Register Transfer Operation Analysis during Data Path Verification

D. Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
India
ds@cse.iitkgp.ernet.in

Abstract

A control part – data path partition based sequential circuit verification scheme aimed at avoiding state explosion comprises two major modules namely, a data path verifier and a control part verifier. The functional specifications of these modules have been identified. Of the two broad tasks involved in data path verification, namely status condition analysis and register transfer operation analysis, a method for the second task along with its termination, soundness and completeness have been treated rigorously. Its performance on some data path architectures has been reported.

Keywords Sequential Circuit Verification, Control Part - Data Path, Data Path Verification, RTL Behaviours.

1 Introduction

State explosion poses a serious problem in sequential circuit verification, irrespective of which method is followed [1], [6]. The designer tackles this problem by resorting to a control part - data path partitioning of the circuit in which all the data storage registers and data transformation / status detection circuits are put in the data path (DP) and the sequential aspects of the behaviour are taken care of in the control part (CP) (Figure 1).

A verifier designed to exploit this existing partition in the circuit structure can avoid the state explosion problem. In such a scheme, the verifier can be organized as comprising two broad modules: (i) a DP Verifier and (ii) a CP Verifier. The DP verifier, like a conventional one, would produce a "yes/no" correctness answer depending upon whether or not, the data path structure supports the register transfer operations and the status checking operations contained in the RTL behavioural description; in addition, if the answer is "yes", then it would extract a behavioural specification of the control part involving the control output lines and the status input lines of the control part. The CP verifier, in turn, would take as one input this behaviour produced by

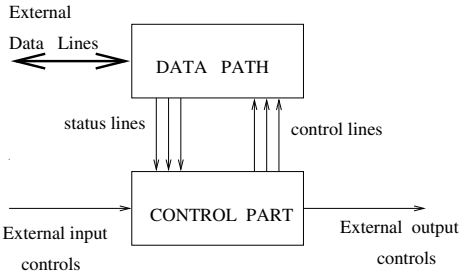

Figure 1. CP-DP Partition in the Circuit Structure

the DP verifier and the structural interconnection description of the CP components, their behaviour and the initial state as three other inputs and produce the final "yes/no" correctness answer. Figure 2 gives the schematic of a CP-DP partition based verifier. It may be noted that there is a one-to-one correspondence between the RTL behaviour and the control level behaviour; the control flow schema of the descriptions remains the same; the only difference is that the register status checking phrases are to be changed to bit level status checking conditional expressions (ST Analysis [8]) and the register assignment statements of the former are to be changed to control signal assignment statements (RT Analysis). Since the DP structure does not have any temporal characteristic, the DP verification mechanism involves analysis of only the spatial interconnection of the components. The CP verification, on the other hand, consists in temporal reasoning in a suitably chosen temporal logic framework [2]. Figure 3 depicts the two modules, the RT Analyzer and the ST Analyzer, of the DP Verifier. In this paper the RT analysis task of DP verification has been described.

There have been attempts to exploit during verification the control part-data path partition in sequential circuits. Word Level Model Checking using Multi-Terminal BDDs, BDD arrays, BMDs, etc. for data path representa-

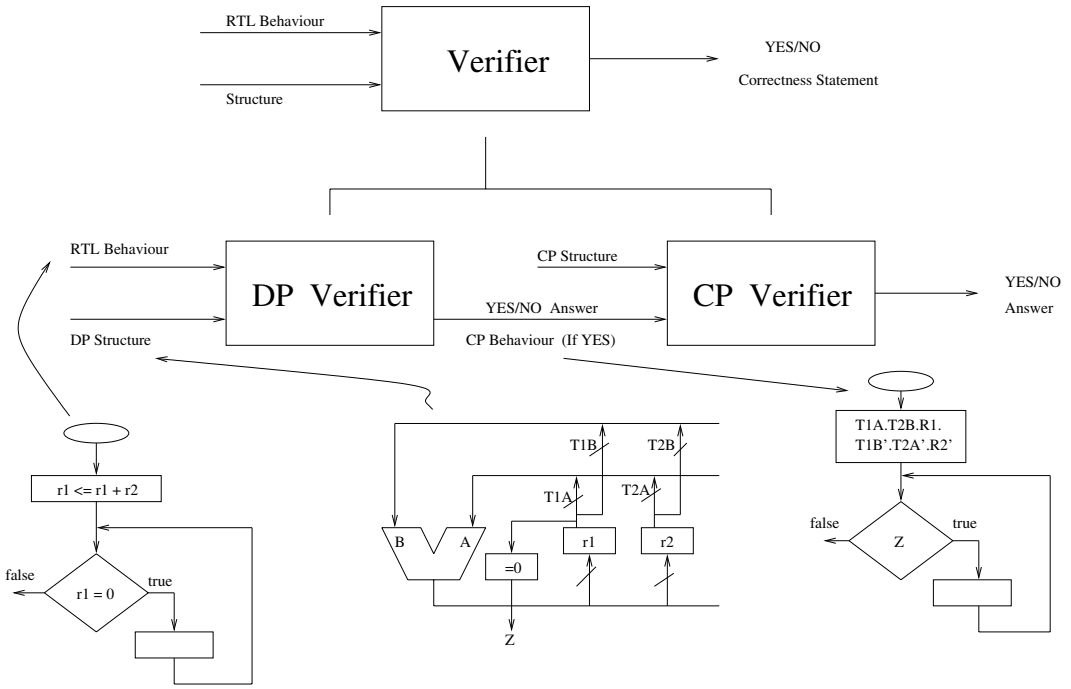

Figure 2. CP-DP Partition based Verifier

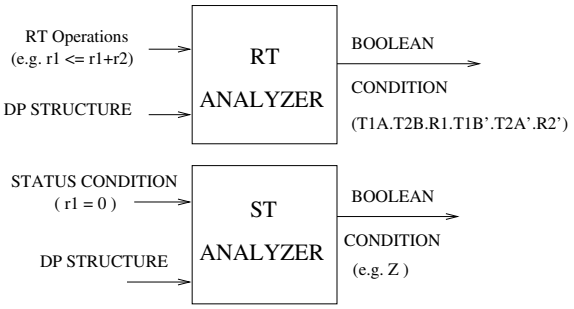

Figure 3. Two Submodules of the DP Verifier

tion have been proposed as solutions [10]. Theorem Proving approaches with predicate based data path representations have also been reported [4, 5, 7, 9, 10]. Various issues pertaining to automation and completeness, however, have not been elaborately described. In many cases, the inputs have been so encoded as to keep the primitive inference steps simpler. In the present work, the emphasis is precisely on identifying the issues in reaching a complete analyzer.

Section 2 gives a logical basis of a CP-DP partition based verifier. Section 3 describes the task of RT analysis in data path verification. The termination, soundness and completeness issues are formally established. A complexity analysis of the basic algorithm along with an extension for concurrent RT operation analysis are also included. In section 4, experimental results on verification of some typical data paths against some single and concurrent RT operations have been reported. The paper concludes in section 5 by identifying some limitations of the method.

2 Logical Basis of CP-DP Partition Based Verification

Let I be the implementation of the entire circuit, S be the behavioural specification and I_1 and I_2 be the implementation of the data path and the control part, respectively.

Obviously, $I = I_1 \wedge I_2$. ... (i)
Recall that the aim of the verifier is to prove $I \supset S$. ... (ii)
The DP verification consists in finding a specification S_1 for the control part such that $I_1 \supset (S_1 \supset S)$. ... (iii)
The CP verification consists in proving that $I_2 \supset S_1$. ...(iv)
Therefore. once DP verification is accomplished, we have $I_1 \wedge I_2 \supset (S_1 \supset S)$ ... (v) [by introduction of assumption to (iii)].
Once CP verification is accomplished, we have,
$I_1 \wedge I_2 \supset S_1$ ... (vi) [introduction of assumption I_1 in (iv)].
Hence, $(I_1 \wedge I_2) \supset S$ or, $I \supset S$ [modus ponens on (v) and (vi)].

3 RT-Analysis

The analysis method described in this section uses the following notation.

3.1 Notation

A: A control signal assertion pattern of the form $\{< c_i, v_i >, 1 <= i <= n\}$, where v_i is the value '0','1' or 'X' (unasserted) of signal c_i.
π_i(A): The i-th projection of A, that is, v_i.
p_b: The given RT operation from the RTL behaviour,
M: The set of all possible data path micro-operations.
p_c: The current RT-expression obtained from the original one (p_b) through a series of rewrite steps. Initially, $p_c = p_b$.

3.2 Phases of RT Analysis

The DP structure and the DP component behaviours together produce a unary function $R_s : M \rightarrow \mathcal{A}$. For RT analysis, R_s captures the DP structure, in its entirety. The RT analysis problem consists of following two subproblems. (i) All possible sequences of micro-operations which accomplish p_b are to be selected from M, and (ii)Corresponding to each sequence selected in (i), it is required to construct, using the function R_s, the control signals that are to be asserted. The second task is relatively simpler and is described first; then the first one is presented.

3.2.1 Construction of the Control Signal Assertion List for a Given Micro-operation Sequence

If, for an RT operation p_b, the concurrent micro-operations that are found to be necessary are μ_1, μ_2 and μ_3, then the control signal assertion pattern A for p_b is found from $R_s(\mu_1)$, $R_s(\mu_2)$ and $R_s(\mu_3)$ by an associative binary operation θ, called *superposition operation*. The operation is defined as follows.

Definition 1 *Superposition of Assertion Patterns: Let A_1 and A_2 be two arbitrary control signal assertion patterns. The assertion pattern $A_1 \theta A_2$ obtained by* superposition of A_1 and A_2 *satisfies the following conditions. For all i, $\pi_i(A_1\ \theta\ A_2)$*
= π_i (A_1), if $\pi_i(A_2)$ = 'X'
= π_i (A_2), if $\pi_i(A_1)$ = 'X'
= π_i (A_1), if $\pi_i(A_1) = \pi_i(A_2) \neq$ 'X'
= undefined, if $\pi_i(A_1) \neq \pi_i(A_2) \neq$ 'X'

3.2.2 Finding all the Micro-operation Sequences that accomplish p_b

In this section the problem addressed is as follows. Given p_b and M, it is required to find from M *all those sequences* of micro-operations which accomplish p_b. We make the assumption that there is only one ALU operation involved in p_b. The micro-operations needed for accomplishing p_b are identified depending upon whether they are capable of *rewriting some left hand side (lhs) signals* in p_c. As p_c is rewritten under certain restrictions the spatial sequence of data flow gets detected simultaneously. Thus, both identifying the micro-operations needed for p_b and ordering them get solved in a single stroke. If a micro-operation m be chosen as the rewrite rule, then *all* the occurrences of its lhs signal d in p_c are rewritten using the rhs expression e of m; also, only the lhs occurrences of signals in p_c are replaced. The process terminates successfully when the lhs of p_c becomes syntactically identical to the rhs. The process is first illustrated through the following example and then formalized.

Example 1 Let p_b be \$dst $\Leftarrow$ \$src1 + \$src2. ('\$' symbol before a register name represents the register content.) Let the relevant micro-operations in M be : {\$dst $\Leftarrow$ res, bus1 $\Leftarrow$ \$src1, bus2 $\Leftarrow$ \$src2, res $\Leftarrow$ bus1 + bus2 }. That is, the data path contains three buses, "res" (the result bus) and "bus1", "bus2" for the operand buses. Each of these micro-operations is used as a rewrite rule to rewrite occurrences of signals in p_b. The sequence of transformations of p_b is as follows.
\$dst $\Leftarrow$ \$src1 + \$src2 [The original p_b]
res $\Leftarrow$ \$src1 + \$src2 [by the mciro-op: \$dst $\Leftarrow$ res]
bus1 + bus2 $\Leftarrow$ \$src1 + \$src2 [by the mciro-op: res $\Leftarrow$ bus1 + bus2]
\$src1 + bus2 $\Leftarrow$ \$src1 + \$src2 [by the mciro-op: bus1 $\Leftarrow$ \$src1]
\$src1 + \$src2 $\Leftarrow$ \$src1 + \$src2 [by the mciro-op: bus2 $\Leftarrow$ \$src2]
The process terminates here and the sequence of micro-operations needed to route the data from the source(s) to the destination is available in reverse order in which they have been used.

For each register transfer expression, there may be more than one micro-operation applicable. At each step, the results of rewriting using these micro-operations have to be collected for subsequent rewritings. Also, the set of signals replaced and the sequence of micro-operations are to be enhanced. A tree appears to be the most obvious structure to depict the progress; it is referred to as *rewrite-tree*. Each node of the tree (data type NODE) is associated with (i) a current RT_expression p_c, (ii) a sequence of micro-operations "so_far", (iii) a set of signals called "replaced", (iv) links to "left_child", and "right_brother". The following recursive function "analyze_rt" constructs the rewrite tree in a DFS manner.

Algorithm 1 *void analyze_rt (NODE root)*
begin
$p_c \leftarrow$ root.p_c; let p_c be "$e_l \Leftarrow e_r$";
if *(e_l syntactically identical to e_r)* **return**;
else
begin

$M_s \leftarrow$ *choose_rewrite_micro_op (root, M);*
if *($M_s \neq \{loop\_det\}$ or $M_s \neq \{no\_more\}$* **then**
begin
$\forall m \in M_s$ **begin**
$n \leftarrow$ *create_node(); $n.p_c \leftarrow$ rewrite (root.p_c, m); n.so_far $\leftarrow m \|$ root.so_far;*
n.replaced $\leftarrow$ lhs(m) $\cup$ root.replaced; add_child (root, n); analyze_rt(n);
end /* $\forall m \in M_s$ */
end
else /* $M_s = \{loop\_det\}$ or $M_s = \{no\_more\}$ */
root.so_far $\leftarrow m \|$ root.so_far, where $m \in M_s$;
end /* *else – e_l not syntactically identical to e_r* */
end

The function "choose_rewrite_micro_op" chooses a subset M_s of M for rewriting the RT expression "root.p_c" (= "$e_l \Leftarrow e_r$"), based on the following logic:

1. Select for replacement the leftmost signal s from e_l which does not occur in e_r; if no such s, then $M_s \leftarrow$ {no_more}. Thus, the lhs of the original RT operation is rewritten from left to write.

2. if $s \in$ root.replaced, then $M_s \leftarrow$ {loop_det} – a signal which has already been replaced reappears indicating a loop in the data path.

3. if none of the above, then $M_s \leftarrow \{m \in M$ such that

   (a) lhs(m) = s, and

   (b) the rhs(m) does not contain any register other than the source registers (occurring in the rhs of "root.p_c") – ensures that no *source register* is disturbed for accomplishing p_b.

   (c) if rhs(m) contains an ALU-operation, then rhs (root.p_c) should also contain the same operation. $\}$

   if $M_s = \phi$ then $M_s \leftarrow$ {no_more};

3.3 Correctness of the function *analyze_rt*

Theorem 1 *(Termination) The function always terminates.*

Proof: The function always constructs a finite rewrite tree because each node has a finite number of children, as many as the drivers of the signal s being replaced, and each branch has a maximum depth equal to the number of non-register signals in the data path. ■

Theorem 2 *Let L be the set of sequences associated with the ("so_far" fields of the) leaf nodes of the rewrite tree not having "loop_det" or "no_more". p_b is accomplished by each member of L (Soundness) and only by them (Completeness).*

Proof: (Soundness) Consider any sequence $\sigma \in L$ of the form $< m_1, m_2, \cdots, m_k >$. Each m_i of the form $s_i \Leftarrow r_i$ captures a (transformed) data flow from the signals in r_i to s_i. Thus, through the application of σ (in reverse order), if e_l becomes same as e_r, then p_b is accomplished by σ.

(Completeness – by contradiction) Let p_b be of the form $s_l \Leftarrow e_r$. Let $\sigma = < m_1, m_2, \cdots, m_k >$ be not in L but its application accomplish p_b. Since σ applies on p_b, m_k must be of the form $s_l \Leftarrow r_k$. Thus, it will be put in M_s by the function and p_b will be rewritten as "$r_k \Leftarrow e_r$" = p_k say. Application of σ entails application of its last but one member m_{k-1} at the next step. Let m_{k-1} be $s_{k-1} \Leftarrow r_{k-1}$. Let $s_i \neq s_{k-1}$ be the leftmost signal of r_k which does not occur in e_r. Thus, the function chooses s_i and not s_{k-1} as the next micro-operation for rewriting. Now, there must be an m_i in σ for rewriting s_i in p_k. Moreover, *since there is just one ALU operation involved in each RT-operation being analyzed*, the same signal does not appear in more than one rewrite step. Thus, the subsequence $< m_{i+1}, \cdots, m_{k-1} >$ of rewrites (in reverse order) will not create any new occurrences of s_i. *Hence the order in which members of σ are used for rewriting is not of concern*. By a simple inductive extension of this piece of argument, therefore, it follows that the function, because of "leftmost-only" replacement strategy, applies a rewrite sequence which is only a permutation σ' of the sequence σ. Thus, if σ is not in any leaf, then σ' is. ■

We have the following corrollary of the above theorem whose proof is obvious.

Corollary 1 *If the set of sequences of micro-operations returned by "analyze_rt" function be ϕ, then the RT-operation cannot be accomplished in the given data path.*

3.3.1 Complexity of RT-analysis

For each rewrite step, that is, for each node in the rewrite tree, there are three major tasks to be performed, namely,
(i) a suitable signal s has to be chosen for rewriting,
(ii) choosing the subset M_s of micro-operations with s on the lhs and no register signal on the rhs other than the source register(s) of p_b , and
(iii) rewriting all the occurrences of signal s in p_c.
The first step takes constant time because the assumption that there can be only one ALU in each path from the

source(s) to a destination permits at most two signals on the lhs of any RT-expression. For each of them it has to be checked whether it is a non-register signal by accessing the signal table. The table can be directly probed because the RT-operations (expressions) and the micro-operations are all encoded in terms of index values to the signal table. The complexity of the second step is as follows. The micro-operation table M is hashed in signal name. Each signal will appear at the rhs of only certain number of micro-operations maximum of which can always be specified. Therefore, finding M_s from a hashed M takes constant time. When table M is not hashed but sorted on the lhs-signal of the micro-operations, finding M_s takes O($log_2 \parallel M \parallel$) time.
Thus, the complexity of each rewriting is O(1) or O($log_2 \parallel M \parallel$) depending upon whether M is hashed or sorted in signal names, respectively.

The number of nodes of the rewrite tree, each node of which involves above complexity, depends upon the operation involved in the RT-operation p_b. Obviously, it is the maximum for binary (ALU) operation. In any case, it is always constant (for each ALU). For example, for a binary operation, the root node has as many successors as there are buses. Some of these are (ALU) input buses and the remaining are the output buses. (There can be more than one ALU in the data path.) Each node corresponding to an output bus will have one successor corresponding to the ALU operation resulting in an RT-expression involving the input buses. The input buses do not expand any more because they can only be driven by registers and the heuristic prevents such micro-operations from being used except for one of the source registers. The two ALU input buses, therefore, can be rewritten by two source registers accounting for two linear subtrees of depth two. Thus the rewrite tree for an RT-operation with a binary ALU operation will have 8 nodes. For n ALU's in the data path, there are $8n$ nodes generated. The complexity figures for RT-analysis can be summarized as follows.
For data path with n ALU's and the micro-operation table hashed, it is $O(n)$.
For data path with n ALU's and sorted micro-operation table, it is $O(n.log_2 \parallel M \parallel)$.

3.3.2 Analysis of Concurrent RT-operations

Let $P = \{p_1, p_2, \cdots, p_n\}$ be the given set of concurrent RT-operations to be analyzed to find all possible sequences of micro-operations that accomplish them. One straightforward approach for analysis could be to invoke "analyze_rt" with each member p_i of P. Let S_i be the set of sequences of micro-operations that can accomplish p_i. Let σ_{ij} be the j-th sequence in the set S_i. Let $\mathcal{T}$ represent the n-ary Cartesian Product of the sets S_i's; that is, $\mathcal{T} = X_{1 \leq i \leq n}\{S_i\}$. The final set S of sequences of micro-operations needed to accomplish P can be constructed using the following definition.
$S = \{\sigma =< \sigma_{1j_1}, \sigma_{2j_2}, \cdots, \sigma_{nj_n} > \in \mathcal{T} \mid$ for any data path signal s,
(i) no member of σ contains more than one micro-operation having s as its lhs, and
(ii) if there are more than one micro-operation occurring in different members of σ having same lhs signal s, then they are same micro-operation$\}$.
The first condition ensures that in accomplishing p_i, the same signal is not assigned more than once. (This was the termination condition for "analyze_rt" and serves to terminate the analysis of individual p_i's.) The second condition states that a micro-operation can be shared by other concurrent RT-operations. Thus, first the sets S_i's can be obtained by analyzing p_i's in isolation. The set S can then be constructed by constructing ordered n-tuples of sequences from S_i's ensuring that no two sequences contain micro-operations which assign to the same signal differently.

4 Experiments

The DP-structures given in Figure 4 are analyzed for both single RT-operations and concurrent RT-operations; the performances of the analysis functions are given in Table 1 and Table 2; 'P/N' indicates possible/not possible, N_1, N_2 stand for the number of rewrite-tree nodes and that of the concurrent tree nodes generated. Real-life circuits such as, Bit-wise-shift-and-add multiplier, Booth's Multiplier, Divider and Tamarack Processor [3] have also been analyzed for all RT-operations in their RTL behavioural descriptions. Some faulty/inadequate data paths vis-a-vis the given RT-operations have been tested. It has been found that in many cases, the analysis points to the faults such as, "intermediary register(s) in the data path" (indicating non-realizability of the RT operation in one time step), "loop detected in the data path", etc.

5 Conclusion

A CP-DP partition based verification scheme has been proposed and validated. The DP verification problem has been discussed in detail. The task comprises two major subtasks namely, the status checking analysis (ST Analysis) and the register transfer operation analysis (RT Analysis). An RT-analysis algorithm has been presented and its termination, soundness and completeness have been treated rigorously. Complexity of the algorithm has been found to be at worst O(n) or O($n.log_2 \parallel M \parallel$), where n is the number of ALU's and M is the set of micro-operations supported by the data path. A simple extension for concurrent RT-operations has been discussed. The method has been tested

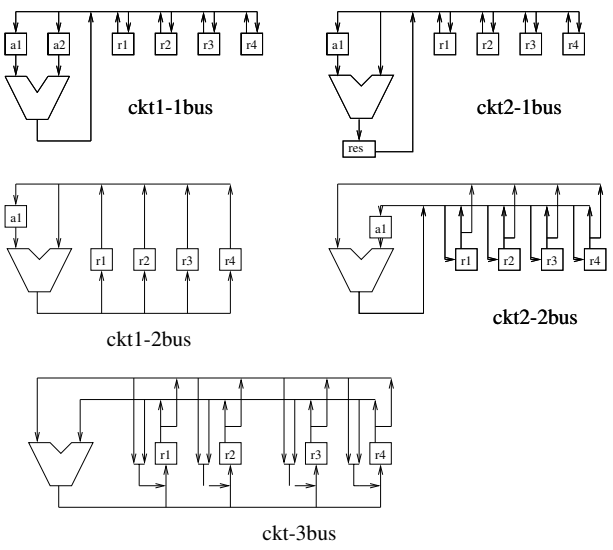

Figure 4. Example Data Paths for RT Analysis

Ckt_name	RT-op fed for analysis	P / N	N_1
ckt1_1bus	r1 ← a1 + a2	P	2
	r1 ← r1 + r2	N	3
ckt2_1bus	res ← r1 + a1	P	2
	r1 ← r1 + r2	N	3
ckt1_2bus	r1 ← r2 + a1	P	3
	r1 ← r1 + r2	N	8
ckt2_2bus	r1 ← r1 + a1	P	3
	r1 ← r1 + r2	N	4
ckt_3bus	r1 ← r1 + r2	P	8
	r1 ← r1 + r1	P	4

Table 1. Performance of "analyze_rt"

Ckt_name	Concurrent RT-op fed for analysis	P / N	N_2
ckt1_1bus	{a1 ← r1, a2 ← r1}	P	4
	{a1 ← r1, a2 ← r1, r1 ← r1 + r1}	N	6
ckt2_1bus	{res ← r1 + a1, r2 ← r1}	P	4
	{res ← res + a1, r1 ← res}	P	4
	{res ← res + a1, r2 ← r1}	N	3
ckt1_2bus	{r1 ← r1 + a1, a1 ← r1}	P	5
	{r1 ← r1 + a1, r2 ← r1}	N	4
	{r1 ← r1 + a1, r2 ← a1}	N	4
ckt2_2bus	{r1 ← r1 + a1, a1 ← r1}	N	5
ckt_3bus	{r1 ← r1 + r2, r3 ← r2, r4 ← r1}	P	30
	{r1 ← r1 + r2, r4 ← r1 + r2}	P	16
	{r1 ← r1 + r2, r4 ← r1 + r3}	N	14

Table 2. Performance of "analyze_concur_rt"

on two 1-bus, two 2-bus and a 3-bus data path architectures for both valid and non-valid RT-operations.

The method has also been successfully applied to arithmetic circuits such as, bit-wise shift and add multiplier, Booth's multiplier, divider, etc. In fact, the data paths of these problems, having no bus, have been found to be simpler than those of Figure 4. Although the TAMARACK CPU [3] has been verified using this method, for a full fledged CPU, the instruction set specification involves RT operations which may need more than one time step. More sophisticated analysis is, therefore, needed for the purpose. The present analyzer enhanced in this direction is also likely to be useful for synthesis of behaviours higher than RTL. Again, a CP-DP partition based approach is likely to face hurdles for many circuits where the partition is not there or not easily discernible. A pipelined architecture is a case in point. It will be interesting to examine whether the method can be enhanced for such cases .

References

[1] M. C. Browne, E. M. Clarke, D. L. Dill, and B. Mishra. Automatic Verification of Sequential Circuits Using Temporal Logic. *IEEE Transactions on Computers*, 35(12):1035–1044, December 1986.

[2] M. Hira and D. Sarkar. Verification of Tempura Specification of Sequential Circuits. *IEEE Tr. on CAD of ICS*, 16(4):362–375, 1997.

[3] J. J. Joyce. *Multi-Level Verification of Microprocessor-Based Systems*. PhD thesis, University of Cambridge, May 1990.

[4] D. Kapur and M. Subramaniam. Mechanizing Verification of Arithmetic Circuits: SRT Division. In *Proceedings 17th Conf FST and TCS, LNCS 1346*, pages 103–122, 1997.

[5] M. Langevin. Automated RTL Verification Based on Predicate Calculus. In *Proc. 2nd. Intl Conf Computer-Aided Verfication*, pages 116–125, 1990.

[6] S. P. Miller and M. Srivas. Formal Verification of the AAMP5 Microprocessor: A case study in the Industrial Use of Formal Methods. In *Proc. Workshop on Indl.-Strength Formal Specification Techniques (WIFT'95)*, Boca Raton, Florida, April 1995.

[7] H. Nakamura et. al. A Data Path Verifier for Register Transfer Level Using Temporal Logic Language Tokio. In *Proc. 2nd Intl. Conf. Computer Aided Verification, LNCS 531*, pages 76–85, 1990.

[8] D. Sarkar. Status Condition Analysis during Data Path Verification of Sequential Circuits. In *Proc. the 13th Intl. Conf.on VlSI Design, India.*, pages 70–75, 2000.

[9] K. Schneider et. al. Control Path Oriented Verification of Sequential Generic Circuits with Control and Data Path. In *Proc. European Design and Test Conf.*, pages 648–652, 1994.

[10] X. Zhao. *Verification of Arithmetic Circuits*. PhD thesis, School of Computer Sc., Carnegie-Mellon University, 1996.