10th IEEE Workshop on RTL and High Level Testing (WRTLT'09), pp. 29-32, Nov. 2009.

Path-Based Resource Binding to Reduce Delay Fault Test Cost*

Michiko Inoue† Satoshi Ohtake Yu-ichi Uemoto
Hideo Fujiwara
Graduate School of Information Science
Nara Institute of Science and Technology

Abstract

This paper proposes a new high level synthesis method that reduces the number of non-false paths as well as the number of paths during *resource binding*. The method is useful to avoid over-testing as well as to alleviate test generation in combination with false path identification methods.

keywords: high level synthesis, resource binding, false path identification

1 Introduction

High level synthesis transforms a behavioral description of a circuit into an RTL circuit. It determines a basic structure of a circuit, and hence, affects test cost. A false path in a circuit is a path that is never activated in normal operation mode. Such paths do not need to be tested, or should not be tested if they have more delay than a system clock cycle and can be activated in test mode due to design for testability such as full scan. Therefore, false path identification is utilized to avoid over-testing as well as to alleviate test generation.

Several false path identification methods have been investigated at gate level[1] and RTL[2, 3, 4, 5]. RTL false path identification is efficient since it can treat a bundle of gate level paths as one *RTL path*. False path identification runs efficiently if the circuit has less paths and works effectively for test generation if the circuit has more false paths. This paper proposes a new high level synthesis method that reduces the number of non-false paths as well as the number of paths during *resource binding*.

The proposed method adopts a path-based assignment strategy where resource sharing is considered based on a data flow between variables in a behavioral description. The path based assignment strategy was introduced by Kim et al.[6] where it is used to reduce the interconnect area and give no consideration into false paths or test generation. This paper uses this strategy to reduce not only the number of RTL paths but also the number of RTL *true paths* (non-false paths). We demonstrate the effectiveness of the proposed method by experiments for some high level synthesis benchmarks.

2 Preliminaries

Resource Binding A circuit is represented by variables, operations and their flow in a behavioral description, while it is represented by registers, operational modules and their connections in RTL description. In high level synthesis, *scheduling* determines when each operations are executed and *resource binding* assigns variables and operations to registers and operational modules, respectively. In this paper, we treat resource binding after scheduling. We use a scheduled data flow graph (SDFG) as a behavioral description after scheduling.

Definition 1 *SDFG is a directed graph $G = (V, E, t, s)$ where V is a set of primary inputs, primary outputs and operations, $E \subseteq V \times V$is a set of variables, $t : V \rightarrow \{op_1, op_2, \cdots, op_n\}$ represents operation types, and $s : V \rightarrow Z_{+0}$ represents control steps when operations are executed.*

In this paper, we do not assume chaining operations, that is, we assume $s(o_i) < s(o_j)$ for any variable (o_i, o_j). For a variable $v = (o_i, o_j)$, we say that v is an output variable of o_i and an input variable of o_j, o_i generates v and o_j uses v.

Resource binding assigns each operation in V to an operational module (*operational module binding*), each variable in E to a register (*register binding*) and also to a port of operational module corresponding to an operation that uses the variable (*interconnect binding*). Figure 1(a) shows an SDFG with 9 variables and 4 add operations executed at control steps $1, 2, 3$. Figure 1(b) shows an assignment of variables a, b, d, e, f, h and add operations $+1, +3$. Both variables b, d are assigned to the

*This work is supported in part by Semiconductor Technology Academic Research Center (STARC) and Japan Society for the Promotion of Science (JSPS) under the Grant-in-Aid for Science Research (B)No.20300018.

†contact author, e-mail:kounoe@is.naist.jp

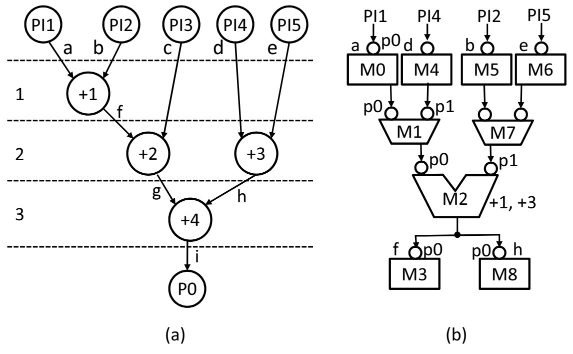

Figure 1: Resource binding example: (a)SDFG and (b)(a part of) RTL circuit.

same port $p0$ of the adder $M2$ and they are transferred to the port through the multiplexor $M1$.

In this paper, we have the same assumption as [3] on high level synthesis and logic synthesis as follows. (1) Every register allocated during high level synthesis is controlled so that it does not load any value unless its input has a valid data. (2) For any combinational RTL module, no logic optimization beyond its boundary is done.

RTL paths and DFG paths An RTL path is an ordered set $\{M_0, M_1(p_1), M_2(p_2), \cdots, M_n(p_n)\}$ where M_0 is a primary input or a register, M_n is a primary output or register, $M_i(0 < i < n)$ is an operational module or a multiplexor, and $M_i(0 \leq i < n)$'s output is connected to a port p_{i+1} of M_{i+1}. For example, an ordered set $\{M_0, M_1(p_0), M_2(p_0), M_3(p_0)\}$ in Fig.1(b) is an RTL path. In general, one RTL path corresponds to a set of paths in a synthesized gate level circuit.

In this paper, a gate level false path means a path which is unsensitizable under the *non-robust sensitization criterion*[6]. An RTL path is *false* if any path in its corresponding set of gate level paths is false for any logic synthesis, and an RTL path is *true* if it is not false.

A DFG path is a 3-tuple (v_i, o, v_o), where o is an operation and v_i and v_o are input and output variables of o. For example, $(a, +1, f)$ in Fig.1(a) is a DFG path and it corresponds to an RTL path $\{M_0, M_1(p_0), M_2(p_0), M_3(p_0)]\}$ in Fig.1(b). In [3], we showed a sufficient condition for RTL false path. An RTL path is false if (1) there is no corresponding DFG path in SDFG, or (2) there is a corresponding DFG path (v_i, o, v_o) but v_i crosses boundaries of control steps more than once.

We define DFG true and false paths as follows. A DFG path (v_i, o, v_o) is *true* if $s(o) = s(o_g) + 1$ holds for an operation o_g that generates a variable v_i. A DFG path is *false* if it is not true. From the above sufficient condition, an RTL path is $false$ if no true DFG path corresponds to the RTL path.

3 Proposed Method

We propose a resource binding method that generates an RTL circuit with small number of RTL paths and small number of RTL true paths. The proposed method first considers a path-based resource sharing in which a set of DFG paths are assigned to the same RTL path. This strategy is effective to reduce the number of RTL paths.

Compatibility for DFG path blocks Though a concept of path-based resource sharing was introduced by [6], it considers only a series of the same type operations that can be assigned to the same self loop. This strategy can simplifies and localizes interconnects between registers and operational modules and can reduce the interconnect area.

In this paper, we generalize this concept. In the proposed method, a set of DFG paths are partitioned into blocks where each block is a set of DFG paths that share the same RTL path. The partition is obtained by repeatedly merging two blocks until no blocks are compatible. To realize this strategy, we introduce a compatibility for two blocks. A compatibility means that two objects can share the same resources.

First we define compatibilities for variables and operations. Two variables are compatible if their life times do not overlap. The life time of a variable is an interval $[s(o_g) + 1, s(o_u)]$ where o_g and o_u are operations which generates and uses the variable. Two operations o_1, o_2 are compatible if $s(o_1) \neq s(o_2)$ and $t(o_1) = t(o_2)$. A compatibility for two blocks is more complicated. It is required that input variables, operations, and output variables appearing in the two blocks are mutually compatible, respectively. In addition, if these variables or operations appear as both input and output variables or appear in the other blocks, it implies resource sharing among them and compatibilities are required among the variables or operations that share the same resource. For a partition of a set of DFG paths, sets of variables or sets of operations that share the same resources are determined. We call such a set of variables and a set of operations *a super variable* and *a super operation*, respectively.

Consider an example for the SDFG in Fig.1. Figure 2(a) shows a set of DFG paths and Fig.2(b) shows compatibilities for an initial partition $\{\{P_a\}, \{P_b\}, \{P_c\}, \{P_d\}, \{P_e\}, \{P_f\}, \{P_g\}, \{P_h\}\}$ where each block has one DFG path. For example, two blocks $\{P_a\}, \{P_b\}$ are not compatible since input variables a and b are not compatible. Figure 2(c) shows compatibilities after two blocks $\{P_g\}, \{P_f\}$ are merged. Since $P_g = (g, +4, i)$ and $P_f = (f, +2, g)$, this merge generates a super variable $\{g, i, f\}$ and a

Pa: (a, +1, f), Pb: (b, +1, f), Pc: (c, +2, g)
Pd: (d, +3, h), Pe: (e, +3, h), Pf: (f, +2, g)
Pg: (g, +4, i), Ph: (h, +4, i)

(a)

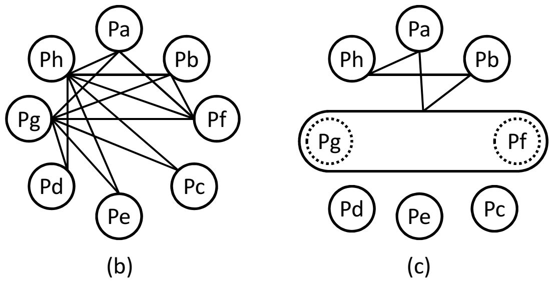

Figure 2: block compatibilities: (a)DFG paths, (b)compatibilities for an initial partition, (c)compatibilities after $\{P_g\}$ and $\{P_f\}$ are merged.

super operation $\{+4, +2\}$. One merge is affected to compatibilities for other blocks if merged variables or operations appear in other blocks. For example, $\{P_h\}, \{P_d\}$ are not compatible in Fig.2(c) since their merge implies resource sharing between super variables $\{+3\}, \{+4, +2\}$ and $+2$ and $+3$ are not compatible.

Outline The proposed method aims to reduce the number of RTL paths first, and to reduce RTL true paths second. In the proposed method, DFG paths are partitioned so that they share RTL paths as much as possible. At that time, we also consider resource sharing between DFG true paths so that the number of RTL true paths is reduced. It has the following 4 steps.

1. DFG path partitioning
   A set of DFG paths are partitioned into blocks where each block is a set of DFG paths that share the same RTL path. We start this step with an initial partition where each block is composed of a single DFG path. Then, we repeatedly select two blocks and merge them into one block until no compatible block pair remains. In each iteration, a pair of blocks such that more compatible block pairs remain after their merge is selected. If there are two or more candidate pairs, a block pair that has true paths for both blocks is preferred.

2. Operational module binding
   Though there is no compatible DFG path blocks at the beginning of this step, there may exist compatible super operations. In this step, we assign super operations to operational modules so that super operations with more common input variables and common output variables can share operational modules.

Table 1: Benchmark characteristics

benchmark	latency	#PI	#PO	#op.	#variable
Lwf	5	2	1	5	7
Tseng	5	3	1	8	11
Paulin	5	4	3	10	11

3. Register binding
   Register binding is similar to the operational module binding. We assign super variables to registers so that super variables with more common operations that generate or use the super variables can share registers.

4. Interconnect binding
   This step assigns super input variables to ports of operational modules. If the corresponding operation is commutative, interconnect binding has some flexibility. We perform an interconnect binding so that each register is connected with less ports.

4 Experimental Results

We evaluate the proposed method using some high level synthesis benchmarks. We compare the results with the RTL circuits with the minimum number of operational modules and registers shown in [7]. Table 1 shows the characteristics of the benchmarks, where #op. means the number of operations.

Table 2 shows the number of RTL paths for the proposed method (*Proposed*) and the results with the minimum number of operational modules and registers (*Minimum module & register*), where *total*, *#true* and *#false* mean the number of total RTL paths, the number of true paths and the number of false paths, respectively. This result shows that the proposed method can reduce both the number of RTL paths and the number of RTL true paths in most cases. Therefore, the proposed method can reduce delay fault test cost.

We also evaluate area of obtained RTL circuits (Table 3). In some cases, the proposed method needs more resources than the optimal cases. However, path-based resource binding works effectively to reduce interconnect area. For two benchmarks, we can reduce the number of multiplexors while preserving the minimum number of other resources.

5 Conclusions

This paper proposes a new high level synthesis method that reduces the numbers of RTL paths and RTL true paths during *resource binding*. The proposed method adopts a path-based assignment strategy that considers

Table 2: The results on the number of RTL paths

benchmark	Proposed			Minimum module & register		
	total	#true	#false	total	#true	#false
Lwf	9	7	2	14	8	6
Tseng	19	13	6	22	13	9
Paulin	26	12	14	27	9	18

Table 3: The results on area

benchmark	Proposed			Minimum module & register		
	#module	#mux	#register	#module	#mux	#register
Lwf	3	3	3	3	4	3
Tseng	7	6	7	7	7	5
Paulin	4	8	6	4	10	6

assignment of variables and an operation of a DFG path together. Experimental results shows that the proposed method effectively reduce both RTL paths and RTL true paths.

References

[1] A. Krstic and K.-T. Cheng, *Delay fault testing for VLSI circuits*. Kluwer Acdemic Publishers, 1998.

[2] Y. Yoshikawa, S. Ohtake, and H. Fujiwara, "False path identification using RTL information and its application to over-testing reduction for delay faults," in *Proceedings of IEEE 16th Asian Test Symposium*, pp. 65–68, 2007.

[3] N. Ikeda, S. Ohtake, M. Inoue, and H. Fujiwara, "RTL false path identification using high level synthesis information," *Technical Report of IEICE*, vol. 107-482, pp. 63–68, Feb. 2008 (In Japanese).

[4] Y. Yoshikawa, S. Ohtake, T. Inoue, and H. Fujiwara, "A synthesis method to alleviate over-testing of delay faults based on RTL don't care path identification," in *Proceedings of IEEE VLSI Test Symposium*, 2009.

[5] Y. Yoshikawa, S. Ohtake, T. Inoue, and H. Fujiwara, "Fast false path identification based on functional unsensitizability using RTL information," in *Proceedings of 14th Asia and South Pacific Design Automation Conference 2009*, pp. 660–665, 2009.

[6] T.Kim and X.Liu, "Compatibility path based binding algorithm for interconnect reduction in high level synthesis," in *Proceedings of International Conference on Computer Aided Design*, pp. 435–441, 2007.

[7] T. Takasaki, T. Inoue, and H. Fujiwara, "A high-level synthesis approach to partial scan design," in *Proceedings of IEEE Asia Test Symposium*, pp. 309–314, 1999.