Institutionen för datavetenskap
Department of Computer and Information Science

Final thesis

TAM Design for Parallel Testing under Bus Bandwidth Limit

by

Kuei-Hsi Tseng

LIU-IDA/LITH-EX-A—10/035--SE

2010-08-27
Final Thesis

TAM Design for Parallel Testing under Bus Bandwidth Limit

by

Kuei-Hsi Tseng

LIU-IDA/LITH-EX-A—10/035--SE

2010-08-27

Supervisor: Zhiyuan He
Examiner: Zebo Peng
Abstract

The complexity of electronic system is increasing rapidly and many of the electronic systems are embedded systems implemented as system-on-chip (SoC). This increasing complexity of SoC leads to longer test application time (TAT). One approach to reduce the TAT is to perform tests to several cores in parallel, which requests transporting test data in parallel instead of sequentially.

In IEEE Std. 1500, it supports parallel test mode by incorporating a user-defined, parallel test access mechanism (TAM) to speed up the testing process. The user-defined TAM means the detail of TAM design is excluded from standard and decided by system integrator. Therefore, we propose a customized TAM structure and two approaches to guarantee full-spatial-parallelism under a bus width limit, and aim to minimize the total number of wire connections. In order to know how close to optimal solution our solutions are, we implement a Simulated Annealing (SA) algorithm to do the comparison.

The experimental results of the two proposed approaches based on benchmark ISCAS’89 and ITC’02 show the parallelism can be guaranteed by our approaches while using only a few wire connections per pin, and the execution times of them are shorter compared with the SA algorithm.
Acknowledgments

First, I would like to thank my examiner Professor Zebo Peng, and my supervisor, Zhiyuan He. Zebo gave me the opportunity to do the master thesis and designated Zhiyuan to be my supervisor. Zhiyuan taught me how things should be done while doing a research, and how to write the document. He helped me find the topic of this thesis, and spent lots of time on the discussion about my thesis with me and helped me get through some bottlenecks when I get stuck.

I also want to thank my good friends, Chin-Han, Fu-Jung, Wei-Chen and Shih-Yen who study in SoC programme with me for cheering me up and giving me some advices when I encountered obstacles to this thesis.
Chapter 1

Introduction

Embedded systems nowadays are everywhere. Most of them contain SoCs. But chips may not work correctly; as there might be some defects during the fabrication process. In order to ensure the quality of final products, we need to test every chip before they are delivered to customers. An automatic test equipment (ATE) is often used to test a SoC. ATEs apply a set of test stimuli generated by automatic test pattern generation (ATPG) software and capture the test responses from the SoC under test. The test is passed if the test response is exactly the same as the expected response. Otherwise the test is failed. How the test data (stimuli and response) are transported between the ATE and the SoC depends on the test access mechanism (TAM). This thesis deals with the TAM design problem and focuses on the minimization of the number of wire connections and at the same time guaranteeing the full-spatial-parallelism. This chapter describes the motivation of our work, defines the problem and gives an overview of organization of this thesis.

1.1 Motivation

The complexity and functionality of electronic systems increase rapidly in recent decades, and millions of transistors are integrated into an integrated circuit nowadays. Due to the existence of defects during fabrication, we need to test the system after the fabrication. Testing process is repeated for all the fabricated products, and the TAT is counted once and again over all the products. A short TAT leads to short time to market (TTM), which is very important to the revenue. Therefore, how to reduce the TAT is an important issue. Different approaches such as test scheduling and TAM design are critical to reduce the TAT.

Test scheduling and TAM design are relevant and affect each other. Given one of them could change the constraints to the other. For example, in Multiplexed TAM Architecture, only one core-under-test (CUT) can be tested at a time; in Daisy-chained
Architecture, CUTs can only be tested sequentially; in Distribution Architecture, CUTs get only partial width of bus line [7, 11]. These architectures all bring some constraints to test scheduling. The most flexible TAM design to test scheduling is that any combination of CUTs are able to be tested in parallel as long as their total number of pins does not exceed the bus width limit, which is called “full-spatial-parallelism”. In order to guarantee full-spatial-parallelism, one naive way is to use full-connections, which means we connect all the bus lines to each pin on a core. However, the cost of full-connections is unnecessarily high. We may connect only partial bus lines to each pin so that there are still alternative paths to transport test data between pins and bus, and full-spatial-parallelism can still be achieved. For example, the full-spatial-parallelism in Figure 1.1 is that $\text{CUT}_1$ and $\text{CUT}_2$ are able to be tested in parallel, the same as $\text{CUT}_1$ and $\text{CUT}_3$, and $\text{CUT}_2$ and $\text{CUT}_3$. Both Figure 1.1(a) and (b) can guarantee the parallelism, but the latter one uses only five wire connections, which has a lower cost in wiring and the resulted control logic. In this thesis, we aim to minimize the total number of wire connections and at the same time to achieve the full-spatial-parallelism.

![Diagram showing possible connections to guarantee full spatial parallelism](image)

(a) Full-connections

(b) Partial-connections

Figure 1.1: Possible connections to guarantee full spatial parallelism
1.2 Problem Formulation

In this thesis, we address the wire-connections minimization problem and, at the same time, we aim for a full-spatial-parallelism. A full-spatial-parallelism means that CUTs can be tested at the same time as long as their total number of pins is less than or equal to the bus width (number of bus lines). For a SoC design, pins on chip are expensive and usually limited; sometimes pins on a core are more than pins on chip, it means that “that CUT” is impossible to be tested at full-parallel speed because the test stimuli or the produced response can not be transported dedicatedly for each pin of CUT. Therefore, we propose a TAM structure which has a memory to widen the bus width as shown in Figure 1.2 ($B_B \geq B_{in}, B_2 \geq B_{out}$). By doing this, we can ensure that every CUT is able to be tested at full-parallel speed individually.

![Figure 1.2: Structure of our test access mechanism](image)

Suppose that a SoC consists of $n$ CUTs, denoted with $C_1, C_2, ..., C_n$ respectively. The CUTs have their own number of pins, denoted with $W_1, W_2, ..., W_n$. The bus width limit $B$ is adjusted according to the number of I/O pins of CUTs, and bus width limit $B$ should be larger than or equal to the maximal number of pins.
among all the CUTs \( n \geq W_i, i = 1, 2, ..., n \). This means any CUT can be tested at full-parallel speed individually. Each pin of CUTs may have multiple connections to bus, \( R_{ij} (1 \leq i \leq n, 1 \leq j \leq W_i) \) denotes the number of connections of \( j \) th pin of \( i \) th CUT connected to bus, and \( P_{ijk} (1 \leq i \leq n, 1 \leq j \leq W_i, 1 \leq k \leq R_{ij}) \) denotes the index of bus line to which \( k \) th connection of \( j \) th pin of \( i \) th CUT is connected.

All the connections between pins and bus construct a “connection scheme”. A bus collision is a scenario that two or more pins try to transport (receive or send) the test data via the same bus line (same path) at the same time. For example, in Figure 1.2(a), bus collisions happen when the 2\(^{nd}\) pin of CUT\(_1\) and the 1\(^{st}\) pin of CUT\(_3\) are using the same bus line (\( P_{121} = P_{311} = 2 \)) on the input side and when the 1\(^{st}\) pin and the 2\(^{nd}\) pin of CUT\(_3\) are using the same bus line (\( P_{311} = P_{321} = 3 \)) on the output side. If we have a set of CUTs that want to be tested in parallel, then we must find at least one way to transport the test data in the connection scheme which contains no bus collision, we called it a “solution path”. For example, Figure 1.2(b) shows that a solution path can be found by changing the bus line of the 2\(^{nd}\) pin of CUT\(_1\) on input side from previous example. (\( P_{122} = 3 \neq P_{311} = 2 \)); on output side, a solution path can be found by changing the bus line of the 2\(^{nd}\) pin of CUT\(_3\) from previous example. (\( P_{311} = 3 \neq P_{322} = 4 \)). As you can imagine, if we add more connections then pins may have more candidate bus lines (paths) to transport the test data, so the chance to find a solution path in a connection scheme will increase.

![Diagram](image)

(a) Pins try to transport data via the same bus line
(b) A way to transport data without bus collision

Figure 1.2: Example of bus collision and solution path
Our goal is all the combinations of CUTs under the constraint are able to find at least one solution path in the connection scheme. The constraint is: the total number of pins of CUTs is less than or equal to bus width limit. For example, if \( B = 7 \), \( n = 4 \), \( W_1 = 7 \), \( W_2 = 3 \), \( W_3 = 2 \), \( W_4 = 1 \), then all the combinations of CUTs under the constraint are: \((C_1),(C_2),(C_3),(C_4),(C_2,C_1),(C_2,C_4),(C_3,C_4),(C_2,C_3,C_4)\), they should be able to be tested in parallel. But among these combinations, most of them are redundant, because if a superset of CUTs can be tested in parallel then its subsets definitely can be tested in parallel; we only have to consider the supersets, in the example, only the supersets: \((C_1)\) and \((C_2,C_3,C_4)\) are taken into consideration and all the combinations of CUTs under the constraint are still covered. We collect the supersets forms “Set of All Supersets of Combinations” (SASC, denoted with \( S \)). There are two supersets in SASC in this example (\( S = \{(C_1),(C_2,C_3,C_4)\} \)).

The problem we address here is to find a connection scheme using the minimum number of connections of all pins such that we can find at least one solution path in the connection scheme no matter which superset in the SASC is under test. That is to say, any combination of CUTs whose total number of pins is less than or equal to the bus width limit can be tested at full-parallel speed.

### 1.3 Contributions

We provide the most flexible TAM design to test scheduling. We propose a new perspective of TAM design introducing the memory as a buffer to deal with the mismatch between pins on chip and pins on cores. And we develop two approaches to find the connection scheme which guarantees the full-spatial parallelism. Both approaches are originated from graph coloring algorithm.

### 1.4 Thesis Organization

The thesis is organized as follows. In Chapter 2, we talk about the background knowledge and related works for our approaches. Different TAM architectures, techniques of graph coloring and max-weight clique problem are described. Chapter 3 presents the how we solve the problem and develop our own approaches based on existing algorithms. Chapter 4 shows the experimental results generated from
applying benchmark ISCAS’89 and ITC’02. Chapter 5 concludes the thesis and discusses the future work.
Chapter 2

Background and Related Work

In this chapter, we give a description of IEEE Std. 1500, which is a standard of test infrastructure for core-based SoC, and a discussion about several TAM design architectures. We introduce two well-known NP-hard problems: graph coloring problem and maximum-weight clique problem, and one algorithm: simulated annealing, which is a probabilistic method for combinatorial optimization problem.

2.1 IEEE Std. 1500

In core-based SoC design, all cores are not manufactured before they are integrated into a SoC, so the testing is applied not only to the interconnects between cores but also to the cores themselves. Therefore, IEEE 1500 standard is created to deal with the problem.

IEEE 1500 standard supports both serial and parallel test mode, but the serial ports are mandatory while the parallel ports are optional because the requirements and goals of different core integrators are quite distinct [1]. The standard keeps the flexibility and leaves the parallel ports undefined for designer. In order to reduce the SoC testing time, designers can utilize parallel test mode by incorporating user-defined parallel TAM on their own. A system overview of IEEE 1500 standard is shown in Figure 2.1, where a system contains $N$ cores and each of them has a wrapper surrounding it. Test data can be transported to the cores through either wrapper serial port (WSP) as shown in Figure 2.3(a) or user-defined parallel TAM system as shown in Figure 2.4(a).
Figure 2.1: System overview of the IEEE 1500 standard

The wrappers is composed of several wrapper boundary registers (WBRs), a wrapper serial port (WSP), a wrapper parallel port (WPP), a wrapper instruction register (WIR) and a wrapper bypass register (WBY) as shown in Figure 2.2. WBRs have some controls on it to switch the mode so that the flow of test data can be changed. There are four functional modes of WBR are defined:

1. Normal mode---The WBR is transparent to the system and the cores works in normal functions.
2. Inward facing mode---This aims to test the cores. The functional inputs of the core are controlled and the functional outputs of the core are observed by the WBRs.
3. Outward facing mode---This aims to test the interconnects between cores. The functional outputs of the core are controlled and functional inputs of the core are observed by the WBRs.
4. Nonhazardous (safe) mode---Both functional inputs and functional outputs are controlled by WBR to a safe state.

When it is in serial test mode (Figure 2.3(b)), WBRs receive and send test data via wrapper serial input and output (WSI, WSO); when it is in parallel test mode (Figure 2.4(b)), test data is transported through user-defined parallel port from/to WBRs. WIR is used to store the instruction to be executed in the corresponding core. WBY is
usually put between WSI and WSO and used when the corresponding core is not involved in the serial testing (Figure 2.4(c)), it is called bypass mode.

Figure 2.2: Core with wrapper

Figure 2.3: Flow of test data in serial test mode
On the whole, IEEE 1500 standard defines a scalable architecture for modular testing without limiting the flexibility. The minimum requirements of hardware components such as WBR, WBY and WIR are defined clearly. And the detail of SoC issue such as TAM design is left to be addressed by the designer, which is the focus in this thesis and will be discussed in next section.

2.2 Test Access Mechanism

Figure 2.5 depicts three architectures of TAM design: 1. Multiplexed Architecture, 2. Daisy-chained Architecture, 3. Distribution Architecture. In Multiplexed Architecture, only one core can be accessed at a time. This means the TAT is the sum of the individual core test time. The design of control is easy but the parallelism only exists inside the wrapper when test data is transported via WPP to/from the core. In Daisy-chained Architecture, all cores are connected serially and insert with bypass registers so that the test data can go through only the relevant cores sequentially. In Distribution Architecture, the total available TAM wires are separated and given to
different cores. This allows cores be tested concurrently and the TAT is the maximum test time of individual cores.

![Diagram of different architectures](image)

**Figure 2.5:** (a) Multiplexed Architecture, (b) Daisy-chained Architecture, (c) Distribution Architecture

There are two types of TAM that have been proposed: test bus [2] and TestRail [3]. Test bus type is the combination of Distribution and Multiplexed Architecture. As Figure 2.6(a) shows, total available TAM wires are separated into different test bus, each test bus get only partial width of TAM wires (Distribution); for the cores connected to the same test bus, only one can be tested at a time (Multiplexed). Figure 2.6(b) depicts that all the TAM wires are distributed to several TestRails, and cores are connected one by one for each TestRail, this implies TestRail type is the combination of Distribution and Daisy-chained Architecture.

Time domain multiplexed TAM (TDM-TAM), presented by Ebadi and Ivanov [6], is another type of TAM design. It is bus-based and uses logic situated locally at each core to be a mask. The test data is divided into frames. Each specific mask for core enables the core extract appropriate data bits from the frames. For example in Figure 2.7, each frame has five bits, Core 1 uses the first bit of each frame, Core 2 uses the second and the fifth, Core 3 uses the third and the forth. Assume that we want to send the data “11” to Core 1, ”00XX” to Core 2 and “XX10” to Core 3 (X = don’t care bit). The two corresponding frames should be “10XX0”, “1X10X”.

All the above TAM designs have their own pros and cons, but none of them can guarantee full-spatial-parallelism discussed in Sec 1.1.
2.3 Graph Coloring Problem

Given an undirected graph containing vertices and edges, finding a vertex-integer-assignment using minimum distinct integers (colors) such that no two adjacent vertices share the same integer (color) is the optimization problem in graph coloring. This can be used to model the problem like resource allocation or scheduling. The minimum number of integers needed to assign a graph is called the chromatic number, which means the least resources needed while modeling the problem. Finding
The chromatic number of a graph has been approved to be a NP-hard problem and lots of algorithms are developed to solve it, both exact algorithms and approximation algorithms.

One well-known heuristic algorithm is the algorithm DSATUR (Degree of Saturation) of Brélaz [8] in 1979; it is a sequential coloring algorithm with a dynamically established order of the vertices. The degree of saturation of vertex $x$, denoted with $\text{deg}_s(x)$, is the number of different colors at the vertices adjacent to $x$. The degree of vertex $x$, $\text{deg}(x)$, is the number of edges incident to the vertex $x$. The degree of saturation changes during the assignment, the degree depends on the graph and remains unchanged during the assignment. DSATUR starts by assigning color “one” to a vertex of maximal degree, the vertices to be assigned later is the vertex which has maximal degree of saturation, if there are two or more vertices tie, the one with maximal degree wins. For example, the order of coloring of a graph which is shown in Figure 2.8(a) using DSATUR is presented in Table 2.1. In Table 2.1, we display the degree of saturation in each step for each vertex and mark the vertex we are going to assign with gray background. At Step1, no vertex is colored yet, so the degree of saturation for every vertex is zero. We then compare the degree of vertices which have the tied degree of saturation, and $V_1$ has the maximal degree, so we assign $V_1$ with integer one. All the vertices adjacent to $V_1$ have to update their degree of saturation, $V_2$, $V_3$, $V_4$, $V_6$ and $V_7$ now have 1 for their degree of saturation, and we start the comparing again. By applying the same rule, the graph is like Figure 2.8(b) when completed. Note that in Step5, although $V_5$ is assigned in Step4, $\text{deg}_s(6)$ did not increase because $V_5$ is assigned with integer one, same as $V_1$.

![Figure 2.8: An undirected graph example before and after assignment](image)
Table 2.1: Detail of coloring example graph step by step

<table>
<thead>
<tr>
<th>Index(degree)</th>
<th>V₀(5)</th>
<th>V₁(1)</th>
<th>V₂(2)</th>
<th>V₃(4)</th>
<th>V₄(5)</th>
<th>V₅(6)</th>
<th>V₆(2)</th>
<th>V₇(3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Step2</td>
<td>---</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Step3</td>
<td>---</td>
<td>1</td>
<td>2</td>
<td>---</td>
<td>1</td>
<td>1</td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>Step4</td>
<td>---</td>
<td>1</td>
<td>2</td>
<td>---</td>
<td>2</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step5</td>
<td>---</td>
<td>1</td>
<td>2</td>
<td>---</td>
<td>---</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step6</td>
<td>---</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Step7</td>
<td>---</td>
<td>1</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>Assignment</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

After 21 years, the algorithm is improved by Walter Klotz [9]. Walter combines DSATUR with backtracking, if the current coloring will not be better than the current optimum, backtrack, until all the possible decisions are made. And, in order to save time from searching in a dead end, from BSC (Backtracking Sequential Coloring), Walter gain the heuristic IBSC(k) by restricted backtracking (Incomplete Backtracking Sequential Coloring), where each vertex may only be k-times the new starting vertex after backtracking. And the result of IBSC(k) has better result than pure-heuristic algorithms and costs less time than BSC as shown in [9].

### 2.4 Maximum-Weight Clique Problem

A clique in an undirected graph is a set of vertices such that for every two vertices in the set, there exists an edge connecting the two. In a weighted graph, which every vertex is assigned with a weight, a maximum-weight clique is a clique that has the maximum total weight of vertices in the graph. For instance, Figure 2.9 shows that two different cliques have the same total weight and both of them are the maximum-weight cliques in the graph (the number in the circle represents the weight). Lots of algorithms can improve the brute-force search for finding a maximum-weight clique in a given graph, but all of them take exponential time to solve the problem.
2.5 Simulated Annealing

Applying simulated annealing to optimization problem is first proposed by Kirkpatrick in 1983. In physical annealing process, first the material is heated; the energy state of material then changes and tends to decrease (but not always) when the temperature reduces. By analogy with physical annealing, each step of SA algorithm replaces the current solution with a neighbor solution under a certain determination. The determination is: If the cost of neighbor solution is lower than the cost of current solution, the replacement proceeds; if the cost of neighbor solution is not lower, we calculate the acceptance probability and do the replacement according to the acceptance probability. This acceptance for higher-cost solutions helps SA not be trapped in local optimality because it allows the uphill moves as shown in Figure 2.10. SA algorithm eventually converge to a near lowest cost solution because it only accepts lower-cost solutions when the temperature is lower than a certain limit. The flowchart of a general SA algorithm could be like Figure 2.11.

Few things need to be noticed when implementing the SA algorithm are: 1. Temperature reduction rate should not be too fast; otherwise it gets easier to be trapped in local optimality. 2. When generating the neighbor solutions, all the solutions should be reachable from each other. 3. Define a stop condition, it can be: a number of iterations without changing the current solution, a fixed amount of execution time, etc.
Figure 2.10: SA allows uphill moves to deal with local optimality

Figure 2.11: Flowchart of simulated annealing
Chapter 3

Analysis and Solutions

This chapter points out the characteristic of our problem and its similarities with existing NP-hard problems. We describe how our problem can be solved by combining our idea with existing techniques.

3.1 Characteristics of Our Problem

To solve the problem, we first propose a TAM structure which has a memory to widen the bus width as shown in Figure 1.2. Then, we want to find a connection scheme using the minimum number of wire connections in total, and the connection scheme satisfies that we can find at least one solution path in it for every superset in SASC. The information of the connection scheme is the bus line assignment for pins, which is a discrete integer permutation. The exhaustive search is usually not feasible in this kind of problem because the domain of candidate solutions becomes extremely large as the size of the problem increases.

For all the pins in the same superset, they should not be assigned to the same bus line because they might be used to transport the test data simultaneously at some time. But the width of bus line is limited, it may not be sufficient to give every pin in the superset a dedicated bus line. When this situation happens, we add redundant connections for pins so that they have the alternative path to avoid bus collision with other pins. The details of how we add redundant connections will be discussed later in this chapter.

3.2 Similarities with Existing NP-hard Problems

After the observation from previous section, we can now utilize these characteristics to solve the problem. Our problem is a resource allocation problem, which usually can
be dealt with by graph coloring problem. The vertex is analogous with the pin, because the former is dedicated to the color and the latter is dedicated to the bus line. To find a chromatic number is like to find the minimum bus width for guaranteeing full-spatial-parallelism without adding redundant connections for pins. If the chromatic number is larger than the bus width limit, then we have to add redundant connections for pins.

We first create a graph from our problem by the following rules: 1. Each pin constructs a vertex. 2. For every two vertices belong to the same superset in SASC we draw an edge between the two, which implies the two representing pins should not be assigned to the same bus line. Once the graph is created, we can observe that vertices belong to the same CUT form a clique, which means every pair of vertices in one CUT are connected. It is because that when a CUT is under test, all the pins in it are used to transport data. For example, a SoC design containing four CUTs, $B = 6, n = 4, W_1 = 6, W_2 = 3, W_3 = 3, W_4 = 2$ constructs a graph shown in Figure 3.1.

![Figure 3.1: An example of graph created from a SoC design](image)

The graph created from a SoC design is a perfect graph. A perfect graph is a graph in which the chromatic number of every induced subgraph equals the size of the largest clique of that subgraph. So the chromatic number can be found by finding the largest clique in the graph. Further, we replace the vertices in one CUT with a
weighted vertex, whose weight equals the size of CUT. New edge is drawn if two CUTs are in at least one same superset in SASC. The generation of new graph is like Figure 3.2, vertices of CUT\(_1\), CUT\(_2\), CUT\(_3\) and CUT\(_4\) are replaced with weighted vertices having weight 6, 3, 3 and 2. CUT\(_2\), CUT\(_3\) are connected; CUT\(_2\), CUT\(_4\) are connected; CUT\(_3\), CUT\(_4\) are connected. The chromatic number is the weight of maximum-weight clique; in this case, a clique contains CUT\(_2\), CUT\(_3\) and CUT\(_4\). By this transformation, the problem size of finding chromatic number reduced from the number of pins to the number of CUTs.

**Figure 3.2: Transformation from original graph to weighted graph**

### 3.3 Overview of Solutions

In this thesis, we proposed two approaches: Postponed-Assignment approach and Immediate-Assignment approach to solve our design problem concerning full-spatial parallelism. In order to check if our approaches produce low-cost connection schemes, we implement a SA algorithm for comparison purpose. This section gives an overview of how the solutions work.

We have a SoC design as input, and then we compute the SASC to construct the graph. After the graph is generated, we find the weight of maximal-weight clique as ideal bus width (IB) because the weight here represents the number of bus lines we need, to achieve full-spatial parallelism without adding redundant connections. We
compare the ideal bus width to the bus width we set. If the ideal bus width is not larger than the bus width ($IB \leq B$), we get the connection scheme by applying DSATUR algorithm to do the single connection assignment. If the ideal bus width is larger than bus width ($IB > B$), we can select either the Postponed-Assignment approach or the Immediate-Assignment approach to solve the problem, or applying SA algorithm for comparison purpose. The flowchart of solutions is depicted in Figure 3.3.

![Figure 3.3: Overview of solutions](image)

### 3.3.1 Graph Construction

Once we have the SoC design, we get the information of how many CUTs the design has ($n$) and how many pins each CUT has ($W_1, W_2, \ldots, W_n$). Before we start to construct the graph, we need to know the SASC so that we can draw the edges between vertices. First, we set the size of the largest CUT as bus width limit ($B = \max(W_1, W_2, \ldots, W_n)$) because it is the minimum bus width which can guarantee that each CUT is able to be tested at full-parallel speed. Then we use the formula $\sum W_{\text{selected cores}} \leq B$ to enumerate the feasible combinations of CUTs and form SASC.

We reorder the index of CUTs according to their number of pins in descending order. The new index starts from the largest CUT to the smallest CUT ($W_i \geq W_j, i \leq j$) and this will help us find all the feasible combinations of CUTs faster. Because we use a counter counting from one to $2^n (1, 2, \ldots, 2^n)$ and the binary expression represents the combination of CUTs. For example, “011000” represents the combination containing CUT2 and CUT3. If a combination is failed to pass the bus width limit ($\sum W_{\text{selected cores}} > B$), we find the first non-zero bit (bit $x$) from the right.
side (bit0), then we can add “2 to the power of \(x\)” instead of one to the counter. It is because when the sum of the current combination is larger than the bus width limit, the sums of the next \(2^x-1\) combinations are definitely larger than the bus width limit also. These combinations can therefore be skipped. Figure 3.4 illustrate how this process works. Among all these feasible combinations, we eliminate the combinations which are the subset of the others. We keep only the supersets, and forms “Set of All Supersets of Combinations” (SASC). How do we determine whether the combination is a superset or not? We check if other CUTs can join the current combination and still be feasible or not \((B - \sum W_{\text{selected cores}} < \min(W_{\text{unselected cores}}))\). Because if the difference between bus width limit and the sum of current combination is “not” less than the size of smallest unselected CUTs, it implies the current combination can be extended to a larger set of CUTs by adding the smallest unselected CUT into it. Therefore we can say it is not a superset yet.

We now construct the graph from the SoC design and the SASC we got, and later on we will use some techniques in graph coloring and maximum-weight clique problem to solve our problem. Each pin forms a vertex, so we will have \(\sum W_i (i = 1,2,\ldots,n)\) vertices in total. Then we connect every pair of vertices if their CUTs belong to the same superset in SASC.

$$B=8 \quad n=6 \quad W_1=8 \quad W_2=6 \quad W_3=4 \quad W_4=2 \quad W_5=1$$

**Figure 3.4:** Find the feasible combinations faster by predicting those combinations we can skip
3.3.2 Find Ideal Bus Width

Now we want to find the ideal bus width to see whether we can achieve full-spatial parallelism in single connection assignment or not. We use the original graph to generate a weighted graph as we discussed in use Section 3.2 because it reduces the problem size and is faster. The weight of maximal-weight clique in the weighted graph is the ideal bus width. For instance in Figure 3.2, \( B = 6, n = 4, W_1 = 6, W_2 = 3, W_3 = 3, W_4 = 2 \), the ideal bus width is 8, because the vertices of \( \text{CUT}_2, \text{CUT}_3, \text{CUT}_4 \) forms a largest clique and they all have to assign to different integer values (indices of bus line). In this situation, we have to make more connections from the pins to the bus line because we set bus width limit to be six (\( B = 6 \)), it is obviously not wide enough for every pin in the clique having only one connection without bus collision. The reason why we do not simply widen the bus width limit is shown in Figure 3.5. If we choose to widen the bus width limit, we may have to take more supersets into consideration and these supersets may leads to a larger ideal bus width and so on. The worst case we could get in the end is the ideal bus width equals the number of all pins (\( \sum W_i (i = 1,2,...,n) \)). Preventing taking more supersets into consideration and getting into the loop shown in Figure 3.5, we only calculate the ideal bus width once. If the answer is not larger than the bus width limit set by us, then we can simply apply DSATUR algorithm to have the feasible connections scheme to achieve full-spatial parallelism. If not, it means that we have to add more connections by one of our three proposed approaches.

![Figure 3.5: Loop relation between supersets and ideal bandwidth](image)

Figure 3.5: Loop relation between supersets and ideal bandwidth
3.4 Postponed-Assignment and Immediate-Assignment Approach

The Postponed-Assignment approach and the Immediate-Assignment approach are both based on DSATUR algorithm which we introduced in Section 2.3. Generally, it can be divided into three parts. The first part is DSATUR Assignment where we make the first connection of each pin. The second part is Verification; we determine if each of the supersets in SASC is guaranteed to be tested in parallel or not. The supersets which are failed in Verification can be fixed in the last part, Adjustment. The flowcharts of the two approaches are shown in Figure 3.6. The only difference between the Postponed-Assignment approach and the Immediate-Assignment approach is that when DSATUR Assignment can not find a valid bus line for pins. In the Postponed-Assignment approach, we leave those pins unassigned and postpone the assignment of them to Adjustment; in the Immediate-Assignment approach, we do Redundant connection Assignment forcing those pins to be assigned immediately and meanwhile some redundant connections are made.

![Flowchart](image)

Figure 3.6: (a) Postponed-assignment approach, (b) Immediate-assignment approach
3.4.1 DSATUR Assignment

In Postponed-Assignment approach

In this step, some vertices (pins) will be assigned with a positive integer (index of bus line). We use pseudo-code in Figure 3.7 to demonstrate how the DSATUR assignment in the Postponed-Assignment approach works. First we have to define five properties of each vertex:

1. \( \text{Index}(x) \): Index of vertex \( x \).
2. \( \text{NS}(x) \): Number of superset to which \( x \) belongs.
3. \( \text{Deg}(x) \): Number of edges adjacent to vertex \( x \).
4. \( \text{DSAT}(x) \): Number of different first values at the vertices adjacent to vertex \( x \).
5. \( \text{Assigned}(x) \): One, if vertex \( x \) is assigned; Zero, if vertex \( x \) is unassigned

The first three properties are static properties, which mean that they will not change during the assignment, and the last two are dynamic properties; they should be updated in the iterations.

Given the bus width limit and a list of all the vertices (lines 01 through 03), we choose one vertex which has the highest priority to assign in each iteration (lines 04 through 17). How do we decide the priority? We want the unassigned vertex with the largest \( \text{DSAT} \) (lines 09 and 08). If there is a tie, we choose the one with the largest \( \text{Deg} \) among them (line 07). If we still can not find only one vertex, we try to find the one with largest \( \text{NS} \) then the one with smallest \( \text{Index} \) among those vertices (lines 06 and 05). After the five judgments, we now have vertex \( x \) with the highest priority (line 10). By detecting its neighbors finding the smallest unused integer value, we find the candidate number \( k \) for vertex \( x \) (line 11). We need to check if the candidate number exceeds the bus width limit or not (line 12). If the answer is no, we assign \( x \) with \( k \) (line 15). If the answer is yes, it implies that we are running out of bus lines, all the bus lines are assigned to the pins which might be active with it simultaneously. We do not assign any value to vertex \( x \), but we set it assigned so that it lose the priority and we will not find it as the vertex with highest priority again (line 13).
01: \[ m = \sum_{i=1}^{n} W_i; \] /* Sum of all the pins */

02: \( B = \) bus width limit;

03: \( AV = [v_0, \ldots, v_{m-1}]; \) /* \( AV \) is a list containing all the vertices */

04: for (\( i = 1 \) to \( m \) ) loop

05: \( \text{Sort}(AV, \text{Index}, a); \) /* Sort \( AV \) according to \( \text{Index} \) in ascending order */

06: \( \text{Sort}(AV, NS, d); \) /* Sort \( AV \) according to \( NS \) in descending order */

07: \( \text{Sort}(AV, \text{Deg}, d); \) /* Sort \( AV \) according to \( \text{Deg} \) in descending order */

08: \( \text{Sort}(AV, \text{DSAT}, d); \) /* Sort \( AV \) according to \( \text{DSAT} \) in descending order */

09: \( \text{Sort}(AV, \text{Assigned}, a); \) /* Sort \( AV \) according to \( \text{Assigned} \) in ascending order */

10: \( x = AV[0]; \) /* The first vertex(index=0) of \( AV \) has the highest priority to be assigned */

11: \( k = \text{SUI}(x); \) /* \( k \) is smallest unused integer among all the neighbors of \( x \) */

12: if \( (k > B) \) then

13: Do not assign any value to vertex \( x \) but set \( x \) as assigned;

14: else

15: Assign \( k \) to \( x; \) /* Pin \( x \) connects to bus line \( k \) */

16: end if-then-else

17: end for

Figure 3.7: Pseudo-code of DSATUR assignment in Postponed-Assignment approach

In Immediate-Assignment approach

Figure 3.8 depicts the pseudo-code of DSATUR assignment in the Immediate-Assignment approach. Basically, we use the same process in the Postponed-Assignment approach to choose one vertex to assign. But the difference is that when the candidate number \( k \) for vertex \( x \) is larger than bus width limit (line 15), we call the function \( F \) to deal with the problem. We collect the unassigned vertices in \( x \)’s CUT as violation vertices (line 16), which mean the vertices which can not be assigned by DSATUR assignment. In function \( F \), we first try to find the assigned vertices which are not adjacent to violation vertices and use their values to assigned to violation vertices (line 32). Once the vertex is assigned with a value, we remove in from the violation vertices. If there are still violation vertices, we try to find a smallest completed CUT whose size is not smaller than the number of violation
vertices and use the values of them to assign to violation vertices (lines 35 and 36). Those vertices we shared their values with the violation vertices are now having bus collision problems, so we have to do the compensation for them. We assigned another values to them representing alternative bus lines they can choose to transport data (lines 37 through 41). If there are still violation vertices, we assign the values which are not in the taboo list (lines 46 through 48).

\[
01: \quad m = \sum_{i=1}^{n} W_i; \quad /* \text{Sum of all the pins} */ \\
02: \quad B = \text{bus width limit}; \\
03: \quad AV = [v_0,...,v_{m-1}]; \quad /* AV \text{ is a list containing all the vertices} */ \\
04: \quad CC = []; \quad /* CC \text{ is a list containing the completed CUTs} */ \\
05: \quad VV = []; \quad /* VV \text{ is a list containing violation vertices we are going} \\
\text{to assign in function } F */ \\
06: \quad \text{mod } e; \quad /* \text{mode for function } F */ \\
07: \quad \text{while (not all the vertices are assigned) loop} \\
08: \quad \text{Sort}(AV, Index, a); \quad /* \text{Sort } AV \text{ according to } Index \text{ in ascending order} */ \\
09: \quad \text{Sort}(AV, NS, d); \quad /* \text{Sort } AV \text{ according to } NS \text{ in descending order} */ \\
10: \quad \text{Sort}(AV, Deg, d); \quad /* \text{Sort } AV \text{ according to } Deg \text{ in descending order} */ \\
11: \quad \text{Sort}(AV, DSAT, d); \quad /* \text{Sort } AV \text{ according to } DSAT \text{ in descending order} */ \\
12: \quad \text{Sort}(AV, Assigned, a); \quad /* \text{Sort } AV \text{ according to } Assigned \text{ in ascending order} */ \\
13: \quad x = AV[0]; \quad /* \text{The first vertex(index=0) of } AV \text{ has the highest priority to be} \\
\text{assigned} */ \\
14: \quad k = \text{SUI}(x); \quad /* k \text{ is smallest unused integer among all the neighbors of } x */ \\
15: \quad \text{if } (k > B) \text{ then} \\
16: \quad VV = \text{unassigned vertices in } x \text{'s CUT(including } x \text{ itself);} \\
17: \quad \text{mod } e = 0; \\
18: \quad F(VV, \text{mod } e); \quad /* \text{Deal with the violation vertices} */ \\
19: \quad \text{else} \\
20: \quad \text{Assign } k \text{ to } x; \quad /* \text{Pin } x \text{ connects to bus line } k */ \\
21: \quad \text{if } (x \text{ is the last vertex in the CUT}) \text{ then} \\
22: \quad \text{Add CUT of } x \text{ into } CC; \\
23: \quad \text{end if} \\
24: \quad \text{end if-then-else}
25: end while

26: \( F(\text{vertices } G, \text{int } M) \) function start
27: \textbf{if} (\( M = 0 \)) \textbf{then}
28: Build a taboo list from all the values of assigned vertices in \( G \)'s CUT;
29: \textbf{else}
30: Build a taboo list from all the values of all the vertices in \( G \)'s CUT;
31: \textbf{end if-if-then-else}
32: Find the assigned vertices which are not adjacent to \( G \), assign their values which are not in the taboo list to the vertices in \( G \), once the vertex in \( G \) is assigned, remove it from \( G \) and update the taboo list;
33: \textbf{if} (\( G \) is not empty) \textbf{then}
34: Try to find a CUT \( Y \) in \( CC \), whose weight is smallest but not less than \# of vertices in \( G \);
35: \textbf{if}(\( Y \) is found) \textbf{then}
36: Assign the values of vertices of \( Y \) which are not in the taboo list to the vertices in \( G \), once the vertex in \( G \) is assigned, remove it from \( G \) and update the taboo list;
37: \textbf{if} (\( M = 0 \)) \textbf{then}
38: \( G2 = \text{vertices of } Y \) whose values are assigned to the vertices in \( G \);
39: \texttt{mode} = 1;
40: \( F(\text{group2,mode}) \);
41: \textbf{end if}
42: \textbf{end if}
43: \textbf{else}
44: Add CUT of \( x \) into \( CC \);
45: \textbf{end if-if-then-else}
46: \textbf{if} (\( G \) is not empty) \textbf{then}
47: Assign the vertices in \( G \) with unused values in the taboo list;
48: \textbf{end if}
49: Add CUT of \( x \) into \( CC \);
50: \textbf{function end}

Figure 3.8: Pseudo-code of DSATUR assignment in Immediate-Assignment approach
3.4.2 Verification

After finishing previous steps, lots of the supersets in SASC now have at least one solution path. So we are going to find out which supersets still have no solution path and we record them as failed supersets.

For each superset, we first find out the associated vertices. Then we sort these vertices according to the number of assigned values in ascending order and build an empty taboo list. The sorting will help us find a solution path quicker if it exists. The vertex with less number of assigned values means that it has less alternative paths to choose. So, when we are trying to permute a solution path from associated vertices, we should let vertices with less number of assigned values choose the bus line first.

We start from the first vertex. For each vertex, if the current value we select is not in the taboo list, then the value is added into the taboo list (make a value decision) and repeat the same movement for the following vertices. If the current value of current vertex is already in the taboo list, then we select the next value of current vertex, and if current vertex is running out of values, we need to backtrack to the last value decision we made trying another branch in the search tree. We claim the superset has no solution path if the backtracking reaches the top of the search tree and runs out of alternative branch. When the problem size becomes larger, we can set the backtracking limit for giving up if the backtracking costs too much time. When the backtracking limit is reached, we give up that superset and record it as failed superset. Figure 3.9 gives an example. We start from $V_1$ and choose value 1. Then we choose value 2 in $V_2$. Next, the first value of $V_6$ is 1, which is already in the taboo list, so we have to choose its second value, 4. But in $V_4$, all its value: 1, 2, 4 are already in the taboo list. When we can not find any candidate value, we have to backtrack to the last decision we made, in this case, backtrack to $V_6$. And in $V_6$ we can not find any other value to choose, so we backtrack further to $V_2$. In $V_2$, we can choose another value, 3. And applying the same rule, we finally find a solution path for these four associated vertices then we can claim the superset is guaranteed. In the figure, the red line is the first search path we made, and the red dash line is the backtracking. The blue line is the second search path we made, which succeeds.
3.4.3 Adjustment

In this step, we will use a simple way to add connections. The existence of failed supersets means we need more connections for pins. For each failed superset, we first sort the associated vertices according to the number of assigned values in ascending order. Then we build an empty taboo list, and try to add one new value to the taboo list from each sorted vertex, until we can not find a new value to add from current vertex. In this situation, we do not do backtrack. We simply assign to current vertex with the smallest integer which is not present in the taboo list and keep trying to add value until the last vertex. The idea is that the vertices with larger number of assigned values are more flexible, they have higher chance to find a value adding to taboo list. So we serve them later and serve the less flexible vertices first. By doing so, we have lower chance to assign new values for vertices (add new connections for pins) which will increase our cost.

Till the end, we select different values from every associated vertex and store the values in the taboo list. This implies the superset now has at least one solution path. We repeat this movement for all the failed supersets. In this way, we can guarantee all the failed supersets now have at least one solution path.
3.5 Simulated-Annealing Algorithm

The reason for implementing this Simulated-Annealing algorithm is try to produce an optimal solution so that we can compare it with the solutions coming from our approaches. The process is basically like what we discussed in Section 2.5. Before introducing how to generate a neighbor solution, we have to explain the relation between connection cost, connection distribution and connection scheme.

1. Connection cost: The total wire connections.
2. Connection distribution: How the connections are distributed among pins. Each pin will be given at least one connection. For example, if we have 3 pins and the connection cost is 4, the connection distribution could (2, 1, 1), or (1, 2, 1) or (1, 1, 2).
3. Connection scheme: Given a connection distribution, how the connections are connected to the bus lines. It is the neighbor solution we generate.

First, we generate a new connection distribution from the current solution. The method is to change the number of connections of each pin according to a probability (1/3). Each pin may: 1. Get an extra connection, 2. Be taken away one connection, or 3. Keep the same number of connections. Next, we use the new connection distribution to generate a new connection scheme. Each connection has 50% chance to inherit from the current solution, which means around half of the connections will connect to the same bus line as the current solution. This can ensure the reachability for solutions. To have a better efficiency, we apply three rules when generating a new connection scheme (neighbor solution):

1. For pins in the same CUT, their first connections are connected to different bus line.
2. Connections belong to the same pin are connected to different bus line.
3. Connection distribution in the same core will be re-permuted as even as possible. Ex: A connection distribution in a CUT is (4,6,3), we will re-permute it to (4,5,4). Because the later connection distribution makes the solution more powerful but it remains the same cost.

Rule1 and rule2 can prevent generating some dummy invalid solutions and rule3 improve the quality of generated solutions.

In the algorithm, we have the following parameters need to be tuned.

1. \textit{initial\_CP}: Initial connection per pin. We set it as the ceiling of the connection
per pin generated from the Immediate-Assignment approach so that we do not set it too low to find valid solutions. This number multiplied by the number of pins will be the initial connection cost we set.

2. **unmoved_limit**: It is the number or limited iterations we set, if we can not find a neighbor solution to replace the current solution within limited iterations, the program will be terminated.

3. **inner_limit**: Number of iterations to generate neighbor solutions in one temperature.

4. **temperature_degradation**: Cooling speed, the larger we set the slower cooling is.

At the beginning, we set the initial cost to be `initial_CP` multiplied by number of pins to find an initial current solution. At each temperature, we try to find several valid neighbor solutions (lines 02 through 07), and we must verify the neighbor solutions we generate just like what we discussed in subsection 3.4.2 (line 06). We pick the lowest-cost generated neighbor solution which passes the verification (line 08). If the cost is lower than the cost of current solution, we replace the current solution with the solution we just found (lines 09 and 10). If it is not, we replace the current solution with it according to the acceptance probability (lines 11 and 12). The acceptance probability is:

\[
\exp\left(\frac{current\text{-}cost - new\text{-}cost}{temperature}\right)
\]

This probability decreases when temperature decreases so that we can control the program to converge to an optimal solution. When we find a neighbor solution to replace the current solution, we need to reset the counter for unmoved times (lines 14 through 16) and continue the next searching. We do the cooling process at the end of iteration (line 17).
01: while (unmoved_times < unmoved_limit) loop
02:     while (inner_times < inner_limit) loop
03:         Find neighbor solutions:
04:             1. Change the connection distribution (this might change the connection cost).
05:             2. Change the connection scheme:
06:                 Each connection has 50% chance to inherit from the current connection scheme, the
07:                     others will be regenerate according to two rules:
08:                         a. For pins in the same CUT, their first connections are connected to different bus line.
09:                         b. Connections belong to the same pin are connected to different bus line.
10:         3. Verify the generated neighbor solutions to see if it is valid.
11:     end while
12:     Evaluate the cost of valid neighbor solutions and choose the one which has the lowest cost.
13:     if (lowest cost < current cost) then
14:         Replace the current solution with it.
15:     else
16:         Replace the current solution with it according to the acceptance probability.
17:     end if-then else
18:     if (did the replacement) then
19:         clear the unmoved_times
20:     end if
21:     temperature = temperature * temperature_degradation;
22: end while

Figure 3.10: Pseudo-code of Simulated-Annealing algorithm
Chapter 4

Experimental Results

This chapter shows the result of our work. The benchmarks we use are ISCAS’89 and ITC’02.

4.1 Postponed vs. Immediate

ISCAS’89 provides only the information of cores, so we randomly select cores in ISCAS’89 as cores of SoCs in our benchmark. The number of cores varies from 8 to 28 and we calculate the average result from 20 generated SoCs for each case. As for ITC’02, we use the original design in it as our benchmark. Each core has the input pins and the output pins, so we have two bus systems (input and output) in one SoC as shown in Figure 1.2.

For the input side of SoCs consisting of ISCAS’89 benchmark cores, Figure 4.1 shows the number of pins in SoC and number of supersets we need to guarantee with different number of cores. It tells that the number of pins is proportional to the number of cores, but the number of supersets grows exponentially. It implies that the problem size for verification step increases much faster than the other parts of the solution. Therefore, the execution time of both Postponed-Assignment and Immediate-Assignment approach grow exponentially with increasing number of cores (Figure 4.2). As for the output side of SoCs consisting of ISCAS’89 benchmark cores, exponential growth of supersets happens also (Figure 4.3 and Figure 4.4). Comparison of the wire connection cost for two approaches are described in Table 4.1. Each pin needs at least one connection to the bus line, except that, the other connections are called redundant connections. The definition of saving ratio is:

\[
\text{Saving ratio} = \frac{\text{\# of Connections}_{\text{Postponed}} - \text{\# of Connections}_{\text{Immediate}}}{\text{\# of Connections}_{\text{Immediate}} - \text{\# of Pins}}
\]

It is the percentage of redundant connections cost we can save by applying Immediate-Assignment approach instead of Postponed-Assignment approach. If the
saving ration is positive, it implies that the Immediate-Assignment approach performs better in the case. Among all the cases, only the case “8 cores in output side” shows the Postponed-Assignment approach gets better result. We check the number of supersets in that case from Figure 4.3, and find out it is pretty low, so we might over-compensate the pins shared their values with violation vertices. The efficiency of compensation decrease due to the low number of supersets we need to guarantee.

Figure 4.1: Number of pins and number of supersets in ISCAS’89 input side

Figure 4.2: Execution time of Postponed and Immediate in ISCAS’89 input side
Figure 4.3: Number of pins and number of supersets in ISCAS’89 output side

Figure 4.4: Execution time of Postponed and Immediate in ISCAS’89 output side
Table 4.1: Comparison of connection cost and saving ratio in ISCAS’89

<table>
<thead>
<tr>
<th># of Cores</th>
<th>Postponed</th>
<th>Immediate</th>
<th>Saving Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of connections</td>
<td>C/P</td>
<td># of connections</td>
</tr>
<tr>
<td><strong>Input Side</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>252.8</td>
<td>1.23017</td>
<td>239.35</td>
</tr>
<tr>
<td>12</td>
<td>521.5</td>
<td>1.90921</td>
<td>461.6</td>
</tr>
<tr>
<td>16</td>
<td>1042.75</td>
<td>2.65804</td>
<td>902.85</td>
</tr>
<tr>
<td>20</td>
<td>1421.25</td>
<td>3.10758</td>
<td>1295.9</td>
</tr>
<tr>
<td>24</td>
<td>2238.35</td>
<td>3.95958</td>
<td>2188.6</td>
</tr>
<tr>
<td>28</td>
<td>3173.4</td>
<td>4.43429</td>
<td>3086.9</td>
</tr>
<tr>
<td><strong>Output Side</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>365.7</td>
<td>1.0905</td>
<td>369.65</td>
</tr>
<tr>
<td>12</td>
<td>817.7</td>
<td>1.57583</td>
<td>783</td>
</tr>
<tr>
<td>16</td>
<td>1666.6</td>
<td>2.69829</td>
<td>1604.8</td>
</tr>
<tr>
<td>20</td>
<td>3650.7</td>
<td>5.00748</td>
<td>3423.5</td>
</tr>
</tbody>
</table>

Note: C/P = connections / pin

For ITC’02, we ignore the results of circuits which can be guaranteed without adding redundant connections. In Table 4.2, all three negative saving ratios come from the cases with low number of supersets, which imply that we over-compensate in these cases.

Table 4.2: Comparison of connection cost and saving ratio in ITC’02

<table>
<thead>
<tr>
<th>SoC design</th>
<th># of pins</th>
<th># of supersets</th>
<th>Postponed</th>
<th>Immediate</th>
<th>Saving Ratio (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># of connections</td>
<td>C/P</td>
<td># of connections</td>
<td>C/P</td>
<td></td>
</tr>
<tr>
<td><strong>Input Side</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d281</td>
<td>1523</td>
<td>7</td>
<td>1832</td>
<td>1.20289</td>
<td>1895</td>
</tr>
<tr>
<td>d695</td>
<td>584</td>
<td>79</td>
<td>1629</td>
<td>2.78938</td>
<td>1560</td>
</tr>
<tr>
<td>g1023</td>
<td>1689</td>
<td>74</td>
<td>4508</td>
<td>2.66903</td>
<td>899</td>
</tr>
<tr>
<td>h953</td>
<td>426</td>
<td>8</td>
<td>508</td>
<td>1.19249</td>
<td>540</td>
</tr>
<tr>
<td><strong>Output Side</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d695</td>
<td>1261</td>
<td>22</td>
<td>1884</td>
<td>1.49405</td>
<td>1839</td>
</tr>
<tr>
<td>g1023</td>
<td>1898</td>
<td>89</td>
<td>4693</td>
<td>2.4726</td>
<td>4081</td>
</tr>
<tr>
<td>h953</td>
<td>450</td>
<td>9</td>
<td>571</td>
<td>1.26889</td>
<td>628</td>
</tr>
<tr>
<td>u226</td>
<td>102</td>
<td>8</td>
<td>111</td>
<td>1.08824</td>
<td>106</td>
</tr>
</tbody>
</table>

Note: C/P = connections / pin
From the results of ISACAS’89 and ITC’02 we can tell that the Immediate-Assignment approach performs better than the Postponed-Assignment approach in most of the cases except the simples. The reason is that the compensation in the Immediate-Assignment approach turns out to be unnecessary in simple cases.

4.2 SA vs. Immediate

Simulated-Annealing algorithm gives us a chance to find the optimal solution. But due to the random generation of neighbor solution, when the problem size comes large, the chance to generate a feasible solution decreases.

We do the experiment for six cases and the results are in Table 4.3. Case A, B, C are the small SoC designs containing 5, 6, 10 cores generated by us. Case ISCAS’89_in and ISCAS’89_out are the benchmark used in section 4.1 which are the smallest in input side and output side of ISCAS’89. Case u226_out is the smallest design in ITC’02 benchmark. For all the cases, we set unmoved_limit extremely large, and apply a CPU time limit (6 hours) to prevent the situation that the program can not find a feasible solution at all. We also tune the parameters: inner_limit and temperature_degradation with different combinations and list the best results here. In the table, the number of connections in red color is the global optimal solution.

As shown in Table 4.3, the SA algorithm can find the global optimal solution in case A and B within a time limit (6 hour), but not in the other cases. In case C, ISCAS’89_out, and u226_out, the SA algorithm can find some higher-cost solutions comparing to Immediate-Assignment approach, but these solutions are clearly not global optimal solutions. As for case ISCAS’89_in, the SA algorithm can not even find one feasible solution for the problem. All these results show that the implemented SA algorithm works only for simpler cases. The key point is the limited neighbor solution generator. Since our problem is complex and highly constrained, it is hard to have a neighbor solution generator which generates only valid solutions. Although we already apply three rules trying to improve the efficiency of generator, we still can not prevent the growing-proportion of invalid solutions coming out from generator when the problem size increases. On the other side, the Immediate-Assignment approach can find feasible solutions within a very short time for all the cases and the solutions in case A and B are even proved to be near optimal.
<table>
<thead>
<tr>
<th>SoC design</th>
<th># of pins</th>
<th># of superset s</th>
<th>Simulated-Annealing</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cost(# of connections)</td>
<td>CPU time (sec)</td>
</tr>
<tr>
<td>A</td>
<td>16</td>
<td>4</td>
<td>18</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>39</td>
<td>5</td>
<td>45</td>
<td>417</td>
</tr>
<tr>
<td>C</td>
<td>95</td>
<td>11</td>
<td>--</td>
<td>Time out</td>
</tr>
<tr>
<td>ISCAS’89_in</td>
<td>117</td>
<td>14</td>
<td>--*</td>
<td>Time out*</td>
</tr>
<tr>
<td>ISCAS’89_out</td>
<td>72</td>
<td>5</td>
<td>--</td>
<td>Time out</td>
</tr>
<tr>
<td>u226_out</td>
<td>102</td>
<td>8</td>
<td>--</td>
<td>Time out</td>
</tr>
</tbody>
</table>

*: The SA program can not find any feasible solution within time limit.
Chapter 5

Conclusion and Future Work

This chapter concludes what we have done and points out the possible direction for further research.

5.1 Conclusion

The goal of this thesis is to provide a most flexible TAM design to test scheduling, which means guaranteeing full-spatial parallelism. We propose a TAM structure using memory as a buffer to deal with the width mismatch between chip and cores, and two approaches to find a connections scheme satisfying our goal.

The proposed Postponed-Assignment and Immediate-Assignment approaches are both DSATUR based methods. In the Immediate-Assignment approach we do the compensation for the vertices which share their values with violation vertices. We can save execution time by setting upper bound in verification step, which is very useful when the problem size becomes large.

The implemented Simulated-Annealing algorithm tries to search for optimal or near-optimal solutions. The creation of neighbor solution has three stages. First it generates a new connection contribution from the current best solution. Secondly it generates a new connection scheme for the new connection contribution. Thirdly, the new connection scheme needs to be verified. If the verified new connection scheme has lower cost, it replaces best solution. On the other hand, the verified new connection scheme with higher cost can only replace best solution according to the acceptance probability. The implemented SA algorithm will keep updating the current best solution until it reaches the stop condition.

The experiment shows that the Immediate-Assignment approach and the Postponed-Assignment approach can find a feasible solution in a very shot time as oppose to the Simulated-Annealing algorithm. And the Immediate-Assignment approach performs better than the Postponed-Assignment approach in most of the SoCs except the small ones.
5.2 Future Work

This thesis provides a TAM structure and the approaches to find the connection scheme achieving full-spatial parallelism. One direction for future work is the implementation of TAM design in detail. For example, design the structure in hardware description language. Another direction is to use constraint logic programming to solve the problem optimally.
References


Appendix

Experimental results in detail

ISACAS’89 8~28 cores in input side:

ISACAS’89 8~20 cores in outside:
http://oz.nthu.edu.tw/~u942516/thesis/isca89_out_8~20.xls

ITC’02 in both sides:
http://oz.nthu.edu.tw/~u942516/thesis/itc02.xls

Small cases for comparison of SA and Immediate:
http://oz.nthu.edu.tw/~u942516/thesis/small_case.xls
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida http://www.ep.liu.se/

Copyright

The publishers will keep this document online on the Internet – or its possible replacement – from the date of publication barring exceptional circumstances.

The online availability of the document implies permanent permission for anyone to read, to download, or to print out single copies for his/hers own use and to use it unchanged for non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional upon the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: http://www.ep.liu.se/.

© [Kuei-Hsi Tseng]