

Reducing Compilation Time of Zhong’s FPGA-based SAT solver

Pak K. Chan, M.J. Boyd, S. Goren, K. Klenk, V. Kodavati, R. Kundu, M. Margolese,
J. Sun, K. Suzuki, E. Thorne, X. Wang, J. Xu, M. Zhu
Department of Computer Engineering
University of California, Santa Cruz, California, 95064 USA

Abstract

We present schemes to reduce the compilation time of configurable hardware that solves Boolean Satisfiability. The SAT solver presented by Zhong in last year’s FCCM conference has a large compilation time overhead mainly due to placement and routing of many FPGAs. We attack the problem on 3 fronts. First, we partitioning the SAT solver into instance-specific and instance non-specific components. Secondly, we transform SAT instances to a canonical form; and finally we present a board-level architecture to solve large SAT instances. All these efforts amount to a reduction in placement and routing time to configure the configurable hardware. We are able to reduce the compilation time to mere routing time of the implication circuits for each instance of the SAT problem, given the best scenario.

1 Introduction

In last year’s FCCM conference Zhong et al presented an architecture for an FPGA-based SAT solver [1]. The SAT solver has the potential to be a “killer application” for reconfigurable computing. Zhong’s SAT solver is efficient and well suitable for FPGA realization. However, the “speedup” of Zhong’s SAT solver is nullified by the compilation time. The compilation time for each instance of the SAT problem includes placement and routing time of many FPGA chips that take many hours. In this article, we propose schemes to reduce this compilation time.

The problem of reducing compilation time was solved within a classroom setting with 12 students. Students worked in groups of two. The instructor posed the challenge, gave hints and outlined the points of attack. Groups had 5 weeks to digest the problems, and discuss potential solutions. To prove the concept,

each group implemented and realized an FPGA-based SAT solver observing the points of attack.

It is also interesting to note that von-Neumann machines are ineffective in solving some hard SAT problems. For example, a 4 CPU SGI Onyx Reality Engine 2 was unable to solve a problem of 577 variables and 32177 literals after 1 hour [2].

1.1 Boolean Satisfiability (SAT)

We outline of the problem of Boolean Satisfiability [3].

Instance: A set of variables $X = \{x_1, x_2, \dots, x_n\}$ and a formula in Conjunctive Normal Form (i.e., a collection C of clauses, each clause consists of a disjunction of literals). A literal is a variable or its negation. A *truth assignment* is a set of Boolean values for the variables, and a satisfying assignment is a truth assignment that causes the formula to evaluate to 1. So each clause must have at least one literal evaluating to 1.

The Satisfiability Question: Is there a satisfying truth assignment for C ?

Example 1: $X = \{x_1, x_2\}$.

$$C : (x_1 + \bar{x}_2) \cdot (\bar{x}_1 + x_2)$$

Answer: Yes. $x_1 = x_2 = 1$.

1.2 Zhong’s SAT solver

The Davis-Putnam procedure (less known as Davis-Logemann-Loveland) [4] is a generic backtracking brute-force search that systemically enumerates the space of 2^n possible binary assignments to the n Boolean variables. This is the basis of Zhong’s SAT solver.

Zhong’s SAT solver [1] consists of a chain of search blocks, one for each variable, as shown in Figure 1. The search order of the variables is predetermined

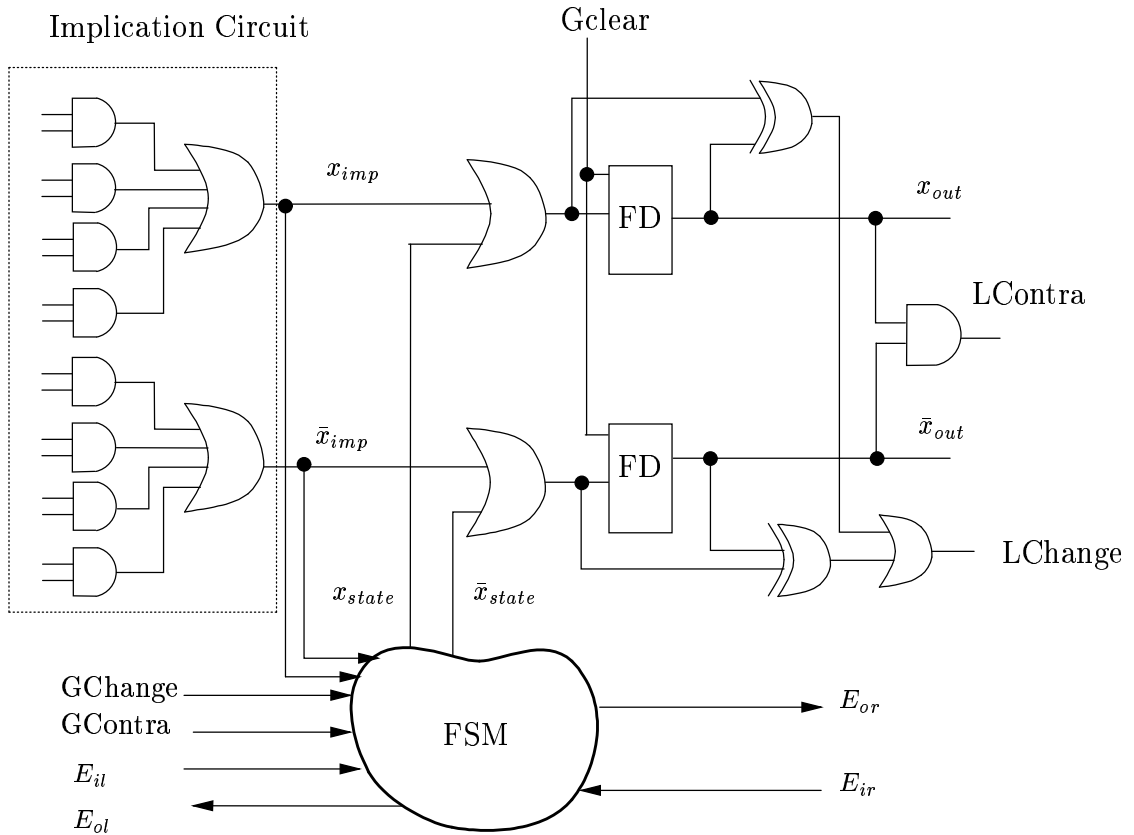


Figure 1: Zhong's Search Block for variable x .

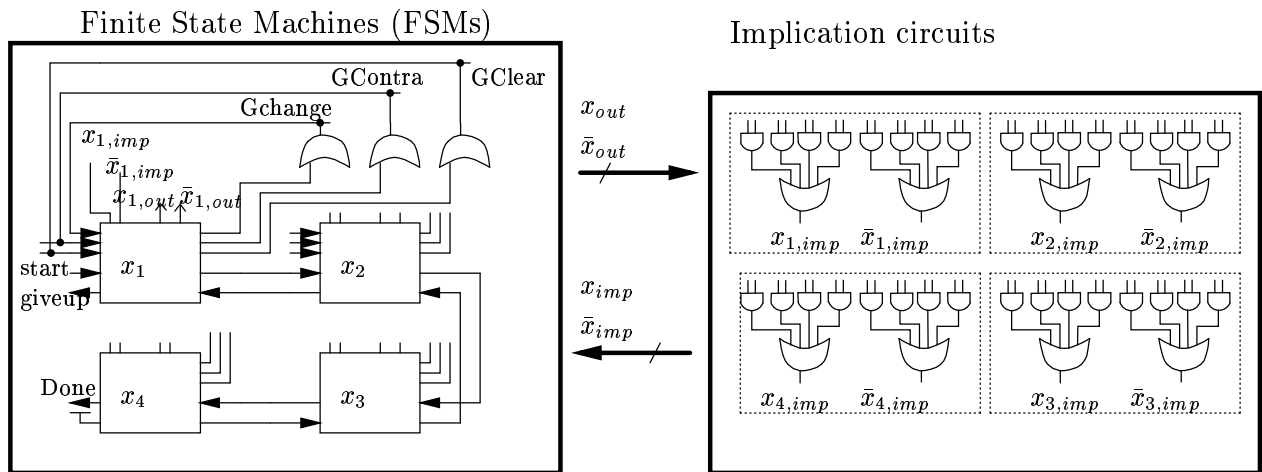


Figure 2: Our improved SAT solver: partitioning the search blocks into FSMs and implication circuits. Implication circuits have the same canonical form.

ahead of time. The assignment of value to a variable is encoded in 2 bits as (x_{out}, \bar{x}_{out}) , with (0,0), (1,0), (0,1), (1,1) representing unassigned, 1, 0, and contradiction, respectively. Each block consists of an implication circuit for a variable x . The rest of the search block is largely a FSM, and logic to detect a *contradiction* and a *change* in the variable’s value. Only one search block is active at any time. Forward search activation comes from left E_{il} and passes on to the right with E_{or} . A FSM is only activated if the value of its variable is not yet implied. Otherwise, it passes control to the next FSM by asserting E_{or} . When active, a FSM machine asserts assignment 1 to its variable. If there is no (global) contradiction, then control will pass forward to the next FSM. If there is a contradiction, then it will try assignment 0 and repeat the implication step. Now if there is contradiction with assignment 0, then control will be pass backward to the FSM upstream by asserting E_{ol} .

This organization is very effective. Although only one FSM is active at any time, the strength of this SAT solver lies in the parallel execution of many implication circuits.

However, the compilation time starting from a set of Boolean clauses to Xilinx FPGA bit streams is very long. In Zhong’s approach [1], a SAT instance represented in *cnf* is translated to VHDL code. The VHDL code is synthesized and then the VirtualWire compiler is used to compile to many Xilinx design files. These design files are then placed and routed by Xilinx *ppr*. The last step is the most time consuming part of the compilation process.

1.3 Compilation time reduction

In this article, we propose schemes to reduce the compilation time involved in Zhong’s SAT solver. We attack the problem on three fronts, outlined below and detailed in later subsections.

For the lack of a better term, we simply call the non-implication circuit part of a search block “FSM” in this article. Each block consists of a FSM and an implication circuit for a variable of the SAT problem instance. For a given variable x , the *support* of its implication depends on the number of variables that appear with x in the same clauses. The implication circuit is instance specific, but the FSM is not.

Our method of attack involves:

1. An improved architecture of Zhong’s SAT solver. In the new architecture, the FSM and implication circuit are on different chips. Since all the FSMs

are instance independent, we need to place and route each FSM chip once and for all. The implication circuits have a particular template, placement is done once and for all, but routing between the modules is instance specific. In essence, there is no instance specific placement. We reduce the compilation time to mere routing time of the implication circuits for each instance of the problem.

2. To achieve the goal outlined above, we need to transform quickly a general SAT instance to a canonical SAT instance. A classical canonical SAT instance is 3-SAT, where each clause has exactly 3 literals. A restricted form of 3-SAT is 3, s -SAT, in which the occurrences of each variable in all clauses has an upper bound of s . Tovey [5, 6] shows a linear-time transformation from 3-SAT instance to 3, s -SAT.
3. A new board-level architecture is proposed to solve large 3, s -SAT instances.

The details are given in the next sections.

2 An Improved SAT engine

Figure 1 shows the organization of the search block for variable x , according to Zhong et al. It is shown in the next subsection that by partitioning the search block into two separate parts (FSMs and implication circuits), a reduction of compilation time can be achieved. We remind the reader that for the lack of a better term, we simply call the non-implication circuit part of a search block “FSM” in this article.

2.1 Partitioning the SAT engine

The basic idea is to separate the instance-specific and instance-non-specific components. The FSMs are instance-non-specific, so they are pre-placed once and for all. For a given problem instance of n variables, where n is less than the maximum number of FSMs, all that is needed is to keep the unused FSMs in their initial states by coercing implied values. An unused FSM will automatically pass control to its neighbor.

Only the implication circuits are instance specific. The illustration in Figure 2 shows that a single chip is delegated to implement *all* the implication circuits. A general scheme will be presented in a later section. As we shall demonstrate in the next subsection, the *logic* for each implication circuit can be instance non-specific. However, routing between the implication circuits are instance specific. The compilation time for

each instance of the SAT problem amounts to routing time between the implication circuits.

2.2 Normalizing SAT instances to 3, s -SAT

We stated earlier that the logic for each implication circuit can be instance non-specific. It is achieved by normalizing the instance to 3-SAT, in which at most 3 literals will appear in each clause, as illustrated in the next example.

Example 2:

$$C : (a + b + c) \cdot (\bar{a} + d + \bar{e}) \cdot (a + e) \cdot (\bar{a} + \bar{c})$$

The value of variable a can be implied from

$$a_{imp} = \bar{b}_{out} \cdot \bar{c}_{out} + \bar{e}_{out}$$

and its complement:

$$\bar{a}_{imp} = \bar{d}_{out} \cdot e_{out} + c_{out}$$

Both of the above expressions are in Sums Of Product form. Furthermore, if we restrict the occurrences of each literal (a variable or its negation) to s times, then each implication circuit has a canonical SOP form. For example, the implication circuit shown in Figure 2 supports $s = 4$.

The transformation from SAT to 3-SAT can be achieved in linear time. A normalization procedure is given in [3], pages 48–49. Additional dummy Boolean variables are introduced during the transformation. Since the values of the dummy Boolean variables can be implied from the values of the original variables of the problem, it is advantageous to append the FSMs of the dummy variables at the end of the FSM chain. Hence, the introduction of the new dummy Boolean variables doesn't incur additional runtime.

The transformation from 3-SAT to 3, s -SAT can be achieved in linear time [5]. The penalty is that new variables are needed for the restriction. If a variable (or its negation) appears $p > s$ times in all the clauses, $\lceil \frac{p}{s-2} \rceil - 1$ new variables and $\lceil \frac{p}{s-2} \rceil$ new clauses and will be introduced. So there is a tradeoff between the value of s and the size of the SAT machine and its performance. In some real problems, s can be as large as 600.

It is conceivable that many Implication-Circuit chip templates can be generated ahead of time, all pre-placed. Each template has many implication circuits, observing the SOP canonical form but with different fan-ins to support variety of variable occurrences s .

The FSM and the implication circuit constitute the bulk of logic in each search block in Figure 1. Using one-hot encoding, each FSM can be implemented using 8 Xilinx XC4000 CLBs. For $s = 6$, each implication circuit requires 3 CLBs. The number of CLBs required to implement each implication circuit increases with s .

2.3 Overcoming pin limitations: muxing and demuxing

Figure 2 illustrates the dataflow between the FSMs and the implication circuits. Inputs to the implication circuits are the outputs from the FSMs. The majority of inputs to the FSMs are from the implication circuits. The FSMs generate the x_{out} and \bar{x}_{out} of all the variables. The implication circuits take them as inputs and generate the implied values of the variables, which will be used by the FSMs to generate x_{out} and \bar{x}_{out} for the next round. So the connection between the implication circuits and the FSMs can be a shared bus, as illustrated in Figure 3.

The number of I/O pins can be reduced using pin multiplexing, but with degraded performance. Let the degree of pin multiplexing be m , and n be the number of Boolean variables. For $m = 2$, we multiplex x_{out} with \bar{x}_{out} (and x_{imp} with \bar{x}_{imp}), and the pin requirement between the FSM and implication circuit is n . With the introduction of multiplexing and demultiplexing logic, each FSM will require 10 CLBs to implement. A counter will also be required to disable the FSMs during multiplexing and demultiplexing.

2.4 Board-Level Architecture

Figure 4 shows a multiple-chip organization of our improved SAT solver. All the chips on the left column are FSMs, whereas the implication circuit chips are on the right. FSM i can accommodate n_i variables, so the total number of variables is $n = \sum n_i$. The outputs from each FSM chip are broadcast to all implication circuits. Obviously, this scheme is limited by the number of I/O pins available on each Implication-Circuit chip. Depending on the degree of multiplexing, m , each Implication-Circuit chip requires $2n/m$ pins.

For example, given FPGA chips with 500 I/O pins each and sufficient logic capacity, a 1000-variable SAT solver can be realized using 4 chips with degree of multiplexing 4. Each FSM chip should have at least 5000 CLBs. The logic capacity of each Implication-Circuit chip depends on the occurrences factor, s .

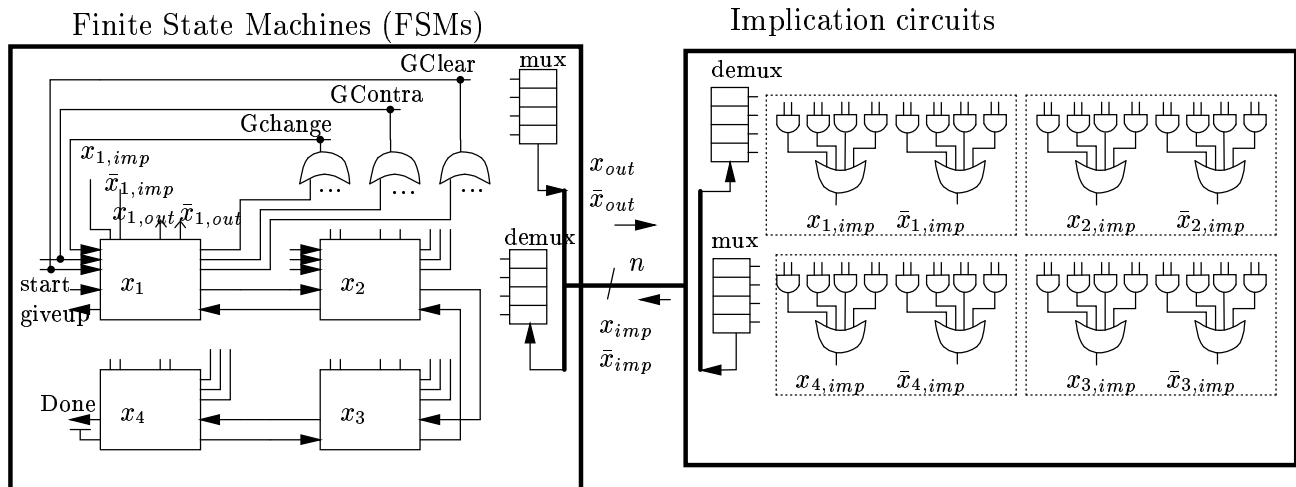


Figure 3: Pin multiplexing and communicating on a shared bus of width n .

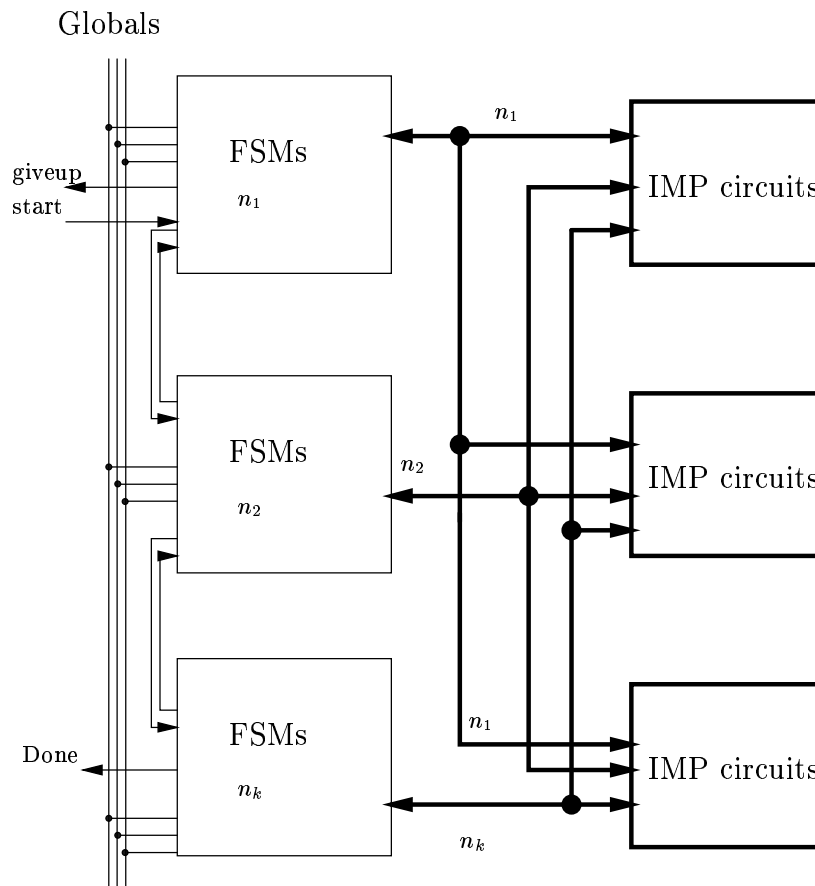


Figure 4: Board-Level architecture for our improved SAT solver.

2.5 Generating the template, from CNF to FPGA design files

There are many options regarding the compilation from a 3-SAT instance to Xilinx design files. Boolean formulae for software SAT solver are typically represented in CNF format, so we choose CNF files as our starting point. Recall that only the Implication-Circuit chips are instance specific, which amounts to translating an CNF instance to routing information for the Implication-Circuit chips.

1. This option doesn't require *any* instance-specific placement. This involves translating CNF to Xilinx LCA format. A net is represented by

```
Addnet netname pin1 pin2 ... pink
```

We could append the instance-specific routing information to a template (pre-placed) Implication Circuit LCA file and route the LCA using our XC4000 router. But our tool only supports up to XC4025 device.

2. This option involves using Xilinx internal tools and Xilinx M1 placement and routing tool `par`.
3. This involves translating a CNF file to Xilinx XNF format. This option involves *some* placement, but it is the easiest to achieve because XNF is a well-documented and is in the public domain. The *relative* placement of the implication circuits are specified using `rlocs`, and the mappings are specified using `HMAP` and `FMAP` guides.

This approach has the additional advantage that many templates for different variable-occurrences factor s can be generated ahead of time. The templates specify the logic gates, the mapping guides, and the relative placement guides. The translator selects a different template according to the occurrences that a particular variable in the SAT instance.

3 Implementation and Results

Within a classroom setting, two fully functional FPGA-based 32-variable SAT solvers were constructed using the principles outlined earlier in the paper. Most importantly, the FSM chips were kept static, and only the Implication-Circuit chips were generated for different instances of the SAT problem. We used Borg prototyping boards [7] to realize all the

SAT solvers, since the Borg boards fit the board-level architecture as illustrated in Fig. 4.

In particular, one FPGA-based engine uses one XC4002APC84-6 and two XC4005PC84-6s. All the FSMs are in the XC4005 chips, the implication circuits are in the smaller XC4002A. The larger XC4005 chips are instance-independent and are 85% utilized, each holds FSMs for 16 variables. They are pre-placed and pre-routed *once and for all*.

Translators were written to convert CNF files to Xilinx XNF format with embedded relative placement and routing information. The Implication-Circuit chip is placed and routed using `ppr`.

The system clock is 20 MHz using degree of pin-multiplexing 2. The SAT solver communicates with a conventional computer via an I/O port, and it reports unsatisfiability or satisfiability of the problem instance. In the latter case, each variable's assignment is shifted out and displayed on the screen. A solution of a problem instance is illustrated below.

```
PCTEST Demo. Needs design file v32pc2.mcs
```

```
BORG PORT address is 0x300
Done input is 6
Problem          SATISFIED.
Output from port is hex 0
Output from port is hex 1
Output from port is hex 19
Output from port is hex 22
Output from port is hex 22
Output from port is hex 22
Output from port is hex 22
```

```
It took 281 clock cycles to determine the answer
This is equivalent to 0.000035 seconds.
```

```
The variables have the following values:
```

v 0	v 1	v 2	v 3	v 4	v 5	v 6	v 7
0	1	0	0	0	1	0	0
v 8	v 9	v10	v11	v12	v13	v14	v15
0	1	0	0	0	1	0	0
v16	v17	v18	v19	v20	v21	v22	v23
0	1	0	0	0	1	0	0
v24	v25	v26	v27	v28	v29	v30	v31
0	1	0	0	0	1	0	0

Note that the SAT solver also reports the number of clock cycles that is required to solve the problem.

4 Extrapolation of Result

To estimate the impact of our ideas of compilation time reduction, we extrapolate our results to the

DIMACS benchmark `hole10`. This benchmark has 110 variables, 561 clauses, and an occurrence factor s of 11. According to Zhong [1], it took the fastest software solver GRASP 8 hours of CPU time to solve this problem. Zhong’s SAT solver completes this problem in 566 seconds. However, Zhong’s SAT solver uses 10 XC4013E devices, each has to be placed and routed for this problem instance. The most “difficult” chip took 2400 seconds to place and route.

Now we estimate the improvement attainable using our scheme. The DIMACS benchmark `hole10` is not in 3-SAT canonical form, so we normalize it to 3-SAT, and 77 new dummy variables are introduced. Now we have a 187-variable problem to solve. Let’s use a board-level architecture as illustrated in Figure 4, and populated with XC4013s. Each XC4013 has 576 CLBs, each FSM can be implemented with 10 CLBs. We would use four XC4013 devices for the FSMs, each XC4013 would hold FSMs for 47 variables, and all pre-placed and pre-routed. The instance-specific implication circuits could be implemented with a single XC4013 chip. As mentioned earlier in Subsection 2.4, each Implication-Circuit chip requires $2n/m$ pins. But with $n = 187$ variables and degree of pin-multiplexing $m = 2$, we need a 187-pin XC4013 chip. Rather conservatively, we assume that the Implication-Circuit chip would take 2400 seconds to place and route. We obtain a speedup of over 80, including the place and route compilation time. Actual place and route time is (861 seconds) 14 minutes and 21 seconds for this implication circuit chip using a Pentium 120 MHz DOS machine.

5 Conclusions

We have presented schemes to reduce the compilation time of configurable hardware that solves Boolean Satisfiability. We discussed schemes to reduce the compilation time to mere routing time of the *implication circuits* for each instance of the SAT problem. To prove the concept, small-scale Xilinx FPGA-based 32-variable SAT solvers were successfully constructed within a classroom setting.

6 Acknowledgments

The authors have benefited from discussions with Professors Martine Schlag and Allen Van Gelder. The first author also acknowledges support from Xilinx.

References

- [1] P. Zhong, M. Martonosi, P. Ashar, and S. Malik, “Accelerating Boolean Satisfiability with configurable hardware,” in *1998 IEEE Workshop on FPGAs for Custom Computing Machines*, pp. 186–195, 1998.
- [2] F. Okushi, “Parallel cooperative propositional theorem proving,” *Annals of Mathematics and Artificial Intelligence*, vol. ??, pp. ??–??, 1998.
- [3] M. R. Garey and D. S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*. San Francisco, CA: W. H. Freeman and Company, 1979.
- [4] M. Davis, G. Logemann, and D. Loveland, “A machine program for theorem-proving,” *Communications of the ACM*, vol. 5, pp. 394–397, July 1962.
- [5] C. A. Tovey, “A simplified NP-complete satisfiability problem,” *Discrete Applied Mathematics*, vol. 8, pp. 85–89, 1984.
- [6] O. Dubois, “On the r, s -sat satisfiability problem and a conjecture of Tovey,” *Discrete Applied Mathematics*, vol. 26, pp. 51–60, 1990.
- [7] P. K. Chan, M. Schlag, and M. Martin, “BORG: A reconfigurable prototyping board using Field-Programmable Gate Arrays,” in *Proceedings of the 1st International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, (Berkeley, California, USA), pp. 47–51, Feb. 1992.