Verification of Data-path and Controller Generation Phase of High-level Synthesis

C Karfa   D Sarkar   C Mandal
Department of Computer Sc. & Engg.
Indian Institute of Technology, Kharagpur
WB 721302, INDIA
{ckarfa, ds, chitta}@iitkgp.ac.in

Abstract

The paper describes a verification method of data-path and controller generation phase of high-level synthesis (HLS) process. The goal is achieved in two steps. In step 1, the generated RTL description is analyzed to obtain the register transfer (RT) operations executed in the data-path for a given control assertion pattern in each control step and in step 2, an equivalence checking method is proposed to verify the correctness of the controller. The method is implemented and integrated with an existing HLS tool, called SAST. The experimental results on several HLS benchmarks prove the effectiveness of the proposed method.

1 Introduction

High-level synthesis (HLS) is the process of generating the register transfer level (RTL) design from the behavioural description. The synthesis process consists of several inter-dependent sub-tasks such as, scheduling, allocation and binding and data-path and controller generation. The operations in the behavioural description are assigned time steps through scheduling process. The allocation and binding process binds the variables to a set of registers and the operations to a set of functional units (FUs) in each control step. The next task is to set the data-path by providing a proper interconnection path from the source register(s) to the destination register for every register transfer (RT) operation. This process of interconnection generation is called *data-path generation*. The objective of this step is to maximize the sharing of interconnection units and hence minimize the interconnection cost, while supporting conflict-free data transfers required by the RT-operations. The data-path generation task, in general, consists in identifying the scope of sharing interconnection paths among data transfer operations which do not take place simultaneously. The minimum number of control signals required to control all the data transfers in each control step is found next. Then, the functionality of each control signal is defined. Finally, the control assertion pattern needed in each control step is found. These processes are collectively called *controller generation*. The controller, represented as an FSM, assigns a value to each control signal, that is, invokes a control assertion pattern, in each control step to execute all the required data-transfers and proper operations in the FUs. As a result, a set of arithmetic operations as well as a set of relational operations are performed in the data-path. The results of the relational operations are stored in single bit registers whose outputs (status signals) are inputs to the controller. The state transitions in the controller FSM depend on these status signals. Finally, a high-level synthesis (HLS) tool produces an RTL with distinct control-path and data-path (CP-DP). The schematic of the RTL produced by any HLS tool is shown in figure 1.

The objective of this work is to ensure the correctness of the RTL design generated by HLS process, i.e., to ensure the correctness of both the data-path interconnections and the controller FSM. The goals are accomplished in two steps as shown in figure 2. First, *finite state machine with data-path* (FSMD) [5] [1] M_2 is constructed from the data-path interconnection information and the controller FSM.The synthesis results after allocation and binding phase is represented as FSMD M_1. In the next step, equivalence between the FSMDs M_1 and M_2 is established to verify the correctness of the controller.

The works reported in [10], [11], [4] try to establish the equivalence between the input behavioural description and the output RTL description of HLS process. The end-to-end HLS verification techniques, however, are not efficient enough as it is not only error prone but also unable to find

[1]An FSMD is an universal specification model, proposed by Gajski et al. in [5], which can represent all hardware designs. This model is used in the present work for encoding the designs to be verified.

the exact sub-task in which the error occurs. In contrast, several works have been reported in the literature to verify certain phases of HLS process. As for example, verification of scheduling process of HLS was reported in [9], [3], [6]. The correctness of allocation and binding phases of HLS are treated in [2], [1], [8]. In this paper, we propose an efficient verification technique for the data-path and controller generation phase.

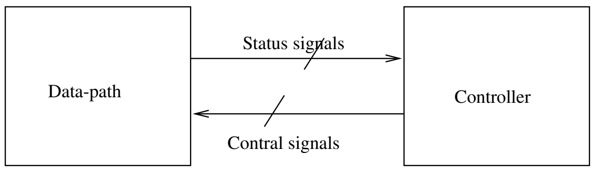

Figure 1. The structure of the RTL description produced by any HLS tool

This paper is organized as follows. The construction of FSMD from data-path and controller is discussed in section 2. The construction process is explained with an example in section 3. In section 4, verification during rewriting process is discussed. The equivalence checking method has been given in section 5. Some experimental results have been given in section 6. The paper is concluded in section 7.

2 Construction of the FSMD from CP-DP Information

Construction of the FSMD M_2 consists of following steps:

- Analyze the data-path vis-a-vis the control signal assertion pattern in each state of the controller FSM to construct the concurrent RT-operations in that state.
- Replicate the control flow of the controller FSM for the FSMD M_2.

The following two informations have to be extracted from the CP-DP description in order to find the register transfer (RT)-operations in each state of the FSMD M_2.

1. *The set of all possible micro-operations in the data-path.* Let this set be denoted as $\mathscr{M}$. A data movement from a data-path component y to another data-path component x is encoded by the micro-operation $x \Leftarrow y$. The data-path components essentially are storage elements (registers), the functional units and the interconnection components (buses, muxes, de-muxes, switches, etc.).
2. *The control signal assertion pattern for every micro-operation in $\mathscr{M}$.* A control signal assertion pattern needed for any micro-operation is represented as an ordered n-tuple of the form $\langle u_1, u_2, \ldots, u_n \rangle$, where u_i represents the value of the control signal c_i and n is the number of control signals; $u_i \in \{0, 1, X\}$, $1 \leq i \leq n$, is the required value of c_i. $u_i = X$ implies that the control signal c_i is not required (relevant) for a particular micro-operation. Let $\mathscr{A}$ be the set of all possible control assertion patterns. So, a function f_{mc} is constructed from the set $\mathscr{M}$ of all micro-operations possible in the given data-path to the set $\mathscr{A}$ of control signal assertion patterns. The DP interconnection is achieved by common signal naming. Thus, *the data-path structure, in its entirety, is captured by the function f_{mc}.*

In each state of the FSM, the controller generates a control signal assertion pattern to execute a set of micro-operations in the data-path to accomplish a set of register transfer operations concurrently. So, the next task is to obtain the set of micro-operations $\mathscr{M}_A$ ($\subseteq \mathscr{M}$) for a given control assertion pattern A. It is, however, not possible to obtain the set $\mathscr{M}_A$ of micro-operations directly from the control signal assertion pattern A by examining its individual control signals in isolation, because a micro-operation may be accomplished by a set of control signals rather than an individual control signal. There is no information available in an assertion pattern to group the control signals so that each group defines a micro-operation around a data-path component. The following definition is in order.

Definition 1 Superposition of Assertion patterns:
Let A_1 and A_2 be two arbitrary control signal assertion patterns. Let $\pi_i(A)$ denote the i-th projection of an assertion pattern A which is the asserted value u_i of the control signal c_i. The assertion pattern, $A_1 \,\theta\, A_2$, obtained by superposition θ of A_1 and A_2, satisfies the following conditions. For all i,

$$\begin{aligned}
\pi_i(A_1 \,\theta\, A_2) &= \pi_i(A_1),\ \mathit{for}\ \pi_i(A_1) = \pi_i(A_2) \\
&= \pi_i(A_1),\ \mathit{for}\ \pi_i(A_1) \neq \pi_i(A_2) \\
&\quad \mathit{and}\ \pi_i(A_1) = X \\
&= \mathit{undefined}\ (U),\ \mathit{for}\ \pi_i(A_1) \neq \pi_i(A_2) \\
&\quad \mathit{and}\ \pi_i(A_1) \neq X.
\end{aligned}$$

Using the above definition and the function f_{mc}, it is possible to construct $\mathscr{M}_A$ from the assertion pattern A by the following definition of $\mathscr{M}_A$: $\mathscr{M}_A = \{\mu_i \mid f_{mc}(\mu_i)\ \theta\ A = f_{mc}(\mu_i)\}$. The superposition of the assertion pattern of each micro-operation in $\mathscr{M}$ and A will be checked one by one to select each member of $\mathscr{M}_A$.

Each RT operation that appears in the RTL behaviour is accomplished by a set of concurrent micro-operations. So, in order to find the concurrent RT-operations accomplished by a control assertion pattern, it is necessary to find the operations realized by the set $\mathscr{M}$ of concurrent micro-operations.

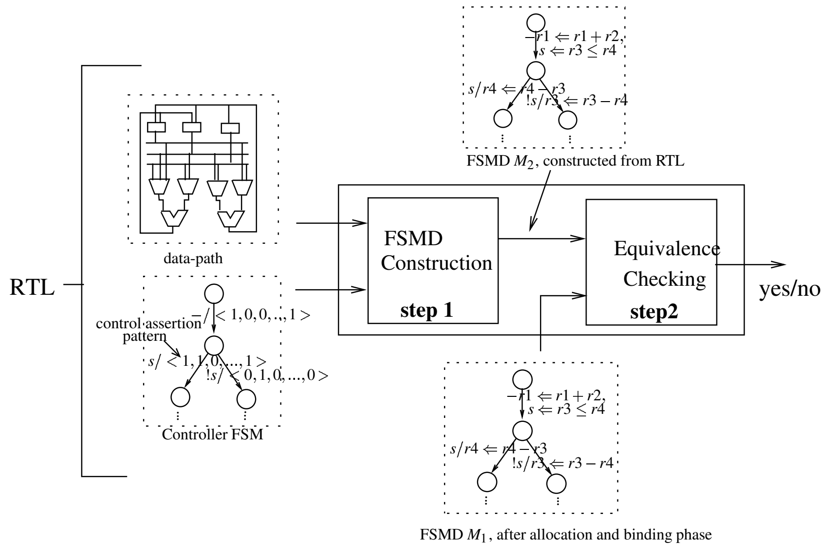

Figure 2. The steps of data-path and controller verification

Finding an RT-operation from a given set of micro-operations is also not trivial because of two reasons. First, there may be more than one RT-operation in that particular state of the FSM. Secondly, there is a *spatial sequence* of concurrent micro-operations needed to accomplish an RT-operation but these are available in an unordered manner in $\mathcal{M}_A$.

The concurrent RT-operations accomplished by the set $\mathcal{M}_A$ of micro-operations are identified using a *rewriting method*. The method also reveals the spatial sequence of data flow needed for an RT-operation in a reverse order (from the destination register back to the source registers). The basic rewriting method consists in rewriting terms one after another in an expression. The micro-operations in which a register occurs in the left hand side (lhs) are found first. Such a micro-operation has the form $r \Leftarrow r\_in$, where r is a register and r_in is its input terminal. Next, the right hand side (rhs) expression "r_in" is rewritten by looking for a replacement (micro-operation) in $\mathcal{M}_A$ of the form "$r\_in \Leftarrow s$" or "$r\_in \Leftarrow s_1 \ <op> \ s_2$". So, after rewriting "$r\_in$", we have the rhs expression, either of the form "s" or of the form "$s_1 \ <op> \ s_2$". In the next step, s (or s_1 *and* s_2 for the latter case) are rewritten *provided they are not registers*. When the expression in hand is of the form "$s_1 \ <op> \ s_2$" (and s_1, s_2 are not registers), then rewriting takes place from left to right in a *breadth-first manner*. Thus, at any point of time, the expression in hand can be of the form "$((s_1 \ <op_1> \ s_2) \ <op_2> \ s3) \ <op_3>_\uparrow \ \ldots$", where the pointer indicates the signal to be rewritten next. The process terminates successfully when all s_i's in the expression in hand are registers. Assuming that there are n DP components, the total number of edges can be n^2. Since in constructing each RT-operation no edge is traversed more than once, the complexity of constructing each RT-operation is $O(n^2)$.

The control structure of the FSMD M_2 can be obtained from the controller FSM and the RT-operations of each control state as constructed by the mechanism described above.

3 Construction of the FSMD M_2: An Example

Let us consider the data-path shown in the figure 3. In this figure, $r1, r2, r3$ are registers, $M1, M2$ are multiplexers, FU is a functional unit and $r1\_out$, $r2\_out$, $r3\_out$, $fLin$, $fRin$, $fOut$ are interconnection wires. The control signal names start with CS. The functionalities of the control signals are as follows:

$fLin \Leftarrow r1\_out \ : CS\_M1_1 = 0 \ \wedge \ CS\_M1_0 = 0$
$fLin \Leftarrow r2\_out \ : CS\_M1_1 = 0 \wedge CS\_M1_0 = 1$
$fLin \Leftarrow r3\_out \ : CS\_M1_1 = 1$
$fRin \Leftarrow r2\_out \ : CS\_M2 = 1$
$fRin \Leftarrow r3\_out \ : CS\_M2 = 0$
$fOut \Leftarrow fLin + fLin \ : CS\_FU = 0$
$fOut \Leftarrow fLin - fRin \ : CS\_FU = 1$
$r1 \Leftarrow fOut \ : CS\_r1Ld = 1$
$r3 \Leftarrow fOut \ : CS\_r3Ld = 1$.

The interpretation of the statement $fLin \Leftarrow r1\_out \ : CS\_M1_1 = 0 \ \wedge \ CS\_M1_0 = 0$ is as follows: "if $CS\_M1_1 = 0$ and $CS\_M1_0 = 0$, then the micro-operation $fLin \Leftarrow r1\_out$ occurs in the data-path". Other statements are interpreted likewise.

The set of all micro-operations $\mathcal{M}$ possible in the data-

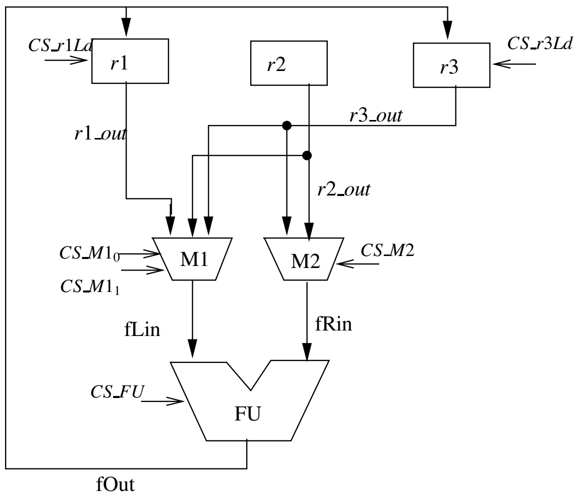

Figure 3. Data-path with control signals

path of figure 3 is as follows:

$\mathscr{M} = \{r1\_out \Leftarrow r1,\ r2\_out \Leftarrow r2,\ r3\_out \Leftarrow r3,$
$fLin \Leftarrow r1\_out,\ fLin \Leftarrow r2\_out,\ fLin \Leftarrow r3\_out,$
$fRin \Leftarrow r2\_out,\ fRin \Leftarrow r3\_out,$
$fOut \Leftarrow fLin + fRin,\ fOut \Leftarrow fLin - fRin,$
$r1 \Leftarrow fOut,\ r3 \Leftarrow fOut\}.$

Let the order of the control signals in a control signal assertion pattern be $CS\_M1_1 \prec CS\_M1_0 \prec CS\_M2 \prec CS\_FU \prec CS\_r1Ld \prec CS\_r3Ld$. It may be noted that the micro-operations $fLin \Leftarrow r1\_out$ and $fLin \Leftarrow r2\_out$ depend on more than one control signal (on both CS_M1_1 and CS_M1_0).

The function f_{mc} from the set of micro-operations $\mathscr{M}$ to the set of all possible control assertion patterns $\mathscr{A}$ is given in table 1. This function can be obtained from the output of any HLS tool containing the RTL behaviour of each component used in the data-path.

Micro-operations	Corresponding control assertion pattern
$r1\_out \Leftarrow r1$	$\langle X,\ X,\ X,\ X,\ X,\ X\rangle$
$r2\_out \Leftarrow r2$	$\langle X,\ X,\ X,\ X,\ X,\ X\rangle$
$r3\_out \Leftarrow r3$	$\langle X,\ X,\ X,\ X,\ X,\ X\rangle$
$fLin \Leftarrow r1\_out$	$\langle 0,\ 0,\ X,\ X,\ X,\ X\rangle$
$fLin \Leftarrow r2\_out$	$\langle 0,\ 1,\ X,\ X,\ X,\ X\rangle$
$fLin \Leftarrow r3\_out$	$\langle 1,\ X,\ X,\ X,\ X,\ X\rangle$
$fRin \Leftarrow r2\_out$	$\langle X,\ X,\ 1,\ X,\ X,\ X\rangle$
$fRin \Leftarrow r3\_out$	$\langle X,\ X,\ 0,\ X,\ X,\ X\rangle$
$fOut \Leftarrow fLin + fRin$	$\langle X,\ X,\ X,\ 0,\ X,\ X\rangle$
$fOut \Leftarrow fLin - fRin$	$\langle X,\ X,\ X,\ 1,\ X,\ X\rangle$
$r1 \Leftarrow fOut$	$\langle X,\ X,\ X,\ X,\ 1,\ X\rangle$
$r3 \Leftarrow fOut$	$\langle X,\ X,\ X,\ X,\ X,\ 1\rangle$

Table 1. The function f_{mc} from the set $\mathscr{M}$ to the set $\mathscr{A}$. The order of the control signals is $CS\_M1_1 \prec CS\_M1_0 \prec CS\_M2 \prec CS\_FU \prec CS\_r1Ld \prec CS\_r3Ld$

Let $A = \langle 1,\ 0,\ 1,\ 1,\ 1,\ 0\rangle$ be the control assertion pattern in a particular state of the controller FSM. The set of micro-operations $\mathscr{M}_A$ for this control assertion pattern A is determined by superposition of the control assertion pattern of each micro-operation and the pattern A is checked one by one to decide whether to include that particular micro-operation in $\mathscr{M}_A$ or not. In particular, $\mathscr{M}_A = \{\ r1\_out \Leftarrow r1,\ r2\_out \Leftarrow r2,\ r3\_out \Leftarrow r3,\ fLin \Leftarrow r3\_out,\ fRin \Leftarrow r2\_out,\ fOut \Leftarrow fLin - fRin,\ r1 \Leftarrow fOut\}$ for this example.

The micro-operation in which a register occurs in the left hand side is $r1 \Leftarrow fOut$. The sequence of rewriting steps for this micro-operation is as follows:

$r1 \Leftarrow fOut$
$\Leftarrow fLin - fRin$ [by the micro-opn. $fOut \Leftarrow fLin - fRin$]
$\Leftarrow r3\_out - fRin$ [by the micro-opn. $fLin \Leftarrow r3\_out$]
$\Leftarrow r3\_out - r2\_out$ [by the micro-opn. $fRin \Leftarrow r2\_out$]
$\Leftarrow r3 - r2\_out$ [by the micro-opn. $r3\_out \Leftarrow r3$]
$\Leftarrow r3 - r2$ [by the micro-opn. $r2\_out \Leftarrow r2$]

So, the RT-operation $r1 \Leftarrow r3 - r2$ is executed by the given control assertion pattern A of a state of the FSM and the forward spatial sequence of the micro-operations for this RT-operation is the reverse order in which they are used in the above rewriting steps; more specifically, therefore, the forward sequence is $r2\_out \Leftarrow r2$, $r3\_out \Leftarrow r3$, $fRin \Leftarrow r2\_out$, $fLin \Leftarrow r3\_out$, $fOut \Leftarrow fLin - fRin$, $r1 \Leftarrow fOut$.

The RT-operations for all other states of the FSM can be found out in a similar manner.

4 Verification During Construction of FSMD M_2

Several inconsistencies in the data-path interconnections and in the control signal assertion patterns that can be detected during construction of the FSMD M_2. are as follows.

- *Inadequate set of micro-operations performed by a control assertion pattern:* It occurs when no micro-operation is selected by the rewriting rule in a certain step of the rewriting process before the terminating condition (that is, all the terms in the rhs expression are registers) is reached. This situation arises due to either of following two reasons: (i) interconnection between two data-path components is not actually set by the control pattern but is required to complete an RT-operation and (ii) the control signals are asserted in a wrong manner which leads to a situation where the required data transfer is not possible in the data-path.

- *Data conflict:* It occurs when more than one replacement are found for a non-register term in any step of the rewriting process. It means that more than one data from different data-path components try to pass through a single data-path component which obviously causes a data conflict. It arises due to wrong control assertion pattern.

- One non-register data-path component can be assigned by only one value in a particular control step. If a non-register term is rewritten twice during the rewriting process, then it implies an improper control assertion pattern resulting in setting up a loop in the data-path without having any register.

5 Verification by Equivalence Checking

In the data-path and controller generation phase, the behaviour represented by the FSMD M_1 is mapped to hardware. The number of states and the control structure of the behaviour are not modified in this phase. Hence, there is a one-to-one correspondence between the states of FSMDs M_1 and M_2. Let the mapping between the states of M_1 and those of M_2 be represented by a function $f_{12} : Q_1 \leftrightarrow Q_2$. The state q_{2i} $(\in Q_2)$ of the FSMD M_2 is said to be the corresponding state of q_{1i} $(\in Q_1)$ if $f_{12}(q_{1i}) = q_{2i}$.

A set of RT-operations are formed for each state transition of the FSMD M_2 from the corresponding control assertion pattern. Now, the question is whether all the RT-operations corresponding to each state transition of the FSMD M_1 is captured by the controller or not. In other words, it is required to verify that all the RT-operations in each state transition in FSMD M_1 are also present in the corresponding state transition in FSMD M_2 and no extra RT-operation occurs in the transition of the FSMD M_2. A transition $q_{2k} \xrightarrow[c']{} q_{2l}$ of the FSMD M_2 is said to be the corresponding transition of a transition $q_{1k} \xrightarrow[c]{} q_{1l}$ of the FSMD M_1 if $f_{12}(q_{1k}) = q_{2k}$, $f_{12}(q_{1l}) = q_{2l}$ and the condition c is equivalent to the condition c'. In the following, a *state based equivalence checking algorithm* is given as algorithm 1 to establish the equivalence between the FSMDs M_1 and M_2.

The successful completion of the algorithm ensures that *any RT-operation occurs in any state transition of M_2 iff it also occurs in the corresponding transition in M_1*. It assures that the control signal generation in each state is correct. The number of iterations of the algorithm mainly depends on the number of transitions in the FSMD M_1. If the number of transitions in M_1 is e and the maximum possible RT-operations in any transition is k, then the complexity of the algorithm is $O(ek)$.

6 Experimental Results

Our proposed verification mechanism has been implemented in ‘C’ and integrated with an existing high-level synthesis tool, SAST [7]. The tool has been run on an Intel Pentium 4, 1.70 GHz, 256MB RAM machine on the outputs generated by SAST for several HLS benchmarks as shown in figure 4. The number of micro-operations possible in the data-path, the number of control signals used to control the micro-operations in the data-path, the number of RT-operations constructed and the number of states in each FSMD for each benchmark problem have been shown as bars in this figure. It might be noted that the number of control signals for each benchmark is less than the number of micro-operations in the data-path. It means that SAST optimizes the number of control signals required to control the micro-operations in the data-path and micro-operations in the data-path do not depend on individual control signals. Our proposed method can successfully find the RT-operations for this case. Furthermore, the number of RT-operations constructed from the control assertion patterns is much higher than the number of states in the FSMDs. It indicates that more than one RT-operation are executed in the data-path for a control assertion pattern. Again, our method successfully finds the RT-operations from a given control assertion pattern. It is also observed during experimentation that the time required to construct the FSMDs and the execution time of the equivalence checking algorithm is not very high and less than a second for most of the cases.

Algorithm 1 State based equivalence checking algorithm

```
Input:  FSMD $M_1$, $M_2$ and the function $f_{12}$.
Output: 'yes/no' answer for "$M_1$ is equivalent to $M_2$".
 1: for each state $q_{1i}$ of $M_1$ do
 2:   Let $q_{2i}$ be $f_{12}(q_{1i})$;  /* $q_{2i}$ is the corresponding state of $q_{1i}$ */
 3:   for each state transition $t$ from $q_{1i}$ do
 4:     Find a transition $t'$ from $q_{2i}$ which has an equivalent condition to that of $t$;
 5:     for each RT-operation $opn$ in $t$ do
 6:       if $opn$ does not occur in $t'$ then
 7:         Report "$opn$ is not present in transition $t'$; hence not equivalent"; exit;
 8:       end if
 9:     end for
10:     if there is some RT-operation which occurs in $t'$ but not in $t$ then
11:       Report "extra RT-operation occurs in $t'$; hence not equivalent" exit;
12:     end if
13:   end for
14: end for
```

7 Conclusions

Advances in VLSI technology have enabled its deployment into complex circuits. Synthesis flow of such circuits comprises various phases where each phase performs the task algorithmically providing for ingenious interventions

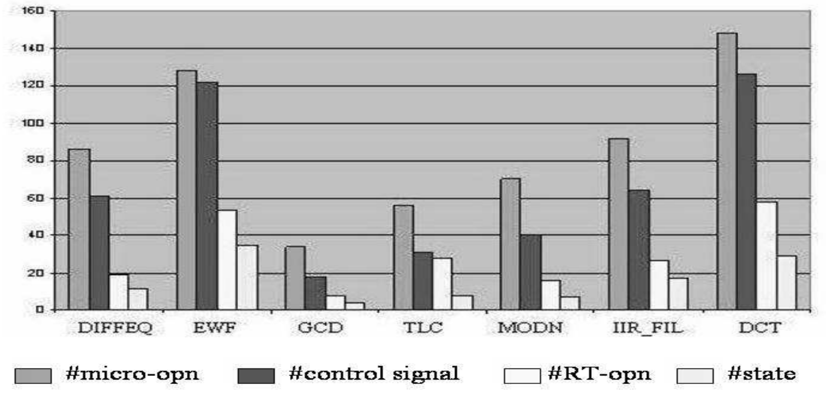

Figure 4. Experimental results of SAST tool for several high-level synthesis benchmarks

of experts. The gap between the original behaviour and the finally synthesized circuit is too wide to be analyzed by any reasoning mechanism. The validation tasks, therefore, must be planned to go hand in hand with each phase of synthesis. The present work concerns itself with the validation of the data-path and controller generation phase of high-level synthesis.

The verification task is performed in two steps. In the first step, an FSMD M_2 is constructed from the data-path information and the controller FSM. In the second step, a state based equivalence checking methodology is used to verify the correctness of the controller behaviour. A *rewriting method* is discussed which is used during the construction of the FSMD M_2; the method finds the RT-operations performed by a given control assertion pattern in each state of the controller FSM. Several inconsistencies both in the data-path and the controller, are revealed during construction of the FSMD M_2. The state based equivalence checking method ensures that any RT-operation occurs in a state transition of M_2 iff it also occurs in the corresponding transition in the FSMD M_1. Experimental results on several HLS benchmarks demonstrates the effectiveness of the proposed method.

References

[1] P. Ashar, S. Bhattacharya, A. Raghunathan, and A. Mukaiyama. Verification of rtl generated from scheduled behavior in a high-level synthesis flow. In *Proceedings of the 1998 IEEE/ACM international conference on computer-aided design*, pages 517–524, New York, NY, USA, 1998. ACM Press.

[2] D. Borrione, J. Dushina, and L. Pierre. A compositional model for the functional verification of high-level synthesis results. *IEEE Transactions on VLSI Systems*, 8(5):526–530, October 2000.

[3] H. Eveking, H. Hinrichsen, and G. Ritter. Automatic verification of scheduling results in high-level synthesis. In *Proc. Conf. Design, Automation and Test in Europe 1999*, pages 59–64, March 1999.

[4] M. Fujita. Equivalence checking between behavioral and rtl descriptions with virtual controllers and datapaths. *ACM Trans. Des. Autom. Electron. Syst.*, 10(4):610–626, 2005.

[5] D. Gajski and L. Ramachandran. Introduction to high-level synthesis. *IEEE transactions on Design and Test of Computers*, pages 44–54, 1994.

[6] C. Karfa, C. Mandal, D. Sarkar, S. Pentakota, and C. Reade. A formal verification method of scheduling in high-level synthesis. In *7th International Symposium on Quality Electronic Design, 2006. ISQED '06*, pages 71–78, March 2006.

[7] C. Karfa, J. Reddy, C. R. Mandal, D. Sarkar, and S. Biswas. Sast: An interconnection aware high-level synthesis tool. In *Proc. 9th VLSI Design and Test Symposium, Bangalore*, pages 285–292, Aug. 2005.

[8] C. Karfa, D. Sarkar, C. Mandal, and C. Reade. Register sharing verification during data-path synthesis. In *ICCTA '07.: Proceedings of the International Conference on Computing: Theory and Applications*, pages 135–140, 2007.

[9] Y. Kim, S. Kopuri, and N. Mansouri. Automated formal verification of scheduling process using finite state machine with datapath (FSMD). In *5th International Symposium on Quality Electronic Design (ISQED'04)*, pages 110–115, Carlifornia, March 2004.

[10] R. Radhakrishnan, E. Teica, and R. Vermuri. An approach to high-level synthesis system validation using formally verified transformations. In *HLDVT '00: Proceedings of the IEEE International High-Level Validation and Test Workshop (HLDVT'00)*, page 80, Washington, DC, USA, 2000. IEEE Computer Society.

[11] S. P. Rajan. Correctness of transformations in high level synthesis. In *CHDL '95: 12th Conference on Computer Hardware Description Languages and their Applications*, pages 597–603, Chiba, Japan, 1995.