the most gate-efficient polymorphic circuits. However, because the multiplex is used to switch the output of different subcircuits to the multiplex’s output, the “Multiplex method based on ABC” do not really utilize the build-in multi-functional property of polymorphic gates.

Table III shows the results of the “parity / majority”. In Table III, the number outside the bracket is the number of gates consumed, and the number inside the bracket is the number of the polymorphic gates. It can be observed that the “Multiplex method based on ABC” still performs the best in terms of gate resource. The Poly_Bi_Decomposition method is comparable with the “Multiplex method based on ABC”.

Table IV shows the results of the “parity / majority” and 8 polymorphic circuits. The number outside the bracket is the number of gates consumed, and the number inside the bracket is the percentage of polymorphic gates.
Table IV shows the experimental results on 8 polymorphic circuits. The circuits, including sao24, 5xp1, z5xp1, ex101, misex3, misex3c and f51m, are taken from the MCNC [17] library. Majority10 is the 10 bit majority Boolean function. Parity10 is the 10 bit parity Boolean function. Sorting-net8 and sorting-net10 are 8 bit and 10 bit sorting-net Boolean function, respectively. It can be observed from Table IV that the Poly_Bi_Decomposition method consumes less gate resource, and the designed circuits have a higher percentage of polymorphic gates. Especially for polymorphic circuits “parity10/sao24” and “5xp1/z5xp1”, the gate resource consumed by the Poly_Bi_Decomposition method is much less than those of the Transformation&Bi_Decomposition method, and the percentage of polymorphic gates of the polymorphic circuits designed by Poly_Bi_Decomposition method is much higher.

VI. DISCUSSIONS

Based on the Bi_Decomposition method in [10], the Poly_Bi_Decomposition method and the Transformation&Bi_Decomposition method are proposed to design polymorphic circuits. The former decomposes the polymorphic Boolean function through a polymorphic gate. The later transforms the polymorphic Boolean function to a single-mode function, and the traditional Bi_Decomposition method is adopted to implement the single mode function. At the end, parts of the circuits are replaced by some polymorphic gates, and the wanted polymorphic circuit is obtained.

Compared with the BDD and multiplex methods in [9], the Bi_Decomposition based methods proposed in this paper can design polymorphic circuits with a higher percentage of polymorphic gates. The BDD method in [9] consumes many multiplexes, and only the lowest level of the circuit consists of polymorphic gates. As for the multiplex method in [9], each single functional subcircuit is designed by traditional logic gates, and polymorphic multiplexes switches one subcircuit’s output to the final output. This approach ignores the build-in multi-functional characteristic of polymorphic gates. Oppositely, the Poly_Bi_Decomposition method makes full use of the multi-functional property of polymorphic gates. The polymorphic circuits obtained have a higher percentage of polymorphic gates, and the number of logic gates consumed is relatively reasonable.

The two methods proposed in this paper show their benefits in different aspects. The Poly_Bi_Decomposition method makes full use of the multi-functional property of polymorphic gates, and the circuits designed by the Poly_Bi_Decomposition method have a higher percentage of polymorphic gates. The methodology behind the Transformation&Bi_Decomposition is more universal. In fact, with the transformation method in Section IV-B, any traditional circuit design methods (such as ABC [14]) can be used to design polymorphic circuits. Usually, the percentage of polymorphic gates in circuits designed by the Transformation&Bi_Decomposition method is low, but the traditional circuits design methods with high performance, such as ABC [14], can be used to design polymorphic circuits directly.

VII. CONCLUSIONS

Based on the Bi_Decomposition method, the Poly_Bi_Decomposition method and the Transformation&Bi_Decomposition method are proposed to design polymorphic circuits. The Poly_Bi_Decomposition method makes full use of the build-in multi-functional property of polymorphic gates, the circuits obtained consumes less gate resource and have a higher percentage of polymorphic gates. Meanwhile, the methodology behind the Transformation&Bi_Decomposition method can be used to design polymorphic circuits through any traditional circuits design method.

ACKNOWLEDGMENT

This work is partly supported by the Fundamental Research Funds for the Central Universities. This paper is based on Zhifang Li's PhD dissertation (in Chinese) in School of Computer Science and Technology at University of Science and Technology of China in June, 2011.

REFERENCES

[1] A. Stoica, R. Zebulum, and D. Keymeulen, "Polymorphic Electronics," in Proc. the 4th International Conference on Evolvable Systems: From Biology to Hardware, Tokyo, Japan, 2001, pp. 291-301.
[2] A. Stoica, R. Zebulum, and D. Keymeulen, "Taking evolutionary circuit design from experimentation to implementation: some useful techniques and a silicon demonstration," Computers and Digital Techniques, IEE Proceedings, vol. 151, no. 4, pp. 295-300, 18 July 2004.
[3] A. Stoica, R. Zebulum, and D. Keymeulen, "On Polymorphic Circuits and Their Design using Evolutionary Algorithms," in Proc. IASTED International Conference on Applied Informatics (AI2002). Innsbruck, Austria, 2002.
[4] R. Ruzicka, L. Sekanina, and R. Prokop, "Physical Demonstration of Polymorphic Self-checking Circuits," in Proc. the 14th IEEE International On-line Testing Symposium, Los Alamitos, US, 2008, pp. 31-36.
[5] R. S. Zebulum and A. Stoica, "Four-Function Logic Gate Controlled by Analog Voltage," NASA Tech Briefs 2006.
[6] H. Liang, W. Luo, and X. Wang, "Designing Polymorphic Circuits with Evolutionary Algorithm Based on Weighted Sum Method," in Proc. the 7th International Conference on Evolvable Systems: From Biology to Hardware. Wuhan, China, 2007, pp. 331-342.
[7] L. Sekanina, "Evolutionary design of gate-level polymorphic digital circuits," in Proc. Applications of Evolutionary Computing: EvoWorkshops 2005. Lausanne, Switzerland, 2005, pp. 185-194.
[8] L. Sekanina, L. Starecek, Z. Kotasek, and Z. Gajda, "Polymorphic Gates in Design and Test of Digital
[9] Z. Gajda and L. Sekanina, "Gate-Level Optimization of Polymorphic Circuits Using Cartesian Genetic Programming," in Proc. the Eleventh conference on Congress on Evolutionary Computation. Trondheim, Norway, 2009, pp. 1599-1604.

[10] B. Steinbach and C. Lang, "Exploiting Functional Properties of Boolean Functions for Optimal Multi-level Design by Bi-decomposition," Artificial Intelligence Review, vol. 20, pp. 319-360, 2003.

[11] W. Luo, Z. Zhang, and X. Wang, "Designing Polymorphic Circuits with Polymorphic Gates: a General Design Approach," Circuits, Devices & Systems, IET, vol. 1, no. 6, pp. 470-476, December 2007.

[12] Z. Li, W. Luo, L. Yue, and X. Wang, "On the Completeness of the Polymorphic Gate Set," ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 15, no. 4, p. article 32, 2010.

[13] L. Sekanina, "Design Methods for Polymorphic Digital Circuits," in Proc. the 8th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop (DDECS 2005). University of West Hungary, Sopron, Hungary, 2005, pp. 145-150.

[14] Berkley Logic Synthesis and Verification Group, "ABC: A System for Sequential Synthesis and verification," [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc/.

[15] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, Logic Minimization Algorithms for VLSI Synthesis: Kluwer Academic Publishers, 1984.

[16] A. Mishchenko, B. Steinbach, and M. Perkowski, "An Algorithm for Bi-Decomposition of Logic Functions," in Proc. 38th Design Automation Conference (DAC 2001). Las Vegas, Nevada, USA, 2001, pp. 103-108.

[17] S. Yang, "Logic Synthesis and Optimization. Benchmarks, Version 3.0," Microelectronics Center of North Carolina 1991.