Execution Condition Analysis in High Level Synthesis: a Unified Approach

O. Peñalba, J.M. Mendías, M.C. Molina
Departamento de Arquitectura de Computadores y Automática
Universidad Complutense de Madrid
Ciudad Universitaria s/n, 28040 Madrid, Spain
{olgape, mendias, cmolinap}@dacya.ucm.es

Abstract

The degree of conditional hardware reuse achieved after a high-level synthesis process depends on two factors: the number of mutually exclusive (m.e.) operations pairs that an algorithm can detect and the description style used by the designer when specifying the system. In this paper, we propose a method that deals with both aspects. It includes a mechanism to analyze the input description and identify all the m.e. operations pairs in a simple and homogeneous way, independently of the conditional constructs (IF or CASE) used to specify the control flow of the system. It also provides a collection of formal transformations on the input description which produces a specification of the same behavior that leads to an improved implementation—in terms of the degree of conditional reuse that is achieved.

1. Introduction

Two operations whose results are never needed both in the execution of the system can be implemented with the same hardware component and scheduled in the same cycle. This property is called mutual exclusiveness. The importance of detecting this property between two operations lies in the possibility of reducing the hardware cost without increasing the latency, that is, the conditional reuse of resources.

In the last decade several analyses of this problem have been carried out and several solutions proposed. By means of a more exhaustive control-flow and data-flow analysis the number of m.e. operations pairs that can be identified has increased in recent approaches. Nevertheless, the possibility of conditional reuse is restricted not only by the m.e. operations pairs not detected by an algorithm, but also by the way in which specifications are written by designers. In relation to this, all the previous approaches—like most HLS algorithms—have an important limitation which prevents them from exploring the maximum potential reuse in a description: an unnecessary and strict fidelity to the description given by the designer.

To see the sharing possibilities that may be behind the source code given by the designer, consider the VHDL descriptions in Figure 1(a) and Figure 1(b). Both behaviors are identical. However, while the code of Figure 1(a) requires a schedule of three clock cycles, if only one adder is available, Figure 1(b) shows how mutual exclusiveness allows a schedule in just two clock cycles with a single adder. The hardware sharing in Figure 1(b) has been achieved by decomposing a complex condition. Likewise, Figure 1(c) and Figure 1(d) show how replicating a common computation can produce the same kind of hardware sharing.

The approach we present in this work proposes a behavior-preserving transformation of the original specification that ignores the implicit design decisions (related to hardware sharing) taken by the user when writing the description. The manipulation of the specification is possible thanks to the chosen formal internal representation and to the existence of an underlying formal framework, where the correctness of every transformation has been mathematically proved. These transformations increase the chance of conditional sharing and enable a very simple and homogeneous algorithm to be defined in order to identify all the m.e. operations pairs in the description.

The next section of this paper contains an overview of the previous approaches. In section 3 we briefly discuss the internal representation used. In section 4 we present our algorithm (a basic approach) along with some experimental results. In section 5 an extension to the basic approach is introduced, which performs some semantic analysis in order to handle relational operators in the conditions, increasing the number of identified m.e. pairs. This extension enables the integration of both types of

1 This work has been supported by the Spanish Government Research Grant TIC99-0474
conditional constructs, IF and CASE, in the same mutual exclusiveness identification method. Finally, some conclusions are presented.

2. Overview of previous approaches

Wakabayashi and Yoshimura [1] proposed a method using condition vectors (CV) over control data flow graphs (CDFG). Operations are assigned CVs in a bottom-up traversal of the CDFG. This approach is simple and can identify mutual exclusiveness between operators in conditional branches. It also performs a data-flow analysis on each of the conditional branches. However, it cannot identify mutual exclusiveness across conditional blocks.

Juan, Chaiyakul and Gajski [2] introduced the concept of the condition graph (CG) and proposed an algorithm for identifying m.e. pairs that is not based on language constructs, providing a method whose results are independent of description styles. This is the only approach so far that considers the CASE construct [3] and the conditions containing relational operators. They also proposed, for the first time, the separation of the mutual exclusiveness identification from the scheduling algorithm. This approach fails in identifying those m.e. pairs involving operations whose results are used in a condition.

The approach by Li and Gupta [4] includes a classification of the possible m.e. pairs in an input description, distinguishing three types: structural m.e. pairs (those which appear in different conditional branches), behavioral m.e. pairs (those which belong to different control paths) and data-flow m.e. pairs (those whose results are only needed by operations that are m.e.). This is not a universal classification as m.e. pairs are not grouped according to any general feature, but to the way they are detected in this particular approach, and it depends on the features of the description language. It may fail in identifying some behavioral m.e. pairs and, hence, some data-flow m.e. pairs.

Other methods can be found in [5] [6] [7].

The main limitations of the previous approaches are the lack of some semantic analysis of the conditions that leads to identifying the actual set of m.e. operations pairs (only the Condition Graph approach performs some semantic analysis) and the restriction to the description as it is given by the designer, without considering the conditional sharing possibilities behind the specification style.

3. Internal representation

In our approach, the internal data representation mechanism used is the **equational specification mechanism** [8] [9]. There are several reasons that justify this choice. The first one is that this mechanism is intuitive, flexible, easy to handle, and referentially transparent. The second reason is that it has a mathematical model and a formal support that permit the correct manipulation of the description. This is crucial for the implementation of the transformations presented in the introduction, because they can be proved to preserve the system behavior. The third reason is that it enables a single and unified method to be defined in order to detect all the m.e. pairs in a description.

Within this mechanism, a system is represented by a set of equations describing the data-flow. Each equation defines how to compute an internal signal (or variable) or
an output port, in terms of a composition of operators whose arguments are other signals and/or input ports.

The correct manipulation of the description, which makes it possible to guarantee the correctness of aggressive transformations, is carried out by means of equations and derivation rules from the underlying formal calculus. Every single equation or rule is previously proved to be correct using an external theorem prover, like, for instance, the Larch Prover [10]. Thus, the final description is always semantically equivalent to the original one.

Getting equational specifications from procedural ones is not difficult. It is enough to compile the source code and describe the resulting graph in terms of a set of mutually recursive equations.

Figure 2 shows a VHDL description and Figure 3 a corresponding equational representation.

4. EQS Algorithm: a first approach

The EQuational Specification based algorithm [11] for detecting mutual exclusiveness between operations makes use, like the Condition Vector approach, of the simple

```
entity example2 is
port(a,b,c,d,e,f,g: in integer;
   x,y: in boolean;
   u,v,w,x: out integer;
   clk: in bit);
end example2;
architecture beh of example2 is
begin
main: process
variable T1: boolean;
variable T2,T3,T4,T5: integer:

T1 := (a&b)&c;
T2 := d&e;
T3 := e1;
if (y) then
  if (T1) then
    u := T3+d;
    w := d;
    else
      if not(a&1) then
        u := T3+d;
      else
        u := d;
      end if;
    end if;
  end if;
  if not(T1) then
    w := a&f;
  end if;
  if (not(T1) and (a&1)) then
    v := T3+e;
  else
    v := b&g;
  end if;
else
  T4 := T3+e;
  T5 := T4+f;
  u := T5+g;
  v := d;
  case a is
  when 1 => w <= a&d;
  when 2 => w <= a&2;
  when others => w <= a&d;
  end case;
  if (x) then
    z := a&g;
  else
    z := d;
  end if;
  case a is
  when 2 => g <= a&d;
  when 6 => g <= f1;
  when others => g <= g1;
  end case;
  wait until clk='event and clk='1';
end process main;
end beh;
```

Figure 2. A VHDL description with conditional sentences.

```
int_sig1 = a+b
int_sig2 = int_sig1-c
u = if y then (if int_sig2 then (c+d) else (if (a=1) then d else (if (d=1) then d else (d+e))) )
else ((c+1)+a)+((c+1)+a)+y+11
v = if y then (if int_sig2 then b+y)
else (if (a=1) then d else (if (d=1) then d else b+y))
else d
w = if y then (if int_sig2 then d else e+f)
else case a is
1: a+d
2: a+e2
default: a*d
x = if (a=1) then a*d
else d
s = case a is
2: a*d
4: f1
6: g
default: g1h
```

Figure 3. An equational specification describing the same behavior as the VHDL code in Figure 2.

idea of attaching an execution condition to every operator. To decide if two operators are m.e. we just compare their execution conditions.

The scheme of this algorithm is shown in Figure 4. It consists of four steps:

- Operations sprout. In this step the input specification is transformed as shown in the introduction to enhance the later reuse possibility. After this transformation, several instances of the same original operation are considered as different. But this is not a problem, because when processing a description for HLS, the number and positions of the operators in the specification do not have to be preserved to achieve the goals of the HLS process, as long as the behavior is preserved and the solution satisfies the cost/performance requirements.

- EC extraction. In this step all the signal definitions are traversed to get the execution condition (EC) of every operation. An EC is a collection of pairs (c, v), where c is an atomic condition, which does not contain any logic operators, and v the corresponding value for the execution of the operation, which can be true or false. After the operation sprout, all the operations have just one EC. The EC for some operations in Figure 3 are shown in Figure 5.

- Operations unification. Those operations consisting of the same operator applied to the same arguments and having the same EC are equivalent from the hardware reuse point of view. So, they have to be unified (note that most systems and designers consider that two operations are identical when they have the same arguments and operator, independently from their EC).

- Mutual exclusiveness identification. Finally, the EC of all the operations are pair-wise compared to detect if they are m.e. Two operations are m.e. if their EC share a condition that has a boolean value in one and the opposite value in the other.
4.1. Operations sprout

It consists of splitting conditions and replicating common expressions to make each signal definition tree dependent from the others. After this step all the conditions affecting an operator appear in nested conditional constructs and all the operators appear everywhere they are needed. Therefore, there is only one kind of m.e. pair: the structural m.e. pair.

**Splitting conditions.** This step consists of transforming an IF construct with a composed condition into several nested IF constructs with atomic ones. This is done by applying equations to the signals definitions. The collection of equations to be applied to split all the composed conditions depends on the set of logic operators available in the description language. For example, for the logic operators and, or, not, and nor, assuming the semantics they usually have, the set of equations in Figure 6 would be enough.

**Replicating common expressions.** This step consists of eliminating the internal signals (or variables) used to store a value in order to calculate it once and use it whenever it is needed. It is performed by applying two transformation rules from the formal calculus: one to substitute every variable or internal signal by its definition —the substitution rule— and another to eliminate those variables or internal signals no longer used —the elimination rule [9].

4.2. Some experimental results.

This first approach extends the solutions space that an optimization algorithm will explore, thus providing the possibility of obtaining implementations with a higher degree of conditional reuse. This is due to the transformations presented.

Table 1 shows the percentage of m.e. operation pairs identified by the Condition Vector, Condition Graph and TDT approaches in relation to the total number of pairs. These data try to give an idea of the possibility of reuse provided by the transformations. The higher the number of m.e. pairs is, the wider the solution space to explore, and so, better solutions can be achieved.

It must be noted that the number of m.e. pairs that are detected may be restricted by the way in which atomic conditions are compared. If this comparison is merely syntactic then several conditions that are semantically equivalent but having different forms are considered as different, and so some actual m.e. pairs are not detected. In the next section we present a partial semantic analysis of the conditions that notably increases the number of m.e. pairs identified.

5. Looking into the conditions: an extension to the EQS algorithm

The conditions that appear in behavioral descriptions are expressions that can include logic, relational and arithmetic operators. After the operations sprout step we

### Table 1. Percentage of m.e. operations pairs in relation to the total number of pairs.

<table>
<thead>
<tr>
<th></th>
<th>CV</th>
<th>CG</th>
<th>TDT</th>
<th>EQS</th>
</tr>
</thead>
<tbody>
<tr>
<td>kim (+) [6]</td>
<td>43.48%</td>
<td>43.48%</td>
<td>43.48%</td>
<td>50%</td>
</tr>
<tr>
<td>juan (+) [2]</td>
<td>20%</td>
<td>46.67%</td>
<td>46.67%</td>
<td>57.14%</td>
</tr>
<tr>
<td>juan (+) [4]</td>
<td>50%</td>
<td>52.77%</td>
<td>61.11%</td>
<td>72.72%</td>
</tr>
<tr>
<td>example1 (+) [11]</td>
<td>50%</td>
<td>52.77%</td>
<td>47.22%</td>
<td>72.73%</td>
</tr>
<tr>
<td>example1 (*) [11]</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>50%</td>
</tr>
<tr>
<td>EXPRESSION</td>
<td>(c, vset)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a = b</td>
<td>(a, [b, b])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a ≠ b</td>
<td>(a, [min, b] ∪ (b, max))</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a &lt; b</td>
<td>(a, [min, b])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a ≤ b</td>
<td>(a, [min, b])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a &gt; b</td>
<td>(a, (b, max])</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a ≥ b</td>
<td>(a, [b, max])</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2. ECs for conditions containing relational operators.

only have to deal with atomic conditions, which are either logic variables or expressions made up of relational operators. If we look at the descriptions included in the HLS Benchmark suites, we can see that most of the conditions contain relational operators. So, taking this kind of operators into account in the analysis of the conditions, which is not difficult in our mechanism, can contribute to a substantial increase in the number of m.e. pairs detected by the algorithm. In addition, this extension will provide a method to identify mutual exclusiveness in descriptions that contain both IF or CASE constructs.

5.1. Removing relational operators

In the basic approach, an EC for an operation was defined as a collection of pairs (c, v), where c is an atomic condition and v the corresponding value: true or false. Within this model, the EC defined by the expressions (a=1), -(a=1) and (a/=1) were considered as

$$EC_{op1} = ((a - 1), \text{true})$$
$$EC_{op2} = ((a = 1), \text{false})$$
$$EC_{op3} = ((a /= 1), \text{true})$$

In this extension the EC will also be a collection of pairs (c, v), but where c is a variable or expression of logic, integer or any other discrete type, and v a set of values of that type. This is based on the idea that false actually represents the complementary set of true. Now the EC defined by those expressions will be written as

$$EC_{op1} = (a, \{1\})$$
$$EC_{op2} = (a, \{\overline{1}\})$$
$$EC_{op3} = (a, \{1\})$$

where "\(" is the complement operation. So \{1\} is a set containing all the values of a’s type, except value \{1\}. Note how, in this approach, the EC defined by the expressions -(a=1) and (a/=1) are identical, providing a more powerful analysis mechanism.

These sets are represented as intervals or unions of intervals. Table 2 shows some EC for expressions with relational operators, where min and max represent, respectively, the minimum and maximum values of the corresponding type (remember that all data types are bounded in hardware implementations).

5.2. Detecting mutual exclusiveness

Within this model, two operations op1 and op2 are m.e. if their execution conditions, EC_{op1} and EC_{op2}, contain pairs (c1, vset1) and (c2, vset2), respectively, such that

$$c_1, c_2 \text{ are syntactically equivalent}$$
$$vset_1 \cap vset_2 = \emptyset$$

According to this, two operations depending, respectively, on conditions (a=1) and (a/=1) are detected as m.e. because EC_{op1} = (a, [1,1]) and EC_{op2} = (a, (\text{min,1}) \cup (1,\text{max})) fulfill the above requirements.

5.3. Extending the EQS algorithm

The REQS algorithm (EQS extended with relational operators handling) keeps the four steps defined for EQS and it also matches the scheme shown in Figure 4. The changes only affect the implementation of some procedures in this scheme: traverse_signal_definition(), unify_operations() and determine_met().

5.4. Mutual exclusiveness analysis with CASE constructs

Once we know how to handle conditions with relational operators it is trivial to handle multiple-selection constructions in the mutual exclusiveness analysis.

Consider a CASE construct like the following one

```
case a is
  case b is
    $k_i : o_{p_i}$
    $k_j : o_{p_j}$
    ...$k_j : o_{p_j}$
  default: $o_{p_{default}}$
```

where $k_i < k_j \quad \forall i, j \in \{1..n\}$ such that $i < j$, and where $a$ is an expression of any discrete type. The EC for the operations that may appear in the branches of this CASE construct are:

$$EC_a \equiv EC_{CASE}$$
$$EC_{op1} \equiv EC_{CASE} \cdot (a, [k_i, k_j])$$
$$EC_{op2} \equiv EC_{CASE} \cdot (a, [k_i, k_j])$$
\[ \ldots \]
$$EC_{opn} \equiv EC_{CASE} \cdot (a, [k_i, k_j])$$
$$EC_{default} \equiv EC_{CASE} \cdot (a, [\text{min}, k_i] \cup (k_i, k_j) \cup \ldots \cup (k_j, \text{max}))$$

Therefore, the method implemented in the REQS algorithm is valid for handling this kind of construct and we can easily integrate it in the mutual exclusiveness.
analysis, increasing the number of m.e. pairs identified. Now we have a global and totally homogeneous model, where every operation has an execution condition that determines exactly when its result is needed, independently of the control flow of the description, that is, independently of the particular IF or CASE constructions used to describe the system.

Figure 7(a) and Figure 7(b) show the collection of m.e. pairs for add and multiply operations in Figure 3, respectively, detected by both approaches: the EQS and the REQS. Shaded cells represent the m.e. pairs detected only by the extended approach, which takes both, the relational operators in the conditions and the CASE constructions, into account. The rest of the cells represent the m.e. pairs detected by both approaches.

6. Conclusions and future work

In this paper, we have presented a new approach to identifying mutual exclusiveness in behavioral descriptions that out-performs all previous approaches in two respects: in providing a powerful and homogeneous model to detect all m.e. operations pairs in a description, and in allowing a higher conditional reuse thanks to a formal transformation of the original description. The correctness of this transformation process is guaranteed because each transformation step applies a proved equation or a rewriting rule from the underlying formal calculus. With this method a larger collection of m.e. pairs is detected in relation to the previous approaches.

The results of the identification method depend on the semantics of the conditions, so it is necessary to study the conditions in depth to detect all the actual m.e. operations pairs in the description. In this approach, some semantic analysis is performed that removes logic and relational operators from the conditions. A more general and powerful analysis is part of our future work.

The internal representation has an important advantage not included in this paper: an easy way to handle dependencies in the temporal iteration space by means of an operator implementing delay, fby [8]. The analysis of mutual exclusiveness involving these kind of dependencies has been disregarded by the previous approaches, although they are very common in behavioral descriptions. This is also the subject of our future work.

References