Author manuscript, published in "13th International Symposium on System Synthesis (ISSS '00), Madrid : Spain (2000)"
DOI : 10.1109/ISSS.2000.874030

hal-00545528, version 1 - 10 Dec 2010

Hierarchical Conditional Dependency Graphs as a Unifying Design Representation in the *CODESIS* High-Level Synthesis System

Apostolos A. Kountouris

MITSUBISHI ELECTRIC ITE
80, Av. Des Buttes de Coesmes
35700 Rennes, FRANCE
kountouris@tcl.ite.mee.com

Christophe Wolinski

IRISA
Campus Universitaire de Beaulieu
F-35042 Rennes CEDEX, FRANCE
wolinski@irisa.fr

Abstract

In high-level hardware synthesis (HLS) there is a gap on the quality of the synthesized results between data-flow and control-flow dominated behavioral descriptions. Heuristics destined for the former usually perform poorly on the latter. To close this gap, the CODESIS interactive HLS tool relies on a unifying intermediate design representation and adapted heuristics that are able to accommodate both types of designs as well as designs of a mixed data-flow and control-flow nature. Preliminary experimental results in mutual exclusiveness detection and in efficiently scheduling conditional behaviors, are encouraging and prompt for more extensive experimentation.

1. Introduction

The topic of efficiently scheduling conditional behaviors having a complex conditional structure, has been thoroughly investigated in previous research work mainly because traditional DFG based heuristics do not efficiently handle this kind of descriptions [1].

Several better adapted heuristics were proposed ([1], [2], [3], [4], [5], [6]). The quality of their results depends heavily on the ability to exploit conditional resource sharing ([2], [4], [6], [7]) and speculative execution ([3], [5], [16], [17]) possibilities as well as shorten path lengths using node duplication techniques [3].

In resource constrained scheduling these techniques permit to better utilize the hardware resources in the datapath and obtain better schedules which result in shorter execution paths and less control logic.

An important issue, also underlined in previous work ([9], [10]) relates to the effects of the syntactic variance of the input descriptions, on the synthesis results. These negative effects intervene in two distinct but interrelated levels as far as scheduling conditional behaviors is concerned; mutual exclusiveness detection and operation scheduling. CDFG based mutual exclusiveness detection techniques [3], [11] using the structure of the input description, produce different schedules for semantically equivalent but syntactically different descriptions. This is due to the variability on the amount of detected mutual exclusiveness [9]. Furthermore, CFG-based scheduling (i.e. PBS [6]) is very sensitive to the statement order in the input description.

From the above it is clear that efficient HLS for control dominated designs relies on the combination of the above techniques and in effectively coping with the problem of syntactic variance.

1.1. A unifying approach

In our previous work of [22], [19] we aligned with the view supported by others [5], [8], [9], in advocating for the need of more flexible internal design representations, to optimize the HLS results and effectively handle both control and data flow dominated designs.

In this paper it is explained why the adoption of an intermediate design representation, like the *Hierarchical Conditional Dependency Graph* (HCDG) unifies and enhances the high-level synthesis of behavioral descriptions. Unification is mainly achieved because the HCDG is well adapted to describe both control-flow and data-flow designs. Representing control and data flow in a uniform manner is key to efficient scheduling/allocation heuristics that combine the aforementioned optimization techniques under a single framework.

Thanks to its origins in formal specification the HCDG constitutes a formal framework on which HLS design activities can be optimized and freed from the negative effects of structural syntactic variance (*if* nesting, order).

Though benchmark results are a good indication on the interest of the proposed approach, further refinement and validation on larger designs is needed. To this end the *CODESIS* interactive synthesis tool has been developed.

Permission to make digital or hard copies of part or all of this work or personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.
ISSS 2000, Madrid, Spain
© 2000 IEEE 1080-1082/00 $10.00

hal-00545528, version 1 - 10 Dec 2010

2. The HCDG internal design representation

The HCDG [20] is a special kind of directed graph that represents data and control dependencies from a uniform dataflow perspective. It consists of the *Conditional Dependency Graph* (CDG) and the *Guard Hierarchy* (GH). To better illustrate the notions of the HCDG a small example will be used throughout this paper. Taken from [8], its C-like representation is shown in figure 1 and its HCDG in figure 2. For details on the HCDG construction process the interested reader is referred to [19].

```
process jian(a, b, c, d, e, f, g, x, y)
in port a[8], b[8], c[8], d[8], e[8], f[8], g[8];
in port x, y;
out port u[8], v[8];
{
    static T1;
    static T2[8], T3[8], T4[8], T5[8];
    T1 = (a +1 b) < c;
    T2 = d +2 e;
    T3 = c +3 1;
    if (y) {
        if (T1) u = T3 +4 d; /*u1 */
        else if (!x) u = T2 +5 d; /*u2 */
        if (!T1 && x) v = T2 +6 e;
    } else {
        T4 = T3 +7 e;
        T5 = T4 +8 f;
        u = T5 +9 g; /*u3 */
    }
}
```

Figure 1. Control-flow dominated description

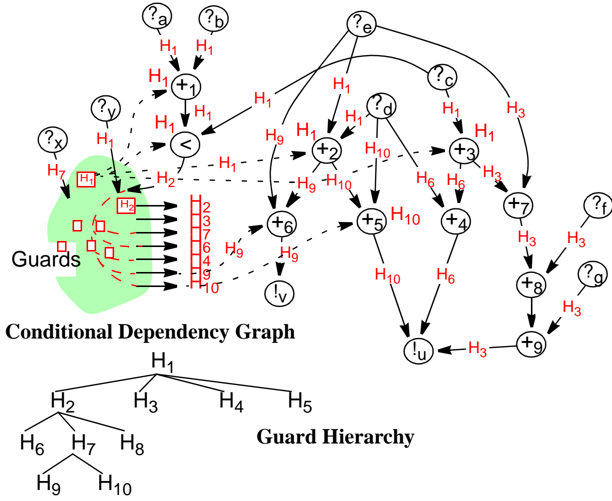

Figure 2. Example HCDG; CDG and GH

The CDG consists of a set of nodes and set of edges both labeled by *guard conditions*, called *guards* in the sequel. Guards (named H_i) are a special type of nodes (shown as rectangles) and represent boolean conditions that control the execution of operations and the assignment of values to variables. They have also been used as a formal control model in [4], [5] using a CDFG representation.

The rest of nodes (ovals) correspond to operations (i/o, computation, data multiplexing and state storage with either register or transparent latch semantics) that compute/assign values to variables. I/O node names are prefixed by ?/! respectively.

Edges represent control and data dependencies. Control dependencies (most of them omitted in figure 2 for readability reasons) are from guard nodes to the CDG nodes labelled by them and are represented by dashed arrows. Solid arrows represent data (computation) dependencies.

The HCDG obeys the principle of static single assignment. Nodes may have more than one definition only under mutually exclusive conditions (e.g. $!u$). In table 1 the guard definitions for the example are given.

Guard	Boolean Definition	Guard	Boolean Definition
H_1	1	H_6	$y \cdot T_1$
H_2	y	H_7	$y \cdot \overline{T_1}$
H_3	$\bar{y}$	H_8	$y \cdot T_1 + y \cdot \overline{T_1} \cdot \bar{x}$
H_4	$\bar{y} + y \cdot T_1$	H_9	$y \cdot \overline{T_1} \cdot x$
H_5	$\bar{y} + y \cdot T_1 + y \cdot \overline{T_1} \cdot \bar{x}$	H_{10}	$y \cdot \overline{T_1} \cdot \bar{x}$

Table 1. Guard Definitions

2.1. Formal semantics and the guard hierarchy

Initially the HCDG was developed as internal representation of systems described in the SIGNAL synchronous formal specification language, used for the specification of reactive, real-time systems. The interested reader is referred to [25] for more details. Being so it disposes of a formal calculus that allows for the compile time proof of correctness properties as well as the definition of correctness preserving graph transformations useful in optimizing the synthesis results [24].

In a discrete time model where time is considered as an infinite sequence of logical instants, a *guard* is the set of logical instants that the boolean condition defining it, evaluates to *true*. The theoretical foundations of the HCDG consider guards as sets and guard formulas as application of *set operations* on these sets. In [21] it is shown how an equivalent representation of guard formulas as boolean functions can be obtained and vice-versa. Guards are *equivalence classes* of the HCDG nodes grouping together nodes labeled by the same guard, thus active at the same logical instants.

The guard nodes of a HCDG are organized in a *Guard Hierarchy* (GH) which is a hierarchical tree-like, representation of the design control (figure 2, bottom). The GH represents the *inclusion relation* between guards.

Inclusion relation. Lets denote by h_i the boolean function corresponding to guard H_i. h_i evaluates to *true* when-

hal-00545528, version 1 - 10 Dec 2010

ever H_i is present otherwise to *false*. The inclusion relation represented by the tree like structure of GH simply states that: $\forall(H_j \in descendants(H_i)) \Rightarrow H_j \subseteq H_i$. Using the boolean definitions the inclusion relation between two guards will be denoted as: $H_2 \subseteq H_1 \equiv h_2 \leq h_1$. In addition, inclusion can be extended to the following cases:

$$H_k = H_i \cup H_j \Rightarrow H_i \subseteq H_k, H_j \subseteq H_k$$
$$H_k = H_i \cap H_j \Rightarrow H_k \subseteq H_i, H_k \subseteq H_j$$

In [21], the guard hierarchy is implemented as a hierarchy of BDD's. Control representations based on BDD's have already been used in previous work ([15], [4], [5]). The originality of the GH lies on the hierarchy construction and not at the use of BDD's which are simply used for their efficiency. Using BDD's two things can be efficiently achieved. First, *equivalence* between guard formulas can be easily established to avoid redundancy. Second, during hierarchization, it is easy to find the *maximum* depth in the tree that a guard node can be inserted, by means of a special factorization algorithm (see [21] for details). This yields an optimally refined inclusion hierarchy.

The some of the advantages of using the inclusion hierarchy information will be shown later on. Briefly, it permits to minimize the number of *mutex* tests [19] in guard exclusiveness detection used for conditional resource sharing especially useful in interactive design environments where speed is important. The hierarchy also enables the development of probabilistic priority functions used in HCDG based list scheduling that efficiently account for conditional behavior [24]. Finally, in [20] it is shown that guard inclusion information is very important in order to *triangularize* a larger number of systems of guard equations than it would be possible by using a rewriting system based only on the axioms of boolean algebra.

2.2. Efficient static mutual exclusiveness detection

Mutual guard exclusiveness will be noted by $\otimes$. Since in the formal foundations of the HCDG guards are *sets* of logical instants, two guards are mutually exclusive if their intersection is empty: $(H_1 \cap H_2 = \varnothing) \Leftrightarrow H_1 \otimes H_2$. In terms of the guard boolean function representations the above translates to: $h_1 \cdot h_2 = false \Leftrightarrow H_1 \otimes H_2$, which is the *mutex* test of [15].

Guard inclusion, as shown in [19], permits to minimize the number of mutual exclusion tests significantly. This optimization relies on the following proposition:

Let $subhier(H) = descendants(H) + \{H\}$ then:

$$H_1 \otimes H_2 \Rightarrow$$
$$\forall((H_i,H_j)) \in subhier(H_1) \times subhier(H_2), H_i \otimes H_j$$

meaning that if two guards H_1, H_2 are mutually exclusive then every guard in the sub-hierarchy of H_1 is mutually exclusive to every guard in the sub-hierarchy of H_2.

A set of benchmarks was used for the experimental evaluation of the mutual exclusiveness identification capabilities of the proposed approach compared to the methods of [26], [8], which are the most powerful methods so far in terms of coverage and insensitivity to syntactic variance. The benchmark from [19], was included to test the capabilities of our approach to reason on conditions defined by simple arithmetic relations [23]. Two semantically equivalent but syntactically different descriptions for each benchmark were used (*desc.1*, *desc.2*). The first, has a maximal conditional nesting as opposed to second one where conditions are flattened and each assignment is in its own conditional block. The results in the table below show that our method has at least as much coverage as the other two methods for a smaller number of *mutex* tests.

2.3. Mutual exclusiveness representation

Guard mutual exclusiveness is represented by a compatibility graph, MEG for *Mutual Exclusiveness Graph*, where vertices represent guards and edges the mutual exclusiveness relation between the guards connected by the edge. For the example the resulting MEG is shown in figure 3. Cliques in the MEG correspond to groups of pairwise mutually exclusive guards. Depending on the resource sharing context (FUs, registers, interconnects) each vertex has an associated list of specification objects

Benchmark		Number of operations	Total number of pairs	Number of mutex pairs	% coverage			Number of mutex tests
					Gupta [8]	Gajski [26]	Ours	
Gupta&Li [8]	desc1	9	36	22	100	86	100	20
	desc2			22	100	100	100	
Gajski [26]	desc1	6	15	7	100	100	100	5
	desc2			7	100	100	100	
Kim [11]	desc1	24	226	120	100	100	100	2
	desc2			120	100	100	100	
Parker [12]	desc1	16	120	55	100	78	100	24
	desc2			55	100	100	100	
test [19]	desc1	8 + 3	28 + 3	18 + 0	78	78	100	11
	desc2			18 + 0	78	78	100	

hal-00545528, version 1 - 10 Dec 2010

being active under this guard and can be allocated to a resource of that type. For instance, during scheduling such a structure permits to easily find groups of mutually exclusive operations that may share the same functional unit of a specific type.

In [22] it is argued that the best adapted algorithm to find such cliques is based on the *initial-graph-partition* algorithm presented in [13]. Other heuristics e.g. [14] are not as well adapted to satisfy our clique construction objectives since clique *maximality* is not always a good optimization criterion when scheduling is considered.

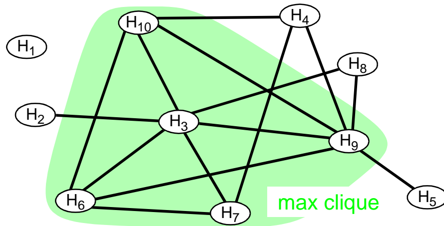

Figure 3. MEG for the example

Amongst other applications HCDGs and guard exclusiveness have also been used to *false path* identification (see [23] for more details) useful in path-based scheduling heuristics as well as more accurate static timing analysis.

2.4. Optimization by HCDG transformations

Constructing the HCDG reflects the way the design is described by the designer. Applying graph transformations semantically equivalent representations are produced. Using guard information transformations like dead code elimination, code motions, node duplication, path length reduction by dependency rearrangement, etc. can be easily performed. In our approach, transformations are of two types; *pre*- and *post*-scheduling.

The objective of pre-scheduling transformations is to remove syntactic variance and bring the HCDG into a form that will eventually yield better scheduling results. Such transformations include, lazy execution guard transformation to increase conditional resource sharing possibilities, dependency rearrangement and node duplication at mutually exclusive guards to shorten path lengths.

The term lazy execution is used to denote the situation when a node produces a value only as often as this value is used by other nodes. Computing the appropriate node guards for lazy execution may introduce additional guards in the guard hierarchy and some control paths may become longer. However the transformed graph contains more conditional resource sharing possibilities and in a scheduling scheme where conditional resource sharing is combined to speculative execution this lengthening of control paths can be effectively amortized. Finally, in certain cases where the result of a node is used at mutually exclusive guards the node can be duplicated at these guards without increasing hardware costs since the duplicated operation nodes are mutually exclusive and may share the same resource during scheduling. Post-scheduling transformations, incorporate scheduling information (i.e. conditional resource sharing and speculative execution) into the HCDG and so the transformed graph can be used in subsequent scheduling iterations or post-scheduling high-level synthesis activities (i.e. allocation/binding etc.).

Comparing figure 2 to figure 4, in the HCDG of the example the initial node guards were modified to enforce lazy node execution (e.g. $+_1$, $+_2$, $+_3$, < initially labelled by guard H_1). Also, the node $+_3$, used under mutually exclusive conditions ($H_6 \otimes H_3$), was duplicated to shorten the control paths. The data merge node (triangle u) is introduced to enforce the single assignment principle for variable u (in the behavioral description) which has multiple definitions (u_1, u_2, u_3) under mutually exclusive conditions, represented by guards H_3, H_6, H_{10} respectively.

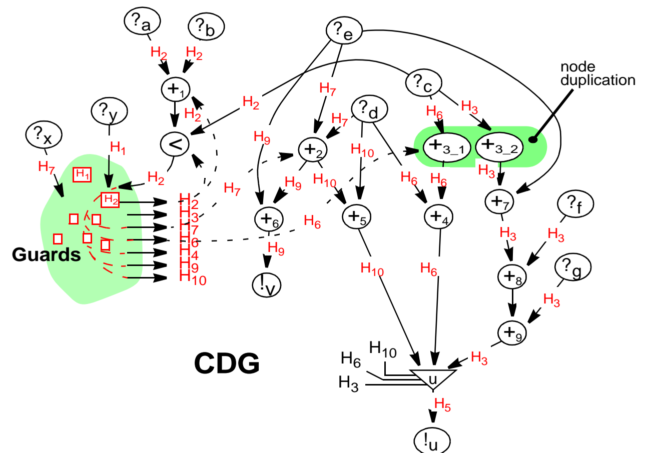

Figure 4. HCDG after optimizing transformations

3. HCDG based List Scheduling Heuristic

In this section a modified list scheduling heuristic that takes advantage of the HCDG features, is described. One important advantage of list scheduling is that its quality depends on the choice of the priority function [1]. In [22] we exploit the guard hierarchy to define a probabilistic priority function that better accounts for the conditional nature of the design. This is combined to an intelligent scheduling policy that employs pre-scheduling optimizing transformations (lazy execution, node duplication), conditional resource sharing and speculative execution.

This process has several advantages. The *list scheduling* priority criterion is satisfied for the greatest number of distinct execution instances (paths) simultaneously because the constructed cliques for conditional resource

sharing contain always the highest priority node and the largest number of other higher priority nodes that can share a resource with it. In respect to [9] and [5], speculative execution is considered only after normally executing nodes have been scheduled. In this way the risk of lengthening execution paths by displacing normally executing operations in favor of speculatively executing ones, is avoided. Finally, conditional resource sharing is exploited during scheduling and not before and so lengthening of execution paths due to inappropriate conditional resource sharing (i.e. [2], [11]), is also avoided.

3.1. Experimental results

The HCDG-based list scheduling heuristic is compared to other similar heuristics (*Kim* [2], *CVLS* [7], [3], *PBS* [6], *Brewer* [5], *ADD-FDLS* [9]) using benchmarks appearing in previous work (*kim*, *waka*, *maha*, *jian* from [2], [7], [12], [8] respectively). For each benchmark the HCDG was constructed, the guard hierarchy was refined, the HCDG was transformed for lazy execution and guard mutual exclusiveness was established using the techniques described in [18]. Results are given in the tables 2 to 5, for various resource constraints (cmp/+/- one cycle resources) and chaining length (cn: 1, means no chaining) in terms of "*total / longest path / shortest path*" numbers of states.

Resources	Kim	PBS	crit. path	Brewer	ours
cmp: 0, +: 1, -: 1, cn: 1	8/8/3	-	-	-/5/-	5/5/4
cmp: 0, +: 1, -: 1, cn: 2	6/5/2	9/5/2	8/8/-	-	5/5/4
cmp: 0, +: 2, -: 3, cn: 1	-	-	-	-/4/-	4/4/2
cmp: 0, +: 2, -: 3, cn: 3	3/3/2	-	4/4/-	-	3/3/2
cmp: 0, +: 2, -: 3, cn: 5	-	4/3/1	-	-	3/3/2

Table 2. Results for the "*maha*" benchmark

Resources	CVLS	Kim	PBS	Brewer	ours
cmp: 1, +: 1, -: 1, cn: 1	7/7/5	7/7/4	-	-	7/7/4
cmp: 1, +: 1, -: 1, cn: 2	-	7/7/3	8/7/3	-/7/-	6/6/3
cmp: 1, ALU: 2, cn: 1	-	-	-	-	7/7/4
cmp: 1, ALU: 2, cn: 2	-	6/6/3	6/6/3	-	6/6/3

Table 3. Results for the "*waka*" benchmark

Resources	Kim	Brewer	ADD	ours
cmp: 2, +: 2, -: 1, cn: 1	8/8/6	-	6/6/5	6/6/6
cmp: 1, +: 2, -: 1, cn: 1	-	-/6/-		6/6/6
cmp: 2, ALU: 2, cn: 1	-	-	-	6/6/6

Table 4. Results for the "*kim*" benchmark

Resources	ours (cn=1)	ours (cn=2)
cmp: 1, +: 1, cn: 1	4/4/3	4/4/3
cmp: 1, +: 2, cn: 1	4/4/2	3/3/2

Table 5. Results for the "*jian*" benchmark

Finally, the insensitivity of the scheduling results to the effects of syntactic variance is shown in table 6. For each benchmark two semantically equivalent but syntactically different descriptions (*descr.1*, *descr.2*) are used. The first, has a maximal conditional nesting as opposed to second one where conditions are flattened and each assignment is in its own conditional block. It is worth noting that for both descriptions the same HCDG was derived.

Bench.	*waka*		*maha*		*kim*		*jian*	
Resources	cmp: 1 +: 1 -: 1	cmp: 1 ALU: 2	cmp: 0 +: 1 -: 1	cmp: 0 +: 2 -: 3	cmp: 2 +: 2 -: 1	cmp: 2 ALU: 2	cmp: 1 +: 1 -: 1	cmp: 1 +: 2 -: 1
descr. 1	7/7/4	7/7/4	5/5/4	4/4/2	6/6/6	6/6/6	4/4/4	4/4/2
descr. 2	7/7/4	7/7/4	5/5/4	4/4/2	6/6/6	6/6/6	4/4/4	4/4/2

Table 6. Insensitivity to syntactic variance

4. The *CODESIS* tool

In order to validate our results in more realistic contexts and quantitatively evaluate the effectiveness of the HCDG and the HCDG-based heuristics the *CODESIS* interactive CAD tool has been developed. Currently the specification front-end is the SIGNAL formal specification language but in the future other standard descriptions languages (e.g. C, VHDL) will be supported. Translation of the HCDG into C and VHDL already exists and allows us to interface to existing implementation tools like software compilers and hardware synthesis (behavioral and RTL) tools. A graphical user interface permits to visualize the HCDG, interactively apply graph transformations, scheduling heuristics and visualize the obtained results.

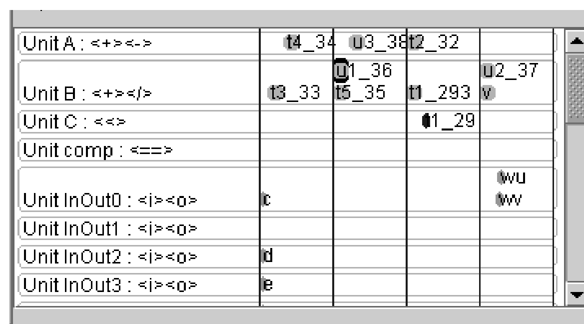

Figure 5. Scheduling results for "*jian*"

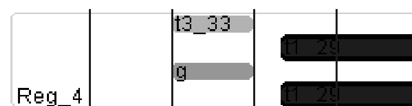

Figure 6. Example of register sharing

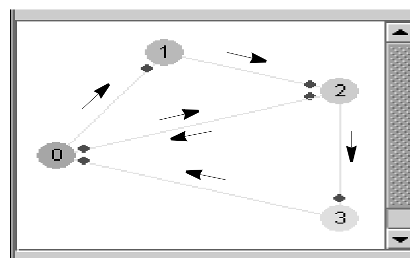

Figure 7. Controller FSM

hal-00545528, version 1 - 10 Dec 2010

hal-00545528, version 1 - 10 Dec 2010

CODESIS screenshots in figure 5 and figure 6 show conditional resource sharing for functional units and registers used in scheduling and register allocation algorithms, for the example. In figure 7 the automatically derived and optimized control FSM is shown.

The design and development of the tool are entirely object oriented in Java allowing for easy extensions and incorporation and use of new features in a plug and play fashion. For instance, new scheduling heuristics can be introduced, different priority functions can be tested, pre- and post-scheduling transformations can be applied in variable order etc. Due to its interactivity, extensibility and visualization capabilities, this tool will be very useful for research, experimentation and educational purposes.

5. Conclusions

The HCDG is a powerful internal design representation with the ability to treat both data-flow and control-flow designs under the same framework. Techniques and heuristics developed for data-flow oriented designs can be readily adapted for the HCDG. In addition several others have been developed to tackle the problems related to control-flow intensive designs.

The HCDG-based scheduling approach exploits most of the existing scheduling optimization techniques, enjoying their combined benefits. Both speculative execution and conditional resource sharing are combined in a uniform and consistent framework similarly to dynamic CV's of [3] and *guards* in [4], [5]. Even more, it does not suffer from effects of syntactic variance at both the mutual exclusiveness detection and scheduling levels, as CDFG or CFG based approaches do. The hierarchical control representation permits to minimize the number of mutual exclusiveness tests and also develop probabilistic priority functions that account for the conditional nature of the design.

Finally, to test our ideas in more realistic contexts a user friendly HLS tool has been built using the HCDG as its internal representation.

References

[1] R. A. Bergamaschi, S. Raje, I. Nair, L. Trevillyan. Control-Flow versus Data-Flow Based Scheduling: Combining Both Approaches in an Adaptive Scheduling System. IEEE Trans. VLSI, 5(1): 82-100, 1997.

[2] T. Kim, J.W.S. Liu, C.L. Liu. A Scheduling Algorithm For Conditional Resource Sharing. Proc. ICCAD 91, 84-87, 1991.

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

[4] Radivojevic, F. Brewer. Analysis of Conditional Resource Sharing Using a Guard-based Control Representation. Proc. ICCD'95, 434-439, Oct. 1995.

[5] Radivojevic, F. Brewer. Incorporating Speculative Execution in Exact Control-Dependent Scheduling. Proc. 31st DAC, 479-484, Jun. 1994.

[6] R. Camposano. Path-based Scheduling for Synthesis. IEEE Trans. on CAD, 10(1): 85-93, 1991.

[7] K. Wakabayashi, T. Yoshimura. A Resource Sharing and Control Synthesis Method for Conditional Branches. Proc. IEEE ICCAD'89, 62-65, 1989.

[8] J. Li, R. K. Gupta. An Algorithm To Determine Mutually Exclusive Operations In Behavioral Descriptions. Euro-DAC'97.

[9] V. Chaiyakul, D.D. Gajski, L. Ramachandran. Minimizing Syntactic Variance with Assignment Decision Diagrams. UCI, Tech. Rep. ICS-TR-92-34, Apr. 1992.

[10] Y-L. Lin. Recent Developments in High-Level Synthesis. ACM Trans. on Design Automation of Electronic Systems (TODAES), 2(1): 2-21, Jan. 1997.

[11] T. Kim, N. Yonezawa, J.W.S. Liu, C.L. Liu. A Scheduling Algorithm For Conditional Resource Sharing - A Hierarchical Reduction Approach. IEEE Trans. on CAD, 13(4): 425-438, Apr. 1994.

[12] A.C. Parker, J.T. Pizarro, M. Mliner. MAHA: A Program for Data Path Synthesis. Proc. 23rd DAC, 252-258, 1986.

[13] R. Puri, J. Gu. An Efficient Algorithm for Microword Length Minimization. Proc. DAC'92, 651-656, 1992.

[14] C.J. Tseng, D.P. Siewiorek. Automated Synthesis of Data Paths on Digital Systems. IEEE Trans. on CAD, 5(3): 379-395, Jul. 1986.

[15] R. A. Bergamaschi, R. Camposano, M. Payer. Allocation Algorithms Based on Path Analysis. Integration, The VLSI Journal, 13(3): 283-99, Sept. 1992.

[16] L.C.V. dos Santos, J.T.J van Eijndhoven, J.A.G. Jess. Combining Code Motion and Scheduling. ProRISC '96.

[17] Kifli, G. Goossens, H. De Man. A Unified Scheduling Model for High-Level Synthesis and Code Generation. Proc. EDTC'95, 234-238, Mar. 1995.

[18] A. Kountouris, C. Wolinski. Hierarchical Conditional Dependency Graphs for Conditional Resource Sharing. Proc. Euromicro'98, Wasteras, Sweden, 1998.

[19] A. Kountouris, C. Wolinski. Extensive Conditional Resource Sharing Based on Hierarchical Conditional Dependency Graphs. Proc. 12th Int'l VLSI Conference, Goa, India, Jan. 1999.

[20] L. Besnard. Compilation de SIGNAL: Horloges, Dependances, Environment. Ph.D., Univ. of Rennes I.

[21] T. P. Amagbegnon. Forme Canonique Arborescente des Horloges de SIGNAL. Ph.D., Univ. of Rennes I, 1995.

[22] A. Kountouris, C. Wolinski. Combining Speculative Execution and Conditional Resource Sharing to Efficiently Schedule Conditional Behaviors. Proc. ASP-DAC'99, Hong-Kong, Jan. 1999.

[23] A.Kountouris, C.Wolinski. False Path Analysis Based on a Hierarchical Control Representation. Proc. ISSS'98, Taiwan, Dec. 1998.

[24] A.Kountouris, C.Wolinski. High Level Pre-Synthesis Optimization Steps Using Hierarchical Conditional Dependency Graphs. Proc. Euromicro'99, Italy, Aug. 1999.

[25] P. Le Guernic, M. Le Borgne, T. Gautier, C. Le Maire. Programming Real Time Applications with SIGNAL. Proc. of the IEEE, 79(9): 1321-1336, Sep. 1991.

[26] H-P. Juan, V. Chaiyakul, D. D. Gajski. Condition Graphs for High-Quality Behavioral Synthesis. Proc. ICCAD'94, San Jose, CA, 1994.