Condition Graphs for High-Quality Behavioral Synthesis

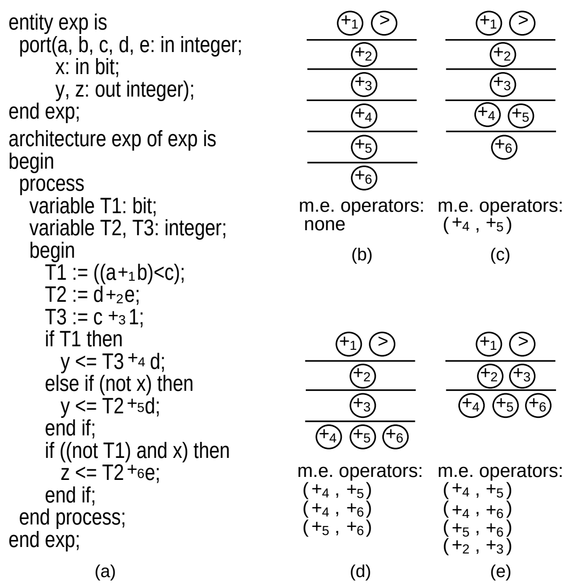
Hsiao-ping Juan, Viraphol Chaiyakul and Daniel D. Gajski
Department of Information and Computer Science
University of California, Irvine, CA 92717-3425

Abstract

Identifying mutual exclusiveness between operators during behavioral synthesis is important in order to reduce the required number of control steps or hardware resources. To improve the quality of the synthesis result, we propose a representation, the Condition Graph*, and an algorithm for identification of mutually exclusive operators. Previous research efforts have concentrated on identifying mutual exclusiveness by examining language constructs such as IF-THEN-ELSE statements. Thus, their results heavily depend on the description styles. The proposed approach can produce results independent of description styles and identify more mutually exclusive operators than any previous approaches. The Condition Graph and the proposed algorithm can be used in any scheduling or binding algorithms. Experimental results on several benchmarks have shown the efficiency of the proposed representation and algorithm.*

1 Introduction

High-level synthesis is a process of producing a register-transfer-level design from a given abstract behavioral description. In general, the major tasks of this process include scheduling the operators from the given behavioral description into control steps and binding the scheduled operators to appropriate resources. For example, two operators in a behavioral description may be scheduled into the same control step but executed by different resources, or they may be executed by the same resource but in different control steps. **However, if we can identify that the results of these two operators will never to be used at the same time, that is, if they are mutually exclusive, then they can be scheduled into the same control step and share the same resource**. Consequently, the number of control steps or the hardware cost is reduced. For example, consider the VHDL description shown in Figure 1(a). Assuming we can not identify any mutually exclusive operators, then simple List Scheduling [4] with a hardware resource constraint of 1 adder and 1 comparator produces a design with 6 control steps as shown in Figure 1(b). However, if we can identify that $+_4$ and $+_5$ are mutually exclusive because they are used in different conditional branches, we can schedule $+_4$ and $+_5$ to the same control step. Thus, one control step is reduced as shown in Figure 1(c). The number of control steps can be further reduced if we can identify more mutually exclusive operators as shown in Figure 1(d) and (e).



```
entity exp is
  port(a, b, c, d, e: in integer;
       x: in bit;
       y, z: out integer);
end exp;
architecture exp of exp is
begin
  process
    variable T1: bit;
    variable T2, T3: integer;
    begin
      T1 := ((a+1b)<c);
      T2 := d+2e;
      T3 := c +3 1;
      if T1 then
        y <= T3 +4 d;
      else if (not x) then
        y <= T2 +5d;
      end if;
      if ((not T1) and x) then
        z <= T2 +6e;
      end if;
  end process;
end exp;
```

(a)
m.e. operators: none
(b)
m.e. operators: (+4 , +5)
(c)
m.e. operators: (+4 , +5) (+4 , +6) (+5 , +6)
(d)
m.e. operators: (+4 , +5) (+4 , +6) (+5 , +6) (+2 , +3)
(e)

Figure 1: An example of behavioral descriptions.

Mutual exclusiveness between operators can be determined by analyzing the input descriptions. Sometimes mutually exclusive operators are obvious from the use of language constructs (such as IF-THEN-ELSE), while others need a sophisticated data-flow analysis. For instance, operators $+_4$ and $+_5$ in Figure 1 are mutually exclusive because they are in different branches of the same IF-THEN-ELSE statement. On the other hand, a data flow analysis is needed to determine that $T2$ is not used when the condition $T1$ is $TRUE$ and $T3$ is used only if $T1$ is $TRUE$. This indicates that $+_2$ and $+_3$ are mutually exclusive. All possible pairs of mutually exclusive operators in example Figure 1(a) are shown in the left most column of Figure 2.

Previous research has addressed the issue of identifying mutually exclusive operators to improve scheduling results. Kim and Liu [7] proposed an algorithm which can identify mutually exclusive operators that are obvious from the use of language constructs. Basically, only operators in different branches of the same IF or CASE statement are identified. Wakabayashi and Yoshimura [8] used *condition vectors* to identify mutually exclusive operations. Their approach can identify the mutual exclusiveness among op-

© 1994 ACM 0-89791-690-5/94/0011/0170 $3.50

erators in conditional branches. In addition, a data-flow analysis is performed on each of the conditional branches. However, they cannot identify mutually exclusive operations across conditional blocks. Path-based scheduling algorithm [2] determined the conditional usage of operators by analyzing every execution path in the control-flow graph. Operators are mutually exclusive if they do not appear in the same path. Furthermore, a false-path analysis [1] can identify mutual exclusiveness among operators in the same path. However, no data-flow analysis is performed. Figure 2 summarizes the result of applying previously known approaches to the example in Figure 1.

mutually exclusive operators	approaches		
	Kim's	Wakabayashi's	Path−based
$+_4$, $+_5$	✓	✓	✓
$+_3$, $+_5$		✓	
$+_3$, $+_6$		✓	
$+_2$, $+_4$		✓	
$+_4$, $+_6$			✓
$+_5$, $+_6$			✓
$+_2$, $+_3$			

✓ : identified

Figure 2: The result of applying previous approaches to identify mutually exclusive operators in Figure 1.

One simple solution to overcome the limitations in previous approaches is to force the users to write descriptions using language constructs and description styles that can be recognized by the mutual exclusiveness identification algorithm used in the synthesis system. This solution is impractical because the users would need to acquire detailed knowledge of the algorithms used in the system.

In this paper, we propose a new approach which can identify mutually exclusive operators in a behavioral description without resorting to language constructs or styles. Furthermore, unlike the previous approaches, we propose the separation of the mutual exclusiveness identification from the scheduling. This separation results in a less complex algorithm which out-performs previous approaches in the identification of mutual exclusiveness. Additionally, the algorithm can be used by any scheduling or binding algorithms.

An overview of our approach is given in the next section. Section 3 outlines the definition and representation of the usage condition of an operator in a description. Identifying whether two operators are mutually exclusive by evaluating their usage conditions is discussed in detail in Section 4. Finally we present the results of our approach on some HLSW benchmarks.

2 Overview of Our Approach

The first step in our approach is to convert the input behavioral description into an *Assignment Decision Diagram (ADD)* representation. The ADD representation can

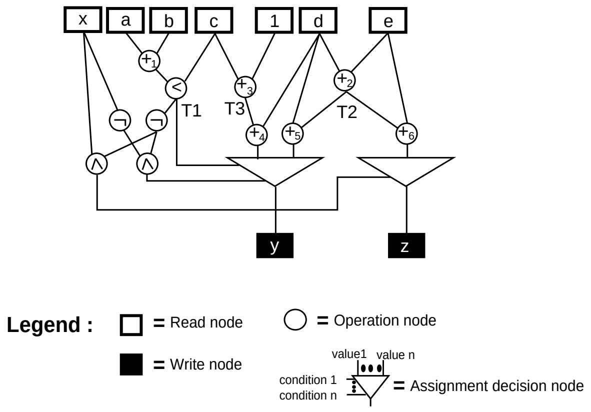


Figure 3: The ADD of the example description.

minimize the syntactic variances in the description due to ordering and grouping of conditions and assignments [3]. Figure 3 shows an example of the ADD representation which is derived from the description in Figure 1. The fundamental concept of the ADD is to represent a given description as a set of all possible conditional assignments to each output port or internal storage unit. The property of assignment values and assignment conditions is represented as a triangular *assignment decision node (ADN)* in the ADD graph. It is the unique property of the ADN that is of interest to our work. Basically, it guarantees that the assignment values to an ADN are mutually exclusive.

The next step is to define and store the usage condition for each operator in the ADD. The usage condition of an operator is defined as the condition under which the result of the operator is to be used. Because the conditions for assignments are explicitly shown in an ADD, the usage conditions of operators can be easily defined in terms of assignment conditions. For example, in Figure 3, the result of $+_6$ is assigned to z only when the condition $(x \wedge \neg(a + b < c))$ is $TRUE$; therefore, the usage condition of $+_6$ is $(x \wedge \neg(a + b < c))$. The usage conditions are represented and stored using a graph representation called *Condition Graph (CG)*. The constituents and constructions of CGs for operators in an ADD will be discussed in greater detail in the next section.

After the CGs for all the operators in the ADD are constructed, the mutual exclusivity of any two operators can be determined by evaluating their CGs. Two operators are mutually exclusive if their CGs never evaluate to $TRUE$ simultaneously. For instance, $+_4$ and $+_5$ are mutually exclusive because their CGs, which represent $(a+b<c)$ and $(\neg x \wedge \neg(a + b < c))$ respectively, would never evaluate to $TRUE$ at the same time.

The conversion from a behavioral description to an ADD representation has been discussed in detail in [3]. In this paper, we shall focus on how to construct CGs for the operators in an ADD and also how to identify mutually exclusive operators by evaluating the CGs.

3 Condition Graphs

The *condition graph* (CG) is a graph used to represent a usage condition of an operator. The usage condition for an operator in an ADD can be written as an arithmetic expression, which always evaluates to either $TRUE$ or $FALSE$. The variables in a usage condition can be bit vectors or integers. The operators in a usage condition consist of three types: (1) *arithmetic operators* such as $\{+,-,\times\}$; (2) *relational operators* such as $\{<,==,>,\leq,\geq,\neq\}$; and (3) *Boolean operators* such as $\{\wedge,\vee,\neg\}$. A CG consists of two types of nodes (read nodes and operation nodes) and edges connecting the nodes. The read nodes represent the variables in the usage condition. The operation nodes represent the types of operations that are performed to compute the usage condition. Thus, a CG can be viewed as a circuit used to evaluate a usage condition and the output is the result of the evaluation.

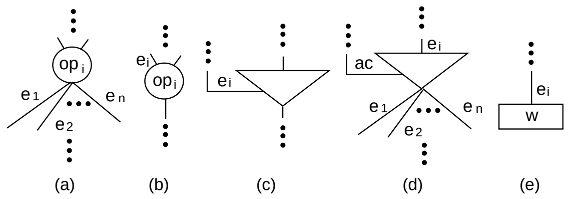


Figure 4: Defining usage conditions.

Let OC_i denote the usage condition of an operator op_i and EC_i denote the usage condition of an edge e_i. The usage condition of any operator in ADD can be defined by the following axioms:

Axiom 1 *Let $\{e_1, e_2, \cdots, e_n\}$ be the set of edges out-going from an operator op_i (Figure 4(a)), then*

$$OC_i = EC_1 \vee EC_2 \vee \cdots \vee EC_n.$$

The usage condition of an edge can be defined by either one of the following axioms according to the types of its destination:

Axiom 2 *If the destination of an edge e_i is an operator op_i (Figure 4(b)), then*

$$EC_i = OC_i.$$

Axiom 3 *If the destination of an edge e_i is an ADN, then (1) if e_i is an assignment condition edge of the ADN (Figure 4(c)), then*

$$EC_i = TRUE;$$

(2) if e_i is an assignment value edge of the ADN and its corresponding assignment condition is ac, let $\{e_1, e_2, \cdots, e_n\}$ be the set of output edges of the ADN (Figure 4(d)), then

$$EC_i = ac \wedge (EC_1 \vee EC_2 \vee \cdots \vee EC_n).$$

Axiom 4 *If the destination of an edge e_i is a write node w (Figure 4(e)), then*

$$EC_i = TRUE.$$

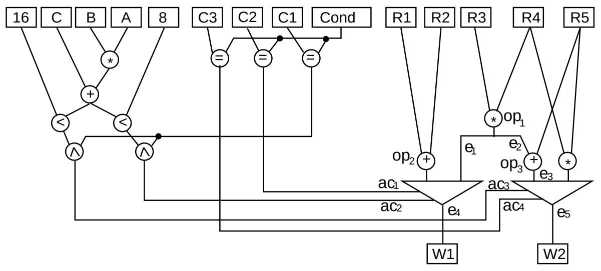


Figure 5: An example of ADD.

The usage condition of an operator can be obtained through a series of applications of the above axioms. For example, consider the operator op_1 in the ADD shown in Figure 5. According to Axiom 1, $OC_1 = EC_1 \vee EC_2$. To obtain EC_1, Axiom 3 and 4 can be applied as follows:

$$\begin{aligned} EC_1 &= ac_2 \wedge EC_4 && \textbf{(Axiom3)} \\ &= ((A*B+C<8) \wedge (Cond == C1)) \wedge TRUE. && \textbf{(Axiom4)} \end{aligned}$$

Similarly, EC_2 can be derived as follows:

$$\begin{aligned} EC_2 &= OC_3 && \textbf{(Axiom2)} \\ &= EC_3 && \textbf{(Axiom1)} \\ &= ac_3 \wedge EC_5 && \textbf{(Axiom3)} \\ &= ((16 < A*B+C) \wedge (Cond == C1)) \wedge TRUE. && \textbf{(Axiom4)} \end{aligned}$$

Therefore,

$$\begin{aligned} OC_1 = &(((A*B+C<8) \wedge (Cond == C1)) \wedge TRUE) \\ &\vee(((16 < A*B+C) \wedge (Cond == C1)) \wedge TRUE). \end{aligned}$$

The CG which represents OC_1 is shown in Figure 6.

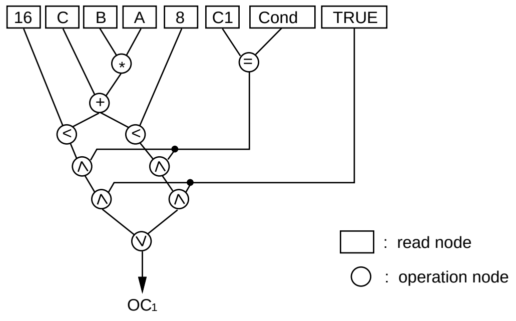


Figure 6: The CG for OC_1.

4 Identifying Mutual Exclusiveness

To identify that two operators, op_1 and op_2, are mutually exclusive we have to show that the corresponding usage conditions, OC_1 and OC_2, of the two operators will never evaluate to $TRUE$ at the same time. One simple approach is to convert OC_1 and OC_2, which could consist of arithmetic sub-expressions, into boolean equations, then prove that $\overline{OC_1 \wedge OC_2}$ is a tautology. However, such an approach is impractical since it requires exponential time and exponential space.

Thus, for practicality, we have constructed a set of lemmas and theorems for identifying sets of mutually exclusive operators that are commonly found in a behavioral description. In addition, we have developed an algorithm to apply the lemmas and theorems to the usage conditions, which are represented in CG, of any two given operators in order to determine whether or not they are mutually exclusive. The main idea of the algorithm is to "pessimistically" assume that the operators are NOT mutually exclusive UNLESS the mutual exclusion can be proved by the lemmas or theorems. This is essential for synthesis because by "pessimistically" identifying two operators as NOT mutually exclusive, even if they are in fact mutually exclusive, the algorithm will only degrade the optimization rather than produce an incorrectly synthesized design. In this section, we shall introduce the set of lemmas and theorems. The proofs of the lemmas and theorems are provided in [6].

Given two operators op_1 and op_2, and thier usage conditions OC_1 and OC_2, op_1 is mutually exclusive to op_2 if and only if $OC_1 \otimes OC_2 = TRUE$, where "$\otimes$" represents the tautology statement $\overline{OC_1 \wedge OC_2} = TRUE$. The results of $\otimes$ can be determined by the following lemmas and theorems:

Lemma 1 *If OC_1 and OC_2 are conditions of the same assignment decision node (ADN), then $OC_1 \otimes OC_2 = TRUE$.*

Lemma 2 *If OC_i can be statically evaluated to $FALSE$ then we can remove OC_i and its corresponding assignment value from the ADD.*

Lemma 3 *If OC_1 can be statically evaluated to $TRUE$ and OC_2 can not be statically computed, then $OC_1 \otimes OC_2 = FALSE$.*

Lemma 4 *$OC_1 \equiv OC_2$ (equivalent) if and only if the CG sub-graph representing OC_1 is identical (isomorphic) to the CG sub-graph representing OC_2.*

In our application, isomorphism between two CG subgraphs is identified in linear time by a one-to-one comparison of the CG without any transformations.

Theorem 1 *If $OC_1 \equiv OC_2$ then $OC_1 \otimes OC_2 = FALSE$.*

Theorem 2 *If $OC_1 \equiv \neg OC_2$ then $OC_1 \otimes OC_2 = TRUE$.*

Theorem 3 *Given OC_1 and OC_2 such that,*

$$OC_1 = OC_{11}\ Rop_1\ OC_{12},$$
$$OC_2 = OC_{21}\ Rop_2\ OC_{22},$$
where $\{Rop_1,\ Rop_2 \in \{<, \leq, ==, \neq, >, \geq\}\}$, *and*
$$OC_{11} \equiv OC_{21},$$
$$OC_{12} \equiv OC_{22},\ then$$
$OC_1 \otimes OC_2 = TRUE$ *if* $(Rop_1,\ Rop_2) \in \{(<,>),(<,\geq),(\leq,>),(==,<),(==,>),(==,\neq)\}$.

For example, if OC_1 is a condition $(x + y < z)$ and OC_2 is a condition $(x + y > z)$ (ie., $OC_{11} = x + y$, $OC_{12} = z$, $OC_{21} = x + y$, $OC_{22} = z$, Rop_1 is $<$ and Rop_2 is $>$) then OC_1 and OC_2 are mutually exclusive.

Theorem 4 *Given OC_1 and OC_2 such that,*
$$OC_1 = OC_{11}\ Rop_1\ OC_{12},$$
$$OC_2 = OC_{21}\ Rop_2\ OC_{22},$$
where $\{Rop_1,\ Rop_2 \in \{<, \leq, ==, \neq, >, \geq\}\}$, *and*
$$OC_{11} \equiv OC_{21},$$
$OC_{12} \neq OC_{22}$, *but OC_{12} and OC_{22} can be evaluated to constant values, then*
$OC_1 \otimes OC_2 = TRUE$ *if* $(Rop_1,\ Rop_2,\ R_c) \in \{(<,>,\leq),(<,\geq,\leq),(\leq,>,\leq),\ (\leq,\geq,<),(==,<,\geq),(==,\leq,>),(==,\geq,<),(==,\neq,==),(==,==,\neq)\}$ *where R_c is a relation between OC_{12} and OC_{22}.*

For example, if OC_1 is a condition $(x == 1)$ and OC_2 is $(x == 2)$ (ie., $OC_{11} = x$, $OC_{12} = 1$, $OC_{21} = x$, $OC_{22} = 2$, Rop_1 = "$==$", Rop_2 = "$==$" and R_c = "$\neq$"), then $OC_1 \otimes OC_2 = TRUE$.

In the case where the usage conditions are complex boolean expressions, the mutual exclusion of the conditions can be proven by decomposing the conditions into sub-conditions and proving the exclusion on the decomposed sub-parts. The decomposition rules are as follows:

Theorem 5 *if $OC_1 \equiv OC_{11} \wedge OC_{12}$, then*
$(OC_{11} \wedge OC_{12}) \otimes OC_2 = (OC_{11} \otimes OC_2) \vee (OC_{12} \otimes OC_2)$.

Theorem 6 *if $OC_1 \equiv OC_{11} \vee OC_{12}$, then*
$(OC_{11} \vee OC_{12}) \otimes OC_2 = (OC_{11} \otimes OC_2) \wedge (OC_{12} \otimes OC_2)$.

To demonstrate the use of these lemmas and theorems, consider the usage conditions of op_2 and op_3 from Figure 5:

$$OC_2 = (Cond == C2)$$
$$OC_3 = (A * B + C > 16) \wedge (Cond == C1)$$

Determining the mutual exclusion between op_2 and op_3 can be accomplished as follows:

$$\begin{aligned} OC_2 \otimes OC_3 &= (Cond == C2) \otimes ((A * B + C > 16) \wedge \\ &\quad (Cond == C1)) \\ &= ((Cond == C2) \otimes (A * B + C > 16)) \vee \\ &\quad ((Cond == C2) \otimes (Cond == C1)) \qquad \textbf{(Theorem5)} \\ &= ((Cond == C2) \otimes (A * B + C > 16)) \\ &\quad \vee 1 \qquad \textbf{(Theorem4)} \\ &= 1 \end{aligned}$$

5 Algorithm and Results

Given two nodes from CGs, o_i and o_j, each of which represents a usage condition of an operator node, we can determine the mutual exclusiveness between the two conditions by using the algorithm *QueryMuEx* shown in Figure 7.

Basically, *QueryMuEx* is a recursive procedure. Each time it is called, it checks whether any of the lemmas can be used to determine the mutual exclusiveness of o_i and o_j. If none of the lemmas is applicable, then *QueryMuEx* calls *Decompose* to decompose the o_i and/or o_j and then recursively applies *QueryMuEx* to the decomposed sub-expression. The decomposition is performed until the mutual exclusiveness of the expressions can be determined.

```
Algorithm QueryMuEx($o_i, o_j$)
begin Algorithm
    if ($ApplyLemmas(o_i, o_j) = unknown$) then
        $return(Decompose(o_i, o_j))$;
    else return $ApplyLemmas(o_i, o_j)$;
    end if;
end Algorithm

Algorithm Decompose($o_i, o_j$)
begin Algorithm
    if ($o_i = \wedge$) then
        return ($QueryMuEx(o_j, LeftPred(o_i)) \vee$
                $QueryMuEx(o_j, RightPred(o_i))$);
    else if ($o_i = \vee$) then
        return ($QueryMuEx(o_j, LeftPred(o_i)) \wedge$
                $QueryMuEx(o_j, RightPred(o_i))$);
    else return FALSE;
    end if
end Algorithm
```

Figure 7: Algorithm QueryMuEx.

Then results from the last recursion is returned to the previous level until the top-most call.

We have tested our algorithm on several benchmarks from the High-Level Synthesis Workshop [5] and previous publications: Wakabayashi's example [8], Kim's example [7], AMD2901, and AMD2910. For each benchmark we wrote three different VHDL behavioral descriptions. Each of the descriptions differs in the use of language construct (eg. IF-THEN-ELSE, and CASE statements) and description style (eg., grouping of conditional assignments). For example, description 1, 2 and 3 are behavioral descriptions of AMD2901 written in different styles.

For each description, we manually compute the number of operators and the total number of pairs of operators that are mutually exclusive. These numbers are given in the *# of operators* and *total # of m.e. pairs* columns, respectively. It should be noted that even though the total number of mutually exclusive operators are computed manually, the computation process is NOT trivial and it is time consuming.

Subsequently, we invoke different algorithms on each description to find all possible pairs of operators that are mutually exclusive for that example. The result of this experiment is reported in terms of the percentage of operator pairs that are found by the algorithm as compared to the number found manually (*total # of m.e. pairs*). Figure 8 shows results obtained using Kim's approach [7], Wakabayashi's approach [8], path-based scheduling approach [2], and our approach.

The results show that our approach can completely identify all possible pairs of mutually exclusive operators. On the other hand, Kim's, Wakabayashi's and path-based approach can identify all possible pairs only for certain description styles.

example	# of operators	total # of m.e. pairs	% of m.e. pairs detected			
			Wakabayashi's	Kim's	path-based	ours
Description 1	6	12	100 %	100 %	100 %	100 %
Description 2	6	12	0 %	0 %	100 %	100 %
Description 3	6	12	50 %	50 %	100 %	100 %
Description 4	16	45	100 %	100 %	100 %	100 %
Description 5	16	45	26.7 %	26.7 %	100 %	100 %
Description 6	14	21	100 %	76.2 %	76.2 %	100 %
Description 7	25	140	100 %	100 %	100 %	100 %
Description 8	25	140	0 %	0 %	100 %	100 %
Description 9	25	140	100 %	68.5 %	68.5 %	100 %
Description 10	27	338	100 %	100 %	100 %	100 %
Description 11	27	338	0.9 %	0.9 %	100 %	100 %
Description 12	25	286	100 %	85.3 %	85.3 %	100 %

Figure 8: Experiment results using different approaches.

6 Conclusion

We demonstrated quality of the proposed approach on several benchmarks of the High-level Synthesis Workshop. The results show that the proposed approach can identify all possible mutual exclusive operators in the benchmarks used, and out-perform all previously known approaches. In addition, unlike previous approaches, the proposed approach is independent of language constructs and modeling styles used in the description.

7 References

[1] R.A. Bergamaschi, "The Effects of False Paths in High-Level Synthesis," *Proc. ICCAD 91*, 1991.

[2] R. Camposano, "Path-Based Scheduling for Synthesis," *IEEE Trans. CAD*, Vol.10, no.1, Jan. 1991.

[3] V. Chaiyakul, D.D. Gajski and L. Ramachandran, "High-level Transformations for Minimizing Syntactic Variances," *Proc. 30th DAC*, 1993.

[4] D.D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis: Introduction to Chip and System Design,* Kluwer Academic Publishers, 1992.

[5] *Benchmarks for the Sixth International Workshop on High-Level Synthesis*, 1992.

[6] H.P. Juan, V. Chaiyakul, and D.D. Gajski, "Condition Graphs for High-Quality Behavioral Synthesis," *Technical Report #94-32, Dept. of ICS, UC Irvine.*, 1994.

[7] T. Kim, J.W.S. Liu, and C.L. Liu, "A Scheduling Algorithm For Conditional Resource Sharing," *Proc. ICCAD 91*, 1991.

[8] K. Wakabayashi and T. Yoshimura, "Global Scheduling Independent of Control Dependencies Based on Condition Vectors," *Proc. 29th DAC*, 1992.