# Hardware/Software Implementation of FPGA-targeted Matrix-Oriented SAT Solvers

Valery Sklyarov, Iouliia Skliarova, Bruno Pimentel, Joel Arrais

University of Aveiro, Department of Electronics and Telecommunications, IEETA,
3810-193 Aveiro, Portugal
{skl, iouliia}@det.ua.pt, {a21394, a21540}@alunos.det.ua.pt

**Abstract.** The paper describes two methods for the design of matrix-oriented SAT solvers based on data compression. The first one provides matrix compression in a host computer and decompression in an FPGA. It is shown that although some improvements have been achieved in this case, there exists a better solution. The second method makes possible to execute operations required for solving the SAT problem over compressed matrices.

## 1 Introduction

The complexity of FPGAs is not always sufficient for implementing SAT solvers and the required resources have to be partitioned between software running on a general-purpose computer and hardware. This involves multiple data exchange, which is either costly or time consuming. To overcome this drawback the following two methods have been explored: 1) the technique for matrix compression/decompression permitting to reduce the size of matrices in software, to transmit them to an FPGA and to restore the original matrices in hardware; 2) matrix transfer in a relatively simple compressed form and solving in hardware a 3-SAT problem [1, 2] over the transmitted matrices avoiding the decompression step.

## 2 Matrix compression/decompression techniques

It is known that the SAT problem can be formulated over different models [3] and we will consider for such purposes ternary matrices [4, 5]. They have been chosen because of the reasons reported in [4, 5]. Fig. 1 demonstrates how the considered technique for matrix compression/decompression has been applied. This technique is useful if the following conditions are valid:

- Additional hardware that is necessary to implement the decompressing circuits is reasonable. The latter can be estimated as follows. Let us assume that FPGA resources can be defined as $R_{FPGA}$, the resources needed to handle matrices with n(2n) columns are $R_n$($R_{2n}$). If $R_n \le R_{FPGA} < R_{2n}$ and after implementing the SAT solver for an n-column matrix the remaining part of the FPGA (i.e. $R_{FPGA}$-$R_n$) is

sufficient to build the decompressing circuits, then additional hardware is reasonable. The values n and 2n are considered because we assume that matrices are kept in FPGA embedded memory blocks with reprogrammable numbers of inputs and outputs. Incrementing an address size (i.e. adding one input) causes the number of the available memory block outputs to be reduced by a factor of 2.

- Let T be the time of data transfer without the use of compression/decompression and the following expression (see Fig. 1) is satisfied: $T>(T_t+T_h)$.

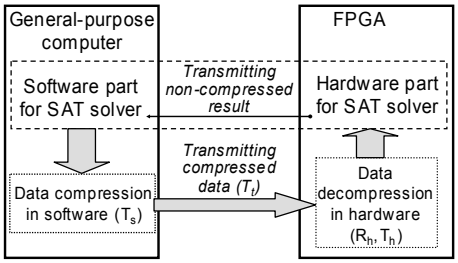


Figure 1. Using compression/decompression technique for data exchange between hardware/software parts of SAT solver

In order to validate these conditions the following technique has been applied.

- Matrix compression and decompression have been employed for the hardware/software SAT solver considered in [5].
- Data compression/decompression have been provided with the aid of slightly modified Huffman coding, which a) settles repeating coefficients for matrix *don't care* values because as a rule the number of *don't cares* is significantly greater than the number of *ones* and *zeros*; and b) is based on effective recursive procedures considered in detail in [6].

Table 1 demonstrates the results of experiments (for data exchange through the Ethernet for RC200 board [7]), which make possible to estimate the advantages and disadvantages of the technique considered in Fig. 1 for different matrices **M**(n×m), where n is the number of columns and m is the number of rows. A ratio is defined as the size of compressed data divided by the size of non-compressed data and R is the percentage of FPGA resources required for the decompressing circuit. Note that the compression in software can be done in parallel with executing the SAT-solver algorithm in hardware (that is why the last column shows just the time $T_h+T_t$).

Table 1. The results of experiments for the circuit in Fig. 1

| **M** (n × m) | T (ms) | R | Ratio | $T_t$ (ms) | $T_s$ (ms) | $T_h$ (ms) | $T_h+T_t$ (ms) |
|---|---|---|---|---|---|---|---|
| 128×128 | 0.903 | 12% | 0.130 | 0.117 | 20 | 0.943 | 1.060 |
| 256×500 | 3.310 | 12% | 0.139 | 0.460 | 220 | 3.800 | 4.260 |
| 256×256 | 3.310 | 12% | 0.105 | 0.347 | 60 | 3.715 | 4.062 |
| 256×1000 | 12.940 | 12% | 0.087 | 1.126 | 230 | 15.380 | 16.506 |
| 256×1500 | 19.260 | 12% | 0.077 | 1.483 | 872 | 23.024 | 24.507 |

An analysis of Table 1 has shown that independently of the good ratio for data compression the total time is increased, i.e. for all the examples $T_h+T_t>T$. Note that the Ethernet-based data exchange is very fast and in case of using other interfaces the compression permits the total time to be shortened. For example, in case of parallel interface the values T and $T_t$ will be increased approximately in 10 times and consequently for all the examples $T_h+T_t<T$. However, we cannot provide significant improvements.

## 3 Executing algorithms over compressed ternary matrices

In accordance with [5] a matrix-oriented SAT solver executes operations over rows/columns of a ternary matrix applying the following set of rules:

1. If a column contains just *don't care* values it must be deleted from the matrix.
2. All rows that are orthogonal to an intermediate vector **w** (that incrementally forms a solution) must be removed from the matrix. All columns that correspond to the components of the vector **w** with values 1 and 0 must be deleted from the matrix.
3. If the matrix contains a row with just one component 0 (1) with an index i then the element i of the vector **w** must be assigned the value 1 (0), i.e. the inverted value.
4. If there is a column j in the matrix without values 1 (0) then the element j of **w** can be assigned the value 1 (0).

The considered SAT algorithm implemented in hardware is depicted in Fig. 2. On the one hand the majority of the involved operations are similar to [5]. On the other hand the algorithm has a number of distinctive features which make possible the required hardware resources to be reduced without a degradation of performance. These features are the following:

1. The rule 1 was avoided because it consumes time but does not simplify the operations over matrix rows/columns.
2. Instead of dynamic selection of the next decision variable (column), a static selection has been employed in such a way that all the columns have been sorted by the number of *non-don't care* values in an ascending sequence and the selection has been performed from the first to the last matrix column. Our experience has shown that such predefined sequence minimizes the required FPGA resources and does not reduce performance (in hardware).
3. Any matrix is addressed in the memory by rows, which means that any row can be read/written in one clock cycle.
4. The rules 2 and 3 are sequentially checked for all matrix rows starting from the first row (see Fig. 2). Note that the relevant to the rules 2 and 3 operations can be executed in parallel over any complete row.
5. During sequential operations over rows a vector, which identifies columns containing just *ones* and *don't cares* (or *zeros* and *don't cares*), is incrementally constructed. It permits the rule 4 to be applied after all rows have been examined (see Fig. 2). Thus the matrix transpose is no longer required.

All the other operations are exactly the same as in [5] and we will not replicate them to keep the description short. Note that any compression technique leads to non-equal sizes of different rows (vectors) and this conducts to irregularity of different SAT solver blocks, such as the matrix memory, the combinational circuit, etc. To cope with this problem the following approach has been employed.

1. The software transforms any matrix that is going to be dispatched to an FPGA in such a way that all the matrix rows contain not more than three *non-don't care* values. It is known that such a technique is called 3-SAT [2] and the respective transformation can be done in polynomial time.
2. Any *non-don't care* value (i.e. any *one* or *zero*) is coded by its index followed by the value. For example, the vector [0-------1-1----] is coded as 0000 *0* 1000 *1* 1010 *1*.
3. If the vector has less than 3 *non-don't care* values then the flag containing all *ones* in the respective code is used. For example, the vector [----------] can be coded as 1111 0 1111 0 1111 0. Thus an r-bit code can be used for any matrix, which has no more than $2^r$-1 columns and the number of matrix rows is limited just by available FPGA resources.

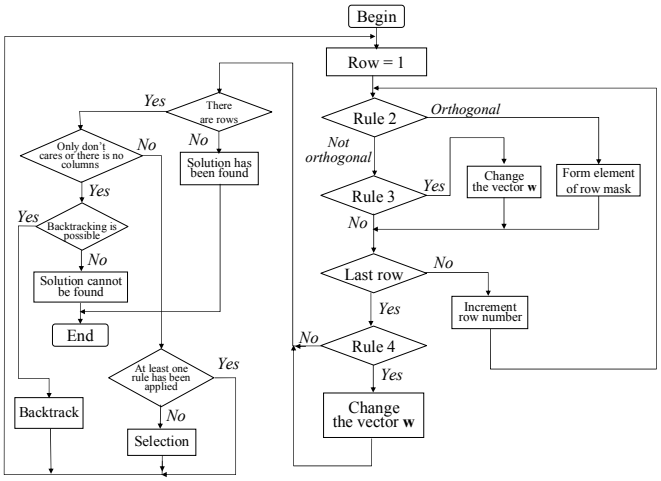


Figure 2. The SAT algorithm implemented in hardware

## 4 The results of experiments and implementation details

A SAT solver, which implements the algorithm in Fig. 2 has been designed in DK2 environment [7] from specification in Handel-C and implemented in Xilinx Virtex-II XC2V1000 FPGA (Celoxica RC200 prototyping board). Mapping, placement, routing and generating the bit-stream for FPGA from an EDIF file created by DK2 have been performed in ISE 6.2.2 of Xilinx. The implemented in FPGA circuits permit to process in hardware matrices containing up to 255 columns and 1500 rows. The clock frequency was set to 45 MHz. As we can see from data in Table 2 the considered

circuit has a high performance. In all the examples we have used randomly generated 3-SAT formulae.

Table 2. The results of experiments with the compressed-matrix-oriented SAT solver

| Matrix (n × m) | The result | Time for solving the problem in FPGA (s) | % of the used FPGA resources |
|---|---|---|---|
| 127 × 128 | Satisfiable | 0.00087 | 30 |
| 127 × 500 | Unsatisfiable | 0.127 | 30 |
| 255 × 256 | Satisfiable | 0.00357 | 54 |
| 255 × 1000 | Unsatisfiable | 0.195 | 54 |
| 255 × 1500 | Unsatisfiable | 0.264 | 54 |

## 5. Conclusion

One of the problems inherent to matrix-oriented SAT solvers is the relatively high volume of data that have to be transferred from a host computer to the accelerator, especially in the case of partitioning the problem between general-purpose software and hardware. Due to the complexity of practical SAT problem instances this partitioning is very common. To reduce the influence of data exchange on the total time of computations, two methods have been explored and analyzed.

## Acknowledgment

This work was partially supported by the Portuguese Foundation of Science and Technology under grant POSI/43140/CHS/2001.

## References

1. J. de Sousa, J. P. Marques-Silva, and M. Abramovici, A configware/software approach to SAT solving, in Proc. 9th IEEE Int. Symp. on Field-Programmable Custom Computing Machines, 2001.
2. P. Zhong, "Using Configurable Computing to Accelerate Boolean Satisfiability", Ph.D. dissertation, Department of Electrical Engineering, Princeton University, 1999.
3. I. Skliarova, A.B. Ferrari, Reconfigurable Hardware SAT Solvers: A Survey of Systems, Proceedings of the 13th International Conference on Field-Programmable Logic and Applications – FPL'2003, Lisbon, Portugal, September, 2003, pp. 468-477.
4. I. Skliarova, A.B. Ferrari, The Design and Implementation of a Reconfigurable Processor for Problems of Combinatorial Computation, Journal of Systems Architecture, Special Issue on Reconfigurable Systems, vol. 49, 2003, pp. 211-226.
5. I. Skliarova, A.B. Ferrari, A Software/Reconfigurable Hardware SAT Solver. IEEE Transactions on VLSI Systems, vol. 12, no. 4, Apr. 2004, pp. 408-419.
6. V.Sklyarov, FPGA-based implementation of recursive algorithms. Microprocessors and Microsystems, Special Issue on FPGAs: Applications and Designs, vol. 28/5-6, 2004, pp. 197-211.
7. Available: http://www.celoxica.com/