Accelerating Boolean Implications with FPGAs

Kolja Sulimma, Dominik Stoffel, Wolfgang Kunz

University of Frankfurt, Germany

Abstract We present the FPGA implementation of an algorithm [4] that computes implications between signal values in a boolean network. The research was performed as a master´s thesis [5] at the University of Frankfurt. The recursive algorithm is rather complex for a hardware realization and therefore the FPGA implementation is an interesting example for the potential of reconfigurable computing beyond systolic algorithms.
A circuit generator was written that transforms a boolean network into a network of small processing elements and a global control logic which together implement the algorithm. The resulting circuit performs the computation two orders of magnitudes faster than a software implementation run by a conventional workstation.

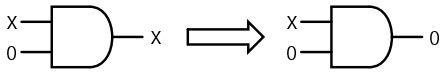

Figure1. Local implication

1 Implications in Boolean Networks

Computing implications for a set of value assignments in a boolean network has many applications, mainly in the area of logic synthesis, testing and verification of digital circuits [3]. For example, it can be used to prune the search space in automatic test pattern generation (ATPG) and to identify circuit transformations in logic synthesis. Figure 1 shows a local implication at a gate with a partial value assignment.

Not all implications of a partial value assignment in a boolean network can be computed by a sequence of local implications at the gates. Figure 2 shows such an indirect implication.

Our implication procedure [4] is known as “recursive learning“ and relies on an AND/OR enumeration of the gates with unjustified value assignments. It computes all implications and represents a solution to the satisfiability problem which is key to many CAD problems.

Recursive learning finds indirect implications by identifying gates for which there are multiple possible value assignments and then recursively computing

the implications for each assignment. If there are implications that are common for all assignments they must be valid indirect implications that can be learned.

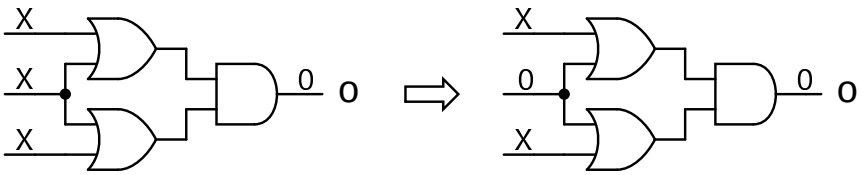

Figure2. Indirect implication

2 FPGA Implementation

The recursive algorithm is rather complex for a hardware realization and therefore the FPGA implementation is an interesting example for the potential of reconfigurable computing beyond systolic algorithms.

The control flow of the recursive AND/OR enumeration is hard to parallelize. Furthermore the control logic is too large to replicate it several times. Furthermore the control flow may be crucial for the success of the algorithm, but most of the runtime is spent in the computation of implications locally at the gates.

Therefore, it was decided to implement a single, sequential control flow and to parallelize the operations on the data structure that represents the boolean network.

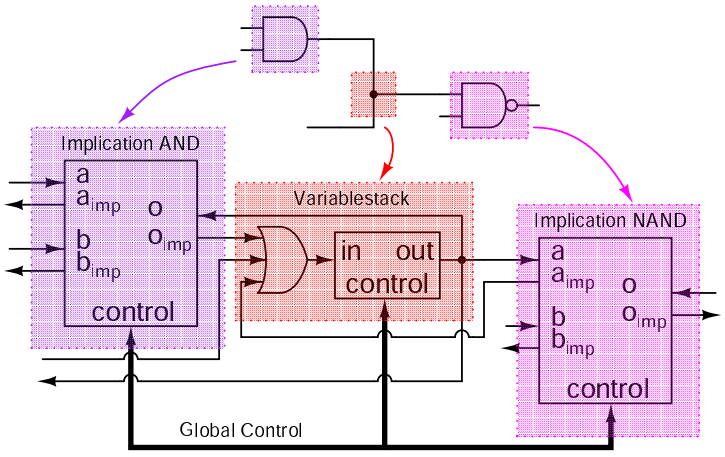

Figure3. Mapping of gates to processing elements.

To take full advantage of the possible fine grain parallelism of FPGAs, a circuit generator was written using the BOOM Package of UC Berkeley [1] that transforms a boolean network into a network of small processing elements (PEs). One element is required for each gate and each variable, respectively, as shown in figure 3. The network of processing elements inherits the structure of the circuit under examination. The PEs communicate exclusively along the edges of the boolean network. A central FSM controller connects to all elements via a few global control signals.

The connections between the PEs are two bits wide because the algorithm operates on four-valued logic. The same encoding as in [6] was used. With this encoding all bits have a monotonic behaviour during the central implication procedure which allows asynchronous operation and simplifies the PE circuits.

There are two types of processing elements, one type representing gates and the other representing variables. Each PE contains the logic necessary to perform the operations required by the algorithm and a stack that allows to save the state of the element up to an recursion depth of 16.

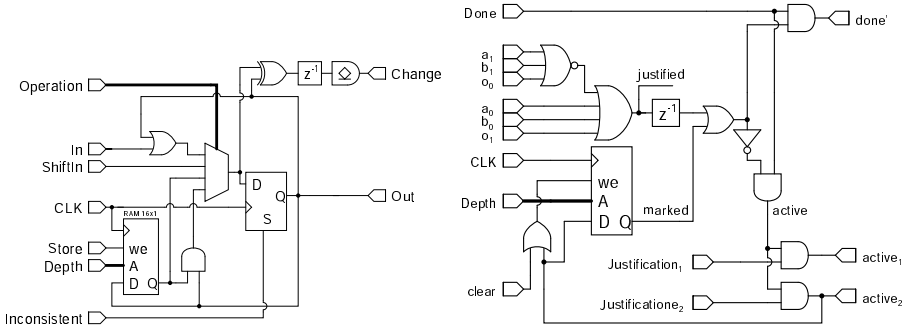

Figure4. PE for a boolean variable (left) and the control portion of a gate PE (right).

Figure 4 shows half of the schematic of a processing element for a boolean variable on the left. It stores the current value of one bit of the encoding of the variable in a flipflop as well as a sixteen entry recursion stack for this bit. The variable type PE can perform five operations on this bit. During initialization the bit can be shifted in from a daisy chain. It can be pushed to or popped from the recursion stack and a logical AND of the current value and the value on the stack can be computed to determine the assignments common to the last recursion call and the one stored on the stack. During the computation of the direct implications a logical OR of the current value and local implications computed by the neighbouring gate type PEs is stored into the flip flop.

The gate type PEs consist of two parts. One controls the AND/OR enumeration in a distributed way. Its schematic is shown in figure 4 on the right. It calculates whether the gate needs to be part of the enumeration (we call this unjustified) and remembers in a recursion stack whether this gate has allready

been part of the enumeration during the current recursion call. This information is used to determine whether this gate should be justified next.

The other part of the gate type PEs compute local implications from the value assignments of the variables connected to the gate. It also assigns justifications to these variables if necessary.

Each processing element requires four Xilinx XC4K logic blocks. This means that with today's FPGAs implications can be performed in circuits with up to 1000 Gates in a single chip. Arrays of multiple FPGAs can be used with only a minor performance degradation, as the implementation still works correctly even if the delay and bandwidth vary from connection to connection.

3 Operation

The FPGA configuration depends on the input network and therefore must be synthesized individually for each boolean network under examination. The circuit can be reused without changes for different initial value assignments.

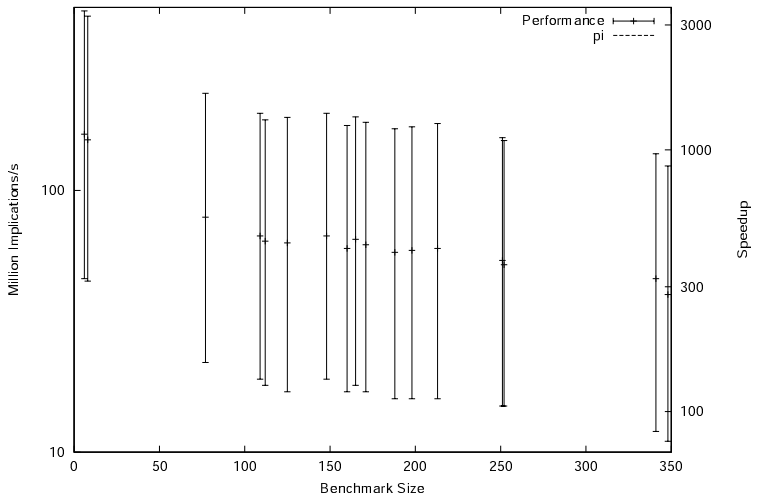

Figure5. Performance of the accelerator by benchmark size

The performance of this approach has been evaluated for ISCAS benchmarks with up to 600 gates. For each benchmark a configuration for XC4000XV FPGAs was generated using the BOOM generator and the Xilinx Foundation synthesis tools. The performance of the various operations required by the algorithm was then analyzed using the Foundation static timing analyzer.

Compared to a 220 MHz Ultrasparc workstation a speedup of two orders of magnitudes was achieved as shown in figure 5. The diagram shows quite large error bars because the parallelism of the computation and hence the achievable

speedup depends highly on the initial value assignment. The center values are typical values for satisfiability problems. The lower values are valid for worst case assumptions that have never been observed during our experiments. The upper bounds are results for assignments close to high fanout nodes.

These results show, that the computation of indirect implications can be sped up by a factor of several hundreds using reconfigurable hardware if the synthesis times can be neglected. This shows that there is a clear potential of reconfigurable computing beyond systolic algorithms. On the other hand, with currently available synthesis tools for FPGAs, resynthesis of a full circuit for each problem instance is a large overhead. In our experiments the place and route times for the larger examples exceeds the typical run time of the software version of the algorithm.

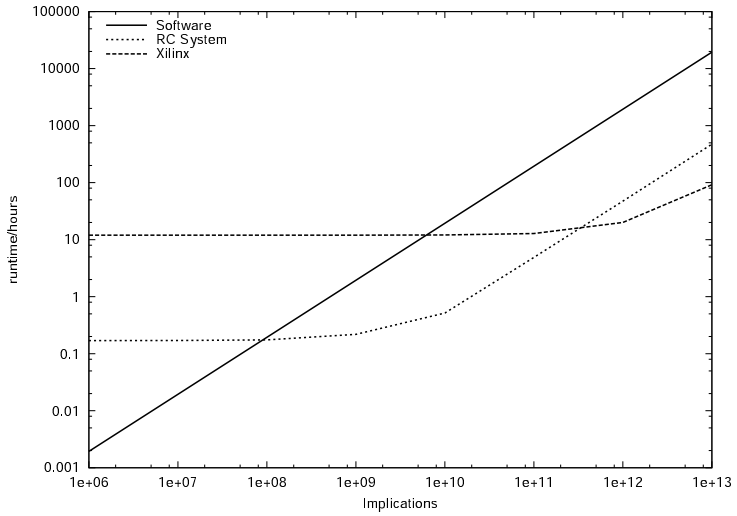

Figure6. Runtime by number of local implications.

Figure 6 shows the runtime including overhead of various implementations of recursive learning by the number of local implications for a 400 gate benchmark. Besides the measurements for the software version and our Xilinx implementation we included a rough estimate for an implementation of our approach on an innovative RC Architecture such as Berkeley's BRASS HSRA or MITs TSF-PGA, [2]. These numbers assume an architecture that allows synthesis of the circuit in 10 minutes but has a ten times lower performance than the highly optimized Xilinx implementation.

The software implementation beats our hardware implementation up to about 5 Giga implications which is about the number of implications usually performed in applications. However, the combined runtime of the hardware implementation

does not increase significantly up to Tera implications. This means, that for typical application we can not improve on the runtime but we can provide a much more thorough search for implications in the same amount of time.

Fast synthesis for reconfigurable computing can potentially push the break even point below 100 Mega implications while still providing a very high peak acceleration. Therefore the development of very fast synthesis algorithms and reconfigurable architectures that support fast synthesis are essential for the presented approach and for reconfigurable computing in general. In the RC context, fast synthesis times are more important than performance optimization of the target circuit.

4 Conclusion

We successfully implemented recursive learning in FPGAs and achieved a very high speedup for typical cases. The high overhead limits the use to applications that spend multiple hours in computing implications in a single circuit, which is quite common in CAD. This limitation hopefully will be eliminated by innovative FPGA architectures and software.

References

1. Michael Chu, Kolja Sulimma, Nick Weaver, Andre DeHon and John Wawrzynek: "Object Oriented Circuit-Generators in Java"; *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines, 1998*
   `http://www.cs.berkeley.edu/projects/brass/documents/Generators_FCCM98.html`
2. Andre DeHon: "Reconfigurable Architechtures for General-Purpose Computing"; A.I. Technical Report No. 1586, Massachusets Institute of Technology, Artificial Intelligence Laboratory, 1996.
   `http://www.ai.mit.edu/people/andre/phd.html`
3. Wolfgang Kunz and Dominik Stoffel: "Reasoning in Boolean Networks"; Kluwer Academic Publishers, 1997. ISBN 0-7923-9921-8
   `http://www.wkap.nl/book.htm/0-7923-9921-8`
4. Wolfgang Kunz and Dhiraj K Pradhan: "Recursiv Learning: A New Implication Technique for Efficient Solutions to CAD Problems Test, Verification and Optimization"; IEEE Transactions on Computer Aided Design, Vol 13, No. 9, pp 1143-1158, Sep. 1994
   `http://www.em.informatik.uni-frankfurt.de/forschung/pubs_arch/iccad94.html`
5. Kolja Sulimma: "Berechnung von Implikationen in Booleschen Netzen mit FPGAs", Universität Frankfurt, 1999. ISBN 3-933966-00-0
   `http://verlag.prowokulta.org/isbn00/`
6. Peixing Zhong, Margaret Martonosi, Pranav Ashar and Sharad Malik: "Accelerating Boolean Satisfiability with Configurable Hardware"; *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1998
   `http://www.ee.princeton.edu/~mrm/pubs.html`