This is a brief overview on the background behind the test set formulas generated by the QBM tool. After establishing its application context, its formal approach to the generation of QBF formulas and the concrete test set formulas are described. Finally, some related work will be credited and the source to obtain the open-source tool will be identified.

1 Application Context

Programmable logic devices enable users to exploit the tremendous concurrency benefits of hardware in their compute solutions without requiring them to go through a silicon production process. The most powerful among these devices are field-programmable gate arrays (FPGAs) that provides hundreds of thousands of logic slices, each one of the capable of computing any arbitrary boolean 6-input function [1]. The configuration of these devices is computed in a synthesis process, which translates a behavioral or functional specification into a structural netlist consisting of the configurable primitives found on the devices and their configurations. These configurations include, for instance, truth tables to be stored in lookup table (LUT) components as well as datapath configurations that establish the proper signal connections.

The synthesis process may start from traditional hardware description languages like VHDL [2], from high-level imperative languages like C++ [3] or from functional description in languages like Chisel [4]. In any case, the specified behavior will result in combinatorial equations that in their entirety define the function for the transition from one system state to the subsequent one also considering external inputs and computing outputs as specified. The efficient mapping of these combinatorial functions to the structural capabilities of the targeted device architecture is crucial as it defines the costs of hardware and power, and even limits the application range of a device. Heuristics transforming data structures like inverter graphs (AIG) [5] are used to balance or reshape the dataflow of the combinatorial computation so that it fits well onto the target structure. Decomposition approaches may be used to identify the utility of special-purpose resources such as carry chains for the implementation of a given function [6].

The mapping heuristics used by modern synthesis tools are very mature and deliver high-quality results. However, there is a growing number of data center applications, most prominently neural network inference [7], which rely on a highly parallel computation carried out by massively replicated operator cores. For them, it is a paramount benefit to use the optimal rather than just a very good implementation of this operator. This makes it also worthwhile to invest extra effort that goes significantly beyond what would be acceptable in a regular synthesis run.

This project has received funding from the European Union’s Framework Programme for Research and Innovation Horizon 2020 (2014-2020) under the Marie Skłodowska-Curie Grant Agreement No. 751339.
2 QBF Formulation

The mapping of a user-specified function to a given reconfigurable logic structure can be expressed as a QBF satisfiability problem as detailed below. Solving this problem yields precise statements:

**UNSAT** that there is no implementation within the range of the granted resources, or

**SAT** that such an implementation, indeed, exists.

It is remarkable that a satisfying assignment of the variables under the outermost existential quantifier of the formulas – which all follow an EAE pattern – corresponds directly to an implementing configuration of the programmable circuit. On the other hand, a problem that has been found unsatisfiable is no longer worth pondering as no engineering effort and experience will yield a solution under the given resource constraints.

A digital combinational reconfigurable circuit can be modeled by its characteristic function \( \Phi \) mapping its \( k \) boolean configuration variables and its \( l \) boolean inputs to its \( m \) boolean outputs:

\[
\Phi : B^k \times B^l \rightarrow B^m
\]  

\[
(c, x) \mapsto y
\]

with \( c = c_0, \ldots, c_{k-1} \)

\( x = x_0, \ldots, x_{l-1} \)

\( y = y_0, \ldots, y_{m-1} \)

\( B = \{0, 1\} \)

The arguments \( c \) specify a concrete configuration of the circuit. They are provided through internal SRAM cells or fuses. Their update is relatively slow or impossible altogether. Hence, the configured values are kept constant over much or even all the operation time.

The arguments \( x \), on the other hand, are typical functional inputs that are provided dynamically in the course of operation. They are evaluated in the context of the current configuration, which defines the effective functionality of the circuit. The circuit is capable of implementing a target user function \( f : x \mapsto y \) if and only if:

\[
\exists c. \forall x. \Phi(c, x) = f(x) \tag{2}
\]

A satisfying assignment \( C : c \mapsto B^k \) identifies an implementing configuration for \( f \).

Eq. (2) is already a valid formulation of the problem of mapping a target user function \( f \) to a configurable circuit with the characteristic function \( \Phi \). Both functions can be joined into a single formula:

\[
\exists c. \forall x. F'(c, x) \tag{3}
\]

A very practical format of \( F' \) is a conjunction of clauses. This makes it a collection of constraints that must be all satisfied. It can, thus, be constructed incrementally. This property is very valuable as it enables the compilation of the specified model into its QBF formulation processing statement after statement.

So as to further simplify the formulation of \( F' \), it is helpful to introduce internal node variables \( n \) that allow the composition of a constraint from simpler subterms or subcomponents. Such node variable may, for instance, represent the interconnections to a component in a structural design. So, the final structure of our problem statement is:

\[
\exists c. \forall x. \exists n. F'(c, x, n) \tag{4}
\]

Solving Eq. (4) then determines whether or not there is a configuration of \( c \) of the configurable circuit so that for all input combinations over \( x \), there is an assignment to the internal node variables \( n \) so that:

- the circuit works physically correct and satisfies all component specifications, and
- the circuit implements the specified user function \( f \).

Note that \( F \) contains two parallel models of how the inputs \( x \) relate to the outputs \( y \subset n \). One model specifies the relation as defined by the user and the other relates the same inputs to the outputs via the configurable circuit description. Observe that the outputs \( y \) are subsumed by the internal node variables in this formulation. Also note that it is only their role that differentiates the variables into the three sets \( c, x \) and \( n \). Otherwise, they may all be viewed as signals or wires carrying a single bit.

For producing a problem statement in QDIMACS format, \( F \) must be specified in conjunctive normal form. This normal form is produced internally by the QBM tool, which is freely available on GitHub[1]. The engineer can specify both

[1] https://github.com/preusser/qbm
the configurable circuit as well as the desired function by structural and behavioral descriptions as described by Preußer and Erxleben [8].

3 Test Set Formulas

The test set formulas are problem statements that ask for the configuration of the lookup tables and for the selection of their appropriate inputs so as to implement binary word adders of different bitwidth within a Xilinx carry-chain structure [11]. All problem statements are satisfiable. The test set includes adder width of four through seven bits. Recent experiments have shown that this range can be expected to span from rather simple all the way to currently infeasible problem complexities [8].

The test set includes, for each problem size, three different formulations for the selection of the inputs to a lookup table from the overall set of inputs, which includes logic one and zero in addition to the actual operand words:

1. Each LUT input is driven by a configurable multiplexer (CMUX) whose configuration determines, which one of all the available input signals is passed through.

2. The first LUT input is driven by a CMUX selecting one from all the available input signals. The CMUX modeled for each subsequent LUT input can only choose from one fewer input signal than the predecessor.

3. A \textit{CHOOSE} operator is driving all the inputs of the LUT. Its configuration determines, which \( k \) of all the available inputs are forwarded. For each possible selection, it only allows one single permutation.

The first option is a naive approach. It models the device capabilities well but introduces practically useless degrees of freedom. For instance, it is irrelevant in what order inputs are provided to a LUT, and it is pointless to feed the same input twice. The second option takes a simple approach to reduce the available degrees of freedom somewhat but only to a degree that each choice for a LUT input can be made individually. The third and last option goes the furthest. However, it is based on identifying the one out of \( \binom{n}{k} \) combinations that selects the needed set of \( k \) inputs from the \( n \) available ones.

So as to appreciate the differences of these different models of the input selection, consider the number of configuration variables that is needed to encode this selection. In the first naive model, we require \( k \cdot \lceil \log n \rceil \) input variables; in the third case, \( \left\lceil \log \left( \frac{n}{k} \right) \right\rceil \) ones. We have:

\[
k \cdot \lceil \log n \rceil \geq \left\lceil k \cdot \log n \right\rceil \geq \left\lceil \log \left( \frac{n}{k} \right) \right\rceil
\]

Observe that the final inequality captures a gap that grows significantly with \( k \).

The drawback of the third approach is that the encoding of the configuration for the selection spans all configuration variables and becomes part of all the clauses that must be generated. The local configurations of the first two approaches, on the other hand, comprise at most \( \lceil \log n \rceil \) variables producing much shorter clauses.

4 Related Work

Previous formal approaches to the boolean matching of combinational computations to FPGA structures were based on a SAT expansion of the problem [9][10]. These approaches are, thus, able to benefit from the enormous advances in the technology of SAT solving [11]. However, the formulation of the matching problem in SAT also implies a heavily inflated problem specification.

This paper demonstrates how the mapping problem is more naturally formulated as a QBF. This approach avoids the inflation of the problem specification by literally reiterating the set of the defining clauses with fresh variables for every possible input combination. Note that this inflation is, in fact, exponential in the number of Boolean inputs. Internally, QBF solvers are not unlikely to utilize SAT solvers themselves [12][13]. However, exposing the natural structure of the problem to the solver and using a solver specialized for this structure, can only benefit the developed tool. The solvers are free in the organization of their search for a solution and may cut away parts
of it on the basis of their domain-specific knowledge. An impetuous naive SAT expansion can only disguise higher-level information on the problem structure and neglects the existence of appropriate tools.

5 Availability

QBM is open-sourced and available at https://github.com/preusser/qbm. It is published together with quick starting instructions and simple example problems like the presented adders.

References

[1] Xilinx, “UG 474: 7 Series FPGAs configurable logic block,” Sep. 2016.
[2] “IEEE standard VHDL language reference manual,” IEEE Std 1076-2008 (Revision of IEEE Std 1076-2002), Jan. 2009.
[3] Xilinx, “Vivado high-level synthesis,” https://www.xilinx.com/products/design-tools/vivado.html.
[4] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman, R. Avižienis, J. Wawrzynek, and K. Asanović, “Chisel: Constructing hardware in a Scala embedded language,” in DAC Design Automation Conference 2012, Jun 2012, pp. 1212–1221.
[5] A. Mishchenko, R. Brayton, and S. Chatterjee, “Boolean factoring and decomposition of logic networks,” in Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design, ser. ICCAD ’08. Piscataway, NJ, USA: IEEE Press, Nov 2008, pp. 38–44.
[6] T. B. Preußer and R. G. Spallek, “Enhancing FPGA device capabilities by the automatic logic mapping to additive carry chains,” in International Conference on Field Programmable Logic and Applications (FPL) 2010, IEEE, Ed., Aug 2010, michael Servit Award Winner.
[7] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, “FINN: A framework for fast, scalable binary-valued neural network inference,” in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA 2017), ser. FPGA. New York, NY, USA: ACM, Feb 2017, pp. 65–74.
[8] T. B. Preußer and F. Erxleben, “Design kernel exploration using qbf-based boolean matching,” in IEEE Workshop on Signal Processing Systems (SiPS 2016), Oct 2016.
[9] A. C. Ling, D. P. Singh, and S. D. Brown, “FPGA PLB architecture evaluation and area optimization techniques using boolean satisfiability,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 26, no. 7, pp. 1196 – 1210, Jul 2007.
[10] S. Safarpour, A. Veneris, G. Baekler, and R. Yuan, “Efficient SAT-based boolean matching for FPGAs,” in Proceedings of the 43rd Annual Design Automation Conference. New York, NY, USA: ACM, 2006, pp. 466–471.
[11] M. Järvisalo, D. L. Berre, O. Roussel, and L. Simon, “The International SAT Solver Competitions.” AI Magazine, vol. 33, no. 1, 2012.
[12] H. Samulowitz and F. Bacchus, “Using SAT in QBF,” in CP, ser. Lecture Notes in Computer Science, P. van Beek, Ed., vol. 3709. Springer, 2005, pp. 578–592. [Online]. Available: http://dblp.uni-trier.de/db/conf/cp/cp2005.html#SamulowitzB05.
[13] F. Pigorsch and C. Scholl, “An AIG-based QBF-solver using SAT for preprocessing,” in Proceedings of the 47th Design Automation Conference, ser. DAC ’10. New York, NY, USA: ACM, 2010, pp. 170–175.