

# Digital Circuit Evolution Using SAT Solver (Evolvable Hardware)



By

**MUMTAZ ALI**

**2010-NUST-MS-EE-30**

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of  
Science in Electrical Engineering

School of Electrical Engineering and Computer Science,  
National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(August 2012)

©Copyright

by

Mumtaz Ali

2012

*to my*

*FAMILY*

# Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Dr. Osman Hasan for the continuous support of my M.Sc study and research, for his patience, motivation, and immense knowledge. His guidance helped me in all the process of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Awais Kamboh, Dr. Rehan Hafiz, and Mr. Nasir Mahmood, for their encouragement, insightful comments, and hard questions.

My sincere thanks also goes Dr. Syed Irfan Ahmed for his constant support during my thesis research.

I would like to thank my fellow Umer Kakli who always helped me in the documentation of my thesis.

Last but not the least, I would like to thank my family: my parents , my brothers and sister for their constant support and patience.

# Abstract

Evolutionary computation uses Darwinian principles to find solutions from a given search space and forms the basis for evolving digital circuits. One of the most computationally expensive steps in evolutionary computation is the comparison of the candidate circuit (chromosome) with the target truth table. We propose to use SAT (satisfiability) solvers to improve upon the efficiency of this process, which is traditionally done using exhaustive simulation. However, traditional SAT solvers, which return the satisfiability of a boolean expression in Yes/No format, cannot be used in this context since we need the percentage (score) of equivalence between two circuits. This thesis presents a SAT solver that fulfills this requirement. We use this novel SAT solver to develop a digital circuit evolution methodology based on the principles of Cartesian Genetic Programming (CGP). The proposed methodology performs exceptionally well for circuits whose behavior can be expressed compactly in terms of CNF (Conjunctive Normal Form) clauses. For illustration purposes, the proposed methodology has been used to evolve digital circuits exhibiting the behaviors of adders, multipliers, muxes, encoders, even parity circuits and a few LGSynth91 benchmarks.

# Table of Contents

	<b>Page</b>
Acknowledgements . . . . .	iv
Abstract . . . . .	v
Table of Contents . . . . .	vi
List of Figures . . . . .	viii
List of Tables . . . . .	ix
<b>Chapter</b>	
1 Introduction . . . . .	1
1.1 Evolvable Hardware . . . . .	1
1.1.1 Motivation . . . . .	1
1.1.2 Scalability of Evaluation Time . . . . .	3
1.2 Preliminaries . . . . .	4
1.2.1 Digital Circuit Evolution . . . . .	4
1.2.2 Functional Equivalence Checking using SAT Solving . . . . .	8
1.3 Proposed Methodology . . . . .	9
1.4 Thesis Organization . . . . .	12
2 Related Work . . . . .	13
2.1 Scalability of Representation . . . . .	13
2.2 Scalability of Evaluation Time . . . . .	14
2.3 SAT Based Formal Verification . . . . .	15
3 SAT Based Fitness Scoring . . . . .	16
3.1 CNF Conversion . . . . .	16
3.2 Proposed SAT Solver . . . . .	18
4 Experimental Results . . . . .	22
4.1 Full Adder . . . . .	22

4.2	8x1 Mux . . . . .	24
4.3	8 Bit Parity . . . . .	25
	4.3.1 LGSynth91 c17 . . . . .	26
4.4	Discussions . . . . .	28
5	Conclusions . . . . .	30

# List of Figures

1.1	Genetic Algorithms based Evolutionary Computing . . . . .	2
1.2	A Candidate Circuit . . . . .	6
1.3	Fitness Function of Conventional CGP . . . . .	7
1.4	An example of Parallel Simulation . . . . .	8
1.5	Proposed Methodology . . . . .	10
3.1	Three Overlapping Sets and their Union . . . . .	20
4.1	Full Adder obtained by Conventional CGP . . . . .	23
4.2	Full Adder obtained by Proposed CGP . . . . .	23
4.3	8x1 Mux obtained by Conventional CGP . . . . .	24
4.4	8x1 Mux obtained by Proposed CGP . . . . .	25
4.5	8 Bit Parity Circuit obtained by Conventional CGP . . . . .	26
4.6	8 Bit Parity Circuit obtained by Proposed CGP . . . . .	26
4.7	c17 Circuit obtained by Conventional CGP . . . . .	27
4.8	c17 Circuit obtained by Proposed CGP . . . . .	27



# List of Tables

1.1	List of Available Functions in CGP . . . . .	5
3.1	Resolution Rules for Some Common Gates . . . . .	18
3.2	Fitness Calculation Example . . . . .	20
4.1	Mean Evaluation Time Per Chromosome for Conventional CGP $t_{cgp}$ and SAT-Based CGP $t_{mcgp}$ . . . . .	29

# Chapter 1

## Introduction

### 1.1 Evolvable Hardware

Evolvable Hardware is a domain in which evolutionary computation or other bio-inspired algorithms are used for automated hardware design, dynamic adaptive hardware, self replication or self repair [2, 16, 5, 1, 17, 11]. This makes evolvable hardware a very interesting multidisciplinary research domain involving biology, computer science and engineering. The focus of this thesis is on using the foundations of evolvable hardware in the context of automated combinational digital circuit design, i.e, a scenario in which evolutionary algorithms are used to evolve combinational digital circuits. This approach is known for providing better synthesis results than the traditional methods [12, 19].

#### 1.1.1 Motivation

Genetic algorithm based evolutionary computing [46], depicted in Figure 1.1, is one of the most commonly used methods for digital circuit evolution. The main strength of genetic algorithms is the ability to automatically search the required solution from a given search space without any prior knowledge about the problem. The input to the genetic algorithm based evolutionary computing is a target truth table and its goal is to find a solution circuit, or *chromosome*, whose truth table matches with the truth table of the target function. Initially, the algorithm starts with a population of random chromosomes. The algorithm has access to a *fitness function* that measures the closeness level of a candidate chromosome and the target function. This fitness function is used to evaluate each chromosome of the

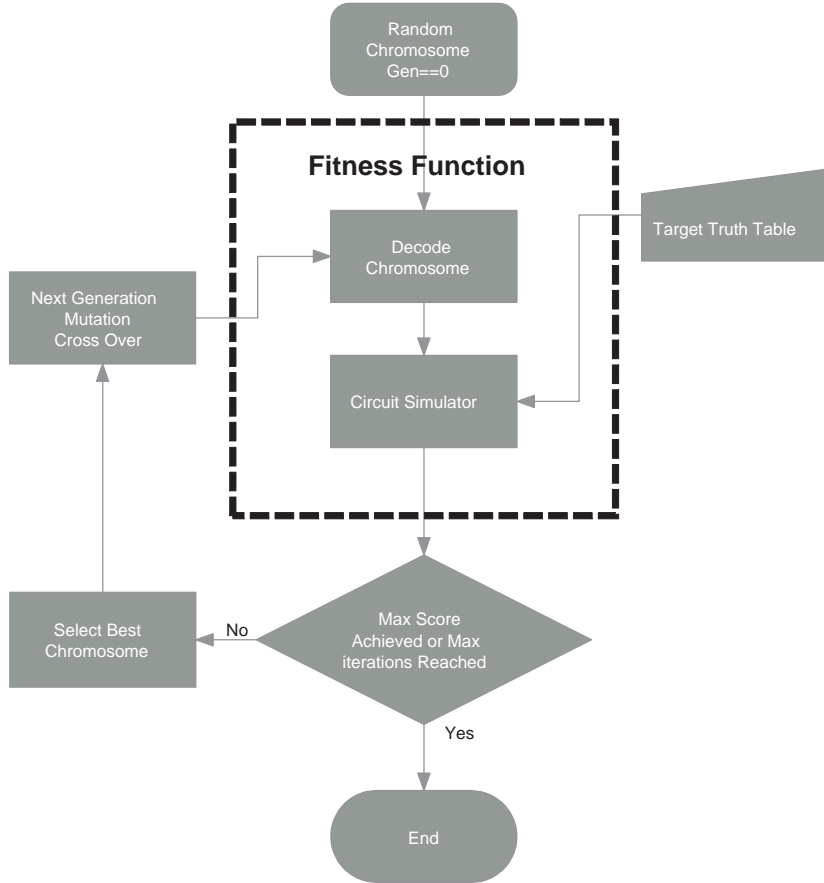


Figure 1.1: Genetic Algorithms based Evolutionary Computing

present generation and assign a fitness score to it. Chromosomes with the best fitness score are selected to produce the next generation of population by recombination/crossover or random mutation. A number of selection strategies have been reported in the open literature. For example, in the tournament selection strategy [26], chromosomes are randomly divided into groups and the chromosomes with the best fitness score from each group are selected to produce the new generation. The genetic algorithm keeps on running until the target solution is achieved, or the maximum number of allowed iterations is reached. A number of genetic algorithm based evolutionary techniques for evolving digital circuits have been reported in the literature. Some promising ones include Koza's genetic programming [3] and Cartesian genetic programming (CGP) [21]. Cartesian genetic programming

is more popular mainly because it is more efficient in terms computation time and required resources [23, 30, 31] than the other biologically-inspired methods.

### 1.1.2 Scalability of Evaluation Time

Traditionally, the fitness scores are calculated using exhaustive simulation in digital circuit evolution. The main idea is to compare the output of the given chromosome and the target function using all the possible input patterns. This kind of exhaustive simulation consumes a significant amount of computation time, which grows exponentially with an increase in the number of inputs or functional complexity of the target function. This enormous computation time requirement is one of the major factors that limits the scope of digital circuit evolution [25].

In this thesis, we propose to solve the above mentioned computation time problem by using SAT solvers [52] to assess the fitness scores in digital circuit evolution. SAT solvers are known to be computationally faster than simulation in the task of equivalence checking [18, 14, 10, 24]. To the best of our knowledge, SAT solvers have never been used for fitness scoring in digital circuit evolution before.

This thesis presents a complete methodology for using SAT solvers to assess the fitness scores in digital circuit evolution. The proposed methodology is primarily based on the Cartesian Genetic Programming(CGP) [20] technique. We have developed a variant of CGP in which chromosomes are evaluated using SAT solving. Since, traditional SAT solvers do not provide a comparison score between the circuits that are being checked for equivalence, so we also implemented our own SAT solver in the reported work. The main principle is to convert the target function behavior and the given chromosome into a CNF and then evaluate their fitness based on the number of inputs assignments for which the CNF is unsatisfiable. The complete code is written in C++. For illustrating the effectiveness of the proposed methodology and our development, we utilize it to evolve digital circuits exhibiting the behavior of adders, multipliers, multiplexers, even parity circuits, encoders and a few LGSynth91 benchmarks. It is worth mentioning that a significant time

gain was observed for circuits in which the number of CNF clauses in the CNF representation is a linear of the number of inputs.

## 1.2 Preliminaries

In this section, we describe some foundational concepts about digital circuit evolution and SAT based equivalence checking along with some commonly used terminology. The information is expected to be helpful in understanding the main contributions of the thesis that are described later.

### 1.2.1 Digital Circuit Evolution

The main principle of digital circuit evolution is based on the genetic algorithms based evolutionary computing as illustrated in Figure 1.1. The target specification is the functionality of the desired circuit in this case and the chromosomes represent digital circuits in coded form. The fitness scores are usually computed by comparing each chromosome with the target circuit using exhaustive simulation.

Cartesian Genetic Programming (CGP), developed by Miller and Thomson, is the most widely used technique of the digital circuit evolution [23, 21, 29, 30, 31]. In CGP, we represent a candidate circuit as a two dimensional ( $n_r \times n_c$ ) grid of programmable nodes where  $n_c$  represents the number of columns and  $n_r$  represents the number of rows. Each node in this 2D grid is programmable and can acquire any one of the 20 available functions, given in Table 1.1. The function acquired by a node is identified by a unique identifier as listed in Table 1.1.

We illustrate the CGP based evolution of digital circuits by considering an example of a candidate circuit given in Figure 1.2. This circuit is composed of a 2x2 grid with four nodes and three inputs and two outputs. In CGP, the inputs and outputs of the nodes are represented by distinct integers such that the inputs are labeled first, starting from the integer 0 and then the output of each node is labeled in a column wise fashion. Thus, in

Table 1.1: List of Available Functions in CGP

Function Number	Function	Function Number	Function
0	0	10	xor(a,b)
1	1	11	xor(a,!b)
2	a	12	or(a,b)
3	b	13	or(a,!b)
4	!a	14	or(!a,b)
5	!b	15	or(!a,!b)
6	and(a,b)	16	mux(a,b)
7	and(a,!b)	17	mux(a,!b)
8	and(!a,b)	18	mux(!a,b)
9	and(!a,!b)	19	mux(!a,!b)

our example circuit of Figure 1.2, numbers 0,1 and 2 represent the inputs and 3,4,5 and 6 represent the outputs of nodes, respectively. Node inputs can be connected either to a node output from one of the previous  $\ell$  columns or to one of the inputs of the circuit. This way feedback loops are restricted. The factor  $\ell$  denotes *level-back* which is used to limit the design space by not allowing a node input to be connected to any column which is behind  $\ell$  allowed columns.

A chromosome representation of the candidate circuit of Figure 1.2 is also provided below it in the form of an integer string. A node in the chromosome is represented by four integers. First three integers show the input connectivity of the node and the last integer identifies its functionality, as given in Table 1.1. The last set of integers in the chromosome shows the output connectivity and thus contains integers equal to the number of outputs. This entry, in the case of our example, is 623, which corresponds to the three outputs of the candidate circuit. In this way, the chromosome captures the complete behavior, including the functionality and structure, of the candidate circuit.

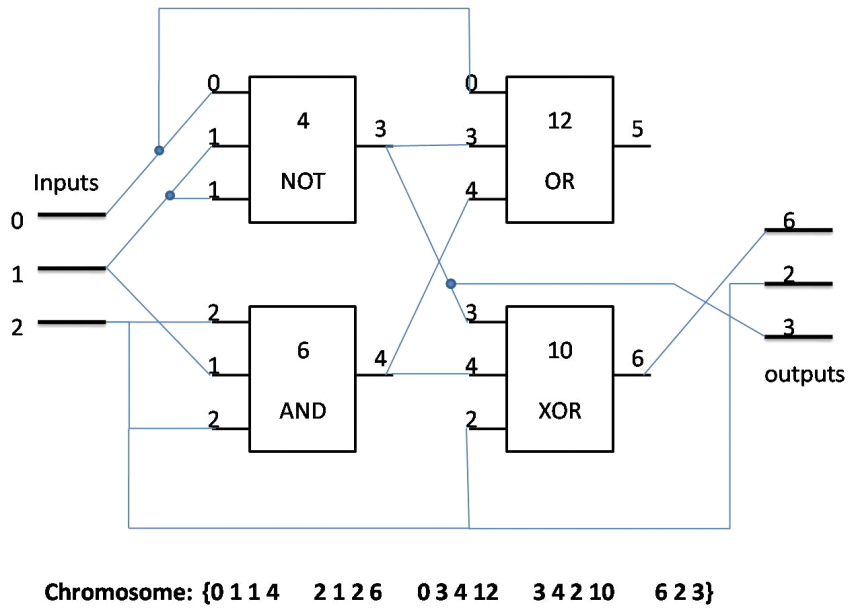


Figure 1.2: A Candidate Circuit

Initially CGP starts with a population of randomly chosen chromosomes, which is usually referred to as the first generation. Fitness function evaluates each chromosome and assigns a fitness score to it. The chromosome with the best fitness score is selected as the parent for the next generation. The selected chromosome is then randomly mutated to produce the next generation. Random mutation is a process in which connectivity of the node input or output is randomly changed. Mutation rate is determined by the user. The parameter  $\lambda$ , which is a user defined term, defines the size of the population in one generation. Each new generation includes the best chromosome from the previous generation and its  $\lambda$  mutated versions. CGP keeps evolving new generations until the solution circuit is acquired or the maximum number of iterations is reached.

The fitness function plays the most important role in the evolution of digital circuits as it is the fitness score that guides the selection of the next generation. In CGP, fitness of a chromosome is calculated using exhaustive simulation as illustrated in Figure 1.3. The main idea is to apply all possible inputs to the candidate circuit and compute the hamming distance between each one of its outputs with the corresponding output of the truth table

of the target circuit. The hamming distance is then used to assess the fitness score of the candidate circuit. This process involves testing for all possible inputs and thus can be quite expensive in terms of computation time. This fact is one of the main limiting factors in the domain of digital circuit evolution.

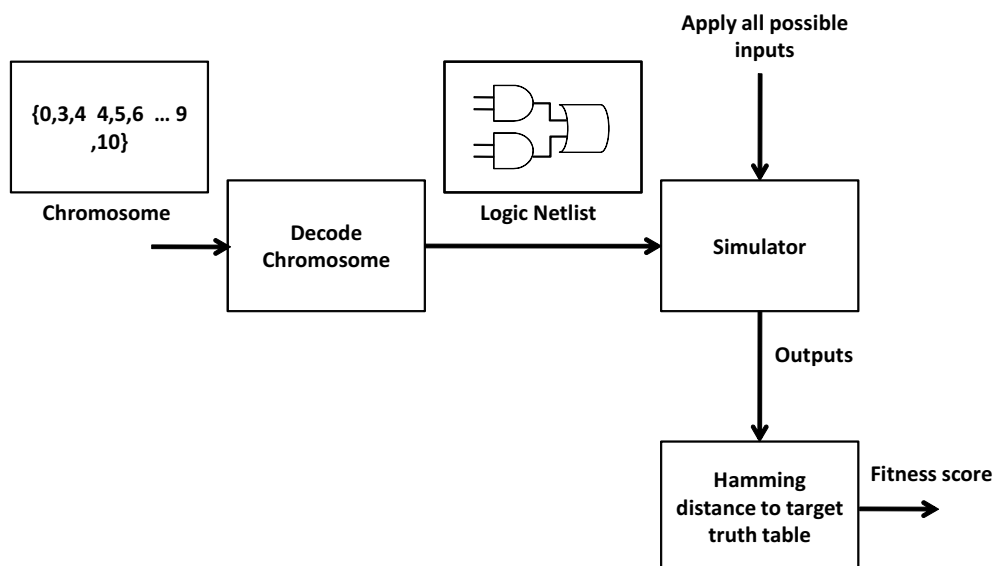


Figure 1.3: Fitness Function of Conventional CGP

Parallel simulation is a technique that can be used to improve the fitness score calculation time for standard CGP [21]. The main principle of parallel simulation is to leverage upon the bitwise logical operations supported by languages like C. This allows us to perform more than one evaluation of a circuit by a single instruction. For example, an integer in C has a width of 32-bits so 32 logical operations for a gate can be executed by one instruction. For example, we can simulate a circuit with up to 5 inputs ( $2^5=32$ ) by applying a single 32 bit vector at each input as illustrated in Figure 1.4. The current thesis is also targeted towards the same goal, i.e., improving the fitness scoring time. However, we propose to use SAT solver based scoring instead of exhaustive simulation. The proposed method performs better than the parallel simulation based fitness scoring, which is the state-of-the-art technique, for a wide range of circuits, as will be demonstrated in Chapter 4 of this thesis.



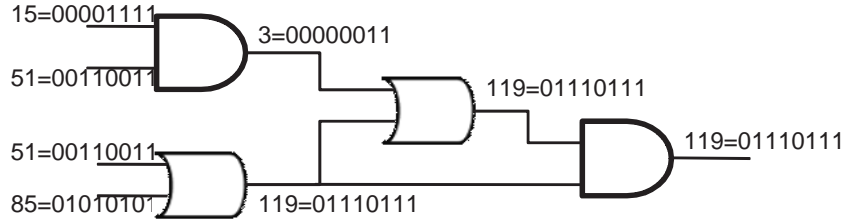


Figure 1.4: An example of Parallel Simulation

### 1.2.2 Functional Equivalence Checking using SAT Solving

Functional equivalence checking [18, 14, 10] is a method in which two different structures are verified to be functionally equivalent. Functional equivalence checking is a common practice in logic synthesis in which a synthesized netlist is functionally verified against the reference circuit.

A SAT solver [53, 52] is an algorithm that automatically determines if the given boolean expression is true atleast for one particular assignment of its variables. This algorithm has found an enormous application in equivalence checking of boolean circuits because it can handle many interesting equivalence checking problems automatically. The main idea is to form the XOR function of the two boolean expressions of the digital circuits, whose equivalence needs to be verified, in the conjunctive normal form (CNF), i.e., a Boolean formula composed of a conjunction of clauses where each clause is formed by a disjunction of literals (Boolean variables). A SAT solver is used to check the satisfiability of the resulting CNF and the two circuits are termed functionally equivalent if and only if the CNF is unsatisfiable, i.e, the XOR of the two outputs is never true for any input variable assignment. Algorithm 1 briefly explains the usage of SAT solvers for functional equivalence checking. A mitter, used on the line 2 of Algorithm 1, represents the bit-wise XOR operation between the outputs of the two circuits and thus is false in case the circuits are equivalent. Each mitter is then converted to its CNF format and is then passed to the SAT solver to check if it is satisfiable for any assignment of input variables. In case a mitter is found to be satisfiable, circuits are not equivalent to one another and the satisfying assignment can

be used for debugging. A modern SAT solver provides a more efficient way to search for a satisfying assignment than exhaustive simulation and therefore outperforms it [15, 45].

---

**Algorithm 1** SAT Based Equivalence Checking.

**Inputs:**

CircuitA: a set of functions  $\{y_1, y_2, \dots, y_N\}$

CircuitB: a set of functions  $\{f_1, f_2, \dots, f_N\}$

**Output:**

A satisfiable assignment

---

```

1: for  $1 = 1 \rightarrow N$  do
2:    $M_i \leftarrow (y_i \oplus f_i)$   $\triangleright M_i$  is a mitter
3:    $CNF_i \leftarrow \text{boolean\_logic\_to\_CNF}(M_i)$ 
4: end for
5:  $x \leftarrow 0$ 
6: for  $i = 1 \rightarrow N$  do
7:    $\{\text{sat}, \text{assignment}\} \leftarrow \text{satsolver}(CNF_i)$ 
8:   if sat is true then
9:      $x \leftarrow 1$ 
10:    break
11:  end if
12: end for
13: if x is 1 then
14:   Print("circuits are not equal for assignment:")
15:   Print(assignment)
16: else
17:   Print("CircuitA and CircuitB are functionally equivalent")
18: end if

```

---

Traditional SAT based algorithms work in Yes/No fashion, i.e., they can merely inform us if a logical formula is satisfiable or not. They lack the ability to find the closeness of two circuits (fitness scores), which is the main requirement in the case of digital circuit evolution. Therefore, traditional SAT solvers cannot be used in this context as is and in order to leverage upon their efficiency compared to exhaustive simulation, we propose a variant of traditional SAT solvers that is capable of fitness scoring.

## 1.3 Proposed Methodology

Figure 1.5 presents a general block diagram of the proposed methodology, which is primarily based on CGP and SAT solving.

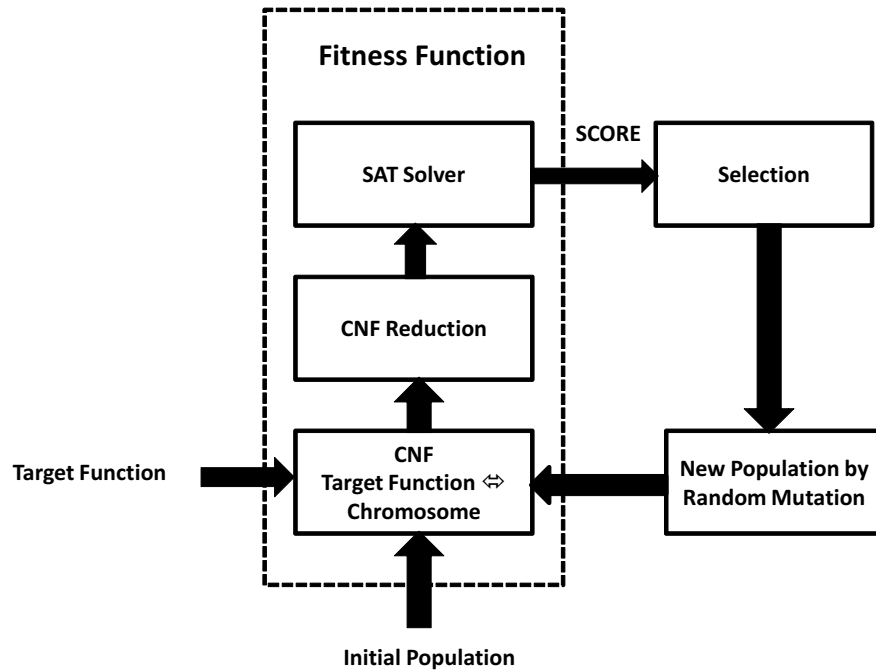


Figure 1.5: Proposed Methodology

The first step is to transform the boolean expression of the target function to its corresponding CNF, which is used in every iteration of the the digital circuit evolution. The next step is to form a combined CNF of the XOR operation of the boolean expression of the chromosomes of the current population and the target function. The combined CNF is reduced as much as possible to minimize the computation overhead and is then given to the SAT solver so that its fitness score can be calculated. It is important to note that the context in which the SAT solver is being used here is different from its traditional usage since the desired output is not a Yes/No kind of an answer to a satisfiability problem but is a number that measures the degree of satisfiability of the given CNF. Traditional SAT solvers do not support this capability and thus, in this work, we developed our own SAT solver that can measure the degree of satisfiability of the given CNF clauses. Chromosome with the best fitness score along with some of its mutated version form the new population for the next generation. This process iterates until a chromosome which is functionally equal to the target function is found or the maximum number of iterations is reached.

We accept the target function and the initial population of chromosomes in the form of minimized minterms and maxterms as the input.

---

**Algorithm 2** Proposed Methodology

**Inputs:**

target\_function: target function  
 $n_r$ : number of rows  
 $n_c$ : number of columns  
seed: a seed value for chromosome  
 $\lambda$ : population size  
 $\ell$ : level back  
 $\mu$ : mutation rate

**Output:**

best\_chromosome: chromosome with the best fitness score

---

```

1: for i=1 to  $\lambda+1$  do
2:   if seed=0 then
3:      $chromosome_i \leftarrow \text{RANDOM\_CHR}(\text{seed}, n_r, n_c, \ell)$ 
4:   else
5:      $chromosome_i \leftarrow \text{seed}$ 
6:   end if
7: end for
8: while target chromosome found or max iterations reached do
9:   Initialize best_score with zero
10:  for i=1 to  $\lambda+1$  do
11:    score  $\leftarrow \text{FITNESS\_FUNCTION}(chromosome_i)$ 
12:    if score > best_score then
13:      index  $\leftarrow i$ 
14:      best_score  $\leftarrow$  score
15:    end if
16:  end for
17:  best_chromosome  $\leftarrow chromosome_{index}$ 
18:  if best_score = max_score or max iteration reached then
19:    break loop
20:  end if
21:   $chromosome_1 \leftarrow \text{best\_chromosome}$ 
22:  for i=2 to  $\lambda+1$  do
23:     $chromosome_i \leftarrow \text{MUTATE}(\text{best\_chromosome}, \mu, \ell)$ 
24:  end for
25: end while
26: Print report for best_chromosome

```

---

Algorithm 2 provides the implementation details corresponding to the proposed methodology outlined above. The algorithm accepts the target function and its number of rows  $n_r$  and columns  $n_c$ , the seed value for the chromosomes and the digital circuit evolution parameters  $\lambda, \ell$  and  $\mu$ . The body of Algorithm 2 is primarily composed of the functions: RAN-

DOM\_CHR, FITNESS\_FUNCTION and MUTATE. RANDOM\_CHR returns a randomly generated chromosome using parameters  $\ell$ ,  $n_c$  and  $n_r$ . FITNESS\_FUNCTION accepts a chromosome and returns its fitness score by using our proposed SAT solving. Whereas, the function MUTATE accepts a chromosome and returns its mutated version using parameters  $\mu$  and  $\ell$ . The proposed methodology algorithm returns the chromosome that is found to be closest to the target function in the given number of iterations. In Chapter 3, we present the implementation details associated with the FITNESS\_FUNCTION, i.e., the SAT solver that returns the fitness score between the target function and a given chromosome.

## 1.4 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 provides a brief literature survey regarding the existing digital circuit evolution techniques. The algorithm for the novel SAT solver is described in Chapter 3. The experimental results are presented in Chapter 4 and finally Chapter 5 concludes the thesis.

# Chapter 2

## Related Work

Scalability is one of the major issues faced by researchers in the domain of digital circuit evolution [25, 16]. The problem of scalability is two-fold. Firstly, the evaluation of the closeness of a candidate solution with the target solution is not scalable. Secondly, the search space grows exponentially with respect to the size of the problem and thus is not easy to handle with the given computation and memory constraints. Various methods have been proposed to tackle these scalability issues and this chapter summarizes them besides providing some related work for SAT solving based equivalence checking.

### 2.1 Scalability of Representation

A search space in circuit evolution is the space of all possible representations of circuits in a chromosome. Thus, the size of the search space is directly related to the length of the chromosome, which is chosen by the designer based on the size of the circuit. As the circuit size grows, the chromosome size has to grow accordingly, which results in an exponential growth in the size of the search space. This in turn increases the computation requirements for finding the solution circuit. Numerous researchers have tried to alleviate this problem. The authors in [33, 16] have proposed to perform digital circuit evolution at the functional level and thus use large building blocks instead of gates as basic components. Some limitations of this approach include the manual definition of such blocks for every design by the designer and the inefficiency of digital circuit evolution algorithms at this higher abstraction level with more complex functionality. Incremental evolution [37, 38, 39] is another option in which a problem is first divided into sub modules or units. First,

evolution is performed to individually evolve these units and then these units act as building blocks for evolution of more complex circuits. In the modular approach, the modules are automatically defined to be reused in the evolution process [4, 8]. Combination of modular and incremental evolution has also been used [13, 41]. Computational developments [34, 35, 32, 6, 9, 7] have brought a new direction to this field with promising theoretical as well as practical results but the issue of scalability is still an open challenge because all the above mentioned techniques compromise on the diversity of the search space.

## 2.2 Scalability of Evaluation Time

Fitness evaluation time also grows exponentially with an increase in the size of circuit. Traditionally, fitness score is calculated using exhaustive simulation where all possible inputs are tested. There are a number of strategies reported so far to tackle this issue. In some cases such as filters, classifiers or robot controllers, where a circuit is required to work only for a subset of inputs, fitness scoring is done by partial simulation. However, this approach is not applicable to circuits where a correct response is required for all possible inputs [42]. If a target system is linear, it is possible to completely evaluate a candidate circuit using only one input vector [44]. Z. Vasicek et al. proposed to verify a candidate circuit against the target solution using a SAT based equivalence checking algorithm. Due to limitation of traditional SAT solvers (which work in yes/no fashion), the method only works for the post synthesis optimization phase where a circuit is initially synthesized using a conventional synthesis tool first and then it is further optimized in evolutionary framework using SAT solvers [45]. In the current thesis, we overcome this restriction by proposing a SAT solver based fitness scoring.

## 2.3 SAT Based Formal Verification

The main focus of SAT solving is to optimize the search for a satisfying assignment of the variables of a given boolean function in terms of time taken and resources utilized. Growing demand for more efficient and scalable verification solutions have fueled the research in SAT based verification techniques in the last two decades [18, 14, 10, 24, 47, 48, 49, 50]. MiniSAT [52] and Picosat [53] are two of the most popular SAT solvers these days.

Converting the boolean expression into their corresponding CNF format is an important step in SAT based equivalence checking algorithms. The main challenge here is to obtain the most compact CNF form, i.e., with the least number of clauses since the number of clauses not only effects the memory consumption but also the performance of a SAT solver in terms of computation time. Tsetin [27] proposed a very promising algorithm for this purpose. It uses new variables for all intermediate nodes of the circuit instead of resolving them. The addition of new variables keeps the number of clauses linear with the size of circuit. Many extensions of Tsetin's classical algorithm have been proposed in the literature, e.g. [28]. However, in our case we cannot use Tsetin's algorithm because we need to calculate all possible assignments in terms of primary inputs and the new variables for intermediate nodes do not allow this. We therefore use the basic method of CNF conversion in the proposed methodology.



# Chapter 3

## SAT Based Fitness Scoring

The following notation is used to explain the CNF conversion process and the proposed SAT solver implementation. Let  $T$  be the target function with  $M$  inputs and  $N$  outputs, which are form the set  $Y=\{y_1, y_2, \dots, y_N\}$ . A chromosome (candidate circuit) is denoted by  $C$  and also has  $M$  inputs and  $N$  outputs. Let  $F=\{f_1, f_2, \dots, f_N\}$  denote the set of variables corresponding to the outputs of  $C$  and  $Z=\{z_1, z_2, \dots, z_{n_r \times n_c}\}$  denote the set of variables corresponding to the outputs of the intermediate nodes of  $C$ . Both  $T$  and  $C$  are circuits with an identical set of inputs and let the set of input variables be denoted by  $X=\{x_1, x_2, \dots, x_M\}$ .

### 3.1 CNF Conversion

The CNF conversion algorithm is given in Algorithm 3. It accepts the target and candidate circuits as inputs and returns their combined CNF. The function `ENCODE_CNF` encodes the CNF as a disjunction (logical OR) of XOR's between the respective outputs from the target function and the chromosome, i.e., the CNF for an  $N$ -output circuit will be encoded as  $E = (y_1 \oplus f_1) \vee (y_2 \oplus f_2) \vee (y_3 \oplus f_3) \vee \dots \vee (y_N \oplus f_N)$ . This way,  $E$  will be false if and only if all the corresponding outputs from target function and chromosome are equal. The first step in the CNF conversion process is to express  $y_i$ 's and  $f_i$ 's in terms of  $x_i$ 's in order to obtain an expression of  $E$  in terms of inputs  $x_i$ 's only. The function `RESOLVE_Y` finds the CNF expression for  $E$  by representing the  $y_i$ 's in terms of their corresponding  $x_i$ 's and returns the resulting simplified CNF in a variable  $CNF_{input}$ . This is done by using the proposition resolution rules given in Table 3.1. The outputs  $y_i$ 's remain the same throughout the evolution process since they represent the target function whereas the outputs  $f_i$ 's

alter in every iteration of the evolution process. In order to minimize the computation cost, we obtain the  $CNF_{input}$  only once and reuse it along with the current  $f'_i$ s to find the net CNF expression for every iteration. The function RESOLVE\_F accepts  $CNF_{input}$  and returns the net CNF expression by representing the  $f'_i$ s in terms of their corresponding  $x'_i$ s.

---

**Algorithm 3** CNF Conversion

**Inputs:**

target\_function: representation of the target truth table

chromosome: a candidate circuit

**Output:**

CNF: combined CNF of target function and chromosome

---

```

1:  $E \leftarrow \text{ENCODE\_CNF}(N)$ 
2:  $CNF_{input} \leftarrow \text{RESOLVE\_Y}(E, \text{target\_function})$ 
3:  $CNF \leftarrow \text{RESOLVE\_F}(CNF_{input}, \text{chromosome})$ 
4: function ENCODE_CNF( $N$ )
5:   Initialize  $E$  as empty set
6:   for  $i=1$  to  $N$  do
7:      $E \leftarrow E \vee (y_i \oplus f_i)$ 
8:   end for
9:   return  $E$ 
10: end function
11: function RESOLVE_Y( $E, \text{target\_function}$ )
12:   for all clauses of  $E$  do
13:     for all literals do
14:       resolve  $y'_i$ s in terms of its corresponding  $x'_i$ s
15:     end for
16:   end for
17:   return  $E$ 
18: end function
19: function RESOLVE_F( $CNF_{input}, \text{chromosome}$ )
20:    $CNF \leftarrow CNF_{input}$ 
21:   for all clauses of  $CNF$  do
22:     for all literals do
23:       resolve  $f'_i$ s in terms of its corresponding  $x'_i$ s
24:     end for
25:   end for
26:   return  $CNF$ 
27: end function

```

---

The next step is to reduce the CNF in order to minimize the computation overhead of the SAT solver. CNF reduction routine is shown in Algorithm 4.  $\Psi$  denotes a CNF clause and  $J$  denotes the number of CNF clauses. The reduction algorithm is based on the following rules.

Table 3.1: Resolution Rules for Some Common Gates

Clause	Resulting Clause
$(\text{AND}(A,B) \vee Y \vee Z)$	$(A \vee Y \vee Z) \wedge (B \vee Y \vee Z)$
$(\text{NAND}(A,B) \vee Y \vee Z)$	$(\neg A \vee \neg B \vee Y \vee Z)$
$(\text{AND}(A, \neg B) \vee Y \vee Z)$	$(A \vee Y \vee Z) \wedge (\neg B \vee Y \vee Z)$
$(\text{NAND}(A, \neg B) \vee Y \vee Z)$	$(\neg A \vee B \vee Y \vee Z)$
$(\text{OR}(A,B) \vee Y \vee Z)$	$(A \vee B \vee Y \vee Z)$
$(\text{NOR}(A,B) \vee Y \vee Z)$	$(\neg A \vee Y \vee Z) \wedge (\neg B \vee Y \vee Z)$
$(\text{XOR}(A,B) \vee Y \vee Z)$	$(A \vee B \vee Y \vee Z) \wedge (\neg A \vee \neg B \vee Y \vee Z)$
$(\text{XNOR}(A,B) \vee Y \vee Z)$	$(A \vee \neg B \vee Y \vee Z) \wedge (\neg A \vee B \vee Y \vee Z)$

1. All duplicate CNF clauses are removed.
2. Subset clauses are removed. For example, the clause  $(x_1 \vee x_2 \vee x_5)$  is a subset of  $(x_1 \vee x_2)$  so it does not provide any new information as long as  $(x_1 \vee x_2)$  is present and thus can be safely removed.
3. Always true clauses are removed. For example, the clause  $(x_1 \vee x_2 \vee \neg x_2)$  is always true and does not affect the overall behavior of the net CNF and thus can be safely removed.

It has been observed through our experimental results, presented in Chapter 4, that the CNF reduction algorithm significantly reduces the CNF size in most of the digital circuit evolution problems and thus significantly minimizes the SAT solver based scoring time.

## 3.2 Proposed SAT Solver

We need a SAT solver that can calculate the number of assignments for which the given CNF is unsatisfiable. It is to note that we are interested only in the number of unsatisfying assignments without specifically knowing them.

The proposed SAT solver algorithm is given in Algorithm 5. A clause is termed as unsatisfiable if and only if all of its literals are false. let  $K$  denote the number of literals appearing in a given CNF clause. There is only one possible assignment of  $K$  variables

---

**Algorithm 4** CNF Reduction

---

**Input:**

CNF: before reduction

**Output:**CNF: in reduced form

---

```
1: for i=1 to J do
2:   if  $\Psi_i$  is always true then
3:     Remove  $\Psi_i$  from CNF
4:   else
5:     for k=i to J do
6:       if  $\Psi_i \subset \Psi_k$  then
7:         Remove  $\Psi_i$  from CNF
8:       else if  $\Psi_k \subset \Psi_i$  then
9:         Remove  $\Psi_k$  from CNF
10:      end if
11:    end for
12:  end if
13: end for
```

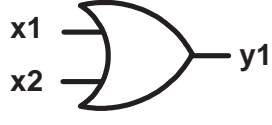
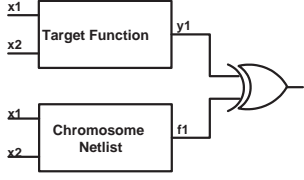
---

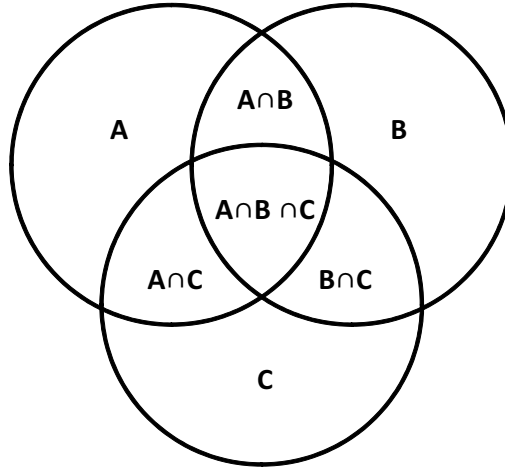
for which a clause may be unsatisfiable. However, since the total number of inputs is  $M$ , which is greater than or equal to  $K$ , the upper bound on the total number of input variable assignments for which the given clause is unsatisfiable is  $2^{M-K}$ . The function `WEIGHT`, used on line 4 of Algorithm 5, accepts a CNF clause and calculates the number of its unsatisfying assignments by using its width, i.e.,  $K$ . Each clause is unsatisfiable for a unique set of input assignments  $i$  and let us denote this set as  $\Phi_i$ . For fitness scoring, we are interested to find the union of all  $\Phi_i$ 's as given in Equation (3.1).

$$\bigcup_{i=1:J} \Phi_i \tag{3.1}$$

Different clauses may share the same unsatisfiable input variable assignments and thus the above mentioned union must be treated like the problem of overlapping sets. Figure 3.1 provides the equation for tackling this problem using an example of three overlapping sets. In a similar way, the function `OVERLAP_WEIGHT`, used on line 5 of Algorithm 5, calculates the union of Equation 3.1 for any number of clauses.

Table 3.2: Fitness Calculation Example

<b>Target Function</b>	$y1 = x1 \wedge x2$
<b>Candidate Circuit</b>	
<b>CNF Encoding</b>	
<b>Encoded CNF</b>	$(y1 \vee f1) \wedge (\neg y1 \vee \neg f1)$
$CNF_{input}$	$(x1 \vee f1) \wedge (x2 \vee f1) \wedge (\neg x1 \vee \neg x2 \vee \neg f1)$
$CNF$	$(x1 \vee x2) \wedge (x1 \vee x2) \wedge (\neg x1 \vee \neg x2) \wedge (\neg x1 \vee \neg x2)$
<b>CNF after reduction</b>	$(x1 \vee x2) \wedge (\neg x1 \vee \neg x2)$
<b>Fitness Score</b>	2



$$A \cup B \cup C = A + B + C - (A \cap B) - (A \cap C) - (B \cap C) + (A \cap B \cap C)$$

Figure 3.1: Three Overlapping Sets and their Union

We have used this net number of unsatisfying assignments as a measure of closeness (fitness score) between the target function and the chromosome. For illustration purposes, we present the whole fitness calculation process of a simple single gate circuit in Table 3.2.

---

**Algorithm 5** Proposed SAT Solver

---

**Input:**

CNF: combined CNF of target function and chromosome

**Output:**fitness score: measure of closeness between target function and chromosome

---

```
1: function SAT_SOLVER(CNF)
2:   initialize score to zero
3:   for i=1 to J do
4:     weight←WEIGHT( $\Phi_i,1$ )
5:     overlap_weight←OVERLAP_WEIGHT( $\Phi_i,J$ )
6:     net_weight←weight-overlap_weight
7:     score←score+net_weight
8:   end for
9:   return score
10: end function
11: function OVERLAP_WEIGHT(CL,J)
12:   pn←1
13:   for i=J-1 to 1 do
14:      $tmp_{pn} \leftarrow CL \vee \Phi_i$ 
15:     if  $tmp_{pn}$  is a valid clause then
16:        $sign_{pn} = -1$ 
17:       pn←pn+1
18:       temp←pn-2
19:       for j=1 to temp and temp 0 do
20:          $tmp_{pn} \leftarrow tmp_j \vee \Phi_i$ 
21:         if  $tmp_{pn}$  is a valid clause then
22:            $sign_{pn} = -1 \times sign_j$ 
23:           pn←pn+1
24:         end if
25:       end for
26:     end if
27:   end for
28:   for i=1 to pn-1 do
29:     overlap_weight=overlap_weight+WEIGHT( $tmp_i, sign_i$ )
30:   end for
31:   return overlap_weight
32: end function
33: function WEIGHT( $\Phi, sign$ )
34:   K←number of literals in  $\Phi$ 
35:   weight←  $2^{M-K}$ 
36:   return sign×weight
37: end function
```

---

# Chapter 4

## Experimental Results

We used the proposed method, described in previous chapters, to evolve circuits that exhibit the behaviour of adders, multipliers, multiplexers, even parity circuits, encoders, and a couple of LGSynth91 benchmarks. Moreover, we evolved the same circuits using state-of-the-art parallel simulation based CGP technique as discussed in Chapter 1, in order to compare the chromosome evaluation time of the two methods. The experimental results are obtained by implementing the proposed methodology using Visual C++ ver. 6 and running on an Intel Duo core 1.86 Ghz processor based machine using the following parameters: i) The digital circuit evolution process has access to a subset of all possible functions, i.e.,  $\Gamma=6,7,10,12$ , described in Table 1.1, to simplify the evolution process. ii) A mutation rate  $\mu = 4\%$  is chosen as it is known to give better convergence, iii) The level back variable  $\ell$  is chosen to be equal to  $n_c$  in order to keep the search space larger.

In this chapter, we first present some of the evolved digital circuits using the proposed methodology. These circuits have been chosen because they highlight some unique characteristics of the proposed approach. Finally, we present a comparison table summarizing the mean evaluation times per chromosome for all the above mentioned circuits using conventional CGP and the proposed SAT-Based CGP for comparison purposes along with some discussions.

### 4.1 Full Adder

A full adder circuit has three inputs and two outputs. It provides the result of adding the three input bits as a sum and a carry bit in the output. We evolved the full adder circuit

using the conventional CGP [21] and the proposed SAT solving based CGP and the finally evolved circuits are given in Figures 4.1 and 4.2, respectively. Both circuits are structurally different but functionality wise they serve the same purpose, i.e, they behave as the full adder circuit. Interestingly, both the circuits are different from the full adder circuit that is usually found in the text books or are designed using the conventional design rules based technique. This clearly indicates the value that digital circuit evolution can bring to the conventional digital design. Interestingly, the full adder circuit obtained via the proposed technique is found to be more efficient than the one obtained via the conventional CGP in terms of both area and speed.

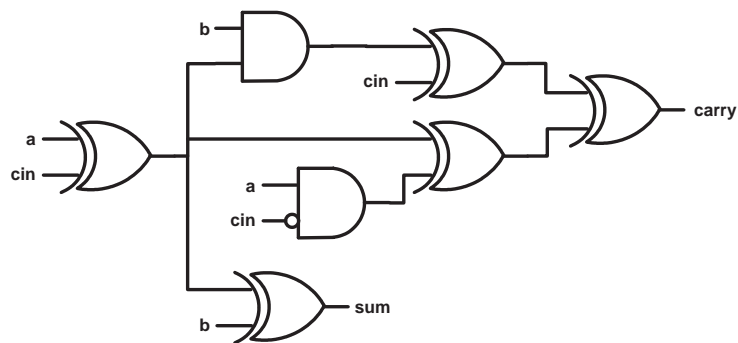


Figure 4.1: Full Adder obtained by Conventional CGP

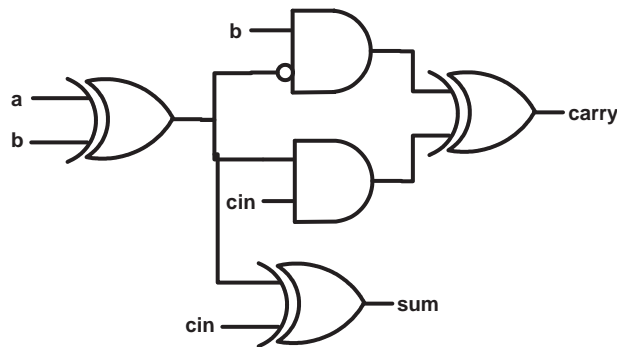


Figure 4.2: Full Adder obtained by Proposed CGP

The conventional simulation based CGP technique took 998 generations and a mean evaluation time of 0.102 milliseconds per chromosome while the proposed SAT solving



based technique took 168 generations and a mean evaluation time of 0.035 milliseconds per chromosome. This difference clearly demonstrates the effectiveness of the proposed SAT based scoring mechanism.

## 4.2 8x1 Mux

An 8x1 Mux selects one of the eight inputs to be passed to the output depending upon the status of three select lines. Hence, it has 11 inputs and 1 output. The large number of inputs makes it susceptible to the heavy computation requirement in the case of conventional simulation based CGP. Figures 4.3 and 4.4 show the circuits of 8 line Mux evolved using the conventional simulation and the proposed SAT solving based CGP techniques, respectively.

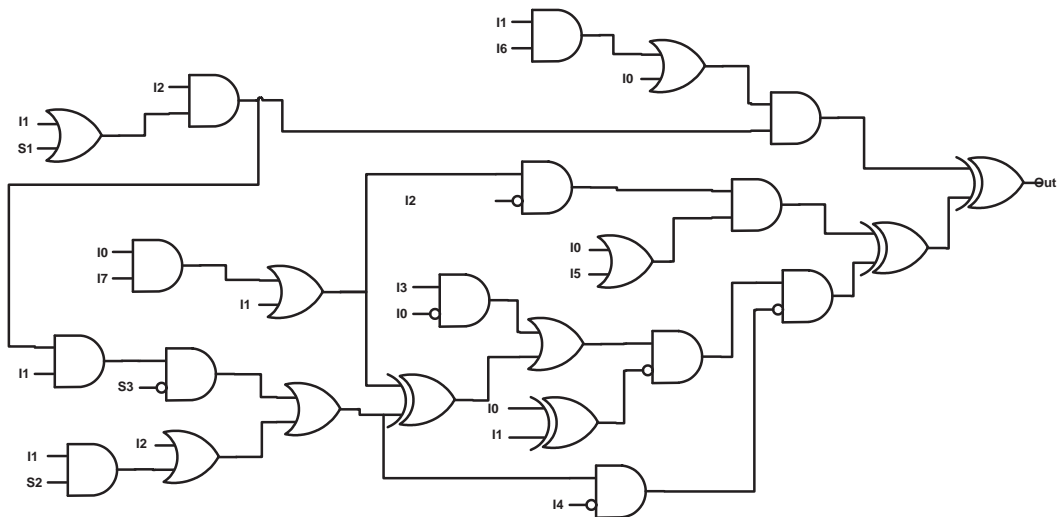


Figure 4.3: 8x1 Mux obtained by Conventional CGP

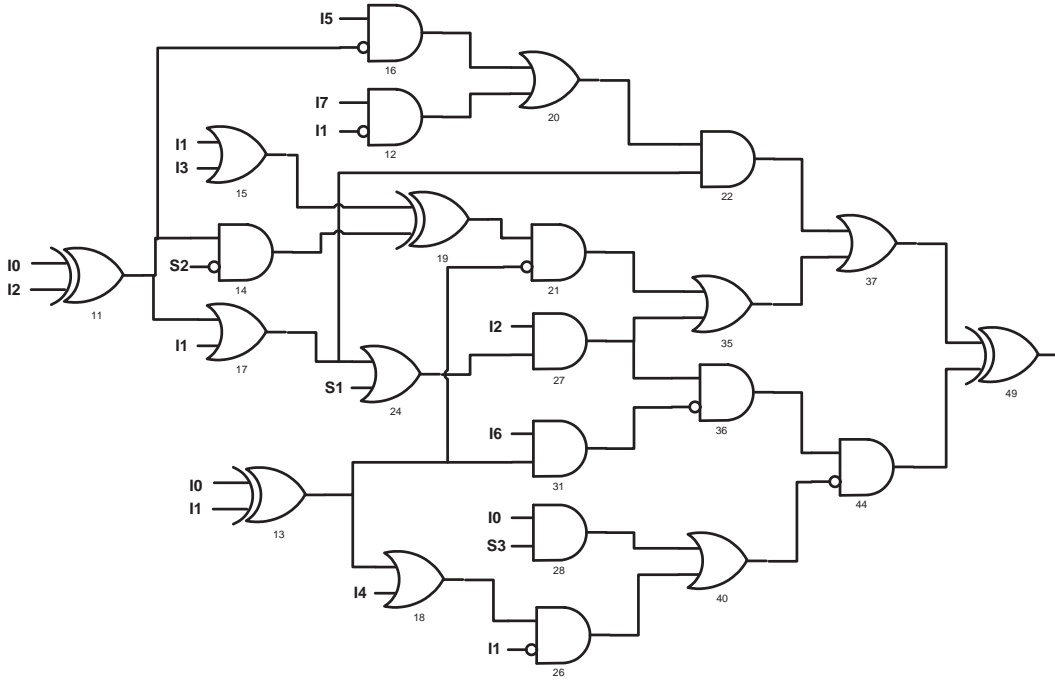


Figure 4.4: 8x1 Mux obtained by Proposed CGP

In terms of computation time, the conventional simulation based CGP took about 850k generations and a mean evaluation time of 0.102 milliseconds per chromosome while the proposed SAT solving based technique took about 100k generations and a mean evaluation time of 0.035 milliseconds per chromosome. This difference is mainly due to the effectiveness of the proposed SAT solving based scoring method for circuits where the number of CNF clauses have a linear relationship with the number of inputs.

### 4.3 8 Bit Parity

The output of an 8 bit parity circuit is true if the number of ones in its input vector is odd. The CNF representation of parity circuits is known to be large and its size grows exponentially with the increase in the number of inputs. Figures 4.5 and 4.6 show the evolved circuits of 8 bit parity using the conventional simulation and the proposed SAT solving based CGP techniques, respectively.

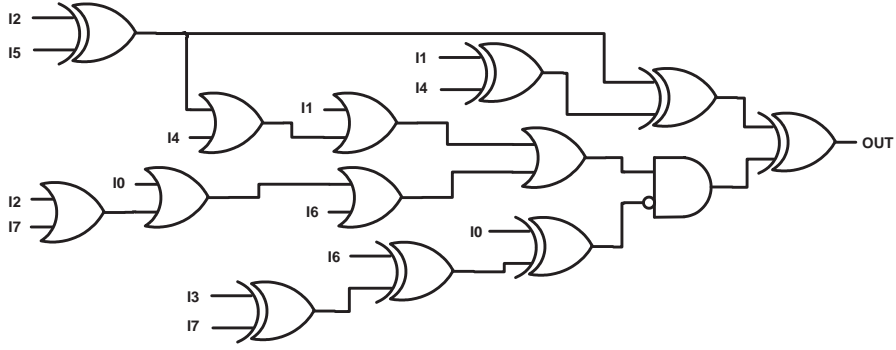


Figure 4.5: 8 Bit Parity Circuit obtained by Conventional CGP

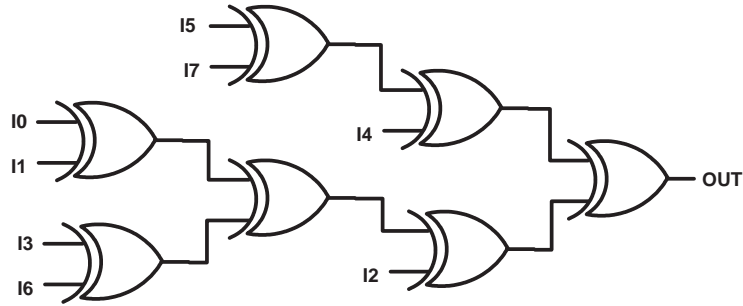


Figure 4.6: 8 Bit Parity Circuit obtained by Proposed CGP

In this case, the conventional simulation based CGP took about 36k generations and a mean evaluation time of 0.32 milliseconds per chromosome while the proposed SAT solving based technique took about 2k generations and a mean evaluation time of 0.7 milliseconds per chromosome. The proposed SAT solving based CGP took more time in the fitness calculation process than the conventional simulation based CGP technique mainly because of the larger CNF representation of the circuit (256 clauses).

### 4.3.1 LGSynth91 c17

c17 is an LGSynth91 benchmark circuit [54] with 5 inputs and 2 outputs. Figures 4.7 and 4.8 show the c17 circuits evolved using conventional simulation based CGP and our proposed SAT solving based CGP techniques, respectively.

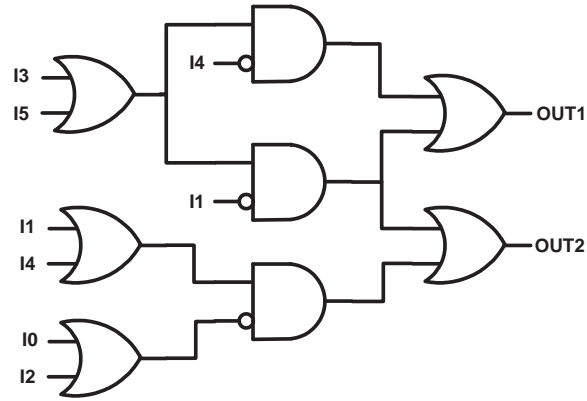


Figure 4.7: c17 Circuit obtained by Conventional CGP

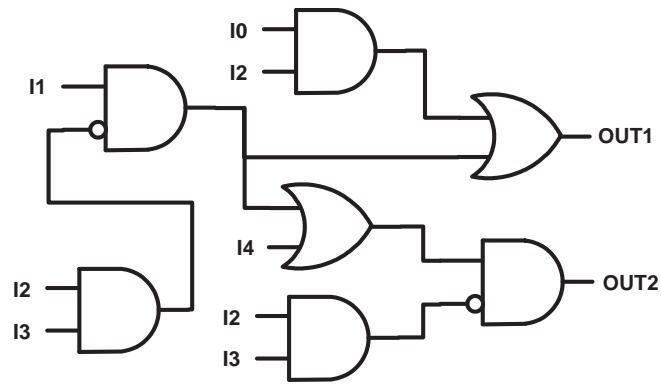


Figure 4.8: c17 Circuit obtained by Proposed CGP

The conventional simulation based CGP took about 29k generations and a mean evaluation time of 0.12 milliseconds per chromosome while the proposed SAT solving based technique took about 66k generations and the a evaluation time of 0.009 milliseconds per chromosome. We observed the maximum time gain for the c17 circuit and this is mainly due to the fact that it has a relatively low number (12) of CNF clauses in its CNF representation.

## 4.4 Discussions

The primary goal of presenting the above case studies was two-fold. Firstly, we wanted to illustrate the correctness of the proposed technique, i.e., it can evolve all kinds of circuits with the desired functionality. All the four examples demonstrate this point. The second purpose was to illustrate the effectiveness of the proposed technique in terms of fitness calculation time for cases where the number of CNF clauses does not grow exponentially with the number of inputs.

Apart from the above case studies, we evolved some other circuits and the results are summarized in Table 4.1. We chose all the major circuits that have been evolved using various digital circuit evolution techniques and the proposed technique was able to successfully evolve all of them. In terms of evolution time, it can be observed that the fitness calculation time is lower for most cases (the highlighted ones are the rest) when compared with the traditional CGP with parallel simulation, which is known to have the fastest evolution time so far [21]. The circuits for which the proposed technique did not perform well (the highlighted cases) are the ones in which the number of CNF clauses grows exponentially with the number of inputs. This restriction was kind of expected since such circuits are known to have problems with SAT solver based equivalence checking as well. In summary, the experimental results show that the proposed technique is able to evolve all kinds of digital circuits and improves the evolution time significantly for most of them as well.

Table 4.1: Mean Evaluation Time Per Chromosome for Conventional CGP  $t_{cgp}$  and SAT-Based CGP  $t_{mcgp}$

Circuit	Number of Inputs	Number of Outputs	Minimized input CNF Clauses	$n_r \times n_c$	Mean Time per Chromosome		Time Gain
					$t_{cgp}$ ms	$t_{mcgp}$ ms	%
Full Adder	3	2	14	1x10	0.102	0.035	291.43
3 Bit Adder	7	4	48	1x25	0.21	0.53	39.62
2x2 Multiplier	4	4	22	1x10	0.12	0.035	342.86
3x3 Multiplier	6	6	102	1x40	0.15	2.2	6.82
Parity 5	5	1	32	1x12	0.115	0.018	638.89
Parity 6	6	1	64	1x15	0.15	0.22	68.18
Parity 8	8	1	256	1x20	0.32	0.7	45.71
4x2 Encoder	4	2	8	1x15	0.095	0.018	527.78
8x2 Encoder	8	2	24	1x20	0.35	0.06	583.33
2x1 Mux	3	1	4	1x10	0.11	0.017	647.06
4x1 Mux	6	1	8	1x15	0.17	0.072	236.11
8x1 Mux	11	1	16	1x30	1.06	0.41	258.54
Lgsynth91 c17	5	2	12	1x13	0.12	0.009	1333.33
Lgsynth91 majority	5	1	11	1x15	0.094	0.016	587.50
<b>16 line MUX and 16 Decoder 10000 chromosome run</b>							
16x1 MUX	20	1	32	1x40	536	2.5	21440
16x4 Encoder	16	4	48	1x40	44.8	5.2	861.54

# Chapter 5

## Conclusions

This thesis presents a novel digital circuit evolution approach based on the principles of SAT solving. The main idea is to leverage upon the computational efficiency of SAT solvers to expedite the process of fitness scoring, which is traditionally done using exhaustive simulations. We proposed a CGP based digital circuit evolution technique where the equivalence problem of the target function and the given chromosome is represented in terms of a CNF and then the number of input assignments for which this CNF is unsatisfiable is found using SAT solving algorithms. This number is used to judge the fitness of the chromosomes and the digital circuit evolution is carried out using the traditional genetic algorithm based approach. We implemented the proposed methodology in C++. The experimental results illustrate the effectiveness of the proposed approach as we are able to correctly evolve all the state-of-the-art evolved digital circuits. Moreover, using the proposed approach, significant reduction in evolution time was observed for circuits in which the number of CNF clauses is a linear function of the number of inputs.

To the best of our knowledge, SAT-based method has never been used in the domain of digital circuit evolution to measure the degree of closeness between two circuits. Thus, this thesis opens the doors to a new area of research in both of these domains. The proposed idea pushes the boundaries of digital circuit evolution and thus can be used to evolve more complex digital circuits in terms of gate count and inputs. Whereas, smarter fitness scoring criterion than the ones reported in this thesis can be explored by the SAT solving community to further reduce the computational times and optimize the evolved circuits. Finding a smarter CNF conversion that results in a more compact CNF representation will help to improve the time efficiency proposed methodology. A worth investigation direction

in this regard would be to use SMT solvers [50] instead of SAT solvers for fitness scoring. The current methods are limited to combinational digital circuits, so in future we would look to find ways to evolve sequential circuits using evolutionary approach.



# References

- [1] G. Greenwood and A.M. Tyrrell, *Introduction to Evolvable Hardware*, New York:IEEE Press, 2007.
- [2] T. Higuchi et al., "Real-World Applications of Analog and Digital Evolvable Hardware", *IEEE Trans. Evol. Comput.*, vol.3, no.3, pp. 220–235, 1999.
- [3] J.R. Koza, "Genetic Programming", *On the Programming of Computers by Means of Natural Selection*, Cambridge:MIT Press, 1992.
- [4] P. Kaufmann and M. Platzner, "Advanced Techniques for the Creation and Propagation of Modules in Cartesian Genetic Programming", in *Proc. of Genetic and Evolutionary Computation Conf.*, GECCO. 2008, pp. 1219–1226.
- [5] T. Higuchi et al., *Evolvable Hardware*. Berlin:Springer, 2006.
- [6] D. Mange et al., "Towards Robust Integrated Circuits: The Embryonics Approach", *Proc. IEEE*, vol.88, no.4, pp. 516-541, 2000.
- [7] S. Zhan et al., "A Developmental Gene Regulation Network for Constructing Electronic Circuits", in *Int. Conf. on Evolvable Systems: From Biology to Hardware*, ICES, LNCS, vol. 5216, Springer, Berlin, 2008, pp. 177-188.
- [8] J.A. Walker and J.F. Miller, "The Automatic Acquisition, Evolution and Reuse of Modules in Cartesian Genetic Programming", *IEEE Trans. Evol. Comput.*, vol.12, no.4, pp. 397-417, 2008.
- [9] J.R. Koza et al., *Genetic Programming III: Darwinian Invention and Problem Solving*. San Francisco, CA:Morgan Kaufmann Publishers, 1999.

- [10] F.V. Andrade et al., "Improving SAT-based Combinational Equivalence Checking Through Circuit Preprocessing", in *26th Int. Conf. on Computer Design, ICCD*, 2008, pp 40-45.
- [11] J.R. Koza et al., *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Dordrecht:Kluwer, 2003.
- [12] J. Cong and K. Minkovich, "Optimality Study of Logic Synthesis for LUT-based FPGAs", *IEEE Trans. Comput. Aided Des. Integ. Circuits Syst.*, vol.26, no.2, pp. 230-239, 2007.
- [13] A.P. Shanthi and R. Parthasarathi, "Practical and Scalable Evolution of Digital Circuits", *Appl. Soft Comput.*, vol.9, no.2, pp. 618-624, 2009.
- [14] Berkley Logic Synthesis and Verification Group: ABC: A System for Sequential Synthesis and Verification. <http://www.eecs.berkeley.edu/~alanmi/abc/>
- [15] S.Z. Shazli and M.B. Tahoori, "Soft Error Rate Computation in Early Design Stages Using Boolean Satisfiability", in *proc. of the 19th ACM Great Lakes symposium on VLSI, GLSVLSI'09*, NY, USA, 2009, pp. 101–104.
- [16] L. Sekanina, *Evolvable Components: From Theory to Hardware Implementations*, Natural Computing Series, Berlin:Springer, 2004.
- [17] X. Yao and T. Higuchi, "Promises and Challenges of Evolvable Hardware", *IEEE Trans. Syst. Man Cybernet. Part C*, vol.29 no.1, pp. 87-97, 1999.
- [18] S. Disch and C. Schollm, "Combinational Equivalence Checking Using Incremental SAT Solving, Output Ordering, and Resets", in *Asia and South Pacific Design Automation Conf.*, 2007, pp. 938-943.
- [19] Z. Gajda and L. Sekanina, "When Does Cartesian Genetic Programming Minimize the Phenotype Size Implicitly?", in *Genetic and Evolutionary Computation Conf.*, ACM, New York, 2010, pp. 983–984.

- [20] J.F. Miller and P. Thomson, "Cartesian Genetic Programming", in *Proc. of the 3rd European Conf. on Genetic Programming*, EuroGP, LNCS, vol. 1802, Springer, 2000, pp. 121-132.
- [21] . J.F. Miller et al., "Principles in the Evolutionary Design of Digital Circuitspart I", *Genetic Programm. Evol. Mach.*, vol.1, no.1, pp. 8-35. 2000.
- [22] R. Zebulum et al., *Evolutionary Electronics Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*. Boca Raton:The CRC Press International Series on Computational Intelligence, 2000.
- [23] E.I. Perez and C.C. Coello, "Extracting and Re-using Design Patterns from Genetic Algorithms Using Case-based Reasoning", *Engineering Optimization* vol.35, no.2, pp. 121-141, 2003.
- [24] E. Goldberg et al., "Using SAT for Combinational Equivalence Checking", in *DATE 01: Proc. of the Conf. on Design, Automation and Test in Europe*,IEEE Press, Piscataway, NJ, USA, 2001, pp. 114-121.
- [25] V.K. Vassilev and J.E. Miller, "Scalability Problems of Digital Circuit Evolution Evolvability and Efficient Designs", *Evolvable Hardware 2000. Proc.*, 2000, pp.55-64.
- [26] D.E. Goldberg and K. Deb, "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms", in *G.J.E. Rawlins (Ed.), Foundations of Genetic Algorithms*, Morgan Kaufmann, Los Altos, 1991, pp. 69-93.
- [27] G.S. Tseitin, "On the Complexity of Derivation in Propositional Calculus", in *Studies in Constructive Mathematics and Mathematical Logic, Part II*, 1968, pp. 115-125
- [28] M.N. Velev, "Efficient translation of Boolean Formulas to CNF in Formal Verification of Microprocessors", in *proc. of 2004 Asia and South Pacific Design Automation Conf.*, NJ, USA, 2004, pp. 310-315.

- [29] J.F. Miller et al., "Designing Electronic Circuits Using Evolutionary Algorithms", in *Quagliarella, D., Periaux, J., Poloni, C., Winter, G. (eds.)*, Chichester:Wiley, 1997, pp. 105-131.
- [30] J.F. Miller and S.L Smith, "Redundancy and Computational Efficiency in Cartesian Genetic Programming", *IEEE Trans. on Evolutionary Computation*, vol.10, pp. 167-174, 2006.
- [31] L. Sekanina, "Evolutionary Design of Digital Circuits: Where Are Current Limits?", in *Proc. of the First NASA/ESA Conf. on Adaptive Hardware and Systems*, AHS, IEEE CS, Los Alamitos, 2006, pp. 171-178.
- [32] S. Harding, J.F. Miller, W. Banzhaf, in 2009 IEEE Congress on Evolutionary Computation. Self Modifying Cartesian Genetic Programming: Parity (IEEE Press, New York, 2009), pp. 285-292
- [33] M. Murakawa et al., "Evolvable Hardware at Function Level", in *Parallel Problem Solving from Nature*, PPSN IV, LNCS , vol. 1141, Springer, 1996, pp. 62-71.
- [34] T.G.W. Gordon and P.J. Bentley, "Towards Development in Evolvable Hardware", in *Proc. of the 2002 NASA/DoD Conf. on Evolvable Hardware*, IEEE Computer Society Press, Washington, DC, US, 2002, pp. 241-250.
- [35] P.C. Haddow et al., "Shrinking the Genotype: Linear Systems for EHW?" in *Proc. of the 4th Int. Conf. on Evolvable Systems: From Biology to Hardware*, LNCS , vol.2210, Springer, Berlin, 2001, pp. 128-139.
- [36] G. Hornby et al., "Automated Antenna Design with Evolutionary Algorithms", in *Proc. 2006 AIAA Space Conf.*, AIAA, San Jose, CA, 2006.
- [37] E. Stomeo et al., "Generalized Disjunction Decomposition for Evolvable Hardware", *IEEE Trans. Syst. Man Cybernet. Part B*, vol.36, no.5, pp. 1024-1043, 2006.

- [38] J. Torresen, "A Divide-and-Conquer Approach to Evolvable Hardware", in *Proceedings of the 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware*, ICES98, LNCS, vol. 1478, Springer, Lausanne, Switzerland, 1998, pp. 57-65.
- [39] J. Torresen, "A Scalable Approach to Evolvable Hardware". *Genetic Programm. Evol. Mach.*, vol.3, no.3, pp. 259-282, 2002.
- [40] S. Yanushkevich et al., *Decision Diagram Techniques for Micro and Nanoelectronic Design Handbook*, Boca Raton:CRC, 2006.
- [41] K. Glette et al., "An Online EHW Pattern Recognition System Applied to Face Image Recognition", in *Applications of Evolutionary Computing*, EvoWorkshops 2007, LNCS, vol. 4448, Springer, 2007, pp. 271-280.
- [42] K. Imamura et al., "The Test Vector Problem and Limitations to Evolving Digital Circuits", in *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, IEEE Computer Society Press, 2000, pp. 75-79.
- [43] T. Pecenka et al., "Evolution of Synthetic RTL Benchmark Circuits with Predefined Testability", *ACM Trans. Des. Autom. Electron. Syst.* vol.13, no.3, pp. 1-21 2008.
- [44] Z. Vasicek et al., "On Evolutionary Synthesis of Linear Transforms in FPGA", in *Proc. of the 8th Conf. on Evolvable Systems: From Biology to Hardware*, LNCS, vol. 5216, Springer, Berlin, 2008, pp. 141-152.
- [45] Z. Vasicek and L. Sekanina, "Formal Verification of Candidate Solutions for Post-synthesis Evolutionary Optimization in Evolvable Hardware", *Genetic Programm. Evol. Mach.*, vol.12, no.3, pp. 305-327, 2011.
- [46] Goldberg and E. David, *Genetic Algorithms in Search Optimization and Machine Learning*, Boston, MA:Addison Wesley, 1989.

- [47] F. Lu et al, "A signal correlation guided ATPG solver and its applications for solving difficult industrial cases", in *Proc. of the 40th conf. on design automation*, DAC, 2003, pp. 436-441.
- [48] S. Kemper, "SAT-based Verification for Timed Component Connectors", *Science of Computer Programming*, vol.77, no.(7-8), pp. 779-798, 2012.
- [49] K. Kanazawa and T. Maruyama, "An FPGA Solver for SAT-Encoded Formal Verification Problems", *International Conference on Field Programmable Logic and Applications*, FPL, 2011, pp. 38-43.
- [50] T. Liu et al., "Bounded Program Verification Using an SMT Solver: A Case Study, Software Testing", *IEEE Fifth Int. Conf. on Verification and Validation*, ICST, 2012, pp. 101-110.
- [51] M. Buro and H. K. Buning, "Report on a SAT Competition". *Bulletin of The European Association for Theoretical Computer Science*, vol.49, pp. 143-151, 1993.
- [52] N. Een and N. Sorensson, MiniSAT, <http://minisat.se>
- [53] A. Biere, "PicoSAT Essentials", *Journal of Satisfiability, Boolean Modeling and Computation*, vol.4, no.(2-4), pp. 75-97, 2008.
- [54] <http://www.cbl.ncsu.edu:16080/benchmarks/LGSynth91/cmlexamples/C17.slif>