

# Accelerated SAT-based Scheduling of Control/Data Flow Graphs

Seda Ogrenci Memik  
Computer Science Department  
University of California, Los Angeles  
Los Angeles, CA - USA  
seda@cs.ucla.edu

Farzan Fallah  
Fujitsu Laboratories of America, Inc.  
Sunnyvale, CA - USA  
farzan@fla.fujitsu.com

## Abstract

*In this paper we present a satisfiability-based approach to the scheduling problem in high-level synthesis. We formulate the resource constrained scheduling as a satisfiability (SAT) problem. We present experimental results on the performance of the state-of-the-art SAT solver, Chaff, and demonstrate techniques to reduce the SAT problem size by applying bounding techniques on the scheduling problem. In addition, we demonstrate the use of some transformations on control data flow graphs such that the same lower bound techniques can operate on them as well. Our experiments show that Chaff is able to outperform the Integer Linear Program (ILP) solver CPLEX in terms of CPU time by as much as 59 fold. Finally, we conclude that the satisfiability-based approach is a promising alternative for obtaining optimal solutions to NP-Complete scheduling problem instances.*

## 1. Introduction

Scheduling is one of the key tasks in high-level synthesis. Given a set of operations with data dependency relations, the scheduling problem associates a control step to each operation such that certain constraints are met. Many of the practical scheduling problem instances are known to be NP-Complete. There exist a plethora of heuristic scheduling techniques in the literature aiming to provide *good* solutions for practical problem instances. Although many of these heuristics often yield high quality solutions, their optimality cannot be guaranteed or optimality is reached in only very restricted cases. Hence, in many of the cases, the *goodness* of the results can only be evaluated by comparison against the optimal solution obtained using exact algorithms.

Traditionally, Integer Linear Program (ILP) solvers and Binary Decision Diagram (BDD) packages are used to obtain exact solutions to many design automation problems.

The scheduling problem has been addressed by researchers focusing on either of these methods (e.g. see [2], [6], [12], [4]). While ILP-based formulations are more popular for scheduling, BDD-based schedulers offer attractive solutions as well. One shortcoming of BDD-based solutions is that the BDD size can become too large for some problem instances. Therefore, they need careful ordering of variables. On the other hand, one advantage of BDD-based solvers is that they construct all possible solutions to a problem which may be useful in some cases. In recent years Boolean Satisfiability (SAT) emerged as a competitive alternative for modeling and solving problems in several areas, such as testing, and logic synthesis. A review on applications of SAT in Electronic Design Automation presents various example applications and points to a promising future for SAT in further domains [8].

In this paper, we introduce a novel application of SAT to scheduling problem in high-level synthesis for (re)configurable systems. We show that the SAT-based scheduler can indeed be used as an efficient tool to tackle computationally intensive problems. Furthermore, we present the impact of advanced lower and upper bound estimation techniques on the performance of SAT solvers. In addition, we discuss simple transformations to handle control nodes. Our experiments show that the SAT formulation yields the optimal result significantly faster than an ILP solver after applying the same bounding schemes to both formulations.

The rest of this paper is organized as follows. In Section 2.1 we formulate the resource constrained scheduling problem as a satisfiability problem. In Section 2.2 we review the techniques to improve the run-time performance of exact solvers and in Section 2.3 we discuss lower bound techniques for the scheduling problem with Data Flow Graphs as inputs. In Section 2.4 we discuss a method to handle Control Data Flow Graphs. Experimental results are presented in Section 3 and conclusions are given in Section 4.

## 2. SAT-based Scheduling

In Section 2.1 we formulate the scheduling problem and describe the transformation into a satisfiability problem. Lower and upper bound techniques employed to accelerate the process of solving the satisfiability problem are discussed in Section 2.2.

### 2.1. Problem Formulation and Boolean Satisfiability Transformation

The two most common formulations of scheduling problem are the *timing constrained scheduling* and the *resource constrained scheduling*. In this paper we focus on the latter formulation. Given a computation represented by a DFG, the resource constrained scheduling problem assigns operations in the DFG to control steps, such that the latency of the schedule is minimized and both the resource constraints and data dependency constraints are met. The resource constraint on each resource  $j$  is  $R_j$ , the number of available instances of that type.

We introduce a Boolean variable  $x_{ijk}$ , where  $i$  represents a particular operation index,  $j$  denotes possible clock steps within the schedule, and  $k$  denotes the resource index. If operation  $i$  is scheduled at step  $j$  on resource  $k$  then  $x_{ijk} = \text{true}$ , else  $x_{ijk} = \text{false}$ . Next, we transform constraints of the scheduling into Boolean expressions:

1. Each operation has to be assigned to a valid control step. For each operation there are  $S \times R$  Boolean variables, where  $S$  denotes the maximum number of control steps and  $R$  denotes the total number of resources.

Exactly one of the  $S \times R$  variables has to be equal to 1. Two sets of clauses ensure that. The first set guarantees having at most one variable equal to 1 (true). The second set guarantees having at least one variable equal to 1. For operation  $i$  these clauses are in the following form:

$$(\bar{x}_{i11} \vee \bar{x}_{i21}) \wedge \dots \wedge (\bar{x}_{i11} \vee \bar{x}_{iSR}) \wedge \dots \wedge (\bar{x}_{i(S-1)R} \vee \bar{x}_{iSR})$$

The second set is one clause containing all variables for the particular operation  $i$ :

$$(x_{i11} \vee x_{i21} \vee \dots \vee x_{i(S-1)R} \vee x_{iSR})$$

2. If there is a data dependency between operations  $u$  and  $v$ , and  $u$  is the parent of  $v$  on the DFG, then  $v$  cannot start before execution of  $u$  is finished. Therefore,

$$x_{ujk} \rightarrow (\bar{x}_{v1k} \wedge \dots \wedge \bar{x}_{vj k} \wedge \bar{x}_{v(j+1)k} \wedge \dots \wedge \bar{x}_{v(j+d_{u,k}-1)k})$$

for all possible resource combinations, where  $d_{u,k}$  denotes the delay of operation  $u$  on resource  $k$ .

3. An operation can only be executed on a matching resource type, i.e.,

$$\bar{x}_{i1k} \wedge \dots \wedge \bar{x}_{iSk} \quad \forall (i, k) \text{ pairs, where the type of operation } i \text{ does not match the type of resource } k.$$

4. At any control step, the number of operations executing on each resource can be *at most* one. A set of clauses are generated for each resource and each control step. For

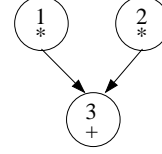


Figure 1. A sample DFG.

resource  $k$ , assuming the resource has a single cycle delay, these clauses take the following form:

$$(\bar{x}_{11k} \vee \bar{x}_{21k}) \wedge \dots \wedge (\bar{x}_{11k} \vee \bar{x}_{O1k}) \wedge \dots \wedge (\bar{x}_{(O-1)1k} \vee \bar{x}_{O1k})$$

⋮

$(\bar{x}_{1Sk} \vee \bar{x}_{2Sk}) \wedge \dots \wedge (\bar{x}_{(O-1)Sk} \vee \bar{x}_{OSk})$ , where  $O$  denotes the number of operations. These clauses are generated for each resource separately. For resources with multi-cycle delays, extra variables are introduced for operations that might have started at an earlier cycle than the current control step and still being executed on the resource.

The SAT formulation of the scheduling constraints described above contains  $O \times S \times R$  variables. The number of clauses generated for the first set of constraints is at most,  $\frac{1}{2} \times O \times (S \times R)(S \times R - 1) + O$ . For the data dependency constraints, the number of resulting clauses are determined by the number of edges in the DFG. For each edge, a set of clauses are generated. The resulting clauses will be in the order of  $E \times (S \times R)^2$ , where  $E$  denotes the number of edges in the DFG. The number of clauses of the constraint that forces operations to be assigned to matching resources is determined by the distribution of the resource types and their respective numbers. Assuming that in the worst case there is one resource of matching type for each operation, the number of these clauses are bounded by  $O \times S \times (R - 1)$ . The number of clauses generated due to the resource constraint is determined by the number of resources and operations compatible with each resource type. The total number of these clauses will be bounded by  $R \times S \times O$ . Finally, the total number of clauses is bounded by,  $\frac{1}{2} \times O \times (S \times R)(S \times R - 1) + O + E \times (S \times R)^2 + O \times S \times (2R - 1)$ .

We illustrate the SAT formulation on a simple example depicted in Figure 1. There are two multiplication (MUL) and one addition (ADD) operations in this DFG. Assume that one single cycle adder and one multiplier with two cycles delay are available. Let the multiplier resource index be 1 and the adder resource index be 2. Therefore,  $R_1 = 1$  and  $R_2 = 1$ . Also let  $S = 5$ , i.e., the maximum number of control steps is 5. In the following, the formulation of the sample scheduling problem is presented.

Clauses generated for the first constraint, for operation 1 are shown below. Similar clauses for operations 2 and 3 are generated:

$$(\bar{x}_{111} \vee \bar{x}_{121}) \wedge \dots \wedge (\bar{x}_{111} \vee \bar{x}_{151}) \wedge (\bar{x}_{121} \vee \bar{x}_{131}) \wedge \dots \wedge$$

$$(\bar{x}_{121} \vee \bar{x}_{151}) \wedge (\bar{x}_{131} \vee \bar{x}_{141}) \wedge (\bar{x}_{131} \vee \bar{x}_{151}) \wedge (\bar{x}_{141} \vee \bar{x}_{151}) \wedge (x_{111} \vee x_{121} \vee x_{131} \vee x_{141} \vee x_{151})$$

Clauses for data dependency constraint for the operation pair (1, 3) are shown below. Similar clauses are generated for the dependency between operations 2 and 3:

$$(\bar{x}_{111} \vee \bar{x}_{312}) \wedge (\bar{x}_{111} \vee \bar{x}_{322}) \wedge (\bar{x}_{121} \vee \bar{x}_{312}) \wedge (\bar{x}_{121} \vee \bar{x}_{322}) \wedge (\bar{x}_{121} \vee \bar{x}_{332}) \wedge \dots \wedge (\bar{x}_{151} \vee \bar{x}_{312}) \wedge (\bar{x}_{151} \vee \bar{x}_{322}) \wedge (\bar{x}_{151} \vee \bar{x}_{332}) \wedge (\bar{x}_{151} \vee \bar{x}_{342}) \wedge (\bar{x}_{151} \vee \bar{x}_{352})$$

Constraints imposing resource and operation type match are shown below. The constraints between operation 3, the ADD operation, and resource 1, i.e., the multiplier, are given here as an example.

$$(\bar{x}_{311} \wedge \bar{x}_{321} \wedge \dots \wedge \bar{x}_{351})$$

Resource constraints for the multiplier are shown below. At any control step only one operation can execute on the multiplier. Since this resource has multi-cycle delay, an operation may have started at an earlier cycle, but would still be considered active at a later step. For the adder there is no resource constraint, since there is only one ADD operation. Control step 1:  $(\bar{x}_{111} \vee \bar{x}_{211})$

⋮

$$\text{Control step 5: } (\bar{x}_{141} \vee \bar{x}_{241}) \wedge (\bar{x}_{141} \vee \bar{x}_{151}) \wedge \dots \wedge (\bar{x}_{151} \vee \bar{x}_{251}) \wedge (\bar{x}_{141} \vee \bar{x}_{241} \vee \bar{x}_{151} \vee \bar{x}_{251})$$

Once the constraints are written, the problem instances can be created for a given resource set and the maximum number of control steps allowed. The objective of the resource constrained scheduling is to minimize the number of control steps required while obeying resource and dependency constraints. The minimum number of control steps for a problem instance can be found through searching over a range of values for maximum number of steps, where at each step the problem instance is given to the SAT solver as input. An upper bound on the latency of the schedule is one possible starting point for this search. If no latency constraint is specified, then an upper bound can be determined by using a heuristic scheduler. For experimental purposes we have implemented a heuristic path-based scheduler to determine an upper bound on the latency of the schedule. Another way to conduct this search is to start with a lower bound and increase it until the problem becomes feasible. We will refer to the first search method as Upper Bound Search (UBS) and the latter as Lower Bound Search (LBS).

As for the particular search strategy, we adopted linear search scheme. Although binary search would require to solve less number of problem instances, the satisfiability solvers are known to spend significantly more time to solve infeasible instances than feasible instances. A binary search strategy might expose several infeasible instances to the solver, whereas a linear search encounters only one infeasible problem. During our experiments with these alternative search methods we observed that a linear search strategy was indeed more time efficient than binary search.

Our general search method is summarized in the following, the start point and the direction of the search is determined by the particular method, i.e., UBS or LBS.

*Bound\_Directed\_Search(DFG, Resource Set, Search\_Method={UBS, LBS})*

```

1 latency_constraint = bound(Search_Method)
2 if (Search_Method = UBS)
3   while (problem is feasible)
4     latency_constraint = latency_constraint - 1
5   end while
6   optimal_latency = latency_constraint + 1
7 else
8   while(problem is infeasible)
9     latency_constraint = latency_constraint + 1
10  end while
11  optimal_latency = latency_constraint - 1

```

## 2.2. Using Bound Techniques to Improve Performance

High computational complexity of the algorithms used by exact solvers has been a major burden for using these techniques in practical scenarios and for large problem sizes. In the ILP domain the problem of quickly obtaining exact solutions has been well researched. Techniques proposed in the past to improve different aspects of the ILP-based solutions can in fact be complementary to each other as well. Developing a well-structured ILP model for the particular problems is one of the important aspects. This has been studied for the scheduling problem and it has been shown that the solution time can be shortened by constructing efficient models [1] and [3]. Another technique is to reduce the problem search space using bounding techniques and eliminating variables from the problem formulation. The benefits of this technique on ILP formulations for scheduling has been studied by Chaudhuri et al. [1] and by Narasimhan and Ramanujan [11]. Another effective way to improve the performance is to incorporate problem-specific information into the underlying algorithm of the solver. Narasimhan and Ramanujan proposed a method to improve computational performance of ILP-based problems by re-formulating the branch and bound algorithm within a generic ILP solver to incorporate information derived from the specific problem [11].

While it has been successfully shown that the speed of ILP-based scheduler can be increased, possible improvements for SAT formulations have not been studied. In this paper we make a first step to contribute to SAT-based approach for solving scheduling problems and particularly investigate the effect of bounding techniques on performance of SAT-based scheduling. Knowledge on the feasible inter-

val of control steps within which an operation can be scheduled can reduce the number of Boolean variables as well as the number of clauses generated for the SAT formulation significantly. In addition, a lower bound on the latency of the whole schedule can help in directing the search of the SAT solver for the optimal solution. As a result, the solution time can be reduced significantly.

### 2.3. Finding Bounds for Schedules of Data Flow Graphs

The simplest bounds for start times of operations in a DFG can be derived from the ASAP and ALAP schedules. An ASAP schedule provides the earliest start time of each operation assuming unlimited resources. Similarly, given a latency constraint, ALAP gives the latest possible start times of operations within this constraint with unlimited resources. Several advanced lower bound techniques for resource constrained scheduling have been proposed in the past to obtain bounds that are tighter than ASAP and ALAP bounds. Rim and Jain proposed a technique to compute lower bounds under resource constraints [13]. Langevin and Cerny developed a recursive lower bound estimation method, which uses Rim and Jain’s estimation as a basis [5]. An extension with provably tighter bounds to the estimation method of Langevin and Cerny is proposed by Ramanujam and Narasimhan [10].

In this work, we have applied Ramanujam and Narasimhan’s [10] lower bound technique on our satisfiability formulation. This lower bound technique obtains bounds by discarding precedence constraints, and then calculating optimal value of the latency of the schedule, which in turn represents a lower bound on the schedule with precedence constraints in place. In a symmetric fashion it has been shown that upper bounds on the latest possible start times of operations can be found. Hence for each operation a valid and tighter bounded execution interval is available. The complexity of the algorithm to derive the bounds is  $O(N^2)$ , where  $N$  is the number of nodes in the DFG. An upper bound on the latency of the whole schedule needs to be provided to the algorithm, which can either be a given design constraint or an upper bound obtained from a heuristic scheduler. For experimental purposes we have generated constraints on the total schedule latency using a geometric path-based scheduler.

For the example from Section 2.1, the earliest and latest start times of each operation are as follows, Operation 1: (1, 3), Operation 2: (1, 3), and Operation 3: (5, 5). Using these bounds, 16 out of the initial 30 variables will be removed from the SAT formulation and several clauses are eliminated. In bigger examples the effect is even more dramatic (see Section 3).

```

a = b + c                                (DFG1)
if (a > d * c)                             (DFG2)
    result = b * 3                          (DFG3)
else
    result = b * 2 + 1                      (DFG4)
output = result + 2                        (DFG5)

```

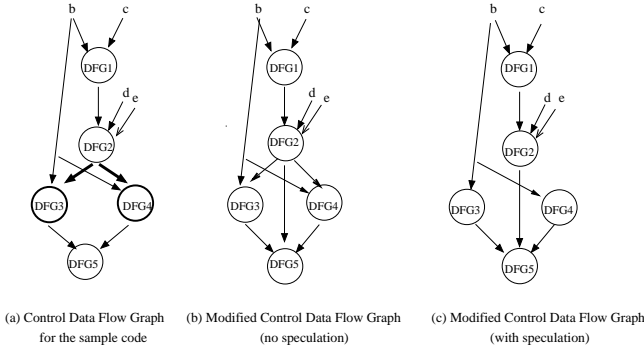
Figure 2. A sample code.

### 2.4. Finding Bounds for Schedules of Control/Data Flow Graphs

In the following, we discuss simple methods to handle control nodes. Consider the simple code fraction given in Figure 2. The CDFG model of this computation depicted in Figure 3(a) is partitioned into regions which constitute individual DFGs. The bold arrows in the CDFG represent the control dependency, i.e., the actual computation uses the result of either *DFG3* or *DFG4* depending on the result of the computation inside *DFG2*. One way to take this relationship into account is to impose a constraint to execute the conditionals first and then start the execution of the selected branch. The transformation for this type of execution is illustrated in Figure 3(b). Additional data dependency edges are created between the primary outputs of *DFG2* and the branch nodes *DFG3* and *DFG4*. The result of the conditional evaluation is also transferred to the common node *DFG5* to enable the correct multiplexing of input data. Another way is to allow speculative execution by starting execution of operations from branch paths while evaluating the condition. A possible transformation to handle this removes the control dependencies between the control node *DFG2* and its successors in the CDFG. Instead dependencies are added from *DFG2* to the node following the speculative branches i.e., *DFG5* as shown in Figure 3(c). This means that nodes *DFG2*, *DFG3*, and *DFG4* are allowed to execute concurrently, whereas node *DFG5* can start executing strictly after the evaluation of conditional node *DFG2* has finished. This speculative execution model was suggested by Radivojevic and Brewer [12] for OBDD-based scheduling. Once a CDFG is transformed using either of the methods, same lower/upper bound techniques can be used to obtain bounds in the presence of control nodes. The additional dependency constraints imposed on the CDFG by the transformations can, however, lead to suboptimal solutions for the original problem.

Benchmark	MUL ALU	Variables (Original) ILP & SAT	Variables (Reduced) ILP & SAT	Clauses (Original) SAT	Clauses (Reduced) SAT
EWF	1 1	1904	872	46148	10217
	1 2	2244	696	84470	8392
FDCT	2 2	3528	3097	140972	77940
	3 3	3780	2917	136396	59028
ARF	2 2	2352	1288	62891	24444
	2 1	1764	966	52791	17100
FFT (MediaBench)	1 1	2660	2024	73121	42167
	2 2	3040	1784	62640	33390
Convolve (MediaBench)	MUL ALU DIV 1 2 1	1224	956	10463	8622
	1 1 2	936	792	6974	3594
Noise_est (MediaBench)	MUL ALU DIV 1 1 1	672	528	4528	2841
	2 2 1	720	480	6452	3110

**Table 1. The number of the variables for the ILP formulation and the number of the variables and clauses for the SAT formulation before and after applying the lower bound technique.**



**Figure 3. CDFG representation of the sample code before and after the control dependency transformations.**

### 3. Experimental Results

In this section we present experimental results to demonstrate the improvements on the run-time of a state-of-the-art academic SAT-solver, *Chaff* [9]. To provide a thorough comparison we also present the impact of the same lower bound techniques on the commercial ILP solver CPLEX.

In Table 1 the number of variables in the ILP formulation and the number of variables+clauses in the SAT formulation are given, both with and without using the lower and upper bound techniques. ILP and SAT are the cases with no bounding techniques applied. ILP-accel and SAT-accel refer to the case where the lower and upper bounds are applied to improve the formulation. For the SAT-based approach several problem instances may have been created

during the search for the optimal solution. Here we present the sizes of the models generated for the first instance of the Upper Bound Search. For the ILP formulation there was not a significant change in the number of constraints, however the constraints in the reduced model contained significantly fewer variables.

Experimental results obtained for a collection of DFGs are summarized in Table 2. DFGs such as EWF, FDCT, and ARF are representative DSP applications. We have also included DFGs (FFT, Convolve, Noise\_est) extracted from two applications in the MediaBench multimedia benchmark suite [7]. The second column gives the DFG sizes in terms of the number of nodes (N) and edges (E). Third column shows the resource set used for each problem instance. We assumed 1 cycle latency for the ALU, 2 cycles for the multiplier, and 4 cycles for the divider. In the fourth column optimal solutions to each instance found by the exact solvers is depicted. The rest of the table presents the run-time results for the ILP solver and the SAT solver before and after utilizing the lower and upper bound techniques. The improved models are referred to as accelerated (accel). For this set of experiments we have applied the Upper Bound Search. Given an upper bound, a linear search through available number of control steps is performed.

As depicted in Table 2, the SAT-based approach finds the optimal solution faster than the ILP approach in almost all cases. In two cases the SAT solver was not able to find the solution within one hour of CPU time limit just like the ILP solver. For the rest of the cases, we observe that the SAT-based approach can be as much as 59 times faster (first row in Table2) than the ILP solution. Note that we applied the same bounds on the ILP models to enable a fair comparison. Also, note that we used *Chaff* as a black box; we did

Benchmark	DFG Size (N/E)	Resources MUL - ALU	Opt. Latency	ILP (sec)	ILP-accel (sec)	SAT (sec)	SAT-accel (sec)
EWF	34/46	1 1	28	1217	6.49	18	0.11
		1 2	21	-	3.51	25.61	0.6
FDCT	42/52	2 2	18	-	-	162.2	20.06
		3 3	14	-	-	345.7	256.1
ARF	28/30	2 2	18	-	-	1316	37.29
		2 1	18	-	-	57.42	42.95
FFT (MediaBench)	42/52	1 1	25	-	-	-	2500
		2 2	17	-	-	489.64	148.74
Convolve (MediaBench)	18/10	MUL ALU DIV 1 2 1	12	1	0.38	0.44	0.29
		1 1 2	12	1.53	0.71	0.77	0.39
Noise_est (MediaBench)	16/8	MUL ALU DIV 1 1 1	13	0.47	0.25	0.06	0.03
		2 2 1	9	0.09	0.07	0.07	< 0.01

**Table 2. Comparison of Chaff and CPLEX. “-” indicates that the problem could not be solved within one hour.**

not use problem specific variable ordering. Further speedup can be achieved by finetuning the SAT solver for scheduling problem.

#### 4. Conclusions and Future Work

In this paper we have explored the potential of satisfiability to be used as a tool for modeling and solving scheduling problem. We have further investigated the impact of lower and upper bound analysis on the problem size, and hence run-time of the satisfiability solver. We have presented experimental results on the performance of the SAT solver Chaff compared against an ILP solver. We have demonstrated that the SAT solver presents a competitive alternative as a tool to find optimal solutions to the NP-Complete resource constrained scheduling problem.

For a complete comparison of alternative exact solution strategies, we need to consider BDD-based methods as well. We can also enhance our models to handle additional features such as pipelining and loops.

#### References

- [1] S. Chaudhuri, R.A. Walker, and J.E. Mitchell, “Analyzing and Exploiting the Structure of the Constraints in the ILP-Approach to the Scheduling Problem”, IEEE Transactions on Very Large Scale Integration (VLSI) System, 2(4) pp. 456-471, December 1994.
- [2] C. H. Gebotys, “Optimal Scheduling and Allocation of Embedded VLSI chips”, Design Automation Conference, 1992.
- [3] C. H. Gebotys, M. Elmasry, “Global Optimization Approach for Architectural Synthesis”, IEEE Trans. CAD/ICAS,12(9), pp. 1266-1278, Sep. 1993.
- [4] S. Haynal, F. Brewer, “Efficient Encoding for Exact Symbolic Automata-Based Scheduling”, IEEE International Conference on Computer-Aided Design, 1998.
- [5] M. Langevin, E. Cerny, “A recursive technique for computing lower-bound performance of schedules”, ACM Transactions on Design and Automation of Electronic Systems (TODAES), 1(4):443-456, October 1996.
- [6] J. H. Lee, Y. C. Hsu, Y. L. Lin, “A New Integer Linear Programming Formulation for the Scheduling Problem in Data Path Synthesis”, Design Automation Conference, 1989.
- [7] C. Lee, M. Potkonjak, W. H. Mangione-Smith, “MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems,” International Symposium on Microarchitecture, IEEE Micro-30, 1997.
- [8] J. P. Marques-Silva, K. A. Sakallah, “Boolean Satisfiability in Electronic Design Automation”, Design Automation Conference, 2000.
- [9] M. W. Moskewicz, C. F. Madigan, Y.Zhao, L. Zhang, S. Malik “Chaff: Engineering an Efficient SAT Solver”, Design Automation Conference, 2001.
- [10] M. Narasimhan, J. Ramanujam, “On Lower Bounds for Scheduling Problems in High-Level Synthesis”, Design Automation Conference , 2000.
- [11] M. Narasimhan, J. Ramanujam, “Improving the Computational Performance of ILP-based Problems”, International Conference on Computer Aided Design, 1998.
- [12] I. Radivojevic, F. Brewer, “Incorporating Speculative Execution in Exact Control-Dependent Scheduling”, Design Automation Conference, San Diego, CA, June 1994.
- [13] M. Rim, R. Jain, “Lower Bound Performance Estimation for the High-level Synthesis Scheduling Problem”, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 13(4), pp. 451-458, 1994.